TECHNISCHE
HOCHSCHULE
LÜBECK

# Bachelor thesis

# Pixel Streaming of interactive 3D applications using the Unreal Engine 4 IDE


Submitted by: Hou, Shiying

Department: Electrical Engineering and Computer Science

Degree program: Information Technology

First examiner: Herr B.Sc. F. Anthony


Date handing out: 8th March 2020
Date handing in: 8th June 2020

(Professor Dr. Andreas Hanemann)
Head of Examination Board

Page 2 Topic of the Bachelor thesis for Ms. Hou, Shiying

**Task description:**

With the help of a feature known as Pixel Streaming, a three-dimensional application, created and packaged with the Unreal Engine 4 integrated development environment, is capable of running remotely on a desktop PC or server in the cloud and, using only a small stack of additional web services, provides high levels of processing that is virtually independent of the capabilities of a client's terminal.
The aim of this bachelor thesis is to explore this advanced technology and demonstrate the acquired level of understanding by developing (and providing details thereof) a Car Configurator application that can run as a streaming application from a test server. Users will be able to connect to functionally interact with the 3D application by connecting to the test server using any modern web browser that supports HTML5.

Herr B.Sc. F. Anthony

TECHNISCHE HOCHSCHULE LÜBECK

**Department of**
**Electrical Engineering and Computer Science**
**Dean's Office - Board of Examiners**

**Statement on the bachelor thesis**

I assure you that I wrote the work independently, without outside help.

Only the indicated sources have been used in the preparation of the work.
The text or the sense of the text are marked as such.

I agree that my work may be published, in particular that the work may be submitted to third parties for inspection or that copies of the work may be made for forwarding to third parties.

16/2020
Date

Signature

# Abstract

When developing network services, the challenge of sharing experiences is usually a crucial point in the collaboration and development. In this new era of experience, people generally respond best to the engaging and immersive interaction with the content. With an increasing amount of content with interactive experiences, the question of distributing the content to various platforms arises.

WebGL and HTML5 are two common solutions to distribute the content in web browsers. The performance of the rendered content, to a considerable extent, relies on the capabilities of the receiving browser and hardware on the client side; the quality of the graphics depends on the browser's display capabilities. This means that in order to share the content to the maximum number of users, either the shared application is defined by the lowest acceptable device, or multiple versions of the application are required to distribute to different platforms.

The Unreal Engine team at Epic Games set out to break the bottleneck and developed Pixel Streaming. Introduced in Unreal Engine 4.21, Pixel Streaming allows to deliver the interactive real-time content to various devices, simultaneously, with the highest possible quality, and keep the experience from the client side. Because the shared application stays and runs on the hosting server, such a distributing solution can maintain the fidelity and interactive features, regardless of the end devices, and provide high-quality graphics than a client-side solution.

This thesis explores the fundamental of the advanced technology. A car configurator application, created and packaged with the Unreal Engine integrated development environment (IDE), runs as a streaming application using the Pixel Streaming Plugin implemented in Unreal Engine 4 (UE4). The development process of the application and the results of the streaming method are presented in detail. With the aid of several tests and the comparison with other distribution solutions, the functionality, usability and utility of the Unreal Engine application together with Pixel Streaming are assessed.

# Acknowledgement

First, I would like to give my sincere gratitude to my supervisor Fabio Anthony for his effective supervision, thought-provoking comments, unfailing attention to the thesis as well as his kindness I could always feel, giving me encouragement and confidence. Without his guidance and help, this thesis would have been impossible. My heartfelt thanks also go to Lenka Kleinau, the head teacher during my undergraduate study in Technische Hochschule Lübeck (THL), for her help of the inspiriting seminars from which I have benefited a lot. I am also sincerely grateful to all the teachers and friends for their cooperation, comments, and wise advice during my college days. Special thanks should also be expressed to my family for supporting and encouraging me and my education all my life. Last but not the least, I offer thanks to all those who have helped me once in my life.

# Contents

# 1 Introduction

This chapter describes in which context the thesis is written. The first section declares the general motivation for the subject. The second section describes the project goals and steps how to achieve it. Then the structure of the thesis is outlined. It follows a description of the development methods and workflow. Finally, the used tools are listed and introduced.

## 1.1 Motivation

Thanks to the combination of technological progress in artificial intelligence, social messaging, Internet of Things (IOT) and mobile connectivity, we have moved beyond the Information Age to the Experience Age. In the Information Age, the idea of communication was to make information available and accessible. In the Experience Age, the primary focus is to create an experience – people want to see, feel, and connect. [6] Society is transitioning from wanting information to seeking experiences related to the information. Instead of being served as by-standers, people prefer to be immersed in the experience they want.

In this Experience Age, people respond best to the creative, high-quality, engaging, and immersive content. An increasing amount of this content incorporates interactive experiences, and 3D models as well as AR or XR overlays have been introduced. [7] In order to attract the most users, producers need to make this content available on different platforms, including smartphones, tablets, PCs, and interactive displays (Figure 1-1).


**Figure 1-1** Shared content across various platforms. Image adapted from [7].

A common approach is to determine the lowest demand of entry and acceptable quality for the target user, and then develop the specified platform based on that. For example, with common or current solutions based on WebGL or HTML5, the integrity and interactivity of the displayed content, to a considerable extent, depends on the consumer's device, especially its hardware, display mechanism, and operating system [7]. This means that to make the content available for the most users, either the shared application is developed based on the lowest acceptable device, or multiple versions of the application must be developed to meet the needs of different user groups. In such a situation, the device that is responsible for displaying real-time content for the end user needs to download the pertinent data and logic, and render the results to the screen. This will not only limit the quality of the experience for all end users, but sometimes needs to sacrifice the image quality for the sake of smaller file sizes and faster downloads as well.

The Unreal Engine team at Epic Games tried to break the usual distribution barrier when streaming high-quality content, and developed Pixel Streaming. Pixel Streaming allows to distribute interactive real-time content, simultaneously, to any device with the highest possible quality, and keep the experience at the client side [8].

Pixel Streaming works as an "out-of-the-box" [8] solution via a plugin for Unreal Engine 4 (UE4). This is accomplished by separating the shared application itself, with the sophisticated hardware running it and streaming the video and audio to the user's device. With the help of Pixel Streaming, a three-dimensional application, created and packaged with UE4 IDE, is capable of running remotely on a desktop PC or a server in the cloud, and using only a small stack of additional web services to share high-end experiences without considering the end client or platform.

In this context, it is worth demonstrating this technology by using a 3D interactive application as an example and investigating how well it distributes real-time content for various devices and gives people a favorable experience. The growing visual quality and flexibility of real-time content seem to have created changes in the industry. Hence, a research and discussion on the advanced technology can be conducted.

## 1.2 Goal

Against this background, the main goal of the thesis is to explore the advanced technology and demonstrate the acquired level of learning by developing a car configurator application that can

run as a streaming application from a test server. Users will be able to connect to the test server using any modern web browser that supports HTML5 and functionally interact with the 3D application.

The Unreal Engine (version 4.24.3) will be used as the main tool in the project context and as a game engine to create the car configurator application. The application as well as the streaming technology will be finally evaluated in terms of functionality and usability.

## 1.3 Thesis Structure

The thesis is structured in six general chapters. After the introduction was given in Chapter 1, Chapter 2 gives at first a general idea of web rendering strategies and covers the fundamental of Unreal Engine 4 that will be used in the example application. The theoretical basement of Pixel Streaming is illustrated in Chapter 3. Subsequently, Chapter 4 describes the implementation details of building the car configurator application and applying Pixel Streaming. Chapter 5 then discusses the results with respect to the functionality and usability based on the performance of the application and a brief user survey. Finally, the conclusion sums up the project findings and the outlook is given at the end of the thesis in Chapter 6.

## 1.4 Development Method

In order to stay flexible during the development, the whole thesis project is managed based on Scrum, an agile project management methodology or framework. It helps advance the project and deliver new application capability about every 2 weeks.

The task planning is managed with a digital Kanban board tool GitKraken Glo that is used for task and issue tracking. The whole management and progress of the project will be reflected on the Kanban board and the project-related materials are managed with GitHub, a Git repository hosting service, and GitKraken, a Git GUI client.

All the tools used in the project's context are listed and described below in section 1.6. The project is developed in Unreal Engine 4 based on blueprints. It is the visual scripting system inside UE4 and is a fast way to start prototyping the target application. Creating 3D models will not be part of the author's work in the thesis context. Therefore, most materials are found on the Internet and some of them may be modified. The source of these materials is referenced in

the appendix (see Appendix D). Some design scribbles and digital icons are produced in Adobe Photoshop.

## 1.5 Development Workflow

At the beginning of the thesis, the focus obviously lies in researching and studying the existing approaches and the required software that relate to the project. After this stage, the development of the car configurator application is the major part. This stage is the iteration for the visual design, and continues as long as a significant progress is made. After completing the application, Pixel Streaming is applied and evaluated. The project needs to fit into the described development methods in section 1.4.

## 1.6 Tools

This section covers the tools that are used during the project. Unreal Engine 4, Blender, and Adobe Photoshop are utilized during the development of the car configurator application. The remaining software is used for project planning and management.

**Unreal Engine 4**

The Unreal Engine is a game engine developed by Epic Games. The latest release is Unreal Engine 4, which launched in 2014. Most of the functional and interactive prototyping are done with the UE4, based on Blueprint Developer and Unreal Motion Graphics (UMG) for graphical user interface (GUI) implementations. The engine also provides fast and accessible options for packaging and exporting a project with much ease. More essential content about UE4 will be introduced in section 2.2 in detail.

**Blender**

Blender is a free and open source 3D computer graphics software toolset that supports the entirety of the 3D pipeline, such as modeling, rigging, animation and so on [9]. In order to perfect the car model, this software is only used to modify some model settings before importing into the UE4. Since only the simple setting of the model is required, Blender is suitable for beginners. The author has only learned the basic operations in Blender.

**Adobe Photoshop**

Adobe Photoshop is an excellent graphic design software. It is widely used by photographers for photo editing and by graphic designers and web designers to create and change images for web pages. The author has expert knowledge of this software and has used it to make some icons or figures required.

**Git & GitHub**

Git is a type of version control system (VCS). Git stores a project or a set of files in a data structure called a repository and helps you track changes to files. It is distributed, where users install locally to manage source code. GitHub is a Git repository hosting service and provides a web-based graphical interface [10]. GitHub is an online service to which developers who use Git can connect and upload or download resources.

**GitKraken & GitKraken Glo**

GitKraken is a Git GUI client which makes developers more productive and efficient with Git. GitKraken Glo is a digital Kanban board that is fully integrated with GitKraken. It is used for task planning during the development, supporting different categories for tasks, assigning people and dates, labelling tasks, and plenty of useful functions for project management.

**Jitsi Meet**

Jitsi Meet is a free video conferencing for web and mobile. It is used to have a video conference on a webpage without installing any software. The author uses this to have the video conference on this thesis with the supervisor.

# 2 Fundamentals

In this chapter an overview on web rendering and common rendering strategies is first given. The second section then describes the basic essential content of Unreal Engine 4. Only information relevant for this thesis will be covered.

## 2.1 Rendering on the Web

When talking about building a website, this can include building out the component model, handling routes and transitions, managing data and state, and even rendering and loading the assets (Figure 2-1). This section mainly focuses on the rendering part.

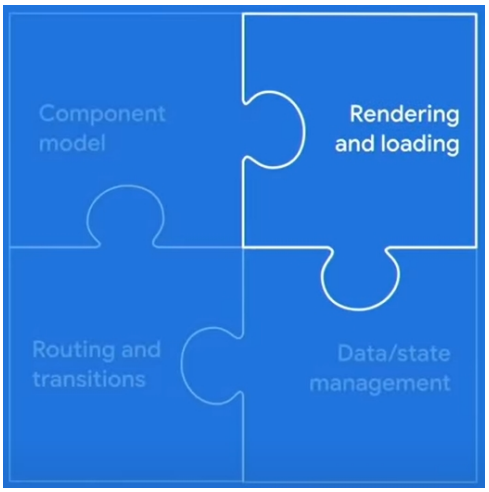


**Figure 2-1** Application architecture on web. Image adapted from [11].

Web rendering is a process of generating HTML output that is ready to be rendered by the web browser and viewable by users. Generally, a browser will request HTML, CSS, JavaScript, and image content from a server and illustrate the page to users according to certain web standards and specifications. While initializing a web page, the browser parses the data from the server in multiple formats and create a visual effect. This process can happen anywhere and anytime depending on the rendering strategy. [12]

### 2.1.1 Modern Web Rendering Strategies

There are several strategies to choose where and when for rendering. Here is a brief introduction

to five common rendering strategies. They are Static Rendering, Prerendering, Client-Side Rendering (CSR), Server-Side Rendering (SSR), and Universal Rendering [12].

**Static Rendering** means utilizing a content delivery network (CDN) to host the recently produced static page content and generating the page ahead of time.

**Prerendering** enables to preload all the elements on the expected page ahead of time so that visitors can see the next page faster.

**Client-Side Rendering (CSR)** means that the browser downloads the entire page and renders the JavaScript to fill the contents. All the processes including data and logic fetching, templating and routing are managed on the client side.

**Server-Side Rendering (SSR)** allows interpreting the page content on the server side, which avoids additional round-trips for fetching data, creating the page, and sending it back to the client.

**Universal Rendering** is a combination of CSR and SSR. Large parts of the data and logic can run on the server and on the browser, attempting to reach the balance between both CSR and SSR.

CSR and SSR are the most relevant to this thesis, so they are described below in detail.

### 2.1.2 Client-Side Rendering (CSR)

The normal steps of Client-Side Rendering are listed below (Figure 2-2):

- The user visits the web page in the browser, sending a request to the website.
- The browser receives and downloads the supporting files (HTML, CSS, JavaScript, etc.) from the server or CDN.
- The browser executes JavaScript codes to fetch dynamic content from the API server.
- The final content is **rendered** and **interactable** through DOM processing in the web browser.

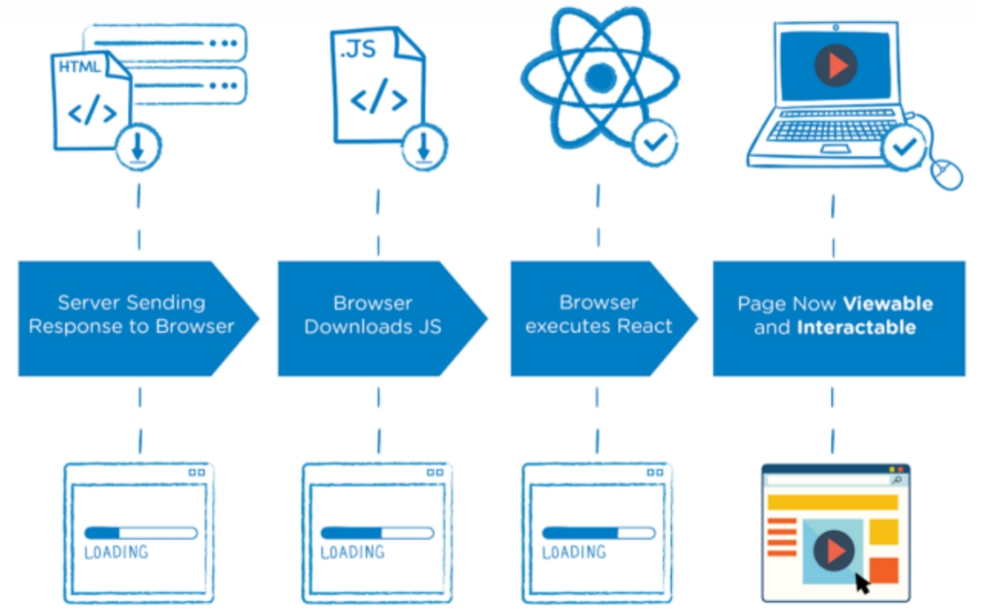


**Figure 2-2** Process of Client-Side Rendering. Image adapted from [13].

Most main tasks are done on the client's browser, so it is called Client-Side Rendering. With CSR, only the data is transferred back and forth and most resources are loaded once during the whole lifespan.

### 2.1.3 Server-Side Rendering (SSR)

Server-Side Rendering generally follows the steps below (Figure 2-3):

- The browser sends a request to the website and the server creates the "Ready to Render" [14] HTML files.
- The web page content can be quickly **rendered** by the browser but it's not interactive.
- The browser downloads and executes the JavaScript.
- Now the interactions can be executed and the page is **interactive**.

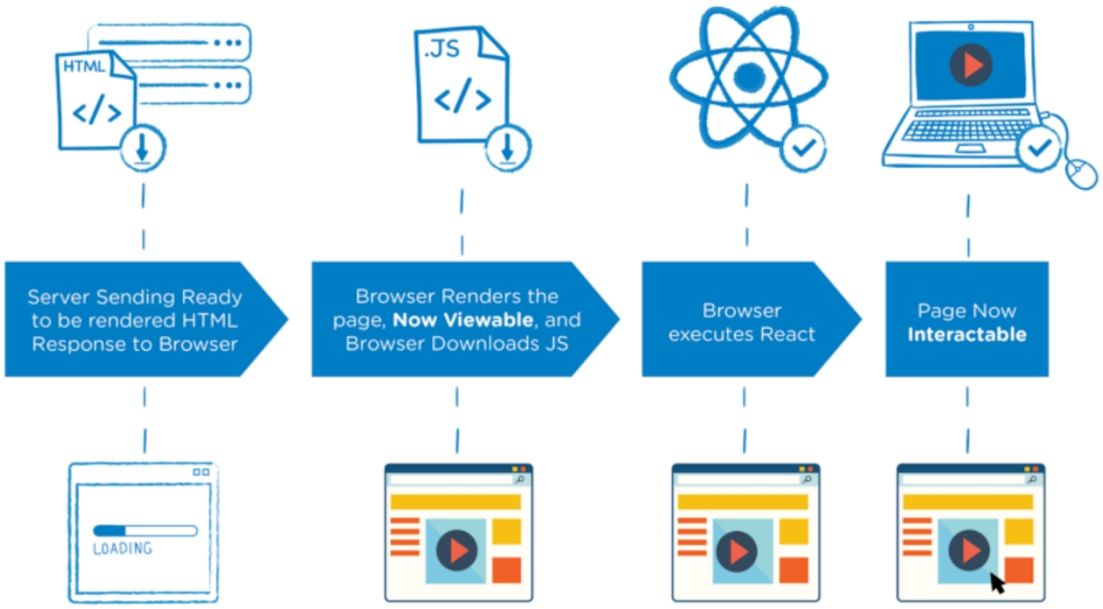


**Figure 2-3** Process of Server-Side Rendering. Image adapted from [13].

Server-Side Rendering prepares the web page content with data directly on the server. Therefore, it is commonly faster to deal with the requests in comparison to Client-Side Rendering, but it comes with a burden on the server side.

### 2.1.4 Differences between CSR and SSR

Since the two rendering strategies are different in content processing, each of them has its own benefits and drawbacks.

For Client-Side Rendering,

Benefits are:

- Fast website rendering after the initial load;
- Rich site interactions;
- Less server resources required.

Drawbacks are:

- Unfriendly search engine optimization (SEO);
- Slower initial load;
- Much burden on the browser.

For Server-Side Rendering,

Benefits are:

- Better search engine optimization (SEO);
- Faster initial page load;
- Great for static sites;
- High performance on a wide range of devices.

Drawbacks are:

- Relatively slow overall page rendering;
- Non-rich site interactions;
- Less flexibility for deployment.

Table 2-1 gives a clear comparison between CSR and SSR. For CSR, the rendering part is mainly done on the client side. For SSR, the page content is first interpreted on the server side and then sent to the client, which reduces the burden and requirements of the client. The answer to which one is better cannot be given. It depends on what you want to do and what you want your page to look like.

| | Client-Side Rendering | Server-Side Rendering |
|---|---|---|
| **Search engine optimization (SEO)** | Bad | Good |
| **Fast initial page load** | No | Yes |
| **Full page reload when navigating** | No | Yes |
| **Server hosting required** | No | Yes |
| **Better user experience** | Yes | No |

**Table 2-1** Comparison between CSR and SSR. Table made from Microsoft Excel.

## 2.2 Basics of Unreal Engine 4

The Unreal Engine, created by the game and software development corporation Epic Games, is one of the world leading game engines. It offers all basic functionality to create modern games

or other applications and supports multiple platforms, such as Linux, Windows, macOS, iOS, Android and HTML5. Furthermore, Unreal Engine 4 provides various free project templates, sample contents, learning projects and plenty of other resources in different fields to lead an easy and quick start. Coding can be done by either the unique visual scripting language Blueprints or by C++ programming. While C++ has more access to the hidden part of the engine, Blueprints are easier to get started. This thesis's project is done completely by the UE4 Blueprint Visual Scripting.

In the following part, only a general overview will be given on the parts that are pertinent to this thesis. Considering this, the below-mentioned topics will be briefly covered:

- Blueprints
- Static Mesh and World Coordinates
- Materials and Textures
- Unreal Motion Graphics (UMG)

### 2.2.1 Blueprints

The Blueprints Visual Scripting system in Unreal Engine is a scripting system using a node-based interface to visually connect events, functions, or variables with wires, thus creating complex functionalities [4]. The two most common used types of Blueprints are Level Blueprints and Class Blueprints.

**Level Blueprints** are global and possess level-wide functionalities meaning that they can also communicate across Class Blueprints. By default, each level is equipped with one Level Blueprint, and this can reference and manage Actors and other things within the level [4]. The Level Blueprints can interact with Class Blueprints (see below) placed in the level.

**Class Blueprints**, on the other hand, are used locally and contain their own variables and functionalities. Most Class Blueprints inherit from several parent classes. The most important is probably the Pawn class which is the physical representation of a player or AI entity in the game world.

Main events to perform the code are *Event BeginPlay* and *Event Tick*. The *Event BeginPlay* event only happens after the corresponding Blueprint starts. The *Event Tick* event is called on every frame and is very expensive regarding the performance since every piece of code will be executed on every game loop. [15]

### 2.2.2 Static Meshes and World Coordinates

Static Meshes are geometries with a set of polygons which make up the main part of the world architecture for levels created in Unreal Engine. They are basic units that can be rendered by the graphics card. The 3D models of Static Meshes can be imported from the external modeling software through the Content Browser. In this project, most of the Static Meshes are offered by Unreal Engine. Only the car model is from the Internet.

The world coordinate system of Unreal Engine 4 is a Cartesian three-dimensional coordinate system (X, Y, Z or R, G, B) where 1 Unreal Engine unit (UU) equals to 1 centimeter.

### 2.2.3 Materials and Textures

In computer graphics, 3D polygons are composed of triangles or squares form the worlds. The most basic and efficient polygon is the triangle and it can be drawn and colored to the screen at high speeds. The process of generating 2D projections of 3D shapes and selecting colors, lights, shadows, or reflections to the triangle is called rendering. [16]

In Unreal Engine, the Material Editor is provided to color 3D shapes. This tool produces visual elements that can be applied as paint to the surfaces of models. There are a lot of parameters can be adjusted in UE4, such as the color, shininess, roughness, opacity, etc. These different visual settings combine a material and it can be applied to a mesh to offer different visual look.

A texture is defined as a two-dimensional image that used in Materials and can be mapped onto a 3D shape. Texture coordinates present how an image is mapped to a 3D shape. The coordinate system is characterized by the UVs where U is the horizontal location and V presents the vertical location.

### 2.2.4 Unreal Motion Graphics (UMG)

To show users more information, typically 2D user interfaces (UI) are being implemented. In Unreal Engine 4, the Unreal Motion Graphics (UMG) tool is provided to create UI elements such as in-game HUDs, menus, or other interfaces. It is implemented within Widget Blueprints in which developers can design information texts or add buttons or sliders to realize different functions.

# 3 Pixel Streaming

This chapter dedicates to give an overview of Pixel Streaming, an advanced technology in Unreal Engine 4. Then several common setups of Pixel Streaming are covered. The chapter additionally discusses the benefits of this technology.

## 3.1 Pixel Streaming Overview

People generally run the Unreal Engine application on the device which runs the gameplay logic and presents the game world on the screen. Despite of the distributed parts of the gameplay logic, the task of rendering the game world needs to be done on each device. Even though the HTML5 version of the application is used to run in the web browser, the gameplay logic and the rendering part still run on the client side.

With Pixel Streaming, people can run the Unreal Engine application remotely across a network, using only a small stack of additional web services. Either a physical desktop in the local network or a remote virtual machine can be used to run the gameplay logic and rendering. The video and audio of the Unreal Engine application is encoded into a stream and then sent to separate devices. People can connect with any modern web browser on their devices, whether desktop or mobile, and view the pre-rendered frames and audio without installing or downloading anything (Figure 3-1). Also, users can interact with the application with a variety of different input methods, which is sent back to the application.

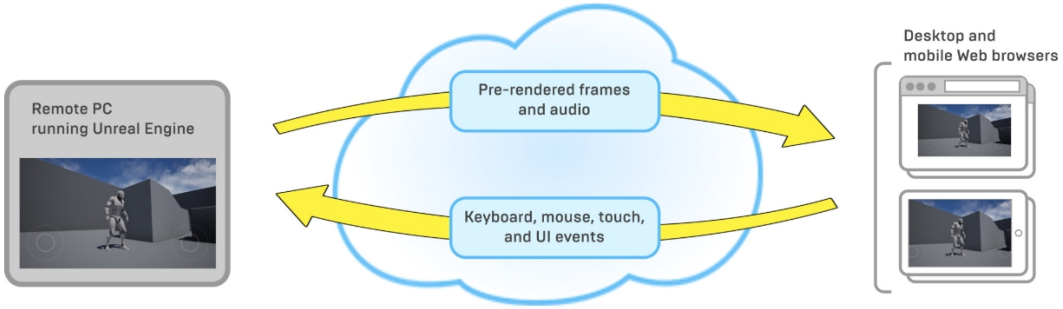


**Figure 3-1** Pixel Streaming Overview. Image adapted from Unreal Engine Documentation [4].

Pixel Streaming is implemented as an Unreal Engine plugin released in UE 4.21. The Pixel

Streaming Plugin in UE4 provides all the required components. Once the plugin is enabled, all you need to do is to start the provided server. It provides sort of a real-time interaction and can be customizable.

### 3.1.1 Technologies

Pixel Streaming mainly uses the following technologies:

**NVIDIA NVENC**

NVIDIA NVENC is a feature in the graphics cards. It is used to code the video stream into H.264, a video compression standard in hardware. This can provide up to 4K, 60 FPS streams, which is a high performance.

**WebRTC**

WebRTC is short for Web Realtime Communications, which is a powerful and advanced technology and standard. It can be added to some applications to obtain the peer-to-peer real-time communication capabilities based on open standards. With WebRTC, people can share video, audio, and data in the web browser, without plugins.

WebRTC is used to do the distribution in the Pixel Streaming system. Actually, it is used right now in many places, from basic web applications using cameras or microphones to advanced video sharing applications, like the video chat apps using today. WebRTC allows to decode the H.264 video on the hardware; as a result, a high hardware specification is not required. All the primary browsers can support it natively nowadays. Additionally, WebRTC can automatically adapt to poor network conditions. It will track and update the video if the user has a bad connection.

**Standard web technologies**

In terms of the web stack, a standard Node.js server and a JavaScript implementation are used in the Pixel Streaming system.

### 3.1.2 Components

**Pixel Streaming Plugin**

The Pixel Streaming Plugin is integrated directly inside the Unreal Engine 4 where NVIDIA NVENC is integrated. It uses H.264 video compression to encode the video stream and sends the stream to the WebRTC Proxy (see below).

**WebRTC Proxy (Removed since UE 4.24)**

The web browsers are connected to WebRTC Proxy rather than the Unreal Engine application. It uses WebRTC to distribute the video and audio streams to web browsers.

Note that WebRTC Proxy is removed since UE 4.24. The web browsers are connected to the Signaling and Web Server (see below) directly and the media stream encoded by the Pixel Streaming Plugin is sent to the server.

**Signaling and Web Server**

The role of the Signaling and Web Server is to coordinate the connections between two ends and host the HTML page that the user goes to.

**STUN and TURN Server**

WebRTC may require two extra servers if connecting over the open network: STUN and TURN Server. The STUN Server is used to get the public IP address and the TURN Server is used for networks that don’t support the WebRTC protocol. Both servers are for network address translation (NAT) issues.

**Browser Code**

The browser code is the JavaScript API that has been included in the Pixel Streaming system.

### 3.1.3 Connections

Figure 3-2 presents the architecture of the Pixel Streaming system and the connection between components. On the left is the Pixel Streaming Plugin which connects directly to the Signaling and Web Server. A direct connection to the web browser is then established. The Signaling and Web Server is responsible for controlling and managing the connections after the initial communication is done. The STUN and TURN Server are optional.

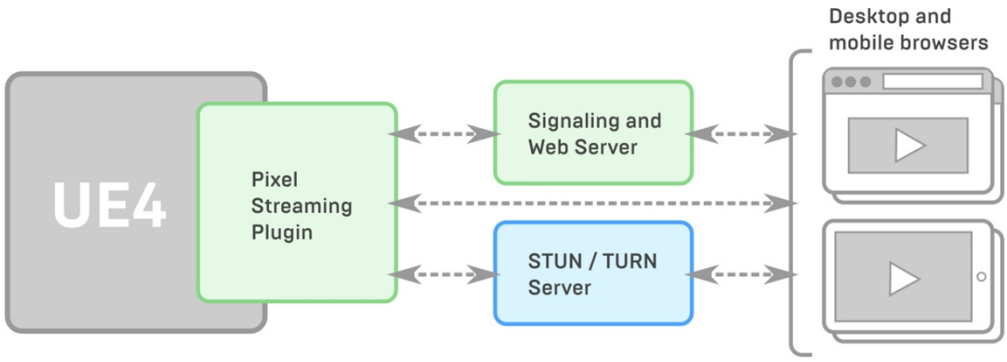


**Figure 3-2** Architecture of the Pixel Streaming system. Image adapted from Unreal Engine Documentation [4].

The Unreal Engine application and the browser need to send their own IP addresses with each other so that the Signaling and Web Server can manage the connection between them. If running in a local area network (LAN), each party is able to access the other party using the private IP address locally in their own network. Otherwise, the STUN and TURN Server are required to connect between the Pixel Streaming Plugin and browsers.

Over the open network, the STUN Server is used to figure out the public IP address of each party and send it to the other party. Then the Signaling and Web Server can continue to handle the connections between them. Alternatively, the TURN Server can be used to relay the media stream. It can connect to both the Unreal Engine application and the browser. Then the media stream is sent from the application to the TURN Server and it forwards the stream to the browser. In this situation, the Unreal Engine application has an indirect connection with the browser.

The STUN and TURN Server are a set of protocols for NAT traversal while building the peer-to-peer communication. Together with the ability to fall back from one server to another, they constitute the Interactive Connectivity Establishment (ICE) framework.

## 3.2 Common Setups for Pixel Streaming

This section gives three common use case setups for Pixel Streaming.

**Connection within a LAN**

This is the most basic setup for Pixel Streaming. In Figure 3-3, the server and the clients are in the same local area network. The Unreal Engine application is placed on a server in LAN. Through the Pixel Streaming Plugin, clients can receive the rendered streams from the server

and communicate with the hosting server.

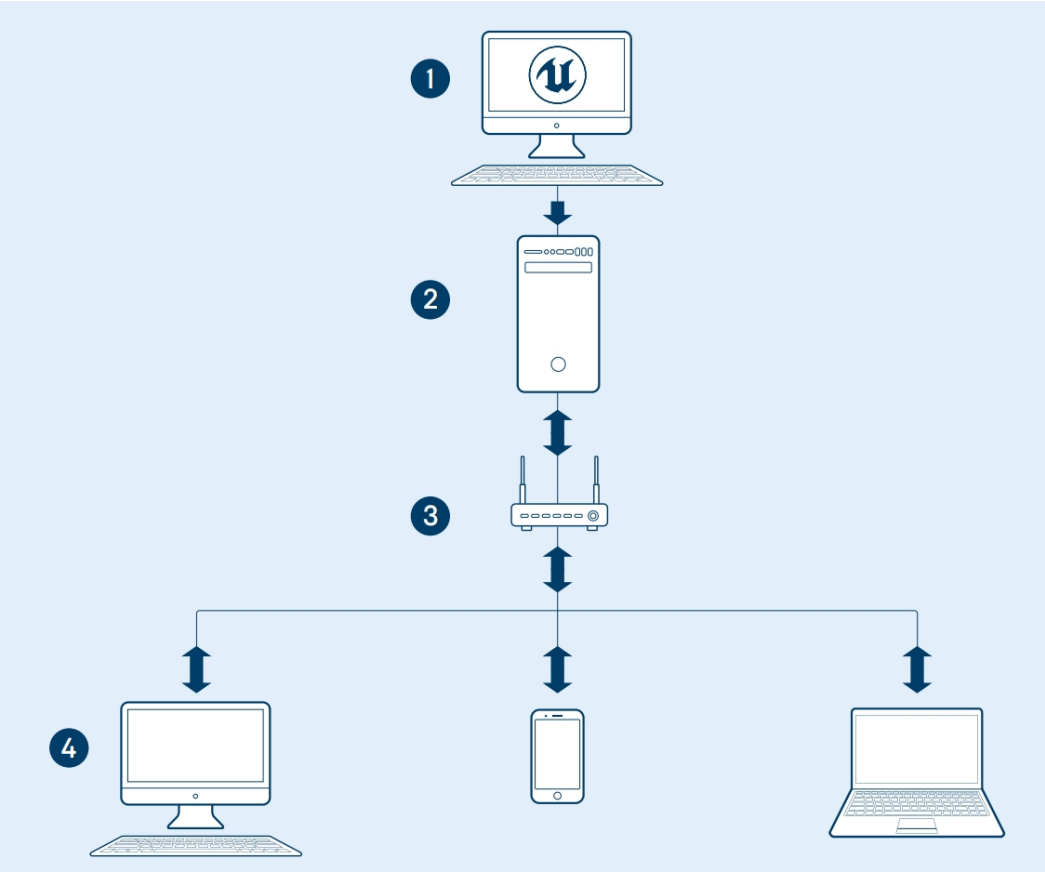


**Figure 3-3** Connection within a LAN. 1: UE4 development. 2: UE4 server app on LAN with WebRTC. 3: Router. 4: User displays/interaction. Image adapted from [4].

### Connection over the open network

Sometimes the content of the Unreal Engine application needs to distribute to users over the open network. In Figure 3-4, the hosting server and clients are in separate LANs. The STUN and TURN Server are required to handle the communication between the hosting server and clients, who share the same content and interactive experience. This setup is suitable for users who need to discuss or present something to specific remote participants.

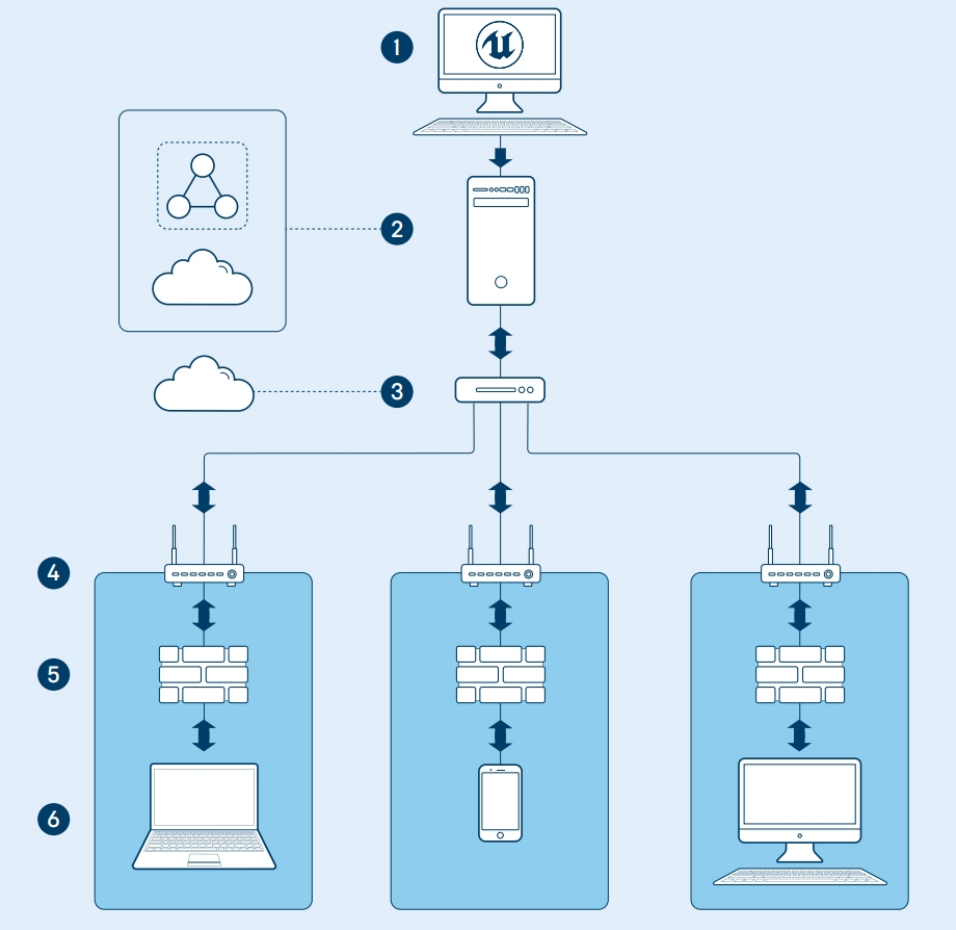


**Figure 3-4** Connection over the open network. 1: UE4 development. 2: UE4 server app in network or cloud. 3: STUN/TURN Server in cloud. 4: Router/entrance to client network. 5: Firewall. 6: Display/interaction devices. Image adapted from [4].

### Unique experience for every client

In the former two setups, all clients share the same user experience. If an application needs to reach a lot of users with different hardware and software and each of them needs to have a unique single experience, then a scalable remote deployment is necessary (Figure 3-5). Each server in the cloud runs a separate stack of Pixel Streaming components for each client so that every user can have his or her own interactive experience. This setup is suitable for the remote collaborative design, consumer applications, configurators, and so on.

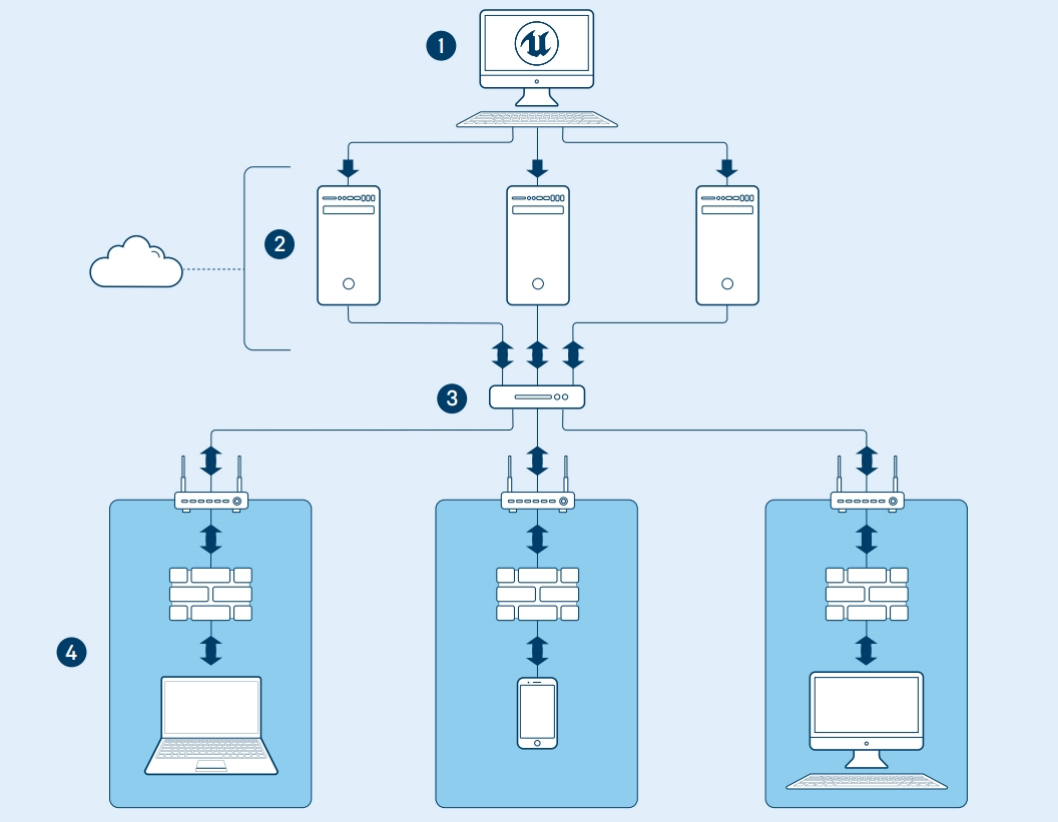


**Figure 3-5** Unique experience for every client. 1: UE4 development. 2: UE4 server apps with WebRTC in cloud. 3: STUN/TURN Server. 4: User displays/interactions. Image adapted from [4].

## 3.3 Benefits

Pixel Streaming provides several possible benefits:

- **Lightweight client.** Everything is done on the server side, so just a modern web browser is enough to use this technology.
- **No installation required.** There is no need for any prerequisite installations. The only thing is to download the media stream when running the application.
- **Multiple platforms supported.** The Pixel Streaming system supports multiple platforms and just one package of the application is required.
- **Deterministic high quality on all platforms and devices.** Running Unreal Engine application on a high-end host system, whether mobile devices or lightweight web browsers can display the same high-quality graphics as on the host device.
- **Multi-user supports.** Pixel Streaming can multicast the Unreal Engine application to

many users so that users see the same stream at the same time.

- **Lower latency.** The WebRTC peer-to-peer communication framework is used to offer the lowest achievable latency between the user and the application.
- **Easy to set up.** Only a small number of components make up the Pixel Streaming system so that everyone can use it easily.
- **Security.** Users only have the video and audio stream, so they can't access the private IP of the client. Therefore, Pixel Streaming has the same security as the web pages. All the data is stored on the server side and only the streams are distributed to the client side.

In summary, Pixel Streaming is a favorable distribution solution which can maintain the highest quality and capacity on the client side in the experience for non-games Unreal Engine application.

# 4 Sample Development Process of Car Configurator Application on Pixel Streaming

This chapter covers the sample development process of a car configurator application using Unreal Engine version 4.24.3. All the functions were implemented by Blueprints Visual Scripting. The Pixel Streaming technology in Chapter 3 was applied to this application and tested on some devices. At first the basic architecture of this application is determined. Then the environment and the car model are selected and designed to provide a realistic and immersive atmosphere. In order to manipulate within the world, UMG is used to provide a functional UI view. In the end, this application is used as an example to do Pixel Streaming. An overview of the workflow is given in Figure 4-1.

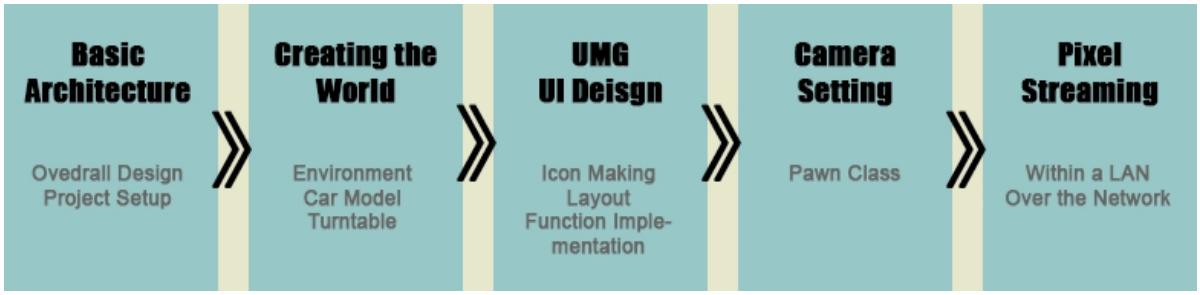


**Figure 4-1** Overview of the workflow for the project. Image created in Adobe Photoshop.

## 4.1 Basic Architecture

In order to make the entire development process organized and clear, the basic architecture must be determined first. The wireframe was first done according to the overall design of the application and detailed functions are listed. Then the project was created in Unreal Engine 4 including the Game Control setup.

Most assets are from the Unreal Engine Official, the detailed source of all assets is presented in the Appendix D. The whole source code of this project is available in the attached USB card (see more information in Appendix E).

### 4.1.1 Overall Design

The goal is to design a real-time interactive car configurator application that users and

customers thus can configure cars. The car configurator allows users to design and build their own cars. To ensure that the car specifically caters to the users' interest, they can choose the car paints, wheel rim colors, add LUT filters and so on. With the power of Blueprints and UMG tools, the following interactions will be created:

- **Easy User Interface.** The user interface should be clear for users to manipulate in the car configurator application.
- **Car Paint Selector.** This allows users to change the car paint from the preset options.
- **Rim Color Selector.** This allows users to change the wheel rim color of the car from the preset options.
- **LUT Setting.** Users can adjust the atmosphere style of the application by choosing the LUT options.
- **Turntable On/Off.** Users can turn on or turn off the rotation of the turntable.
- **Background Music On/Off.** Users can turn on or turn off the background music.
- **Screenshot.** This enables users to have a screenshot and the screenshot is saved in the project folder.
- **Exit Application.** Users can exit the application from the UI button.

Based on these features, the wireframe of this application was created (Figure 4-2). The submenu only appears when the user clicks certain button in the main menu bar.

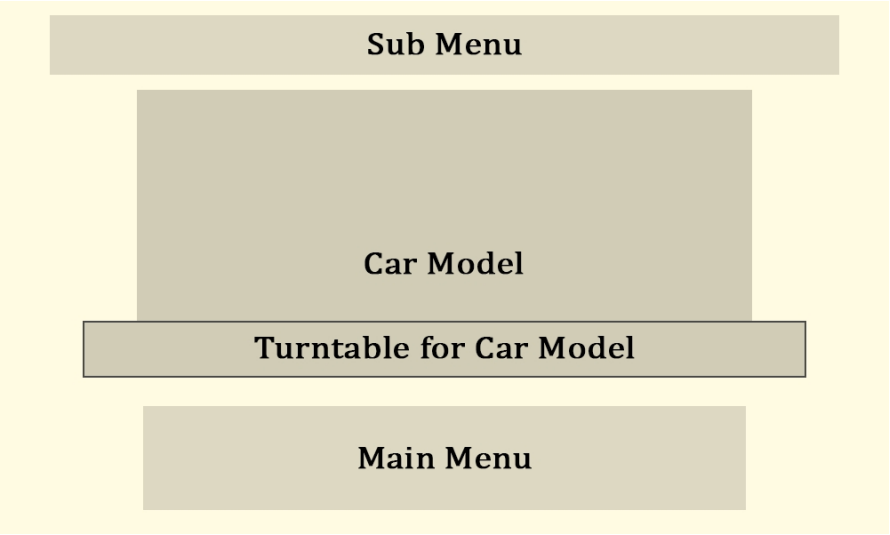


**Figure 4-2** Wireframe of the car configurator application. Image created by Adobe Photoshop.

### 4.1.2 Project Setup

After having determined the overall direction, the car configurator project now can be created in Unreal Engine. This project is called ***CarConfigurator_Hou*** and a default level map named ***CarConfigurator*** was created through the UE4 menu. A project can include multiple level maps, but in this project, there is only one main level map.

Every project needs input from the user and a car configurator is no different. For the default level map, the controls are set up to be the first-person game mode in which players can move around the camera by using the mouse click and the W, A, S, D keys. This will be practicable in most circumstances, but for a car configurator, users should be provided a still camera and visible mouse cursor to operate in the car configurator application. The game controls should be set up so that the mouse doesn't act as normal does which is to be in first-person viewport. This can be realized through the *PlayerController* and *GameMode* Blueprints.

**PlayerController**

The *PlayerController* is the interface between the *Pawn* and the human player controlling it. It implements the functionality for taking the input data from the player and translating that into actions, such as movement, using items, firing weapons, etc. In this car configurator application, a *PlayerController* Blueprint named ***PC_Configurator*** was created. The following option should be checked to make the mouse cursor visible (Figure 4-3).

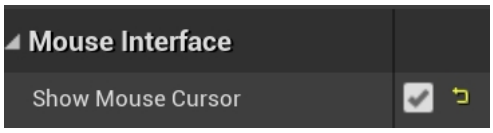


**Figure 4-3 *PC_Configurator*** Blueprint: settings in this blueprint. Screenshot from the UE4 project.

**GameMode**

The *GameMode* Blueprint defines the game's set of rules. The rules can include how players join the game, game pausing, and level transition, as well as any game-specific behavior. The *GameMode* is set for each level, and it can be reused in multiple levels. In this application, a *GameMode* Blueprint named ***GM_Configurator*** was created. The setting of *Player Controller Class* was set to ***PC_Configurator*** created before so that users can operate with the mouse cursor (Figure 4-4).

**Figure 4-4** ***GM_Configurator*** Blueprint: settings in the blueprint. Screenshot from the UE4 project.

Finally, in the *Game Mode* option applied the ***GM_Configurator*** though the *World Setting* panel. Then the basic setup of this application was done.

## 4.2 Creating the World

This section covers the whole construction process of the world, including the environment, lighting, as well as the importing of the car model and the implementation of the turntable.

### 4.2.1 Environment

Two different styles of environment are provided in separate level maps, ***CarConfigurator*** Level and ***CarConfigurator_Sky*** Level. Apart from the environment setting, everything else in these two level maps is the same. In this application, the ***CarConfigurator*** Level is mainly used to create the remaining part. The construction steps of both level maps are demonstrated below. In the following sections, the author primarily uses the first environment.

**CarConfigurator Level**

The first level named ***CarConfigurator*** provides users with a relatively abstract view. A background environment sphere was put in the world (Figure 4-5), thus creating a gray background seen from the inside (Figure 4-6). The gradient and the brightness of the background sphere can be adjusted through two parameters ***Exponent*** and ***Intensity*** in its attached material. In this application, the ***Exponent*** value is 2 and the ***Intensity*** value is 200.

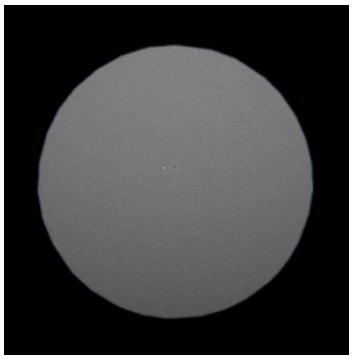

**Figure 4-5** Outside look of the background environment sphere. Screenshot from the UE4 project.

**Figure 4-6** Inside look of the background environment sphere. Screenshot from the UE4 project.

***Exponent*** parameter controls the gradient of this background. The larger the number, the more the dark parts in the background. Table 4-1 shows different effects at different ***Exponent*** values with the fixed ***Intensity*** value 200.

| Exponent | 1 | 5 | 10 |
|---|---|---|---|
| Intensity | 200 | 200 | 200 |
| Effect |  |  |  |

**Table 4-1** Effect of different ***Exponent*** values on the environment sphere. Table created by Microsoft Excel.

***Intensity*** parameter controls the luminous degree of this background. The larger the number, the brighter the background. Table 4-2 shows different effects at different ***Intensity*** value with the fixed ***Exponent*** value 2.

| Exponent | 2 | 2 | 2 |
|---|---|---|---|
| Intensity | 1 | 200 | 1000 |
| Effect |  |  |  |

**Table 4-2** Effect of different ***Intensity*** values on the environment sphere. Table created by Microsoft Excel.

This environment sphere is the main part of the environment. Other parts are listed below (Figure 4-7). All these components interact with each other to provide a gray and abstract scene.

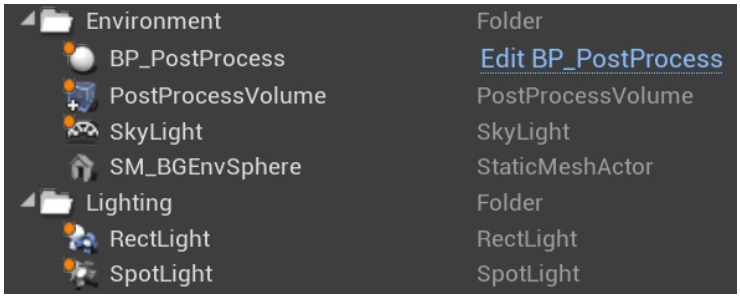


**Figure 4-7** Environment components in ***CarConfigurator*** Level. Screenshot from the UE4 project.

### CarConfigurator_Sky Level

This environment setting provided by the default level simulates the real sky (Figure 4-8). It is made up of the following parts (Figure 4-9). All these components interact with each other to provide a realistic sky scene.

**Figure 4-8** Look of the ***CarConfigurator_Sky*** Level. Screenshot from the UE4 project.

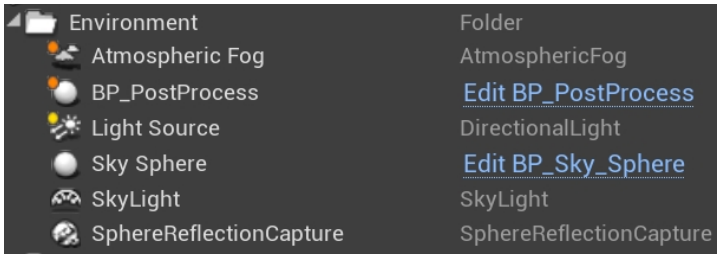


**Figure 4-9** Environment Components in ***CarConfigurator_Sky*** Level. Screenshot from the UE4 project.

## 4.2.2 Car Model and Turntable

### Model Pre-processing

Model creation is not the author's main task, so all models are readily available. The car model used in this application is Audi_R8 which is downloaded from *TurboSquid* [17]. Before importing into UE4, there are two things that need to be set in Blender. The first thing is to normalize the name of each part of the car model. The naming of the components was messy at

first. In order to distinguish clearly when assigning materials, the components need to be renamed according to the UE4 standard. The next step is the allocation and naming of materials. In Blender, whether the material allocation is wrong needs to be carefully checked and each material also needs to have a proper name. The useless materials should be deleted so as not to cause redundancy after importing into UE4.

After ensuring that the model has no major problems, it then can be exported in FBX format. When importing into the UE4, the *Combine Meshes* option should be clicked (Figure 4-10) so that the imported model will be a complete static mesh instead of several scattered static meshes.

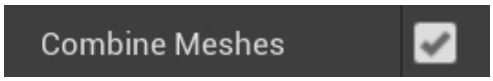


**Figure 4-10** Option when importing the car model. Screenshot from the UE4 project.

UE4 has its own naming convention. The name of the imported model is the same as the Blender file name with prefix "SM" (Figure 4-11). "SM" stands for static mesh. Also, the imported materials along with the model is the same as the components' name in Blender with prefix "M" (Figure 4-12). "M" stands for material.

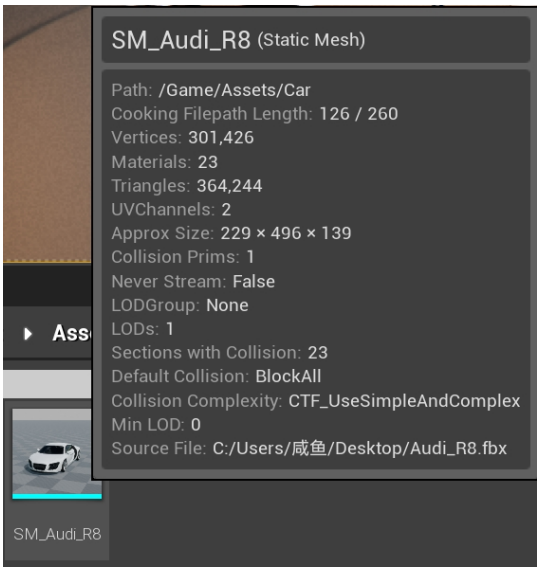


**Figure 4-11** Name of the imported model in UE4. Screenshot from the UE4 project.

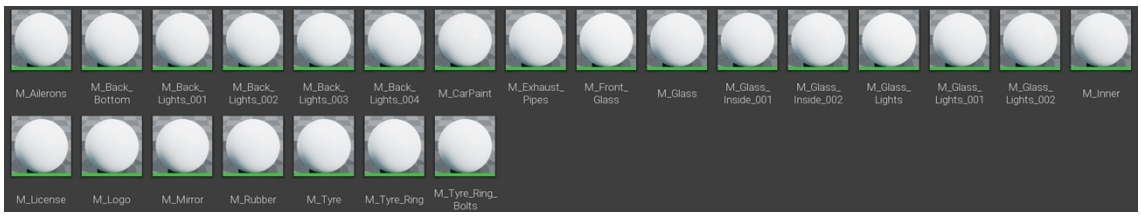


**Figure 4-12** Name of the imported materials in UE4. Screenshot from the UE4 project.

### Applying Materials to Car Model

It can be seen from Figure 4-12 that all the materials that have been set in Blender will be lost and become blank after importing into the Unreal Engine. Here the author will not set up these materials one by one. All the materials applied on the car model come from the automotive material package provided by UE4 (Figure 4-13). It is a collection of 164 high quality automotive-themed materials and textures which have been setup for use in Unreal Engine 4. These materials have been optimized to take advantage of techniques and features such as triplane projection and ray tracing.

**Figure 4-13** Automotive material package provided by Unreal Engine Official. Image adapted from the Unreal Official.

### Blueprint for Turntable

The Blueprint for the turntable is called ***BP_TurnTableSet***, which complies with the UE4 naming convention. It was based on the turntable Blueprint from the *Photo Studio* template that enables developers to create product presentation and cinematics in a photographic studio environment. The attached materials and textures used were also from this template.

The turntable consists of three parts (Figure 4-14). The purple part is the center plate that can rotate. The yellow part is the outer plate and four ramps make up the blue edge part.

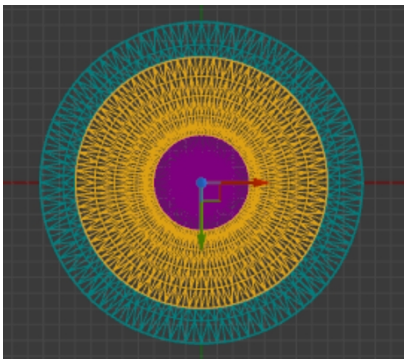

**Figure 4-14** Wireframe of the turntable seen from the top. Screenshot from the UE4 project.

The setups of this Blueprint were done step by step in the *Construction Script* function which executes when instances of a Blueprint are created to perform initialization actions. The construction includes the following steps.

**Step 1:** Select the outer plate mesh (Figure 4-15). There are five types of outer plate to choose from. They are stored in an *Enum* variable called ***FloorTypeMap*** (Figure 4-16). The selected floor type is stored in a variable called ***FloorType***. In this application the floor type is the fifth one called ***Flat Floor***


**Figure 4-15** *Construction Script* function: select the outer plate mesh. Screenshot from the UE4 project.

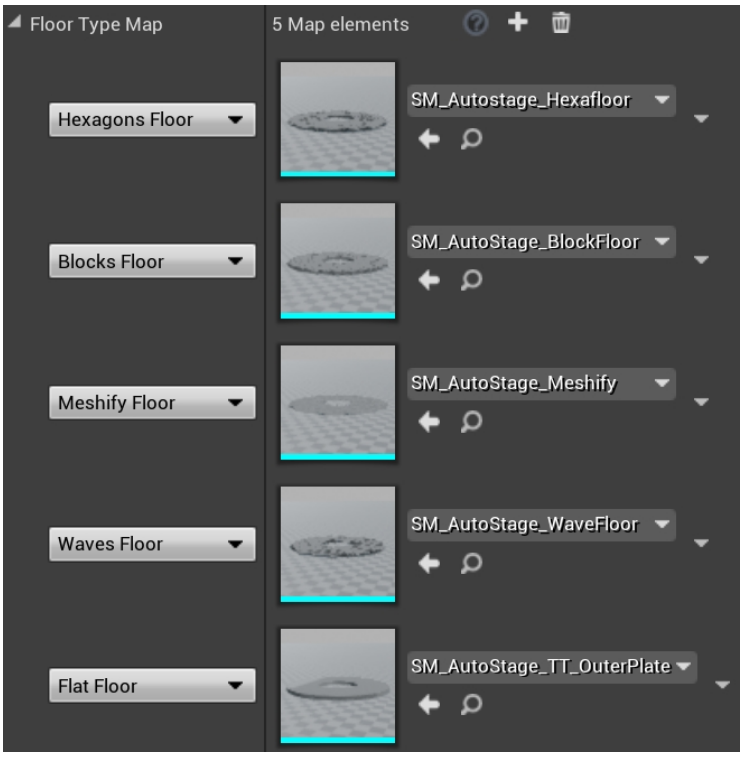


**Figure 4-16** *Construction Script* function: five types of the outer plate. Screenshot from the UE4 project.

**Step 2:** Set the stage materials (Figure 4-17). It is a custom function called ***SetSatgeMaterials*** which is used to set all the materials of the turntable. All components are attached with black matte material and the final view of the turntable can be seen in Figure 4-18.

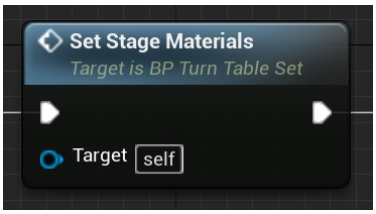


**Figure 4-17** *Construction Script* function: custom function to set the stage materials. Screenshot from the UE4 project.

**Figure 4-18** Overall look of the turntable. Screenshot from the UE4 project.

**Step 3:** Set the rotation of the turntable (Figure 4-19). This function allows the center plate to rotate around the Z axis. The rotation speed will be set later in the *Event Graph* event.

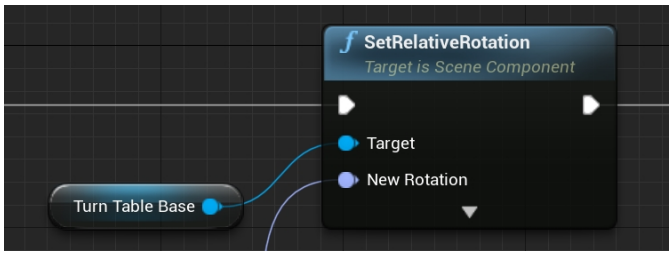


**Figure 4-19** *Construction Script* function: set the rotation of the turntable. Screenshot from the UE4 project.

**Step 4:** Attach the car actor to turntable (Figure 4-20). The ***AttachCarActor*** function is to attach the car to the center plate so that the car can rotate with the center plate together. There are two main points in this function. The car actor should be movable and be set as a child to the center plate component.

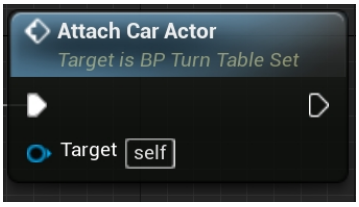


**Figure 4-20** *Construction Script* function: attach the car actor to turntable. Screenshot from the UE4 project.

**Step 5:** Use *Event Tick* event to make rotation. *Event Tick* is a simple event that is called on

every frame of gameplay. The rotation rate is stored in the ***RotationRate*** variable and is set to 10 in this application (Figure 4-21). This value is multiplied with the output of the event *Delta Seconds* so that the turntable rotates 10 degrees per second.


**Figure 4-21** ***RotationRate*** value. Screenshot from the UE4 project.

After these settings, the car model now can rotate with the turntable center plate when running the application.

## 4.3 UMG / UI Design

This section focuses on the development process of the UI design. The interactive UI was implemented by Unreal Motion Graphics (UMG) UI Designer. It is a visual UI authoring tool which can be used to create UI elements such as in-game HUDs, menus, or other interface related graphics to present to users. At the core of UMG are Widgets, which are a series of pre-made functions that can be used to construct your interface (things like buttons, checkboxes, sliders, progress bars, etc.). The Widget is stored as a Widget Component within a Blueprint class, which uses two tabs for construction: *Designer* tab allows for the visual layout of the interface and basic functions while *Graph* tab provides the functionality behind the Widgets used.

This application has only one Widget Blueprint named ***UMG_Menu***. The UI design is implemented in this class. The UMG needs to be presented when running the application. Hence, in the Level Blueprint of ***CarConfigurator***, UMG Widget Menu should be created and added to the viewport (Figure 4-22).

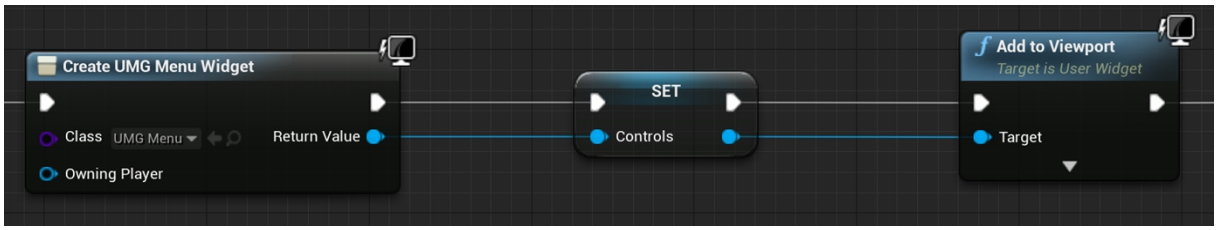


**Figure 4-22** Level Blueprint: add UMG to the viewport. Screenshot from the UE4 project.

### 4.3.1 Icon Making

Before setting up the layout, the icons need to be first prepared. All icons were made in Adobe Photoshop and in PNG format. Considering the functions of the main menu and the submenu in the application mentioned in section 4.1.1, the following icons were created and applied (Figure 4-23).

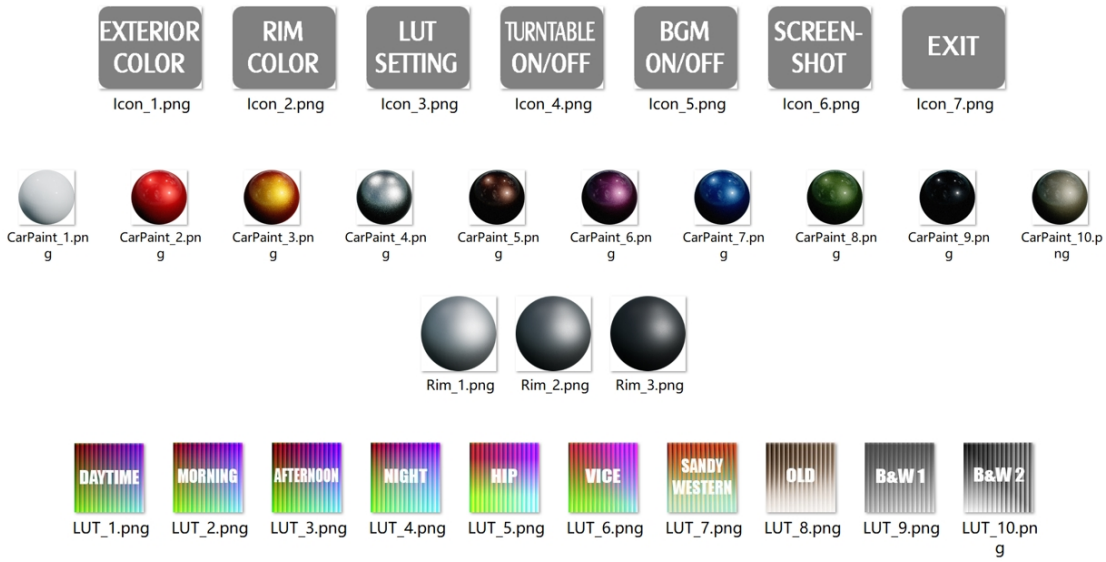


**Figure 4-23** Icons used in UI. Screenshot from the UE4 project.

### 4.3.2 Layout

The visual layout of the interface should be designed in the *Designer* tab of the Widget Blueprint. The basic idea is that the main menu is always kept at the bottom of the application. When clicking on a certain menu option, the corresponding submenu will appear or disappear on the top. Different submenus can switch between each other with animations.

The overall layout is shown below (Figure 4-24). This figure shows the submenu of car paint selector. Another two submenus of wheel rim color selector and LUT setting are in the same place. Each group of icons is placed in one horizontal box.

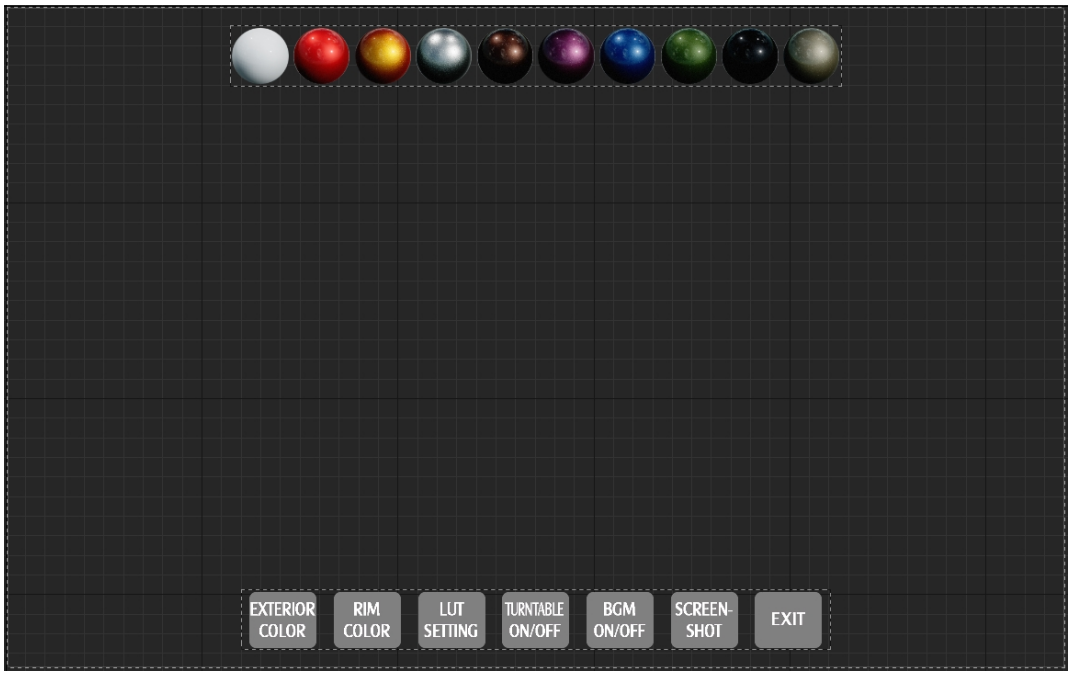


**Figure 4-24** Overall layout of UI. Screenshot from the UE4 project.

### 4.3.3 Function Implementation

Next step is to realize the functionality behind the Widget Blueprint in the *Graph* tab. The menu contains several options to help the user find information and execute functions. It comprises the following six main options:

- **Exterior color**: allows the user to change the car paint.
- **Rim color**: allows the user to change the rim color of the four wheels.
- **LUT Setting**: allows the user to change the LUT filter.
- **Turntable ON/OFF**: controls the rotation or stop of the turntable.
- **BGM ON/OFF**: switches the background music.
- **Screenshot**: allows the user to have a screenshot of the scene.
- **Exit**: exits the application.

#### Exterior Color

While clicking on the button, the submenu of the car paint selector will be appeared with an animation (Figure 4-25).

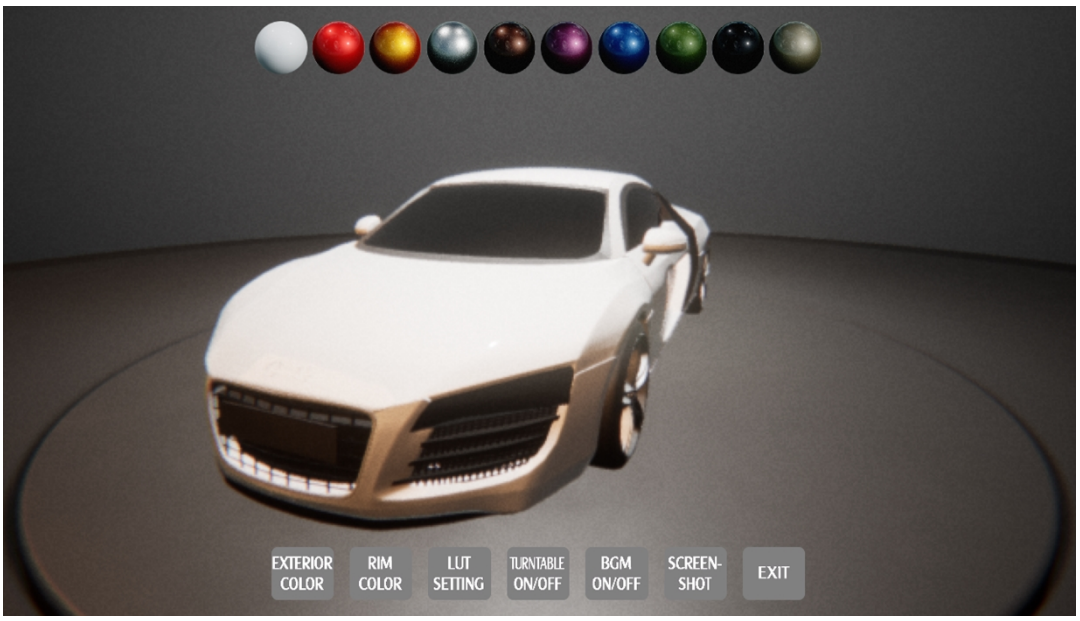


**Figure 4-25** Look when clicking ***Exterior Color***. Screenshot from the UE4 project.

To change the car paint, an *Event Dispatcher* is utilized and called every time a button is clicked on (Figure 4-26). This event, which toggles the car paint of the car, is bound within the Level Blueprint Class, because only here the visibility of the car model could be accessed.

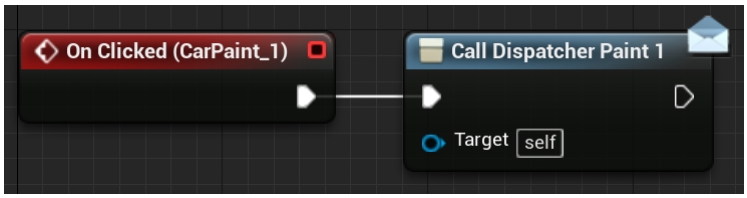


**Figure 4-26** ***UMG_Menu*** Blueprint: call dispatcher when clicking the ***Exterior Color*** button. Screenshot from the UE4 project.

In the Level Blueprint, the corresponding event was created and bound to the dispatcher (Figure 4-27). Once the ***ChangeToPaint_1*** event happens in this example, the car paint will change to the corresponding color (Figure 4-28). The remaining car paint options were implemented in the same way.

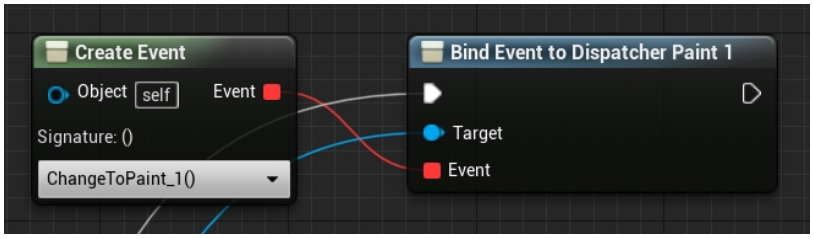


**Figure 4-27** Level Blueprint: bind ***ChangeToPaint_1*** event to the first dispatcher. Screenshot from the UE4 project.

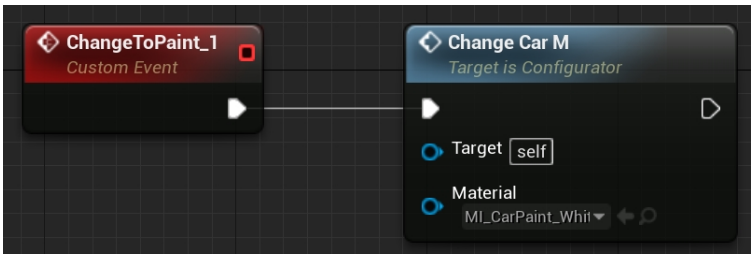


**Figure 4-28** Level Blueprint: ***ChangeToPaint_1*** event. Screenshot from the UE4 project.

**Rim Color**

While clicking on this button, the wheel rim color selector will be appeared with an animation (Figure 4-29).

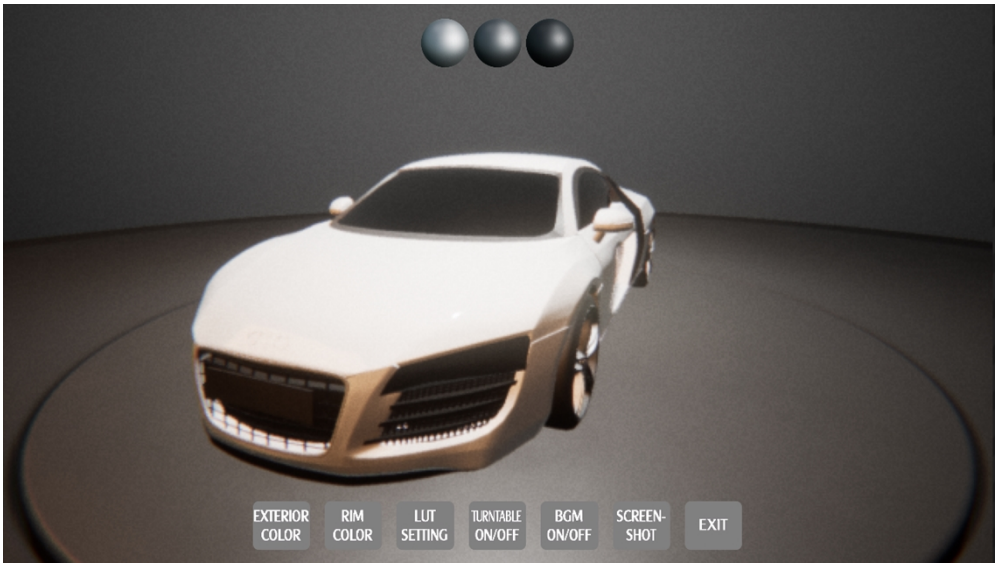


**Figure 4-29** Look when clicking ***Rim Color***. Screenshot from the UE4 project.

To change the wheel rim color, an *Event Dispatcher* was also utilized and called every time a button is clicked on (Figure 4-30) like the settings in former ***Exterior Color*** button. The event, which toggles the wheel rim color of the car, was bound within the Level Blueprint Class.

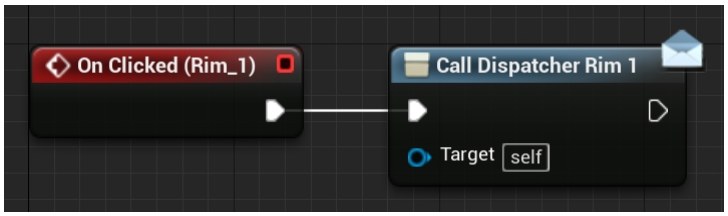


**Figure 4-30** ***UMG_Menu*** Blueprint: call dispatcher when clicking the ***Rim Color*** button. Screenshot from the UE4 project.

In the Level Blueprint, the corresponding event was created and bound to the dispatcher (Figure

4-31). In this example, ***ChangeToRim_1*** event was created and used to change the rim color correspondingly (Figure 3-32).

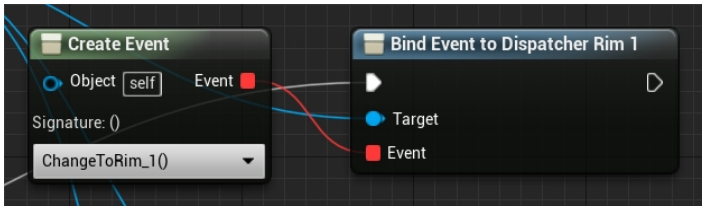


**Figure 4-31** Level Blueprint: bind ***ChangeToRim_1*** event to the first dispatcher. Screenshot from the UE4 project.

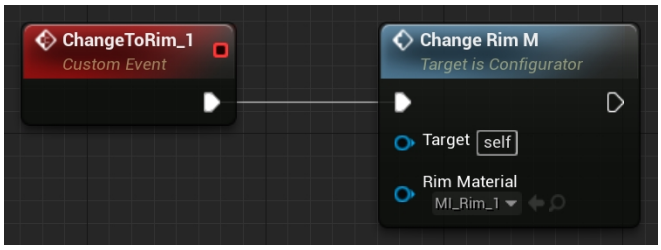


**Figure 4-32** Level Blueprint: ***ChangeToRim_1*** event. Screenshot from the UE4 project.

**LUT Setting**

While clicking on the button, the LUT selector will be appeared with an animation (Figure 4-33).

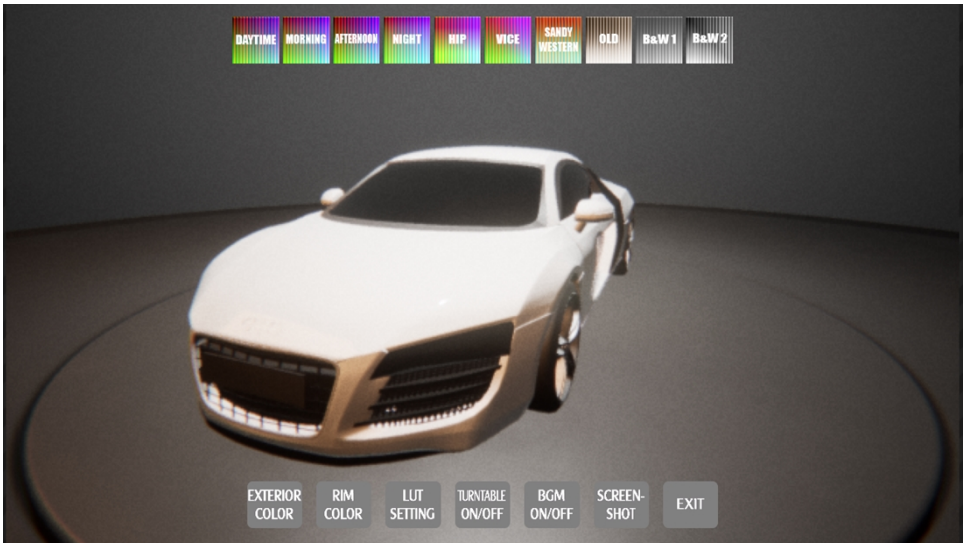


**Figure 4-33** Look when clicking ***LUT Setting***. Screenshot from the UE4 project.

LUT stands for Lookup Table which is mainly used in *Color Grading* in UE4. It can be used to transfer or correct color of the scene using a *Post Process Volume*.

Implementation of these buttons was different from the previous. When one of these LUT

options is clicked, it changes the *Color Grading LUT* setting in the ***BP_PostProcess*** Blueprint in the world. Hence, in order to communicate with this Blueprint in the Widget Blueprint, the ***BP_PostProcess*** Blueprint needs to be first obtained and saved in one variable called ***PostProcessBP*** (Figure 4-34).

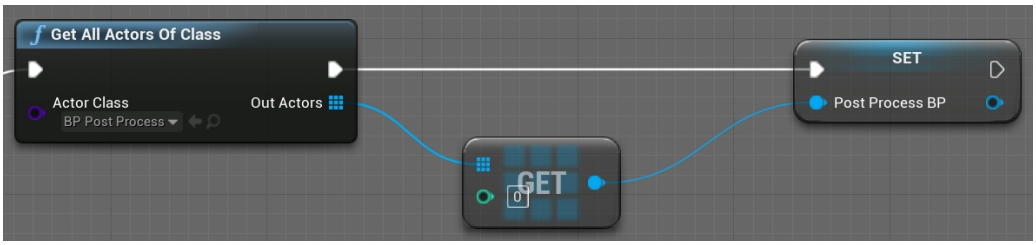


**Figure 4-34** ***UMG_Menu*** Blueprint: get ***BP_PostProcess*** Blueprint. Screenshot from the UE4 project.

Once one of the LUT options is clicked, the function ***ChangeLUT*** in the ***BP_PostProcess*** Blueprint will be called and the corresponding parameters will also be passed into this function (Figure 4-35).

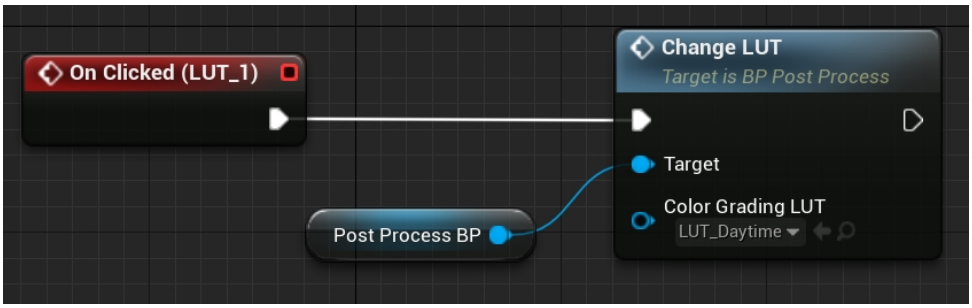


**Figure 4-35** ***UMG_Menu*** Blueprint: go to ***ChangeLUT*** function when clicking. Screenshot from the UE4 project.

Then in the ***BP_PostProcess*** Blueprint, ***ChangeLUT*** function was called and *Color Grading LUT* was set according to the LUT parameter (Figure 4-36).

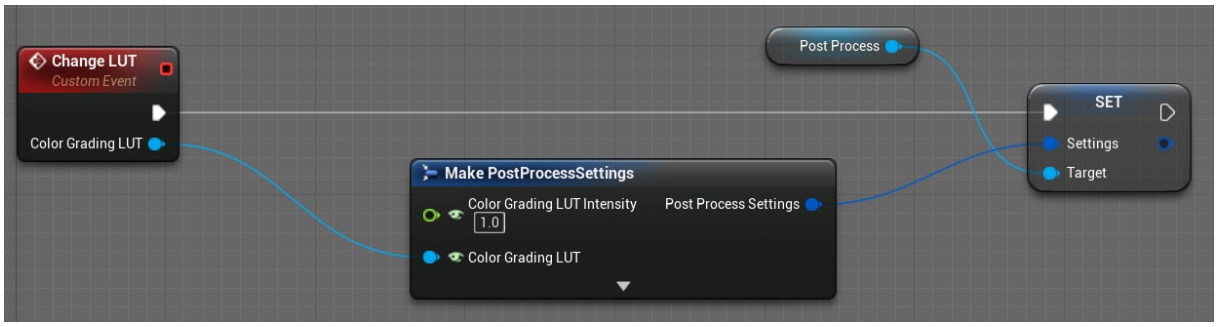


**Figure 4-36** ***BP_PostProcess*** Blueprint: set *Color Grading LUT*. Screenshot from the UE4 project.

### Turntable ON/OFF

This menu does not include submenus. It only controls whether the turntable rotates. The rotation rate was set in the ***BP_TurnTableSet*** Blueprint, so the ***BP_TurnTableSet*** first needs to be obtained and saved in the Widget Blueprint (Figure 4-37).

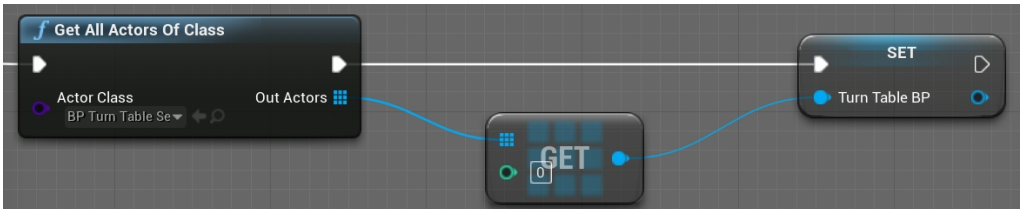


**Figure 4-37** ***UMG_Menu*** Blueprint: get ***BP_TurnTableSet*** Blueprint. Screenshot from the UE4 project.

If the turntable is rotating, the ***RotationRate*** is set to 0 on clicking the button, and vice versa (Figure 4-38). The ***RotationRate*** variable in the ***BP_TurnTableSet*** Blueprint needs to be public so that it can be directly accessed in the Widget Blueprint.

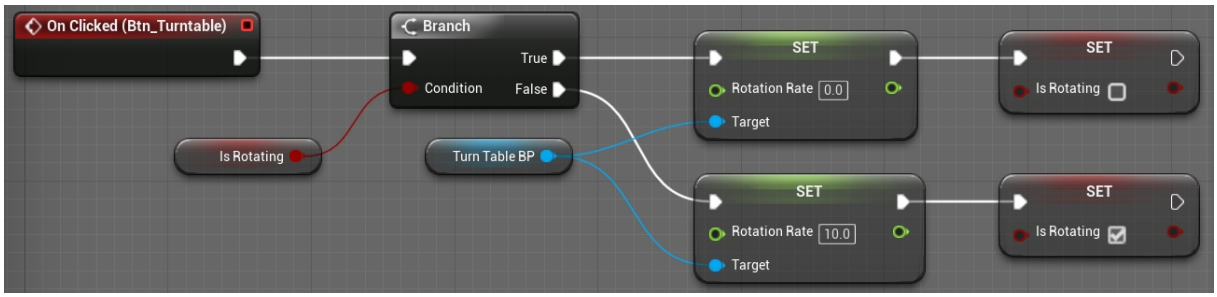


**Figure 4-38** ***UMG_Menu*** Blueprint: how ***RotationRate*** changes when clicking the button. Screenshot from the UE4 project.

### BGM ON/OFF

The behavior of audio playback in the Unreal Engine was defined within Sound Cues (Figure 4-39). It was created by the background music file in WAV format. When the application is running, the background music is open by default. The user can click this button to turn off the background music and can also click it again to reopen the music.

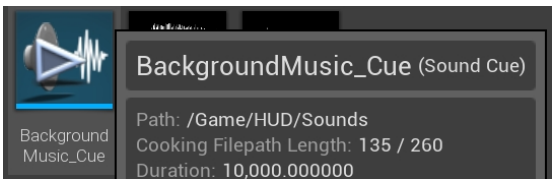


**Figure 4-39** Sound Cue of BGM. Screenshot from the UE4 project.

### Screenshot

When a screenshot is initialized, all submenus are closed and the animation appears. In addition to the animation and sound effects, the screenshot is mainly done by the *ExecuteConsoleCommand* function. It executes a console command called ***HighResShot 2*** [4] which can be used to take a screenshot that is twice as big as the screen resolution (Figure 4-40).


**Figure 4-40** ***UMG_Menu*** Blueprint: screenshot implemented by console command. Screenshot from the UE4 project.

By default, all screenshots will be saved to the following location in your project folder: *Saved\Screenshots\Windows*. When taking the screenshot, the following message will be displayed in the lower right-hand corner of the screen (Figure 4-41). Clicking on the message will display the folder containing the screenshots.

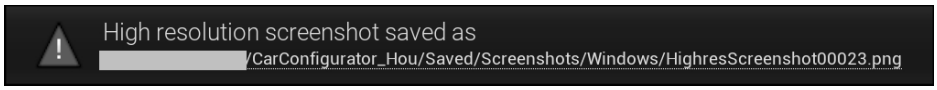


**Figure 4-41** Screenshot saved path. Screenshot from the UE4 project.

### Exit

To exit the application the simple function *Quit Game* is called once the ***Exit*** button is pressed (Figure 4-42).

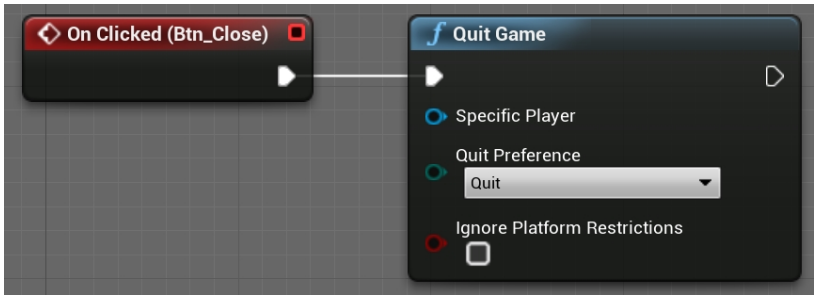


**Figure 4-42** ***UMG_Menu*** Blueprint: quit game. Screenshot from the UE4 project.

## 4.4 Camera Setting

For a car configurator, the camera must be fixed and the user can only rotate left and right to view the car model. In order to realize such a function, a Pawn Blueprint named ***BP_Camera*** was created. This blueprint consists of two components: *SpringArm* and *Camera*. The camera component acts as the main camera for the application scene and the spring arm component is used to position the camera appropriately in 3D space [18]. The camera component should be the child of the spring arm component, as the spring arm is used to position and rotate the camera (Figure 4-43).

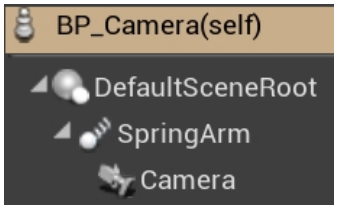


**Figure 4-43** Components of ***BP_Camera*** Blueprint. Screenshot from the UE4 project.

## 4.5 General Overview

Following is the overall look of the car configurator when starting the application (Figure 4-44).

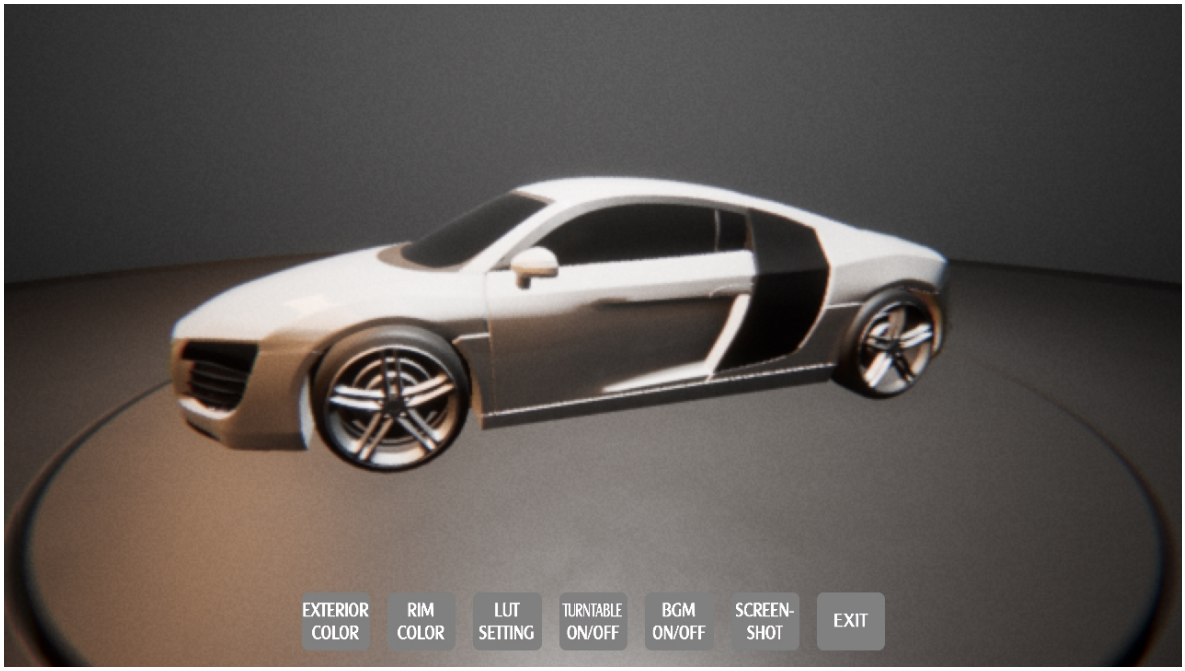


**Figure 4-44** General overview. Screenshot from the UE4 project.

## 4.6 Applying Pixel Streaming

This section talks about the connection process of Pixel Streaming, using the previous car configurator application as the streaming example. Two circumstances were provided. One is to apply Pixel Streaming locally within a LAN, the other is to run the streaming application over an open network, in which the STUN and TURN Server will be used.

### 4.6.1 Connection within a LAN

The following steps show how to stream the car configurator application in a local area network.

**Step 1: Enable the Pixel Streaming Plugin**

To use the Pixel Streaming, the first thing was to enable this plugin in Unreal Engine 4 (Figure 4-45). Then the Unreal Editor will restart again for the plugin changes to take effect.

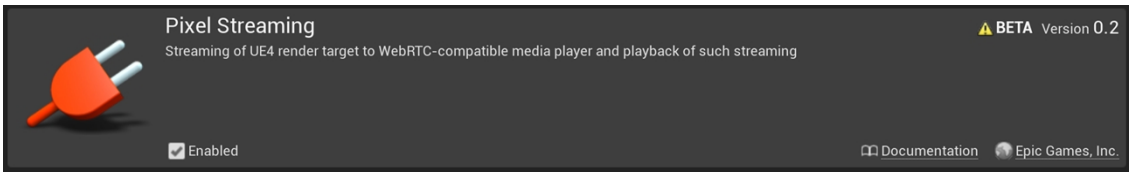


**Figure 4-45** Pixel Streaming Plugin. Screenshot from the UE4 project.

**Step 2: Package the application**

The Pixel Streaming Plugin only works when the application has been packaged as a standalone executable file, so the next step was to package this car configurator application for windows though the menu in UE4.

**Step 3: Add the audio mixer**

In order to let the audio properly stream, the audio mixer is required. A new shortcut of the executable file should be created and the text *-AudioMixer -PixelStreamingIP=localhost -PixelStreamingPort=8888* was appended at the end of the *Target* field in this shortcut's property setting. Now the standalone application has been prepared to stream.

**Step 4: Start the server and run the packaged application**

Inside the Unreal Engine installation folder, the folder of the Signaling and Web Server can be found and this server can be started by running the *run.bat* file. What this will do is downloading

the required things using NPM, the package manager for the Node JavaScript platform, and setting up the server. Remember that Node.js is required for the server to work.

When the server has been started correctly, the following lines will appear in the console window (Figure 4-46). All these ports are configurable in the documentation.

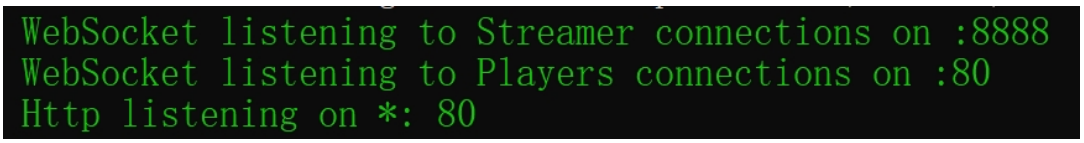


**Figure 4-46** Appeared lines in the console window when starting the server successfully. Screenshot from the console window.

Then the packaged car configurator application can be started from the shortcut created before. When this application connects to the Signaling and Web Server, the following line (Figure 4-47) will appear in the console window. It means that the application runs successfully with the enabled Pixel Streaming Plugin and the Signaling and Web Server is ready to connect the clients to this application.


**Figure 4-47** Appeared lines in the console window when the application is connected to the server successfully. Screenshot from the console window.

**Step 5: Connect clients**

Now the web browsers of different devices can be connected to this car configurator application through the IPv4 address of the server in the LAN. These devices must be in the same local area network as the server that runs the application.

The connection information can be seen in the console window. For example, Figure 4-48 means that one client with ID 101 has been connected to this application and Figure 4-49 means that this connection has been closed.


**Figure 4-48** Appeared lines in the console window when one client has been connected. Screenshot from the console window.

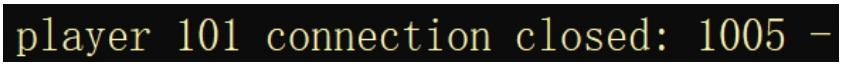

**Figure 4-49** Appeared lines in the console window when the connection has been closed. Screenshot from the console window.

### 4.6.2 Connection over the Open Network

To run the application with the Pixel Streaming Plugin over the open network, the TURN and STUN Server are required. The first two steps are the same in section 4.6.1, so this section will start from the third step.

**Step 3: Add the audio mixer**

The shortcut of the executable file still needs to be created. But the added text becomes - A*udioMixer -PixelStreamingIP={publicIP} -PixelStreamingPort=8888*. The Pixel Streaming IP should be the public IP of the computer that runs the Unreal Engine application instead of *localhost*. Because this time the application is played over the open network.

**Step 4: Start the server and run the packaged application**

Instead of running *run.bat* directly, the names of the STUN and TURN Server are required to set first. In the folder where the Signaling and Web Server is placed, find the following file (Figure 4-50) and set the names of the STUN and TURN Server in the *peerConnectionOptions* part (Figure 4-51) in this file. A free STUN Server can be found online and the public IP of the TURN Server should be the computer's public IP address that runs the application. Then run *runAWS_WithTURN.bat* file to start the Signaling and Web Server.

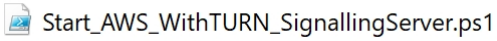


**Figure 4-50** File used to set the STUN and TURN Server. Screenshot from the folder.

```
$peerConnectionOptions = "{ \""iceServers\"": [{\""urls\"":
[\""stun:" + $PublicIp + ":19302\"",\""turn:" + $PublicIp +
":19303\""], \""username\"": \""PixelStreamingUser\"",
\""credential\"": \""Another TURN in the road\""}] }"
```

**Figure 4-51** STUN and TURN Server setup in the file. Screenshot from the file.

**Step 5: Connect clients**

Instead of connecting with the local IP address in a LAN, devices should connect to the streaming application through the public IP over the open network in this deployment.

### 4.6.3 Connection Results

The author has used the web browsers of three different devices to connect to the car configurator application. The first device is the laptop that runs the application; the second one is a tablet; and the third one is the smartphone. The details of these devices are listed in Table 4-3 below.

| | | | | |
|---|---|---|---|---|
| 1 | **Laptop** | Alienware | Intel i7-7700HQ / NVIDIA GeForce GTX 1050Ti | Chrome |
| 2 | **Tablet** | Apple | iPad Pro 2016 | Safari |
| 3 | **Smartphone** | Apple | iPhone XR 2018 | Safari |

**Table 4-3** Details of the used devices. Image created by Microsoft Excel.

The connection results are presented in Figure 4-52. These clients are connected within a LAN. The connection results over the open network are the same. It can be seen that all connected clients share the same content.

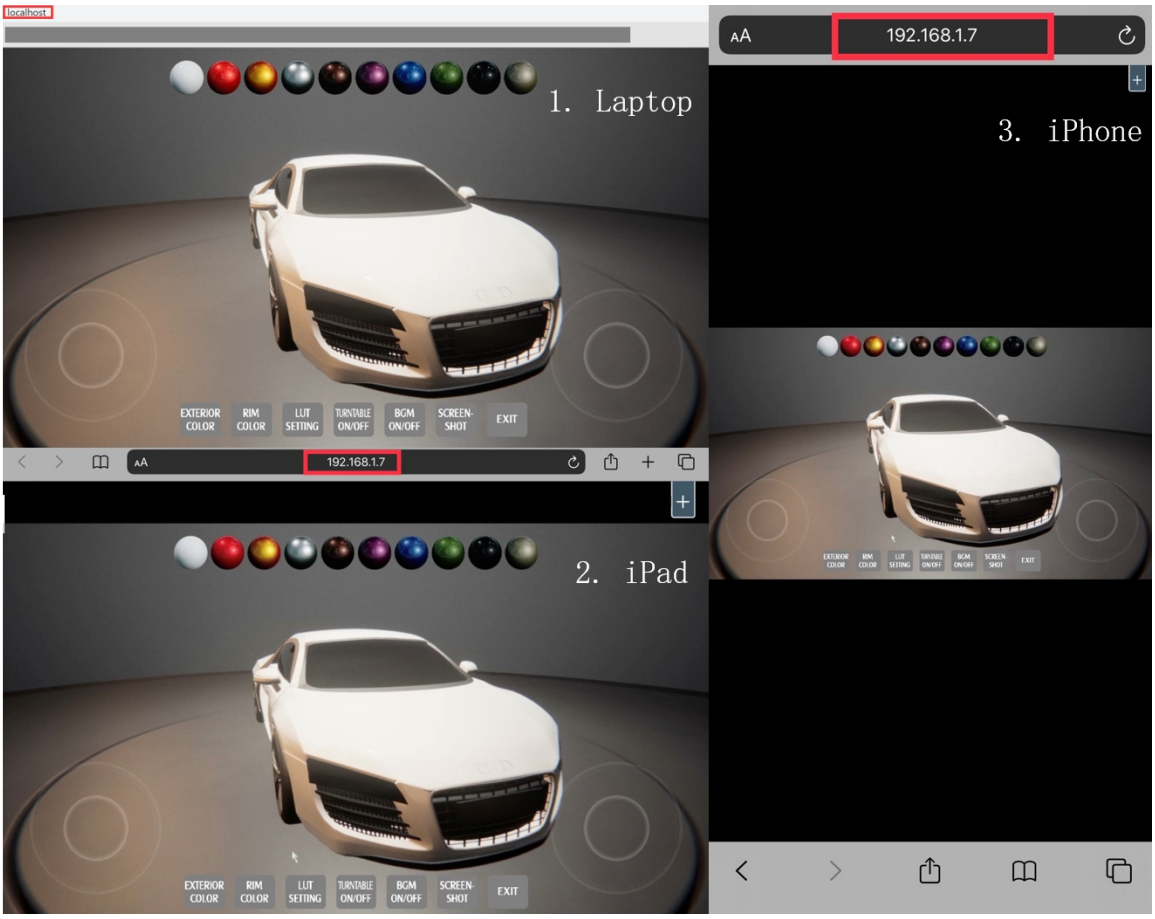


**Figure 4-52** Connection results on different platforms. Image made from screenshots.

When clicking the plus button at the top right of the page, additional options are provided to control the stream (Figure 4-53).

- **Kick all other players.** This option allows to close all connections except for the current one.
- **Enlarge Display to Fill Window.** This option allows the media player to resize to fit the size of the current window.
- **Quality Control ownership.** This option allows to use the current connection to determine the quality of the stream encoding.
- **Show Stats.** This option presents the statistics about the connection.


**Figure 4-53** Additional options on the page. Screenshot from the web page.

# 5 Discussion

This chapter discusses the results of the streaming car configurator application. The first section presents the functionality of this application and this technology. Then the usability is evaluated through interviews with users from three different career fields. This chapter additionally discusses the comparison between Pixel Streaming and other distribution solutions, such as WebGL and HTML5, and finally gives the reasons for developing Pixel Streaming.

## 5.1 Consideration of Functionality

The main objective of the functionality test is to check the functionalities of the streamed car configurator application. The main functions of this application were tested in four different ways of running. According to the description in the previous chapter, the application was first packaged as a standalone executable file and ran on the server side, and then three clients accessed this application through the Pixel Streaming technology. Hence, this application will first be accessed and tested as an executable file on the hosting computer. The next step is to test the media stream of the application on the three web browsers mentioned in section 4.6.3.

**Running the standalone executable file on the hosting computer**

The laptop used by the author was used as a server to run the application during the streaming. The standalone executable file was started on this laptop and tested. Table 5-1 shows the functions of the car configurator application. All these functions were tested and the result was positive. This application ran very smoothly on this laptop.

| | |
|---|---|
| 1 | Car paint selector |
| 2 | Wheel rim color selector |
| 3 | LUT selector |
| 4 | Turntable management |
| 5 | BGM management |
| 6 | Screenshot |
| 7 | Exit application |
| 8 | Scene rotation |
| 9 | Animations |
| 10 | Main menu button hover sound effects |

**Table 5-1** Functions in the car configurator application. Table created by Microsoft Excel.

### Accessing the streaming application on the web browsers

The streaming result on three clients (web browsers of the laptop, tablet and smartphone) can be seen from section 4.6.3. The clients were connected to the server through the streaming technology. The functionality of Pixel Streaming was tested in this part.

First, the functions listed in Table 5-1 were first tested and the result is generally positive. In terms of Pixel Streaming, it is a technology that can distribute the content to any device with the highest possible quality. To confirm this, the framerate was randomly recorded three times on each of the three devices. As shown in Figure 5-1, The frame rate was generally stable, and the operation of the application was generally smooth. The framerate was only slightly lower for a period of time on the tablet (see the yellow part in Figure 5-1), probably due to an unstable connection. Generally speaking, the performance of Pixel Streaming is good.


**Figure 5-1** Framerate on the clients. Screenshot from the UE4 project.

## 5.2 Consideration of Usability

The usability is evaluated through interviews with users from three different career fields. Six people took the test. They are students, office workers or middle-aged people. They experienced the streaming application on their own computers. After that, they were asked to evaluate the streaming application based on the following statement using a five-tier Likert scale. Testers need to choose the response that best characterizes how they feel about the statement from five scales: Strongly Agree, Agree, Neutral, Disagree and Strongly Disagree.

1. The application is easy to use.
2. The application provides a good overview of the functionalities.
3. The application provides a good orientation.
4. The streaming technology provides a good experience.
5. The streaming technology can be useful in multiple fields.

The handling and user-friendliness of the car configurator application and the streaming technology were perceived as overall pleasant as seen in Figure 5-2. All the testers found the application easy to use on their own devices, although some participants sometimes felt a little delay in operation. But this application gave them a good view and orientation of the functionalities due to the friendly UI. Additionally, most of them thought this streaming technology sometimes provided an unstable connection, but they were all optimistic about this technology. In conclusion, the car configurator application together with Pixel Streaming was considered as user-friendly and satisfactory.

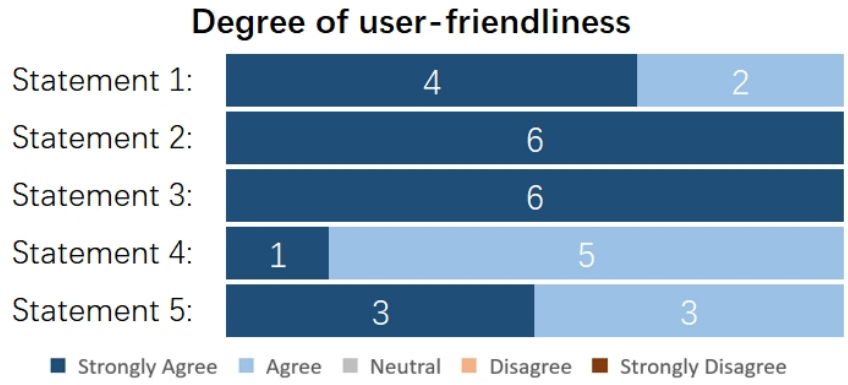


**Figure 5-2** Degree of user-friendliness of the streaming application accessed using a five-tier Likert scale. Chart created in Microsoft Excel.

## 5.3 Comparison of Distributed Solutions

Here two other solutions WebGL and HTML5 for distributing content are described. Then their limitations will be discussed. At last, the reason for choosing Pixel Streaming is given.

### 5.3.1 WebGL

WebGL is a cross-platform JavaScript API that is implemented right into the web browser. Users can interactive with the 2D or 3D graphics rendered in the HTML5 Canvas element in browsers that support it without downloading an application or plugin.

WebGL is based on OpenGL ES (OpenGL for Embedded Systems), a subset of OpenGL, which is designed for embedded systems like smartphones, tablets, and video game consoles [4]. It is executed on a computer's GPU (Graphic Processing Unit). WebGL elements can be combined with other HTML elements, allowing GPU-accelerated usage of physics and image processing and effects [19].

To employ WebGL, both source code written in JavaScript and shader code written in OpenGL ES SL(OpenGL ES Shading Language) are required [4]. Compiling is not needed before deployment.

**Limitations**

As a widely-used distribution solution, the primary limitation is the creation and delivery of the content. The content and interactive elements must conform to the WebGL framework. Therefore, when dealing with the content developed by a real-time engine like Unreal Engine, a lot of extra work needs to be done as can be seen from Figure 5-3.

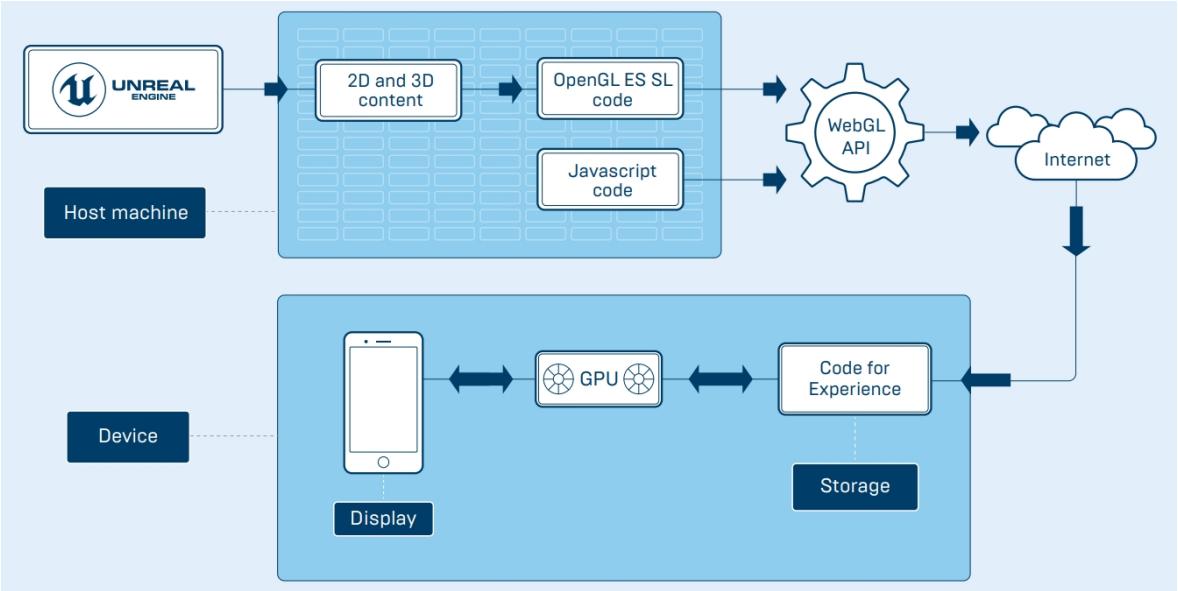


**Figure 5-3** Dataflow of WebGL deployment. Image adapted from [4].

Besides, WebGL is a kind of **client-side** distribution solution. Its performance is completely dependent on the capacity of the browser and hardware on the client side. Client needs to download the data and logic of the distributed application and render them to the screen. Such a solution puts a lot of pressure on the client side.

Furthermore, if the data requires a high level of protection, additional actions have to take on the client side.

### 5.3.2 HTML5

HTML5 is the latest version of HTML, a standard markup language for web page development. There are many solutions within HTML5 to build a 3D interactive experience through the Canvas element, without using WebGL at all. The knowledge of GPU programming and packaging is required. However, the performance of these solutions is insufficient regarding the quality and functionality.

#### Limitations

HTML5 setup is also a kind of **client-side** solution. It runs the application on the client side and the client side needs to download all the data of the application. For streaming a large application, it is a bit difficult to achieve, especially for some mobile devices. Also, HTML5 cannot guarantee the same quality and experience on different platforms and devices. Besides, HTML5 has the same security considerations as WebGL.

### 5.3.3 Reasons for Pixel Streaming

Unlike WebGL and HTML5, Pixel Streaming is a kind of **server-side** solution for distributing content. All the data stays on the server side and only the media streams are transferred to the client side, allowing to stream the content at the highest performance available. Pixel Streaming can share high-end experiences without consideration of the end user or platform. Such a distribution solution is faster and safer than other client-side solutions.

# 6 Conclusion and Outlook

The objective of this thesis is to explore Pixel Streaming, an advanced streaming technology, and demonstrate it through a car configurator application. To embark on this project, common web rendering strategies, especially Client-Side Rendering and Sever-Side Rendering, were first introduced and some key concepts in Unreal Engine 4 that were related to this thesis were listed, aiming to make readers better understand the following chapters. Then Pixel Streaming was presented in detail. By means of the Pixel Streaming system, the Unreal Engine application can run on the server side to avoid the quality limitations of client platforms and other issues in data transmission. Furthermore, to demonstrate the performance of the streaming technology, a car configurator application was developed and run as a streaming application from a test server. Three clients connected to the application though the web browsers and the connection result was generally satisfactory according to the functionality test and usability test. In the end, this thesis evaluated the distribution solutions available to share an Unreal Engine application though Pixel Streaming in comparison to other available solutions. As a typical server-side distribution solution, Pixel Streaming can maintain the highest quality and capability in the experience for Unreal Engine applications.

In view of the above, several potential applications emerge for the following specific field of works. For some car manufactures, they become more and more advanced with the official websites on how to customize the cars. With Pixel Streaming, the car rendering and its customizations can be done on a high-end server and then stream to the website, so that people could see it and start customizing it straightly in the website without downloading anything at all. Another option for car manufactures is the car showroom. Imagine that customers are looking at a big screen and a salesperson is making changes to the car on a separate tablet. The customers can see the changes on the screen and thus have a general idea of what the car looks like rather than just read the printed booklet. Furthermore, an architectural walkthrough could be realized through this streaming technology. If an architecture company is creating a building, then they can show this building through a remote interactive presentation to perspective customers. Customers can see details of the building, or even change something remotely on their own devices without having to have the Unreal client.

However still a number of limitations need to be overcome to arrive at a fully operational streaming technology that could affectively be applied in further developments. One major limitation of Pixel Streaming is the input conflict. If multiple users are connected to one Pixel Streaming system, everyone’s input arrives at the UE4 application, which will cause some

conflicts if they click in different directions or press different keys. In terms of the adaptation, it's all the same for every client, including the graphics quality and bitrate stuff. But in reality, this will be affected by the capability of the user's devices and other factors. Another disadvantage is the limitation of platform support. Pixel Streaming can only run on Windows operating system and NVIDIA machines. Furthermore, the Pixel Streaming Plugin is a beta version at the moment.

In conclusion, with the aid of the developed car configurator application, the performance of Pixel Streaming was successfully demonstrated and accessed. There will be no doubt that Pixel Streaming finds use in future applications when the above-mentioned limitations are to be overcome.

# Appendix A - List of Figures

# Appendix B - List of Tables

# Appendix C - List of Abbreviations

| | |
|---|---|
| AR | Augmented Reality |
| CDN | Content Delivery Network |
| CSR | Client-Side Rendering |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| HUD | Head-Up Display |
| ICE | Interactive Connectivity Establishment |
| IDE | Integrated Development Environment |
| IOT | Internet Of Things |
| IP | Internet Protocol |
| LAN | Local Area Network |
| LUT | LookUp Table |
| NAT | Network Address Translation |
| NPM | Node Package Manager |
| PC | Personal Computer |
| SEO | Search Engine Optimization |
| SSR | Server-Side Rendering |
| UE4 | Unreal Engine 4 |
| UI | User Interface |
| UMG | Unreal Motion Graphics |
| UU | Unreal Engine Unit |
| VCS | Version Control System |
| XR | Extended Reality |

# Appendix D - Source of Assets in Car Configurator Application Project

All assets are placed in the *Content* folder of the car configurator project ***CarConfigurator_Hou***. The source of these assets will be declared here.

**Automotive Materials**

From Unreal Engine Marketplace.

**Car Model (Audi_R8)**

From TurboSquid: https://www.turbosquid.com/FullPreview/Index.cfm/ID/1283969 [Accessed April 20, 2020].

**Turntable**

From Unreal Engine template *Photo Studio*.

**Icons**

Created by Adobe Photoshop.

**Background Music**

Track: Slow Vibing. From https://freemusicarchive.org/genre/Ambient_Electronic [Accessed May 20, 2020].

**Mouse Hover Sound Effect**

Button Sound 16. From https://soundjay.com/button-sounds-2.html [Accessed May 20, 2020].

**Screenshot Sound Effect**

Camera Shutter Click 08. From https://www.soundjay.com/camera-sound-effect.html [Accessed May 20, 2020].

# Appendix E - Contents of Attached USB Card

/01_CarConfiguratorSourceCode

/02_WindowsPackage

/03_SignallingWebServer

BachelorThesis_PixelStreaming_HouShiying.pdf

# Bibliography

[1] A. Cookson, R. DowlingSoka, C. Crumpler, *Unreal Engine 4 Game Development in 24 Hours*. Indianapolis, Indiana: Sams Technical Publishing, 2016.

[2] T. Shannon, *Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings*. Boston: Addison Wesley, 2017.

[3] T. Parisi. *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages*. Sebastopol, California: O'Reilly Media, 2014.

[4] Unreal Engine Official. (2020, January) *Streaming Unreal Engine Content to Multiple Platforms*. [Online]. Available: https://cdn2.unrealengine.com/Unreal+Engine%2FPixelStreamingWhitepaper%2FPixelStreamingWhitepaper-V1.9C-ec1985a0f32c53c6650b2d02c11a67c32fa4e176.pdf. [Accessed May 21, 2020].

[5] Unreal Engine Official. *Unreal Engine Official Documentation*. [Online]. Available: https://docs.unrealengine.com/en-US/index.html. [Accessed March 3, 2020].

[6] S. Bennett. (2019, February) *The Experience Age Has Arrived*. [Online]. Available: https://www.batimes.com/articles/the-experience-age-has-arrived.html. [Accessed May 26, 2020].

[7] Unreal Engine Official. (2020, January) *Discover Pixel Streaming: real-time distributed content for any device*. [Online]. Available: https://www.unrealengine.com/en-US/tech-blog/discover-pixel-streaming-real-time-distributed-content-for-any-device. [Accessed May 21, 2020].

[8] J. Antunes. (2020, January) *Pixel Streaming in UE4: a solution for real-time distributed content*. [Online]. Available: https://www.provideocoalition.com/pixel-streaming-in-unreal-engine-real-time-distributed-content. [Accessed May 25, 2020].

[9] Blender. *Home of the Blender Project*. [Online]. Available: https://www.blender.org/about. [Accessed May 20, 2020].

[10] K. Finley. (2012, July) *What Exactly Is GitHub Anyway?* [Online]. Available: https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway. [Accessed May 25, 2020].

[11] Google Developers. *Web Fundamentals*. [Online]. Available:

https://developers.google.com/web/fundamentals. [Accessed May 25, 2020].

[12] Advanced WebDev. (2020, March) *Modern web rendering strategies*. [Online]. Available: https://advancedweb.dev/web-rendering. [Accessed May 20, 2020].

[13] K. Bhatt. (2020, April) *Client-side Rendering Vs. Server-side Rendering Vs. Pre-Rendering for Web Apps*. [Online]. Available: https://www.bacancytechnology.com/blog/client-side-rendering-vs-server-side-rendering-vs-pre-rendering. [Accessed May 20, 2020].

[14] K. Shah. (2020, January) *Client-Side Vs. Server-Side Rendering: What to Choose When?* [Online]. Available: https://dzone.com/articles/client-side-vs-server-side-rendering-what-to-choos. [Accessed May 20, 2020].

[15] A. Tavakkoli. *Game Development and Simulation with Unreal Technology*. Natick, Massachusetts: A K Peters, 2015.

[16] M. A. Gutiérrez, F. Vexo, D. Thalmann. *Stepping into Virtual Reality*. Manhattan: Springer, 2008.

[17] TurboSquid. *TurboSquid: 3D Models for Professionals*. [Online]. Available: https://www.turbosquid.com/. [Accessed April 28, 2020].

[18] B. Carnall. *Unreal Engine 4.X By Example*. Birmingham: Packt Publishing, 2016.

[19] Web APIs. *Getting started with WebGL*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL. [Accessed June 2, 2020].