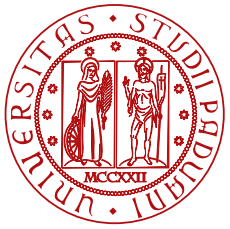

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Development of a virtual reality application for the training of cardiac surgeons

Master Candidate

Boscolo Cegion Nicola

Student ID 2074285

Supervisor

Prof. Savino Sandro

University of Padua

Academic Year
2023/2024

To my parents
and friends

Abstract

This thesis describes the design and development of a virtual reality application for the Meta Quest 2 visor for the training of doctors and nurses in the field of pediatric cardiac surgery. The purpose of the application is the presentation of case studies through the immersive visualization of 3D models derived from imaging techniques. The application allows the creation of virtual classrooms where students can interact with 3D models of physiological or pathological hearts under the supervision of the teacher or independently.

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

VR Virtual Reality

PC Personal Computer

OS Operative system

HMD Head Mounted Display

JSON JavaScript Object Notation

API Application Programming Interface

UE Unreal Engine

SDK Software Development Kit

NDK Native Development Kit

REST Representational state transfer

HTTP Hyper Text Transfer Protocol

APK Android Package

IP Internet Protocol

UI User Interface

RPCs Remote Procedure Calls

FPS Frame Per Second

1 Introduction

1.1 Case introduction

At the university hospital of Padua, there is a very important cardiac surgery center where various heart interventions are performed on many people, the need to teach and visualize the case of operation is very important.
Thanks to MRI, surgeons can see not only pictures of the heart, but also can even make 3D models. Each 3D model can show every detail of the patient heart, this can make surgeons able to analyze and show where and how to resolve the problem.

1.1.1 Virtual reality head mounted displays

The equipment: The university of Padua has some Meta quest 2[1] [fig:1.1] a Head Mounted Display (HMD) for Virtual Reality (VR).
The Meta Quest 2 is a standalone HMD, this means that they don't need other peripherals like an external console or Personal Computer (PC) for working.
For navigating, the Meta Quest 2 uses two wireless controller, but it also has the possibility to just use your own hands to navigate the interfaces. For the context awareness it uses 4 infrared cameras and sensors like multi axes gyroscopes and accelerometers.
They use a custom version of the Android Operative system (OS), this can give

[1] All rights to meta reserved

a certain degree of liberty in creating APPs for the device.

Figure 1.1: Meta quest 2

Use cases of VR: Principally the surgeons are using VR equipment for training and showing critical health conditions of different patients hearts. They are using an APP called Shapes XR, this APP has a web interface for upload 3D models and then show them on the HMD. The app has a multiplayer functionality so that multiple people can look at the 3D models in the environment, even if the developers recommend at max 8 people, they tested with 14 users connected and there weren't any problems.

1.1.2 APP PROBLEMS

How Shapes XR works: Shapes XR lets you create rooms, accessible via a code, were multiple people can create 3D models with basic tools like 3D brushes and standard shapes like cubes, pyramids and so on. It also let you upload a 3D model file on their own website, so that in the home you can download it and start to work on it. It let you also create your own avatar. This is the main feature that the surgeons are using for show the 3D Models

The Problems: Unfortunately Shapes XR is principally used for 3D modelling, so the app has a lot of features like changing the scale of the world or brushes for modelling the objects that aren't useful for the surgeons, and quite distracting

because for a lot of people is the first time using a HMD.
The user experience is extremely important in VR because is difficult to tutoring the user while is using the HMD. Then Shapes XR position the user in an empty 3D plane with little to none point of reference so If a user accidentally uses the teleport function, they may find themselves somewhere far away from the scene they are supposed to be watching.

Feedbacks from surgeons and nurses: The 12/12/2023 I have whiteness a lesson with the integration of the Quest 2 and Shapes XR. First it was pretty chaotic, a lot of people didn't even know how to use the controllers, and they had problems even for putting the code for entering the session. Unfortunately we didn't have time to make a nice lesson for teaching how to use the HMD, the tutorial made by Meta approximates takes 15min to complete, even more if the user want to try the mini-games, so we did not have the time to show it. The main critical points were:

- Inadequate tutoring for teaching how to use the HMD
- Difficulty for accessing the multiplayer room
- Difficulty at moving in the room
- Some people avatar were blocking the visual of some people

How we can see a new app for this use case could be useful, and this is the main reason why this project exist.

2

Requirements

2.1 Functional Requirements

The main features of the project must be:

- **Show custom 3D model at runtime:** The software must be able to download 3D models in OBJ format and rendering them at runtime, with also show coloring made by a surgeon. The 3D model should also be movable and have the possibility to change its size.
- **Multiplayer functionality:** The software must recreate a 3D virtual classroom where a professor can show the 3D model to the students, all students and professor must be synchronized.
- **Upload of 3D models:** There will be a web portal were a surgeon can upload and preview 3D models.
- **Tutoring functionality:** The software will have some tools for learning such as lase pointers, and the possibility to change the position and dimension of the 3D model.
- **Compatibility with Meta quest 2:** The system will need to run on the Meta quest 2.

2.2 Data requirements

The system necessitates the following data (with the following characteristics) to fulfill its objectives:

- **Required Data:**
  - 3D models in OBJ format.
  - Local Internet Protocol (IP) address for multiplayer.
  - Generating and managing codes for multiplayer sessions.

- **Data Format:** All data must be respect its standard, other communication betwin clients and server must use JavaScript Object Notation (JSON) format file. These formats facilitate data exchange with external systems.

2.3 Non-functional requirements

To ensure the development of an effective VR software, the following non-functional requirements must be considered:

- **Use of Game engine:** A game engine is a software made by another company that is capable of creating a video game, by giving the developers some tools for making the development experience quicker and easier.
- **Simple Back end:** The backend must do the least things effectively because so that the IT department does not need to update the server that will host the services.
- **Compliance with Internet standards:** The system will be compliance with Hyper Text Transfer Protocol (HTTP) for all communication betwin server and client, the HMD will use the game engine multiplayer system.
- **Maintaining performance:** VR applications need to perform extremely well for having a good VR experience, Meta allows a minimum of 72 Frame Per Second (FPS), but our target for a better experience will be 90 FPS.

3

Preliminary project

The project is made by three main components: VR APP, backend, Web User Interface (UI). This chapter will talk about them.

3.1 The VR APP

This section will explain the main choices behind the VR APP

The development environment: There are three main way to develop on the Meta Quest 2:

- **Native:** By using native Application Programming Interface (API) that Meta provides for the HMD.
- **Unity:** Is a famous game engine principally used for lightweight video games, It is pretty functional and easy to use, its programming language is C#.
- **Unreal Engine:** Is a famous game engine used for big games, its performance is the best in the market, it uses C++ and a graphical programming language called Blueprint.

As we said in the non-functional requirements we will use a game engine, I have opted for Unreal Engine (UE) because of its performance, the 3D models that will use are complex (~1-10MBytes in size), so it will be used for its performance, also it has a lot of tools for multiplayer and a good documentation other than a big community of developers.

3.2 Unreal Engine

Made by Epic Games, the project was born for the video game Unreal, now is one of the best engine that power a lot of important video games and 3D animation. For this project it will be used the 5.2.1 version because at the time of writing this thesis for the best compatibility with the HMD. The main reason of this upgrade was for faster building time and more advanced API of Meta.

The fundamentals: UE has a 3D preview mode that let the user set the various 3D objects in the scene. In unreal a scene is called "Level", each element in a level is called "Actor", Actors are made by multiple components.
Each element of unreal can be build with a propriety system called Blueprint, a visual programming interface, or via C++, or by combine them. Principally Blueprint can make the developing of the project faster and easier at the cost of performance, C++ perform better, and it has a bigger range of tools. So It is important to combine both languages for having the best performance and flexibility in the project.
Like a lot of game engines, UE tries to render as much frame as much as possible, each cycle of rendering is called tick.
Its important that long actions must be asynchronous respect the game ticks, and if something must be runner in the so-called "game thread", it must be as fast as possible so that the frame rate does not drop to a certain level.
Another important component is the player controller, this is the instance of a player inside a level, a player controller can possess a pawn actor or a character actor.

External library: One challenge for this application is having a correctly multiplayer session between HMD, unfortunately UE does not provide a `VRpawn` that have multiplayer functionality. For having a faster developing experience, it will be used a library called VRexpansion[1], the library is open source under MIT licensing. The main useful components are:

- **VRcharacter:** a character for VR games, all its components can be replicated in a multiplayer session.

[1] `https://vreue4.com/`

- **Graspable Actor:** an actor that can be gripped by the HMD components. It also can be replicated in a multiplayer session.

Events: Events are what start the execution of code inside a Blueprint. Each blueprint have its own standard event, they depend on the object, the basic ones are:

- **Begin Play:** runs one time when the actor is spawned (this is not a constructor).
- **On tick:** runs on every game tick.

But we can also create custom event with Blueprint or C++, they can be triggered whenever we want, they can also be replicated in clients in multiplayer sessions. Like functions, events can have inputs but not outputs.

Multiplayer: UE just support a client-server configuration for manage multiplayer session, it has two main implementation: Stand-Alone server and listening server.
Stand-Alone server consist in having a server that emulates the gaming session, it requires a server powerful enough to run the basic function of the game event if it does not need to render it.
Listening Server does not need a Stand-Alone server, simply one device acts not only as a client but also as a server. Normally client-server is the best in performance and latency, but because this software is simple, the listening server is the right implementation.
For making a good multiplayer we have some blueprints that can help with the synchronization of the data:

- **Game Mode:** runned only inside the server, as it is called it provides the rules how the game should work, it is important for the login and log out of the players.
- **Game state:** it represents the state of the game, all HMD has one, but it is always replicated by the server.
- **Player controller:** runned in every HMD, it represents the player using the HMD.
- **Player state:** it has all the data of a player like the username, its replicated.

An important feature for synchronize events between client and server are Remote Procedure Calls (RPCs), RPCs are events of actors that can per reproduce in multiple device at once by following one of these rules:

- **Client** The RPCs is executed on the owning client connection for this actor.
- **Server** The RPCs is executed on the server, it must be called from the client that owns this actor.
- **Multicast** The RPCs is executed on the server and all currently connected clients the actor is relevant for, Multicast RPCs are designed to be called from the server, but can be called from clients. A Multicast RPCs called from a client only executes locally.

3.3 Network Infrastructure

The Network Infrastructure [fig:3.1] it is simple, a server will be hosted in the IT department of the hospital that will host the Representational state transfer (REST) server for managing 3D Models and multiplayer sessions, and in the same server will host via NodeJS the Web Portal for managing the 3DModels. The multiplayer itself will be manage by the host HMD

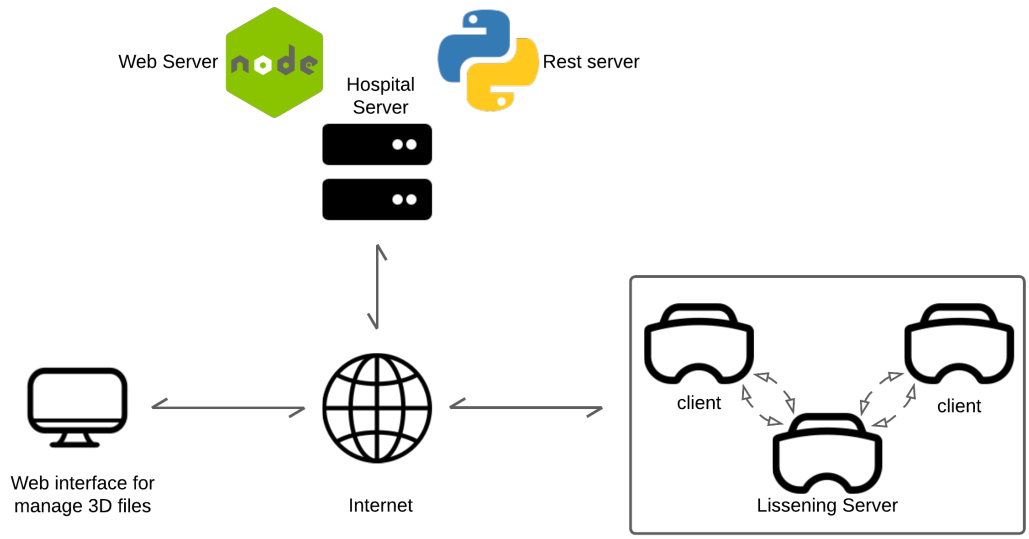

Figure 3.1: Network Schema

The backend: The backend is a simple Python program that it functions as a REST server, were each 3D model is a resource. It also manages the multiplayer session by creating the session code, and save the IP address of the listening server. The library used is called Flask[2], it simply let you run a function respect to a HTTP response received.

The Web UI: The Web UI is made with ReactJS, a popular framework for website, the main reason of this choice is for faster development because its community is massive, in fact will be using a UI library called MUI[3] and a 3D library for rendering 3D models called React Three Fiber[4].
The main functionality of the Web UI will be:

- previewing 3D models with the right colors
- upload 3D models
- delete 3D models

3.4 Development environment

It is important to set the right development environment for this project, even if for the Web UI and backend it is a standard NodeJS and Python environment, it is not that simple for UE
First the choosing the right version is very important because Meta does not always test all UE version, at the start of the developing of the project, the last one was 5.2.1, we also need Visual Studio with the right components for compile in C++, we could use other text editor, but Visual Studio is the most completed one for unreal. For other information look at: [2] After we set up the basic UE environment, we have to pick the right Android studio and Software Development Kit (SDK) for building for Android
Another important component is the Android Native Development Kit (NDK) that enables UE to compile native C++ code. But for installing there is a command line tool that UE shipped with that can install the right version. Because the

[2] https://flask.palletsprojects.com/en/3.0.x/
[3] https://mui.com/
[4] https://r3f.docs.pmnd.rs/getting-started/introduction

requirements differs respect the version of UE for more information look at Epic Games documentation: [1].
Other than the configuration for Android, we also need to add the right components for building for the Meta Quest 2, There are two other components useful for developing:

- **MetaXRPlugin:** plugin for: sideload, publish and setting the project for the HMD, this component is already implemented in the UE fork of META.
- **MetaXRSimulator:** simulator for the HMD, this is extremely important because the compile time of the Android Package (APK) is slow. The simulator uses a PC instance of the game, and it overlays it.

For more information look at: [3]

4

The project

This chapter will explain the development behind the project by starting from the fundamentals.

4.1 The 3D models

In this section will talk about how 3D models are saved and rendered on UE

4.1.1 The OBJ file format

For understanding how UE will show 3D models and how they will be stored, we must talk about their characteristics.

- **Vertices:** points that describe the geometry
- **Faces** indicated were there is a polygon by grouping 3 or more vertices
- **Normal:** there is one for each vertex that is in a face, indicates the direction to which the face is exposed, and for calculating how light is reflected
- **UV:** vectors that helps how a texture should be applied to the model
- **Vertex colors:** RGB vector that indicates a color for each vertex

As we talked about in chapter 2, we will use the OBJ file format for storing files. The React Three Fiber has a native support for OBJ, and the backend server does not need to read the file but just to manage by saving, deleting and send it via HTTP. Unfortunately UE does not have a OBJ file reader usable at runtime but just an importer for what it calls static meshes (3D models that don't have

moving parts). So there is the need to build a parser OBJ to UE custom types. First we need to understand how the OBJ file format is compose of, here a general example code:4.1

```
# List of geometric vertices and colors, with (x, y, z, R, G, B)
v 0.123 0.234 0.345 0.294 0.960 0.258
v ...
...
# List of texture coordinates, in (u, v) coordinates
vt 0.500 1
vt ...
...
# List of vertex normals in (x,y,z) form
vn 0.707 0.000 0.707
vn ...
...
# Faces
f 1 2 3              # simple
f 3/1 4/2 5/3        # with UV
f 6/4/1 3/5/3 7/6/5 # with UV and normal
f 7//1 8//2 9//3     # with normal
f ...
...
```

Code 4.1: OBJ file example

The vertices, UV and normals are simply written, instead the faces have different formats, first not always they use triangles, but also quads, this depends on how the file was exported. For ease of use the parser will support both. The numbers of the face are the indices of vertex, indices starts from 1. Then a face can have also the UV and normals corresponding for the vertex. For our use cases UV aren't needed, but for future-proof they are still been parsed correctly.
Sometime is useful to divide the 3D model in multiple objects the OBJ format represent by dividing the 3D model with a "o". OBJ can also divide the faces in groups by dividing them with a "g". Here an example of how the division works: code:4.2

The software that the surgeons are using for exporting 3D models just support groups, so will implement those, and they will become useful for rendering the model in parts.

```
o object1_name # delaration of the objet name
v...
vn...
g group1_1_name # declaration of a group
f ...
g gorup2_1_name
f...

o object2_1_name
v...
vn...
g group1_2_name
f ...
g gorup2_2_name
f...
```

Code 4.2: OBJ grouping

4.1.2 To unreal types

UE has some custom classes for manage thing like vectors, colors... this classes other to have useful methods they also interface with the blueprint system, we can for example expose variables or functions, so we can call them at blueprint level. This is very important so that we can interconnect the C++ components with blueprints.
Unreal has a component called `procedural mesh`, this component has the possibility to render a 3D model given vertices and triangles, it also has more data that you can feed to the rendered mesh such us: normals, tangents, UV, vertex colors. It can also have collisions and a material.
The `procedural mesh` has also the possibility to load the mesh in parts, so the parser will save the different triangles in the various groups that are defined in the file. This will be important later for loading time.
Vertices are directly read and saved in an array of `FVector` and normals will be saved in the same way. Vertex colors just need to be read and put in a `LinearColor` array, the object itself can be initialized with the data retrieved in the file. UV because are 2D vectors will be saved in an array of `Vector2D`. Unfortunately there is a mismatch between how UE manage correlation between vertices and normals respect the OBJ file standard. Unreal needs that two arrays that contains vertices and normals, so that the vertex in the array at the position `i` must have its normal in the normal array at position `i`. This is still a trivial problem, because there's just the need to load all the normals in memory and

then save them again in the correct order decided by the faces.
UV are being manage in the same way. Another problem is that unreal just accept triangles and not quads, and because it is a common practice to use quads when exporting 3D models the program will convert quads in triangles, this is pretty trivial, for each quad we can divide it in two triangles.
Unreal also works in `Z`-up coordinates that means that the `Z` axis points up, there is another standard called `Y`-up were the `Y` axis points up, unfortunately the OBJ file format does not have any ways to reference scale or if the file is saved in `Z`-up or `Y`-up, so it is important to export the file in `Z`-up.

Because when loading a big procedural mesh it could create a big loss in FPS, for reduce this problem the model will be rendered in parts, by using the different groups found on the file. After some testing I have decided to load groups every `0.6ms`.

4.2 The user experience

Thanks to VRExpansion plugin, we can use the standard `VRCharater`, this `Character` has already implemented the online synchronization, and it also has the components for the controllers and camera management. The controllers can `grip` any `Actor` or `Componet` that implement the interface `VRGrip`, this will be used for moving the 3D models. Other than that the `VRCharater` is an empty blank, will need to implement some functions for making fully functional.
Here are the main components to develop:

- Input management
- Widget Interaction
- Interaction pointer
- Side menu
- Grip framework
- 3D model size management
- Loading sphere

4.2.1 Input management

Input in UE is manage by two data files: `Input Action` and `Input Mapping Context`. In the next two paragraphs will be addressed how the input works, and then will be used in the various components of `VRCharater`.

Input Action Are files that are used to identify a certain input of the controller, each file should be named after an action more than the input used for making clear what they serve. For example:
For using the `A` button find in the right controller, you need to have a file that represents the button, the necessary settings are: Consume input which allows you to take into account that the input has been served, and the type of value that in this case will be `Digital (bool)`.

Input Mapping Context Each file represents all the inputs used by an actor, an actor could change its inputs, so they can be multiple files for different occasion, Here each `Input Action` will be associated with the corresponding input. `Input Mapping Context` can be used for other objects so that they can override the standard behavior of the `Character`, for example by equipping a laser pointer and using the `A` button for toggle it. For setting the `Input Mapping Context` there is a function called `Add Mapping Context`.

5

Conclusions and Future Updates

A	**B**
C	D
E	F
G	H

Table 5.1: Table example

References

[1] Epic Games. *Getting started with Android*. https://dev.epicgames.com/documentation/en-us/unreal-engine/how-to-set-up-android-sdk-and-ndk-for-your-unreal-engine-development-environment?application_version=5.2.

[2] Epic Games. *Getting started with Visual Studio*. https://dev.epicgames.com/documentation/en-us/unreal-engine/setting-up-visual-studio-development-environment-for-cplusplus-projects-in-unreal-engine?application_version=5.2.

[3] Meta. *Creating Your First Meta Quest VR App in Unreal Engine*. https://developer.oculus.com/documentation/unreal/unreal-quick-start-guide-quest/.

Acknowledgments