Seamlessly Displaying Models in Virtual Reality

Pontus Gagnero
Peter Huang

Faculty of Health, Science and Technology

Computer Science

15 ECTS

Supervisor: Tobias Pulls

Examiner: Johan Eklund

2018-01-16

Seamlessly Displaying Models in Virtual Reality

Pontus Gagnero and Peter Huang

This dissertation is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this dissertation which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Pontus Gagnero

Peter Huang

Approved, 2018-01-16

Advisor: Tobias Pulls

Examiner: Johan Eklund

Abstract

Virtual reality is a term used to describe a simulated reality, where the user is shown the simulated world instead of the real world by using a headset. The popularity of virtual reality is rising every year. The goal of this project is to create a solution for industries such as production companies to demonstrate their products in virtual reality as easy as possible. To create a seamless solution three parts are required: a graphical user interface, a converter, and an engine. When these components are combined into a program, a user can easily select a model file that is automatically converted to an engine-supported format. The file is then used in the engine to display the model in virtual reality. We implemented a prototype solution focused on a seamless integration of a custom-made user interface, the third-party library Assimp for the file conversion, and Unity3D as the virtual-reality engine. The prototype was evaluated in terms of supported file formats, the conversion time and corresponding memory usage.

Acknowledgements

We want to thank our advisor Tobias Pulls at Karlstad University for giving us helpful advice and proofreading our thesis. We also want to thank Altran for giving us the opportunity to work with them, and especially David Glennberg for his support.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Virtual reality (VR) is a term used to describe an experience of a reality that is not real, but simulated [1]. As our senses are the only way of perceiving reality, they can be misled. Misleading is done by replacing our senses information with made-up information, e.g., a simulation of reality, or a completely made up reality. As vision is the strongest sensory input that humans have, replacing it with a virtual world can make a person feel as if they are in another world. Lately VR has started getting popular due to it becoming more possible with recent hardware improvements. Previously VR had performance limitations and low capital interest. The growth of VR has become a fast-growing trend over the last couple of years, going from almost nothing except a few novel toys over the years, to an established business [2].

Industries, especially in the manufacturing area, are looking for a solution to be able to display products and 3D models in VR, but at the same time does not want to go through the problems that comes with it, especially regarding file conversions. While this can be solved, there has been no easy, general solution available. They all required downloading the file, converting it and finding a way to render it, and in VR, all separately.

This project is aimed at making an easy-to-use prototype. We designed the prototype and implemented the prototype by combining several libraries and tools into one solution. What follows is the details of our methods, choices and ideas that we have gone through creating this prototype. The solution will give the ability to easily display the models from commonly used 3D file formats in VR, with only a few clicks. File formats are different ways of storing 3D models. As these industries often use different formats to store important information about their models. The difference of the structure of information can be problematic to work with. This makes converting between files an essential part of the project. As we require a specific file to render we need to convert any file that we need to render into that specific file. This means that we need a converter. By making this part

automatic instead of a process where the user has to find a converter, download it and then use it. We decided to put the converter in the program without the user even having to think about the converter for the file format.

This gives the program a purpose in industries where the user does not need to know how to do any of the previously stated steps. The user should simply take a file, of the supported file formats, that they want to show, and the program will display the model of the file without any problems.

In conclusion, the way we decided on structuring our solution is according to Figure 1.1. The figure shows an overview of how the program will work from a user perspective. The program takes a file from one of the supported formats, puts the file through the user interface where some settings can be tweaked and then displays the model in VR.

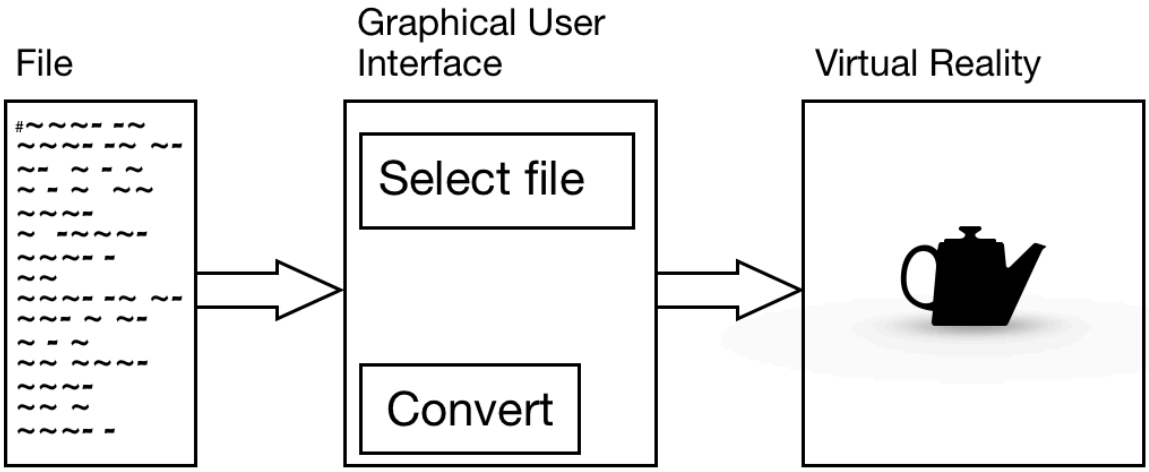


Figure 1.1: Overview of the project.

The result of this project is a prototype. The prototype consists of a graphical user interface (GUI), a converter, and an engine. A user uses the prototype to select a file, converts the file and then imports the file in the engine to render in VR. A number of files are tested for the prototype.

The rest of this report is structured as follows: Chapter 2 introduces VR, the engine and the various development tools used in this project. The motivation for this project

is also explained at the beginning of the chapter. In Chapter 3 we present the design for the user interface, converter, VR environment, VR functionality, and the various camera modes. The implementation chapter, Chapter 4, describes how we implemented the project with the different components presented in the design chapter. We also explain our implementation choices. The results of this project, such as the performance, scaling, mesh, collision, and a potential security risk, and a conversion performance evaluation are presented in Chapter 5. Finally, Chapter 6 we conclude the project and present a general summary, the improvements, and the future work.

2 Background

The project is about designing and creating a generalized solution for viewing 3D models in a virtual environment. These models come in different file formats which needs to be converted. This is of interest to the company Altran who wants to sell this solution to other companies which then will be able to show their models to their own customers. The project includes creating an easy-to-extend generalized solution, so it can be specialized for some use cases.

The following sections defines the different terms for this project, such as the different programs, file formats, and the tools used. Section 2.1 explains the Virtual Reality and HTC Vive. The engine and the plug-ins are described in Sections 2.2 and 2.3, respectively. Section 2.4 mentions what file formats the converter has in the scope for this project. The tools and programming language for this project are described in Sections 2.5 to 2.8.

2.1 Virtual Reality

Virtual reality is a technology that simulates an environment in 3D using VR headset technology. One of the most prominent ones is the Vive made by HTC [3]. Vive contains the base stations for the room set-up, the headset, and the controllers. The base stations are two sensors placed diagonally opposed to each other in the room. The two sensors create a virtual play room and detects the motions of the Vive headset and the controllers. Vive is the most widely used VR headset in the personal computer (PC) market [4].

Current day technology gives the user the ability to give inputs, receive outputs and to interact with the objects within the virtual world. Modern VR is achieved by using both hardware and software specialized for VR to give the user an immersive experience [5].

2.2 Unity3D

Unity3D is a multi-platform game engine from Unity Technologies [6]. The Unity3D engine supports many different types of platforms, ranging from mobile operative system as iOS to a PC with SteamVR [7]. Unity3D supports both Unityscript (i.e., Unity's version of Javascript) and C# (previously Boo[1]) as languages for functions.

A scene in Unity3D is an empty environment, where all the game objects, such as camera, lightning and scripts are placed in the scene. A game object is an entity, which contains components to define what it is, for example adding a camera component to the game object makes the game object a camera. Script components are scripts that can be applied to a game object, they work as object oriented programming and is applied as a component, and can then access the game objects other components and properties.

Unity3D fully supports C# and is also where the majority of documentation for Unity3D is written [8]. The engine is open source compared to many other engines, which means the engine is easier to adapt to the purpose you need the engine for. While Unity3D does not support all the latest technologies natively, it has the basic functionalities and graphical capabilities needed for a VR environment.

2.3 Plug-ins for Unity3D

We use two plug-ins:

pb_STL is an Stereolithography (STL) importer and exporter plug-in for Unity3D made by Karl Henkel. The plug-in is free to use in software as long as the copyright is stated. The plug-in imports an STL file and makes it usable by Unity3D as a mesh of the file. The mesh can be used for rendering it inside the engine.

SteamVR is a plug-in for Unity3D that gives Unity3D VR functionality, it also contains

[1]Boo: https://en.wikipedia.org/w/index.php?title=Boo_(programming_language)&oldid=817784331, last accessed 2018-01-04.

features and examples [7]. SteamVR is developed by Valve. Due to being easy to set-up, the plug-in allows the developer to quickly start with developing for VR.

2.4 3D File Formats

The project requires different 3D file formats to convert from, to give the customer an easier time to make their product work out of the box. Some formats have different functions but in general they contain the mesh of the model, which is what you use to render the objects with. There are a variety of 3D formats due to different CAD (Computer-aided design) software, such as `FBX` format developed by Autodesk [9].

Meshes are elements representing a geometric object [10]. The meshes consist of vertices (points in 3D space), where the edges or lines meets. A triangle is defined by groups of three vertices where the edges are connected. Triangles is what the user see. Triangles in other words are 2D surfaces which only is visible to one side, which is defined by the normal (i.e., normal decides what side is rendered). These surfaces can have colour and/or texture. Together these components make up a basic version of a mesh.

2.4.1 Stereolithography Format

The Stereolithography (STL) file formats stores the information's of the vertices and the normal vector of the model [11]. The vertices and the normal vector is the component for creating triangles which represents the surface of the model. The `STL` file format does not contain the colour, texture, materials or other attributes which allows for quick prototyping of 3D models.

2.4.2 Other Formats

Wavefront Object (OBJ) files are basic 3D files, they only contain the mesh [12]. The `OBJ` format contains simple instructions to specify where the vertices are.

3DS has become a industry standard for file formats [13]. The `3DS` format is relatively simple and works by storing triangles.

COLLADA (DAE) is an open-standard format [14]. The `DAE` format stores the data of the vertex, material and object.

Filmbox (FBX) is an object-based format, which contains motion, 2D, 3D, audio and video data [15]. There still exist meshes inside the format which can be interpreted.

AutoCAD Drawing Interchange Format (DXF) is a complex and partly undocumented file format [16]. The `DXF` format does not contain the dimensions for the coordinates of the mesh.

2.5 Assimp

Assimp is an open source library used for importing and exporting various 3D model formats [17]. The library is used to convert 3D file formats into other file formats. Assimp is written in C++. External bindings are required to use Assimp in another programming language, such as C#.

2.6 C#

C# is an object-oriented programming language developed by Microsoft [18]. The C# language is a type safe language. A known feature in C# is its garbage collection to free memory used by unused objects. The C# programming language has similar syntax to the programming languages C, C++, and Java.

2.7 Integrated Development Environment

Visual Studio is an integrated development environment (IDE) created by Microsoft [19]. The IDE supports a couple of programming languages including C#. Visual Studio has

features (e.g., NuGet package manager, plug-ins, debug, tests) necessary to develop a product with ease. MonoDevelop is an IDE with cross-platform support [20]. C# is the programming language used in MonoDevelop to write programs for Unity3D.

2.8 GTK#

GTK# is a GUI system used by MonoDevelop and .NET [21]. The GTK# GUI system binds with the GTK+ GUI system [22]. Both systems are cross-platform and open source. The system supports different programming languages including C#. A variety of layouts and methods are provided by GTK#.

2.9 Summary

This chapter describes the background, the different terms, hardware and software. We described what VR is and the ways to implement it using both hardware, such as HTC Vive, and software, such as Unity3D and MonoDevelop. The different tools and plug-ins we used to implement the solution are also described. We described the various 3D model file formats and the 3D model geometry.

3 Project Design

To view an object in VR, the object file needs to convert to a supported format for the engine before rendering the object. The file conversion is needed because there are more file formats than the engine supports, and it is easier to handle one file format in the engine than multiple different types of file formats. The design of the GUI is presented in Section 3.1. Section 3.2 explains the converter. The environment, VR camera mode, and the optional camera modes for viewing the rendered object is described in Section 3.3. Figure 3.1 is an overview of the design for file conversion and rendering in engine.

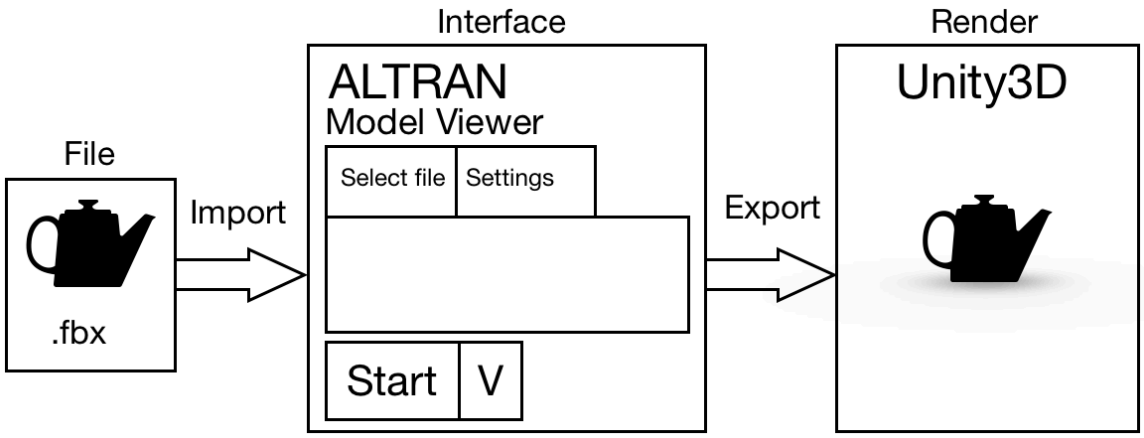


Figure 3.1: Overview of our design.

3.1 Graphical User Interface

To design the GUI, we sketched the GUI in several iterations. We compared each of these sketches until we were satisfied with the look of the GUI. The early sketches are the ideas for the layout and the components, such as the buttons. The different components of each sketch are selected and put together into the final sketch of the GUI. We chose to use the final sketch because the order of components is positioned from the top to bottom. The final sketch was also preferred visually by us, and it shows clearly what the different components

are. The layout has similarities to other programs which influenced our choices. Figure 3.2 shows the four early sketches of the GUI. Figure 3.3 shows the final sketch of the GUI.

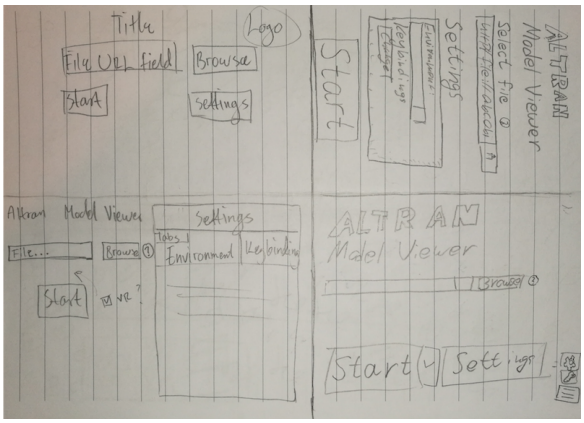


Figure 3.2: Four early sketches of the GUI.

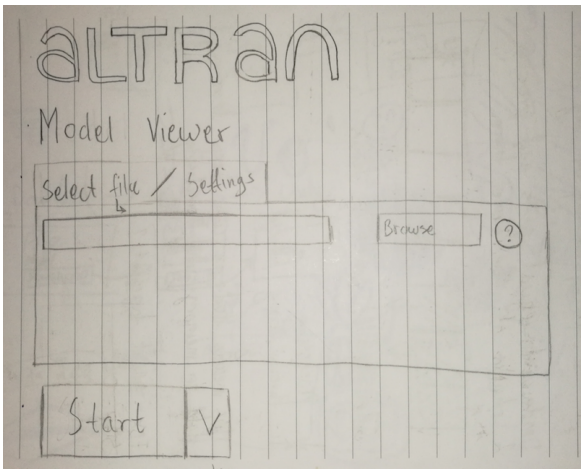


Figure 3.3: The final sketch of the GUI.

The final sketch of the GUI consists of tabs, "Select file" and "Settings". The "Select file" simply opens a new window for selecting a file. The settings tab displays the different settings for the VR world. A "Start" button and a "V" check box/toggle button are positioned at the bottom of the GUI. The "Start" button simply starts the program to convert the 3D model file to another file format and render it in the engine. The user can choose if they want to display the model with VR on or off. The help button contains the

instructions on how the GUI and the converter works. Figure 3.4 shows the final sketch of the GUI.

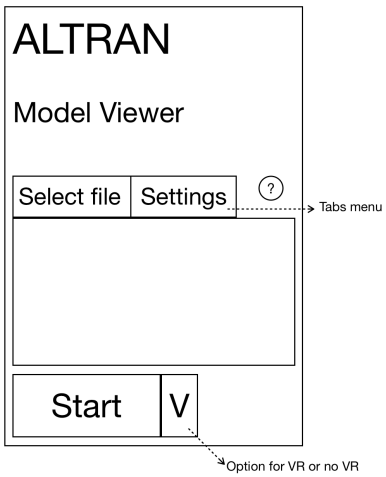


Figure 3.4: The final revision sketch of the GUI.

3.2 Converter

The design of the converter is to be built in the GUI. A converter library will be used. The library converts file formats, such as the non-compatible ones to one of the compatible file format for the engine.

3.3 Engine

The VR environment will be basic, consisting of a simple plane acting as the ground with the rendered object and a logo. Additionally, the environment will have invisible planes on the edges of the environment acting as invisible walls. Figure 3.5 shows a simple picture of the environment.

There are three different camera modes: VR mode, orbit mode and first-person mode. The user can switch between these three modes in real time. The VR mode is the default

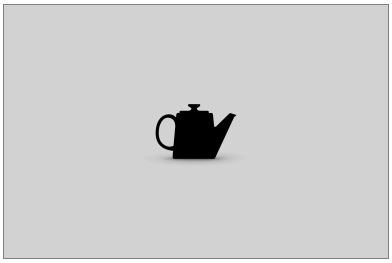
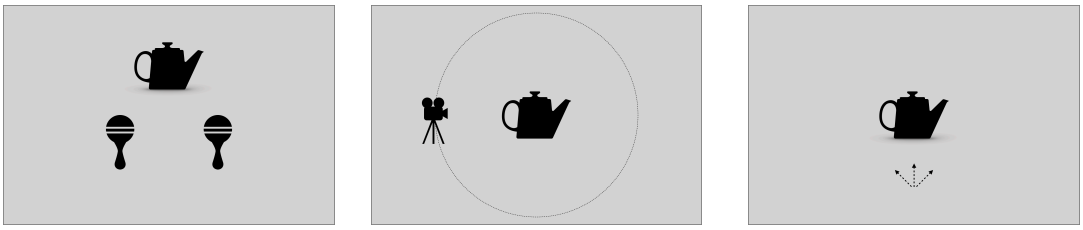


Figure 3.5: An illustration of the environment.

mode and the other two modes are used if VR is not available and for testing purposes. Figure 3.6 shows the three camera modes.



(a) VR mode. (b) Orbit mode. (c) First person mode.

Figure 3.6: Three simple illustrations of the VR, orbit and first-person mode, respectively.

3.3.1 VR Mode

The design of the VR mode includes interactions with the object by using the VR controllers. The user can use the VR controllers to pick up the object, rotate and move it around. The user can teleport across the room. The teleport gives the user an easier way to move around in longer distances. To teleport we need a laser pointer to point in the room and teleport to where the laser pointer is pointed at.

3.3.2 Orbit Mode

The orbit mode is designed to be used without the VR mode. The camera rotates with the movements of the mouse and the camera zooms in or out by scrolling the mouse wheel.

3.3.3 First Person Mode

The first-person mode movements and camera is controlled with the computer keyboard and mouse. This mode is used when the user wants to view and move around the object without being in VR mode.

3.4 Summary

The project is designed for the file conversion to render for viewing in VR. The final look of the GUI is created from multiple sketches. The cameras are VR, first-person and orbit. The VR camera is the default and the other two are optional. The design of the VR mode includes object interactions and teleport for easier movement around the room.

4 Project Implementation

Section 4.1 present the implementation of the GUI. The 3D file format converter is described in Section 4.2. The implementation of VR, environment and related functions are described in Section 4.3. The details for the importing, rendering 3D models, and optional camera modes are given in the same section as the VR implementation.

4.1 Graphical User Interface

The GUI was built using C# with `.NET` functionality. The standard libraries for running it cross-platform were implemented with `.NET`'s MonoDevelop. To obtain the desired result of the design, we worked with the GTK# library. The GTK# library was used to create a GUI that was able to be compiled and run on the Windows and Linux platforms. These platforms were used to develop the product. The library contains functions which we used to create the layout. Structuring the GUI to look like the designed user interface was done by adding graphical containers with a vertical flow with a title label, a file selects and a start button. Figure 4.1 shows the implemented GUI.

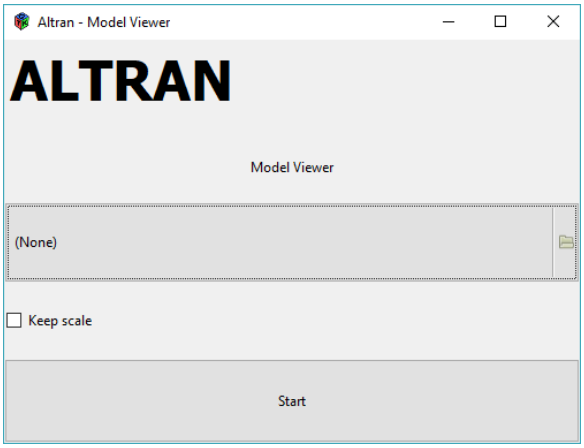


Figure 4.1: The GUI implemented with GTK#.

In the design chapter we had the "Settings" and the help button features in the user interface design, which we decided not to implement, as we only created one environment described in Section 4.3.2, and the help button was not necessary for the program to function. The "V" button for selecting VR viewing or without VR at start up is likewise not implemented, as we implemented a function to quickly switch between the different camera modes described in Section 4.3.3.

We implemented a check box in the GUI for the user to check the box if they want to convert and render the 3D model in their original scale, or leave the box unchecked to render the model with a smaller scale. The reason why we implemented the check box and the function to render the model in their original scale or a scaled down version is evaluated in the next chapter. For the dimensionless formats, such as the `DXF` format, the scale will be decided by Unity3D.

4.2 3D File Conversion

The `STL` format stores the information of the vertices and the normal vectors which are the triangles representing the structure of the 3D models. `STL` does not store information of other CAD attributes such as colour, material or texture, which allows the format to be fast, small and simple. The advantage of the `STL` format is that the format allows for creating 3D model prototypes quickly compared to other format such as the `OBJ` format, which stores additional CAD attributes, such as the material for the model.

Section 4.1 explained how we implemented the conversion button event. The event is triggered by the button. The event is connected to a function which converts the file with the help of the provided file path. In this function we take the path and give it to Assimp which takes the file and converts it to the `STL` file used later. Additional details of the conversion function is explained in Section 4.2.2.

4.2.1 Assimp

There are a few different implementations of Assimp as well as different versions. The versions include different file formats, for example the latest version at the moment is 4.0.1, which contains all formats we have researched. The version we used in this project is Assimp version 3.3.2 for `.NET`, as it is the version which is easiest to implement due to existing as a pre-packaged NuGet package. The Assimp version 3.3.2 does not include formats such as `STEP`. It is possible to use version 4.0.1 or any later version in the program but it would require implementing a C++ to C# wrapper such as the CPPSharp binding library.

Assimp allows two different implementations of it. The implementations are from implementing it as a library function or using the compiled terminal program. After looking at these different methods and trying the terminal option we decided to go with implementing the library version. The Assimp library is written in C++ which can cause problems when it comes to binding it to C# due to implementation time, probability of memory leaks and potential bugs. This is no major problem, but it was a better idea to use the earlier Assimp version with the same core functionality rather than trying to use the latest Assimp version which require external binding only for more formats.

4.2.2 Conversion Code

Assimp was implemented by importing the library AssimpNet, which contains Assimp version 3.3.2 and the `.NET` binders for it. An example of how Assimp is used in our project can be found in Listing 1. We first created a scene, which constructs an object in memory. After which we imported the file to the scene, by giving Assimp the file path as well as a few standard flags to simplify the file. For example, one of these flags is `Triangulate` which splits any faces with more than 3 indices into faces with exactly 3 indices[2]. The imported file is then put in the allocated memory. Once the file has been imported, we

[2]All PostProcessSteps can be found in http://assimp.sourceforge.net/lib_html/postprocess_8h.html, last accessed 2018-01-04.

export the file which takes the data and puts it in the desired file format structure and writes it to a file. In other words, it saves the file from memory to an STL file.

```
var assimpScene = assimp.ImportFile(filePath,
    PostProcessSteps.CalculateTangentSpace |
    PostProcessSteps.Triangulate |
    PostProcessSteps.JoinIdenticalVertices |
    PostProcessSteps.SortByPrimitiveType);

assimp.ExportFile(assimpScene, fileExportPath + "/processed-file", "stl");
```

Listing 1: Assimp usage in C#.

4.3 Implementation in Unity3D

For the engine we had some options. The biggest two competitors for VR development are Unity3D and Unreal Engine. In Unity3D's case it has a big community and a lot of documentation. Unity3D also supports C#, which is the preferred language choice of the team. Unreal Engine has better graphics and a more standard code base but does not support C#, and does not have as much documentation on VR, which means it would take us more work. Unity3D on the other hand uses C# but with their own variety of C# so it contains differences. The Unreal Engine has a smaller community with less algorithms for rendering in real-time. The Unreal Engine is a bit more "heavy weight", which means it takes longer to load and start up. The project is aiming to be as seamless as possible and as a result it was decided that Unity3D should be used.

Unity3D has VR capabilities and supports different platforms. Libraries or assets exist in Unity3D as free to use. This gives us the ability to work faster without programming the basic functions. Only the platform output setting was needed to be changed to get Unity3D working on all relevant platforms. For the project we decided to use a specific

Unity3D version called 2017.1.1xf which is available on Linux and Mac as to be able to co-develop on Unity3D.

4.3.1 Manager

The manager is a game object in Unity3D which contains the `ImportModel` script. This script is important due to how it handles the import and render of the file. The manager exists in the hierarchy for the current game scene. We wrote the `ImportModels` script which contains the code and functions for importing the model and render it. Figure 4.2 shows the flow of the script, it shows a flow chart representation on how the code is structured. The code itself is described in Section 4.3.5.

`ImportModels` is one of the main scripts that connects all the other scripts together. The `ImportModels` script works by connecting the camera as well as importing, modifying and instantiating the object. The script is added to the manager, as shown in Figure 4.3. The figure shows the hierarchy of the scene with all the different game objects, and the inspector shows the manager game object which contains the `ImportModel` script, and the properties for the instance of the attached script. The script properties contain the file name of the processed file which will be imported, and the default material for the file and the size that the script will scale the object to.

4.3.2 World Environment

The world environment is created within Unity3D. The environment is made up with a ground plane as a square with the area $8UU^2$ (Unity units, which is similar to meters). The plane is given a place-holder texture. In addition, a plane with Altran's logo is placed on the edge of the playable area. This would most likely be changed to the customer's logo. We implemented invisible walls in the environment. A wall is created with a plane and the mesh renderer and mesh collider are removed to generate the invisible plane, and then a box collider is added to the invisible plane to produce the invisible wall. Then we duplicated

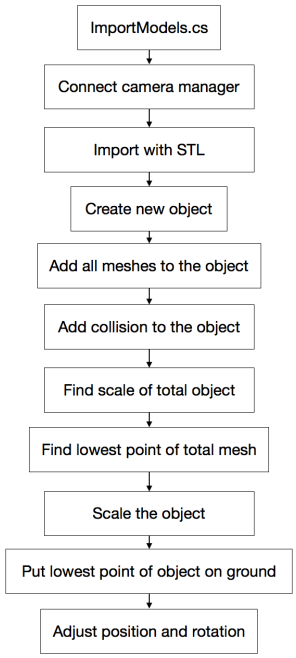
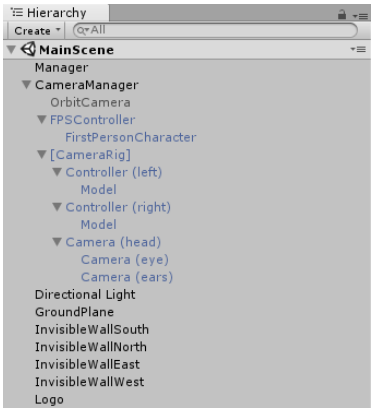
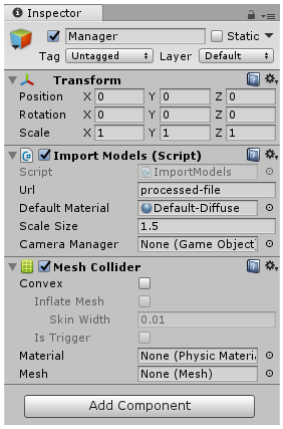


Figure 4.2: The steps for importing the model in ImportModels.cs.



(a) The hierarchy of all game objects in the scene.



(b) The inspector of the manager.

Figure 4.3: The hierarchy and the manager's inspector.

the invisible wall until we had a total of four invisible walls which are then positioned at the edges of the environment. Figure 4.4 shows the implemented environment.

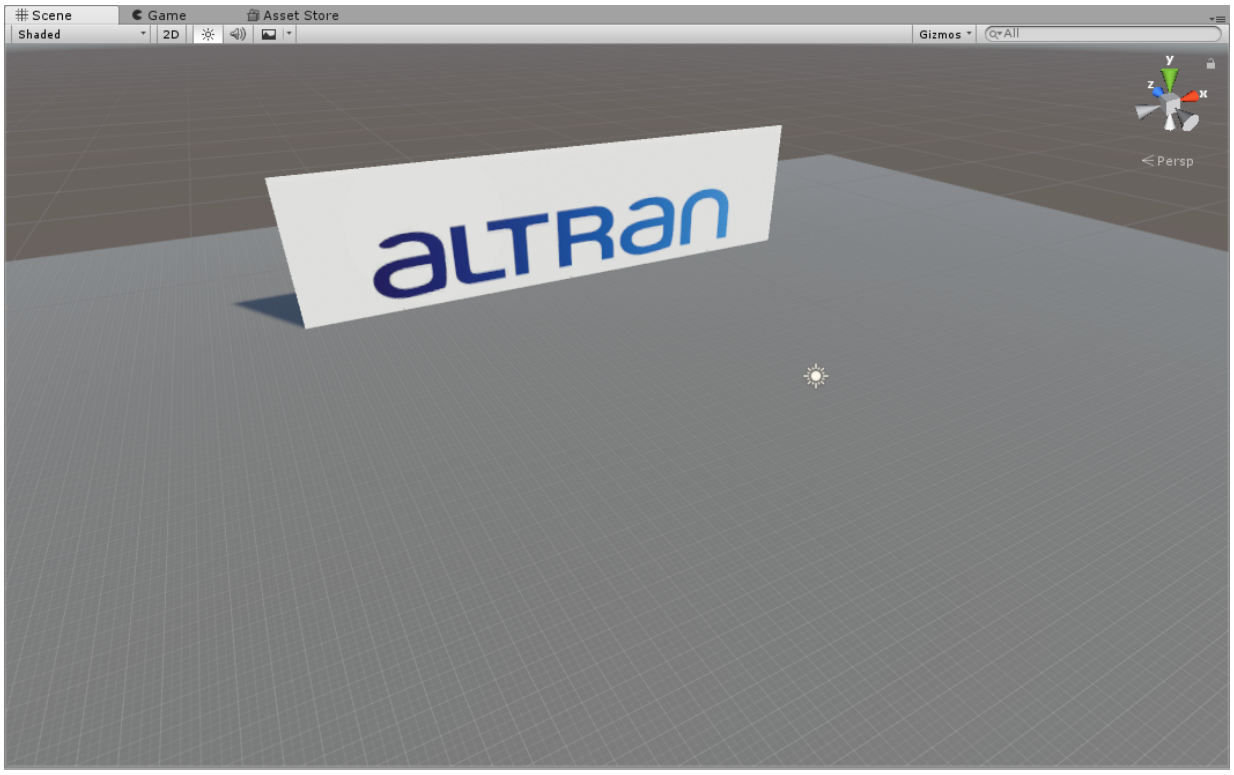


Figure 4.4: The implemented environment and logo.

Meshes can be used both as a graphical object and collider, where graphically it is used with a material assigned to it with the help of a mesh renderer. The mesh renderer component renders the mesh graphically to be visible. The mesh can also be used as a collider with the help of a mesh collider component, where the mesh is used without any simplifications to interact with any other collisions, such as the player. Changing the mesh collider to a box collider changes the complexity of the mesh to a simplified version which has less complex collision and is also lighter performance wise.

4.3.3 Camera Modes

We implemented a camera manager that manages the three different camera modes. The first one is the VR camera which is the primary and default camera mode. The other two are the first-person and orbit cameras which is optional if VR is not used or available. To switch between these cameras, we implemented a function in the camera manager to cycle between cameras by pressing the "F" key on the keyboard. The switching between the cameras is quick and in real time. Listing 2 a snippet of the camera cycling code is listed. The code works on an ordered list of cameras named `cameras`. The code works by changing the current camera to the next one in the list, or if there is no next one it will default to the first camera (camera 0). Then it disables all cameras and after that it enables the current one.

```
currentCamera   = (currentCamera < cameraList.Count - 1)
                ? currentCamera + 1
                : 0;

cameraList.ForEach(x => x.SetActive(false));

cameraList[currentCamera].SetActive(true);
```

Listing 2: Code for cycling between the three camera modes.

VR was implemented using Valve's free library, SteamVR. SteamVR is found on the Asset Store as a free product and is downloaded and unpackaged to the project folder. Using SteamVR's prefab (an asset template) which contains the camera, objects and scripts required for a stripped-down version of VR to be functional. In Section 4.3.4 we go into more details on what scripts we added.

The orbit camera mode is implemented by adding a new camera object to the camera manager. The camera is focused on the target. The script is using the mouse movements

to translate into movement which works as an orbit which goes around target. This is done by using a clamp function to keep the distance constant, as well as the X- and Y-mouse movements to move vertically and horizontally respectively while keeping the same distance from the object and camera locked onto the target. It is possible to move the camera closer or further away from the object by using the mouse wheel. Listing 3 shows the codes for moving the camera with the mouse and the calculation of the camera distance when zooming in.

```
xAngle += Input.GetAxis("Mouse X") * mouseSpeed;
yAngle += Input.GetAxis("Mouse Y") * mouseSpeed;

yAngle = Mathf.Clamp(yAngle, yPositionMin, yPositionMax);
cameraDistance = Mathf.Clamp(cameraDistance - Input.GetAxis("Mouse
    ScrollWheel"), cameraDistanceMin, cameraDistanceMax);
```

Listing 3: Code for the mouse movement and the clamp functions.

The first-person camera mode is implemented by importing the standard assets from Unity3D. The standard assets have the first-person controller prefab which were added to the camera manager as a choice. As Unity3D's default assets contains more than the minimum some functions had to be removed from the first-person controller. The sound for the footstep, jumping and landing are removed as the sounds does not serve any purpose. In addition, the character movement is controlled with the following keyboard buttons, "WASD" to move around, "Space" to jump, "Shift" to run, and the camera orientation is controlled with the computer mouse.

4.3.4 Virtual Reality Functions

SteamVR contains a few pre-made scripts and objects. SteamVR has models for the HTC Vive controllers and a couple of standard functions which we used for the implementation

of the project. However, it did not contain all scripts that we needed for the project, we also had to add a script that picks up objects, we call this the `PickUp` script.

We implemented the `PickUp` script by attaching a script to the controller, which triggers the script once the trigger button is pressed down. If an object is inside the controller's collider, the object is 'picked up'. The script works by disabling physics on the object which is being picked up, by making it kinematic in Unity3D and attaching it as a child to the controller object and therefore it will follow the controller around. Figure 4.5 shows the track pad and the trigger button of the Vive controller.

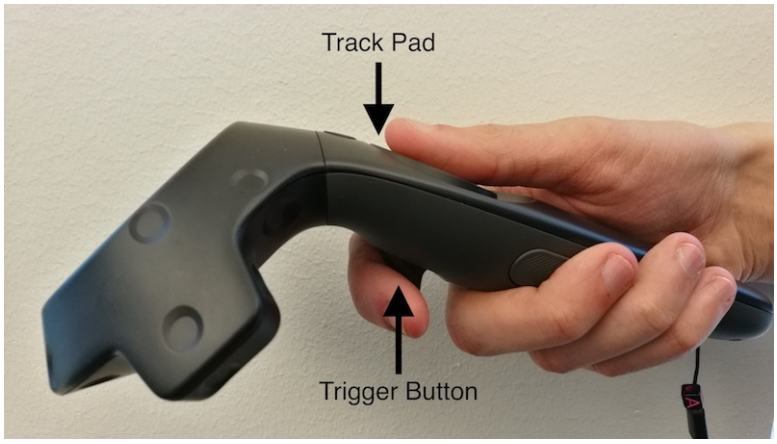


Figure 4.5: The layout of the HTC Vive controllers.

We also added a teleportation script to the controllers which is provided by SteamVR plug-in that adds a laser pointer to the controllers and the teleportation ability. The laser pointer can be seen in Figure 4.6.

The laser pointer itself is added as a component which reacts to the button press and increases in size to indicate that it is being used. The laser is constantly pointing straight out of both controllers. The colour of the laser was set to red which gives it visibility. The teleportation was also added for both the left and right controller. If the trackpad button is pressed the user is instantaneously put at the location that the laser pointer is pointing at, if it is inside the playable area. Figure 4.7 indicated that the laser pointer activated.

Figure 4.6: A picture of the laser pointers.

Figure 4.7: A picture indicating when the laser pointer is activated.

4.3.5 Importing Meshes in Engine

The object is added in two different steps. The first step is to read the object into memory using an external file source. The second is to add the mesh to the engine.

In the first step a function takes the STL file and then translates it into a mesh structure and splits it if necessary. Once it is loaded into a mesh structure it creates a new mesh object and sends it as a list of meshes to the second step.

The second step takes the meshes that were returned from the first step and adds every one of them, as a child, to a shared empty parent game object. Every mesh also adds a mesh collider to the parent game object. The game object is then used to scale the meshes. To do this calculation the highest dimension needs to be found, so the meshes are being looked through to see what vertex is the furthest in all directions. Then the biggest dimension is taken and used to uniformly scale the game object to be maximum $1.5UU$ in any dimension. This is done by taking the desired length (in this case $1.5UU$ as previously mentioned), and dividing it by the biggest dimension's length. The formula looks like this:

$$\text{Scale} = \frac{\text{desired max dimension}}{\text{biggest mesh dimension}} \tag{4.1}$$

The scale is then specified as the game object's size. This makes the game object have no dimension longer than $1.5UU$. It also calculated the lowest vertex point and subtracted it as the lowest point to make it in level with the ground plane. The game object is also made kinematic and gravity was removed. This implies that the game object will only move if the player moves it.

4.4 Summary

The GUI was implemented with the GTK# GUI library to resemble the designed user interface. We explained the conversion functions and the library used for conversion of the 3D file formats to STL, which then are imported to Unity3D for rendering. With Unity3D,

we implemented the VR and its functions, the environment, and the optional way to view the rendered model without VR.

5 Evaluation

The result of the implementation part is a prototype of the project. While several features (e.g., help button, environment settings) are not included, or implemented in a different way compared to the project design chapter, is due to the priorities we made during the implementation and the time limit. We focused on the important features to achieve the main goals of the project.

The prototype has the features to select a file and convert the supported and tested 3D file formats to the `STL` format. The STL file is then imported to Unity3D for rendering and viewing in VR or, as an option, without VR. The environment is an empty room with the rendered object. In VR the user can pick up the object and move it around. With the laser pointer, the user can point with it anywhere in the room and teleport there.

Sections 5.1 and 5.2 evaluate the 3D formats used in this project. We discuss the scaling, mesh and collision of the rendered models in Sections 5.3 and 5.4. A security concern and an issue surrounding the converted-to format are discussed in Sections 5.5 and 5.6, respectively.

5.1 3D File Formats

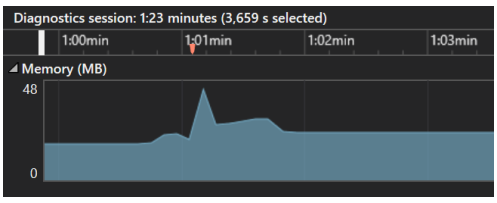
We used the Assimp 3.3.2 library for the conversion between the 3D file formats. We tested that our implementation works with a number of the 3D file formats: `FBX`, `DXF`, `3DS`, `STL`, `OBJ`, and `DAE`. Support for a wide range of formats is a key aspect of our work. The tested files are sample files which were available on a modelling site[3].

5.2 Conversion Time and Memory Usage

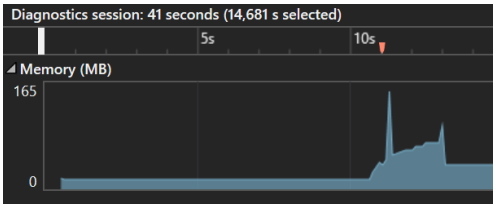
We measured the execution time it takes for the conversion of the 3D file formats to convert to the `STL` format. The data of the memory usage is likewise collected. The data

[3]Modelling site: `https://free3d.com/3d-models/all`, last accessed 2018-01-04.

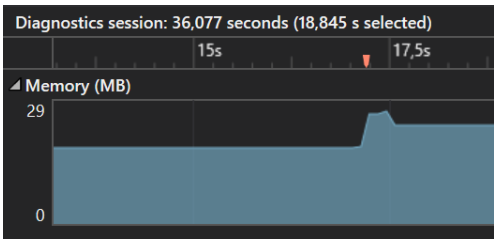
are collected from the Visual Studio Performance Profiler for the memory usage. Figure 5.1 present the graph of the memory usage in the converter program and their peak memory usage and Table 5.1 shows the data for each conversion sample. The converter uses more memory and more time when it converts larger files. Both the GUI and the Assimp part together uses approximately 18 megabytes in idle state before conversion is done for the first time. Afterwards the program tends to expand in memory size for no clear reason even while in idle state. The expanded memory size is still the same after Unity3D was exited. This means that the memory leak or allocation probably has to do with the conversion either from Assimp or the bindings between Assimp and C#.



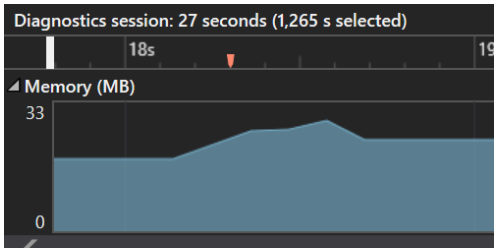
(a) Sample 1.



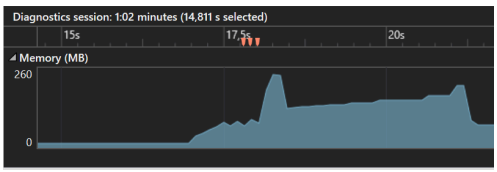
(b) Sample 2.



(c) Sample 3 in 3DS format.



(d) Sample 3 in DAE format.



(e) Sample 4.

Figure 5.1: Converting time and memory usage.

Table 5.1: Conversion time and peak memory difference.

Name	File format	File size (MB)	Peak memory difference (MB)	Time (s)
Sample 1	DXF	2.92	24.1	1.081
Sample 2	OBJ	7.05	127.7	2.608
Sample 3	3DS	0.29	8.4	0.572
Sample 3	DAE	0.87	9.5	0.578
Sample 4	OBJ	20.31	218.0	4.454

5.3 Scaling of the Rendered 3D Models

The scaling of the converted and rendered 3D models in Unity3D does not represent their actual scale or size. We have tested a couple of models, some of them have much larger scale compared to their actual scale or size. The cause for the scaling issues with the models can be from the CAD modelling programs, the modeler, Unity3D, the file format, i.e., `DXF` file format which is dimensionless, or in any combination of those. We created a temporary solution to scale these models down to acceptable scale since the model scaling is not the focus of the project, but it is a concern, as the correct scaling of these models should represent their actual size in reality to get an accurate view of these models in VR. The solution is optional for the user to choose by checking the `Keep scale` check box in the GUI before rendering in VR.

5.4 Mesh and Collision

The meshes in Unity3D has a limit of 65000 vertices due to storing the arrays in a three-dimensional vector called `Vector3`. This means that Unity3D can only store a maximum of 65000 vertices in every mesh, as seen in Figure 5.2. A solution to this would be to either limit the size of the mesh or split the mesh into smaller meshes. The `pb_STL` plug-in has support for mesh splitting which was used to split the meshes. The rendered meshes

turned out to have their mesh faces inverted, which means that the faces/planes of the mesh were facing to the wrong direction. The inverted mesh faces were solved by adding a `ReverseNormals` script to the instantiation function in the `ImportModels` script, which the `ReverseNormals` is attached to all meshes.

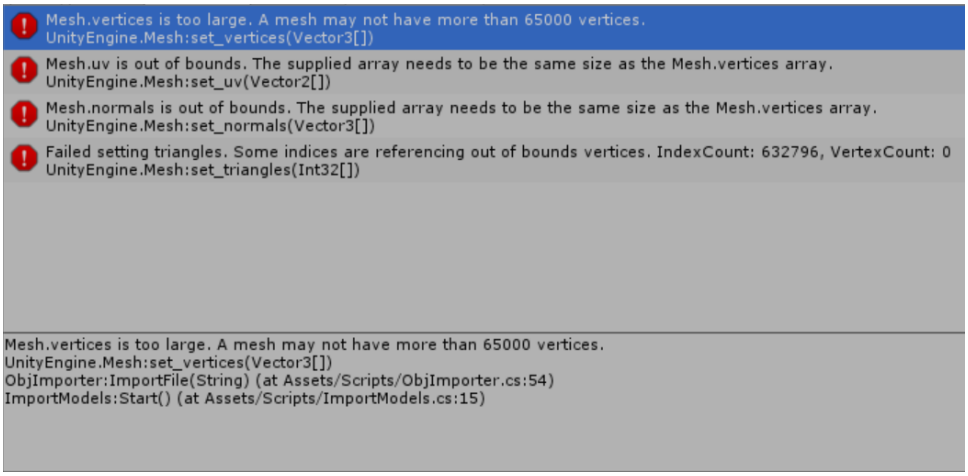


Figure 5.2: Maximum vertices error from Unity3D.

The collider does not account for empty space in between the mesh faces. This makes only the faces of the mesh grabbable and means that the collisions are too complex. There is a way to simplify colliders but the built-in method in Unity3D makes the program run extremely slow when grabbing an object in VR. As a consequence, the method was not used and now the program runs a bit slower, but it does not slow down to a stutter when grabbing an object in VR.

5.5 Converted Format

At the beginning of the project, we planned the converted 3D file format to be in `OBJ` format. This was due to us thinking the format was easy to use in Unity3D. As the project went on, we found that the better solution was to use the `STL` format instead of the `OBJ` format due to already existing library for Unity3D that supported `STL`. The reason for switching from `OBJ` to `STL` was because we could not split the larger meshes into smaller

meshes. The pb_STL plug-in provided the mesh splitting functions, therefore made us change to work with STL.

5.6 Securely Executing Unity3D

The program has a potential concern relating to the execution of the Unity3D program. The execution is done by simply executing the executable Unity3D binary file at a hard-coded location in the file system. We do not check for a valid signature or perform any other validation mechanism before executing. Malware may then be able to execute itself with the same permission as our program if it is able to replace the Unity3D executable on the file system. The security concern should be resolved in future versions.

5.7 Summary

We evaluated the 3D file formats used in this project. The findings from converting the 3D files were that the larger file takes more time to convert and uses more memory. We discussed the scaling of the 3D models, the mesh and the collision when we import the 3D files to Unity3D for rendering and viewing in VR. The results of the project are a prototype with the features to select a file and convert it and then import to Unity3D for rendering.

6 Conclusion

In this project we built a user interface which selects the provided 3D model file and then converts the file to the STL file format. Then the STL file format is imported by Unity3D and render the model for viewing in VR. Additionally, we added an option to view the model outside of VR is implemented if VR is not available.

In the design chapter we designed the layout of the user interface and converter program. We defined what the environment consists of, and the VR functions for the VR mode. In the implementation chapter we described how we implemented the user interface, converter and VR and the optional camera modes. We explained which features we implemented in the user interface and the Unity3D program, and what features were not implemented, or implemented in a different way.

In the evaluation chapter we assessed the design and the work from the implementation chapter. We took a set of sample files and measured the conversion and the memory usage when the file converts. We described the results of the rendered models, such as the scaling, mesh and collision. VR development with Unity3D has been easy and fast to get started with, as there are plug-ins, such as SteamVR with the basic VR functions to use.

6.1 Development Problems

We initially developed the project to be functional cross-platform, but it currently only works on the Windows operating system. The user interface and the converter were implemented in the Linux operating system. Issues surfaced when the implementation part with Unity3D started. The issue is that the SteamVR in Unity3D is not supported on Linux [23]. The rest of the VR development time was spent in the Windows environment since HTC Vive currently only works on recent versions of Windows [24]. To avoid the same problems again for future projects, we would choose to develop in one platform instead of trying to develop to other platforms.

6.2 Future Work

This is the project's potential future work. The future work includes what was not finished and the potential expansions to the project.

6.2.1 Scaling

Scaling caused problems when it comes to importing 3D formats. The imported models are not necessarily specifying their dimensions. Sometimes the dimensions are not specified in the CAD software at all as they are sometimes dimensionless (e.g., DXF). This means that there is no information concerning if the modelers are giving their 3D models the right dimensions or the right scale. When considering VR, the correct scale is important for a immersive experience. This is an assumption which we do not take. An option to view the 3D models without scaling is given in the user interface.

A implementation of a scaling feature with VR controllers is a potential future work. The feature includes the functions to grab the models and stretch it apart or together to decrease or increase the scale. The scaling feature would allow the user to scale the models in VR to get the feeling how to models look in different sizes and at different detail levels.

6.2.2 Features in the User Interface Program

There are features in the user interface program presented in the design chapter which were not implemented. The settings feature to configure various properties for the VR environment, such as the environment in the scene. Additionally, there should be a help button to give instructions on how the program works and which 3D file formats are supported by the program. The file browser should also contain restrictions on what file formats can be chosen.

6.2.3 VR environment

In the current state of the program, we have only one VR environment to be in. More scene environments should be developed in the future to give the user different VR experiences and variations to view the models.

6.2.4 3D File Formats

There are more 3D file formats than the current ones that we would like to implement. To develop the program further, we would want to implement the latest Assimp version, as the library supports more 3D file formats than the current version. To implement the latest Assimp library we have to use other methods, as the library is not written in C# which we used to develop our project. A possibility is to wait for an update of the currently used NuGet package (AssimpNet 3.3.2).

6.3 Concluding Remarks

Throughout the development period we discovered that not all 3D file formats are suitable for Unity3D. The scaling and splitting of the 3D models took a big part of the development time, as it was more substantial than we thought.

References

[1] V. R. Society, *What is Virtual Reality?* [Online]. Available: https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html (visited on 09/20/2017).

[2] D. Reisinger, *Why virtual reality is about to go mainstream.* [Online]. Available: http://fortune.com/2015/10/07/virtual-reality-mainstream/ (visited on 12/20/2017).

[3] H. Corporation, *VIVE VR System.* [Online]. Available: https://www.vive.com/us/product/vive-virtual-reality-system/ (visited on 09/18/2017).

[4] K. Leswing, *Here are the top 5 most popular VR headsets and what they cost*, 2017. [Online]. Available: http://www.businessinsider.com/vr-headsets-comparison-popularity-price-features-details-2017-6?r=US&IR=T&IR=T/#samsung-gear-vr-1 (visited on 12/28/2017).

[5] W. R. Sherman and A. B. Craig, *Understanding Virtual Reality : Interface, Application, and Design.* Ser. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, 2003, ISBN: 9781558603530.

[6] U. Technologies, *Unity*, 2017. [Online]. Available: https://unity3d.com/public-relations (visited on 09/11/2017).

[7] V. Corporation, *SteamVR Plugin*, 2015. [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/32647 (visited on 09/25/2017).

[8] aleksandr, *Documentation, Unity scripting languages and you*, 2014. [Online]. Available: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/ (visited on 09/20/2017).

[9] Autodesk, *Adaptable file format for 3D animation software*, 2017. [Online]. Available: https://www.autodesk.com/products/fbx/overview (visited on 11/05/2017).

[10] W. contributors, *Vertex (geometry)*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Vertex_(geometry)&oldid=807211018 (visited on 12/18/2017).

[11] ——, *STL (file format)*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=STL_(file_format)&oldid=815307726 (visited on 12/18/2017).

[12] ——, *Wavefront .obj file*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Wavefront_.obj_file&oldid=791922420 (visited on 12/18/2017).

[13] ——, *.3ds*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=.3ds&oldid=815637032 (visited on 12/18/2017).

[14] ——, *Collada*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=COLLADA&oldid=802428443 (visited on 12/18/2017).

[15] ——, *FBX*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=FBX&oldid=804976920 (visited on 12/18/2017).

[16] ——, *AutoCAD DXF*, 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=AutoCAD_DXF&oldid=805906606 (visited on 12/18/2017).

[17] A. D. Team, *Open Asset Import Library*, 2015. [Online]. Available: http://assimp.sourceforge.net/ (visited on 10/04/2017).

[18] B. Wagner, L. Latham, P. Onderka, and M. Wenzel, *A Tour of the C# Language*, 2016. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (visited on 09/11/2017).

[19] Microsoft, *Visual Studio IDE*, 2017. [Online]. Available: https://www.visualstudio.com/vs/ (visited on 10/12/2017).

[20] M. Project, *Cross platform IDE for C#, F# and more*, 2017. [Online]. Available: http://www.monodevelop.com (visited on 10/09/2017).

[21] M. Project, *GtkSharp*, 2017. [Online]. Available: http://www.mono-project.com/docs/gui/gtksharp/ (visited on 10/09/2017).

[22] T. G. Team, *What is GTK+, and how can I use it?* 2017. [Online]. Available: https://www.gtk.org/ (visited on 10/04/2017).

[23] Plagman, johnv-valve, anoadragon453, triage-valve, kisak-valve, eric-schleicher, Christoph-Haag, connerbrooks, and JoeLudwig, *SteamVR Release Notes and Known Issues*, 2017. [Online]. Available: https://github.com/ValveSoftware/SteamVR-for-Linux (visited on 12/26/2017).

[24] H. Corporation, *What are the minimum system requirements?* 2017. [Online]. Available: https://www.vive.com/us/support/category_howto/what-are-minimum-system-requirements.html (visited on 01/03/2018).