COMP3710 - Final report

REDESIGN UI FOR ECOVR USING LEAP MOTION

Zena Tamerat Wolba (u5179306)
# Contents

Abstract ................................................................................................................................. 2
Introduction ............................................................................................................................ 2
  Background ........................................................................................................................... 2
  Problem ................................................................................................................................. 3
Technology ............................................................................................................................... 4
Research process .................................................................................................................... 5
  Unity widgets Leap Motion .................................................................................................. 5
    Toggle button widget ......................................................................................................... 6
    Slider widget ..................................................................................................................... 7
    Scroll widget .................................................................................................................... 7
 VR Interface Design Pre-Visualisation Methods ................................................................. 8
 Unreal Engine 4 Leap Motion integration demos .............................................................. 9
    Jenga ................................................................................................................................. 9
    Carillon ............................................................................................................................ 11
    Hollow .............................................................................................................................. 12
Design & innovation ............................................................................................................... 13
  Timeline menu .................................................................................................................... 14
  Challenges .......................................................................................................................... 14
  Sensor data toggle menu ................................................................................................... 17
    Challenges ....................................................................................................................... 17
References ............................................................................................................................... 18
Appendix 1 – Project description .......................................................................................... 19
Appendix 2 ................................................................................................................................ 20
Appendix 3 – READ ME ........................................................................................................ 20
  How to open this project? .................................................................................................... 20
  How to view code? ............................................................................................................... 20
  List of program files submitted ......................................................................................... 21
  Testing ................................................................................................................................. 21
Abstract

Regardless of who you are the area of Virtual Reality (VR) is incredibly fascinating. In the 80/90s several attempts were made by companies such as Nintendo to make VR an everyday reality in our lives, but all of these attempts were hopelessly impractical and failed to capture our imagination. In 2016, things have changed. In fact, the concept of owning a device capable of transporting ourselves to an alternate reality is no longer fanciful thinking. Today we have a large range of consumer VR headsets to choose from anyone of which quickly demonstrate that this technology is a window into the future. While this amazing technological advancement has shocked and excited us, it has also opened a whole new list of issues hitherto unknown to mainstream computer science. Rising to the top of this list is the issue of Interface design. Because of the unique challenges presented by VR, it seems that all we have learned about interface design for 2D displays (PCs, laptops, mobile phones) is no longer relevant, leaving an enormous gap for innovative, bold new ideas. Hand tracking devices (such as the Leap Motion controller) have gone a long way in helping developers create ways of closing this gap. The world of VR is not compatible with traditional input devices such as keyboard and mouse, and instead hand tracking devices have the promise of providing a means of interaction that feels absolutely natural. It is for these same reasons that this project was undertaken: to replace an existing, traditional UI and replace it with a new 3D UI that would use hand tracking to create an immersive and intuitive VR experience. The program in question is called EcoVR, a project which began in 2015 as part of ANU Techlauncher. EcoVR relies heavily on the VR experience, but until now has relied on outdated menus and interactions which are frustrating to use and also spoil the sense of presence that is so critical to VR applications.

Introduction

Background

VR technology has made a significant resurgence in the last 2 years. After a long period of stagnation throughout the 80s and 90s it looked as though the dream of a mass produced Virtual Reality headset that was enjoyable to use, as well as affordable, was too far way. This changed when in 2012 Palmer Luckey, a student of California State University with a passion for electronics, founded his start-up company “Oculus VR”, and promised the world that his company could finally make VR headsets a feasible solution for immersive gaming. The VR headset was called the “Rift” and was made possible through an incredibly successful crowdfunding campaign. The Rift triggered the resurgence of VR headsets back onto the scene and before long companies such as Sony and Microsoft had begun working on devices to rival it. Fast forward to 2016 and Samsung, HTC, Microsoft, Sony have all released (or will soon release) their respective VR headset each managing to find its own appeal in this rapidly growing market.

The return of VR headsets was also a major catalyst for the beginning of the EcoVR project. EcoVR is a project that started from a Techlauncher project led by Dr. Tim Brown in the beginning of 2015. The project is a “proof of concept to explore new methods of visualising complex time-series environmental data, on the landscape.” To realise this goal, a spatially accurate 3D interactive model of the ANU Research site forest, at the Australian National Arboretum, was created using the Unreal 4 gaming engine with native support for the Oculus
Rift VR headset. The virtual landscape was created using real data such as: a Digital Elevation Model (DEM) and point cloud data from 3D LiDAR scans. Time-series environmental data such as: soil moisture and air temperature, gathered by a mesh network of environmental sensors, can also be overlaid and played-back by the user. EcoVR also features a UI that consists of 2D traditional menus and widget that have been heavily customised to work in the 3D environment of the program. The player must use a keyboard/mouse or gaming controller to navigate the interface and interact with the virtual environment. There are significant disadvantages to this approach and they make for a slow, unintuitive and often frustrating experience. This project proposes building a new UI that will make use of hand tracking to make it possible for the user to interact with the program using their hands alone. Hand tracking will be provided by the Leap Motion controller.

The Leap Motion controller is a USB peripheral device developed by Leap Motion, Inc. The Leap Motion controller tracks hands and fingers using infrared cameras and LEDs to track the user’s hands and fingers in 3D space. Its functionality is similar to products like the Microsoft Kinect, but as it only focuses on the hands, rather than the entire body, it enjoys a higher resolution and can therefore offer greater precision tracking. Leap Motion hardware has remain unchanged since 2012 and instead the software responsible for interpreting the images from the IR cameras and translating that into 3D position data has been steadily updated.

**Problem**

The existing User Interface for ECOVR is made up of 2D planes (menus) that exist in the 3D environment and house all the widgets necessary for interacting with the virtual world. Player input is handled by Keyboard/mouse or Xbox controller, allowing the player to control their movement, interact with the UI and toggle which menu is currently being viewed. The UI has been designed with VR headset users in mind, but has significant limitations that disrupt the experience. Therefore the purpose of this project is to redesign the UI from scratch using Leap motion technology, to create a more intuitive and immersive experience for users of VR headsets. The Leap Motion controller will allow the Keyboard/Mouse and Controller to be substituted for the more intuitive hand gesture control. Using specific hand movements the user will be able toggle menus on/off, press button widgets, adjust slider widgets and then smoothly transition back to traditional input mode.
Technology

Unreal Engine is a gaming engine that was first introduced in 1998 by Epic Games. Although it is used predominantly for gaming development, Unreal Engine has also been used successfully for a variety of applications outside of the gaming world. The latest generation is Unreal Engine 4 (UE4), released in 2012, which provides everyone from students to professional developers access to “a complete suite of game development tools made by game developers, for game developers”.

As of the 2nd of March 2015, Unreal Engine 4 along with all future updates was made openly available to anyone, with the only caveat being a 5% royalty paid to UE4 after the first $3,000 of revenue made. Along with this Epic games also introduced the “Unreal dev grants” scheme, that awards cash prizes to promising and creative projects developed in UE4 of between $5,000 and $50,000. These factors played the biggest role in the decision to develop EcoVR in UE4, beginning in early March 2015. Throughout the lifetime of the project the strengths and inherit weaknesses of UE4 became clear, but its large suite of features, continuous support and promise of developer grants made it the right choice for EcoVR.

As this project proposed designing an entire new UI for EcoVR specifically for the Leap Motion controller, it was necessary that UE4 supported this device. This was certainly the case with the release of the “Unofficial Leap Motion plugin” in 2015. The plugin was created by a member of the UE4 online developer community with the alias “Getnamo” as a replacement for the built-in plugin for UE4 (released in 2014) which was incomplete. Getnamo’s plugin was so successful that it is now integrated into the latest release of UE4 (4.11).
Research process

Designing User Interfaces that we can physically interact with in a 3D space (volumetric interfaces) often sparks images in our minds of popular Sci-Fi movies such as *Minority Report* and *Iron Man*. These user interfaces are always packed with information such as moving graphics, and the users are portrayed as being highly adept at using these complex but powerful interfaces. However, films are designed mainly to entertain and in reality good User Interfaces design for VR and human interactions aims to reduce clutter and create an intuitive experience. This of course is a challenging task which presents a variety of unique problems and an equally large variety of interesting possible solutions. Despite this 3D interface design, which is a necessary step forward for VR and hand tracking applications, is still very much in its infancy.

Fortunately, User Interface design has been a large focus for the Leap Motion development community with many implementations introduced for different platforms (i.e. Unity, Unreal Engine 4 etc.). During the early stages of this project I conducted research into existing solutions for managing hand tracking input in VR applications. This included playing through VR demos with the following precise objective:

*Find current implementations of traditional User Interface elements (buttons, sliders etc.) in 3D space for use with Leap Motion, plus innovative interfaces that provide new interactions.*

**Unity widgets Leap Motion**

Unity is a cross-platform game engine developed by Unity Technologies. The Unity gaming engine was initially released in 2005 and is extremely popular for developing 2D and 3D applications by beginner and professional game developers alike. With the introduction of the Oculus Rift Development Kits in 2014 Unity also began support for VR applications. Unity also has played a large part in the blending of Leap Motion input into mainstream application development. The Unity Gaming engine features full Leap Motion integration and also boasts a suite of 3D interface widgets which offer Unity developers the tools to easily create intuitive interfaces. These include the toggle button, slider and scroll widgets. Each of these widgets are made up of 2D planes presented in a 3D space, which compress against one another or move back and forth according to the player’s touch.
**Toggle button widget**

The toggle button, as the name suggests, is a button that has the same functionality as a check button in a traditional interface: once pressed its appearance changes to indicate a change in state. The toggle button is made up of three 2D planes which are (in order of front to back): the button text, the button surface, and the “anchor surface”. When the button is pressed the button surface (and associated text) is compressed into the anchor surface. If the button is pushed all the way into the anchor surface it will change colour and play a brief animation to signal the change of state.

![Figure 3: Unity button widget (Littlefield, A 2014)](image)

The toggle button is designed to be used by one finger, which causes issues sometimes due to inconsistencies with the Leap Motion controller failing to track individual fingers correctly. However, from my experience I’ve learned that when buttons are scaled and spaced correctly players can use less precise hand inputs without the annoyance of unintentionally activating a neighbouring button. The Leap Motion documentation suggests possible uses for the toggle button widget include:

- **menu items** (e.g. toggle subtitles, start game, etc.)
- **toggling doors to open/close**
- **reaction-based games** (e.g. drumming)
- **in-game controls** (e.g. an ignition button or cockpit switch)**
Slider widget

![Image of Unity slider widget](image)

Figure 4: Unity slider widget (Littlefield, A 2014)

The slider widget is a slider bar in 3D space which allows the user to select a value along a predefined scale. This widget was very important as the functionality that it implemented was directly applicable to this project. Using the Unity Slider widget can best be described as highly intuitive and satisfying. As with the Toggle button widget above, the slider widget is composed of three 2D surfaces which form the slider handle and the slider bar itself. Just like the toggle button you need to activate the slider handle by pushing into it, before you can change the value of the slider. The Leap Motion documentation suggests possible uses for the Slider widget include:

- **menu items (e.g. adjust brightness)**
- **in-game controls (e.g. airplane accelerator)**

Scroll widget

![Image of Unity scroll widget](image)

Figure 5: Unity scroll widget (Littlefield, A 2014)

The final widget we will discuss here is the Scroll widget. Unlike the toggle button and slider widgets this is not useful for the purposes of this project however it still provides an interesting new interface interaction for Leap Motion users. The Scroll widget allows users to scroll through content by reaching into the content window and dragging left and right. 2D surfaces are used to construct the Scroll widget and as with the toggle button, the user must press into the scroll pane to activate the widget before interacting with it. From experience, I would say that the scroll widget works well and it is especially useful that it features a mini
scroll bar above to indicate the user’s current position within the content selection. The Leap Motion documentation suggests possible uses for the Scroll widget include:

- user instructions
- chat logs
- scrolling through pictures

**VR Interface Design Pre-Visualisation Methods**

The field of 3D Interface design is at the cutting edge of research into making VR experiences more user-friendly and intuitive. My research into this area led me to an 18-minute video released in October 2015 by motion designer Mike Alger (now working for Google’s VR design department) that delves into the current state of the art in VR interface design using the Leap Motion and Oculus Rift DK2. Alger’s area of focus is:

“research and design of user interfaces for ergonomic multitasking in virtual reality, focused primarily on the interaction design of operating systems and browsers.”(Alger, M 2015)

Alger believes that the future of good interaction design relies on designers who can create UIs that exploit what we know about human nature and the way our brains work. He also outlines some of the significant difficulties that face Interface designers in the future as technology moves towards VR and hand tracking, and away from the traditional Monitor, Keyboard and Mouse. Alger recognises the ability of VR to make using operating systems more efficient. For instance, VR offers 360 degree screen space and will lead to less interruptions due to switching between task windows. When using VR it is also possible to customise your surroundings to suit you (i.e. beach, nature park, martian planet etc.) which leads to happier workers.

![Figure 6: Top-down view of comfort zones for VR users](image)

I’ve learned a lot from this video and I certainly believe it gave me the best introduction possible to designing UIs in a 3D space. It opened my eyes to a whole range of issues I had previously been unaware of. For instance, figure 6 above represents the zones that content
should or should not be placed to ensure ease of use for the user. The main content zone is the most common and ideal place for UI elements. Objects on the far right or left fall into the peripheral zone that is not ideal but still OK. It is uncommon for the user to look into the curiosity zone because it requires significant effort to do so and risks cable snag which is annoying and could potentially damage equipment. Finally, designers should avoid placing content in the no-no zone (or closer to the user’s eyes than 0.5 metres) as this causes significant eye-strain.

Figure 7: Mike Alger’s custom button widgets made in Unity. Demonstrates Alger’s experimentation with colours and movement to create interactions that feel natural (Alger, M 2015)

Alger has little doubt that VR will revolutionise how we interact with operating systems in the near future. He also concedes that to achieve this goal will require a considerable thought, development and time. However, in this video Alger has clearly outlined the basic design principles that we can all aim to follow when developing UIs for VR. This video has been an invaluable resource throughout this project and the most rewarding part of my research into this area.

Unreal Engine 4 Leap Motion integration demos

Jenga

VR Jenga was the first UE4 demo I played which featured Leap Motion support. The application was developed by the author of the Leap Motion plugin for UE4, a leading member of the UE4 online development community with the alias: “Getnamo”. The demo was designed to show off all the latest features that were now possible to implement in UE4 for the Leap Motion. This included fully rigged 3D hands with full hand tracking (including each individual bone), a suite of hand gestures to provide new input event handlers, the ability to use Leap Motion in Unreal Engine 4 while using VR headsets (i.e. Oculus Rift) and much more.
The demo itself was a starting place for me to gain some experience as a user of the Leap Motion controller in UE4 and to be honest my first impressions were not good. The objective of the game was to remove blocks from one stack and place them on-top/inside another. To grab a block you perform a *grab gesture* (make a fist) with your outstretched hand, at which point the next block your hand touches will become attached to your hand. This was not immediately clear however and I initially wasted a lot of time trying to manually pick up blocks or nudge them with my hands. To release a block you open your hand triggering the *grab release gesture*.

There was also another feature called *telekinesis* which was a workaround for a problem that I had never considered prior to playing this demo: how do you pick up objects off the ground? The issue is, that by default the player’s virtual body is in the standing position, but users are usually seated behind a desk making it impossible/impractical to crouch down to pick up an object in the virtual world. For this to work a user would have to play while standing up with their VR headset’s tracking camera placed at a suitable distance from their bodies which is of course impractical in most home or office environments. Telekinesis attempts to solve this issue by allowing users to look at an object they wish to pick up and then perform a *circle gesture*. Unfortunately, I found this feature worked very inconsistently during my own play through and despite repeated testing and attempts to recreate the issue in my own project, I was not able to establish the reason for this.

All in all, I learned a lot in a short time about UE4 support for the Leap Motion at this current point in time. The Jenga demo definitely brought to my attention the difficulties with regards to user interactions for Leap Motion control. There were in fact instructions that could be accessed from within the game which explained the control system. However, even after reading these I could not consistently pick up blocks and place them quickly causing me to lose patience. This was an eye-opening moment for me as I realised the importance of designing for interactions that are intuitive and predictable.
Carillon

Carillon is a game powered by UE4 that allows users to compose music using hand gesture control. It is in a sense a free virtual instrument, assuming you have access to a Leap Motion controller. The game first caught my attention because of its novel UI and the interactions it made possible. To control the virtual music machine, the user must interact with a set of gears floating in a row in front of them. Hand gestures control everything in the game. The user first swipes to open a row of gears, then performs a grab gesture to select a specific gear, rotates it at different frequencies to produce different sounds and finally swipes again to deselect.

![Figure 9: Player selecting a gear in Carillon – selecting and turning the gears affects the sounds produced by the enormous music machine](image)

On the whole the control scheme of Carillon felt intuitive and didn’t take me long to master. This is mostly down to clever design, but because you never actually move from the conductor’s position which eliminates the need for controls dedicated to locomotion and keeps interactions relatively simple. I was very happy with the integration of the Leap Motion in this demo and throughout my play through it felt like a useful tool, rather than an annoyance. Lastly, the inclusion of support for the Oculus Rift is a positive note (excuse the pun) and definitely helps sell this extremely unique and interesting musical experience.
Hollow

Hollow was the final UE4 demo I played and the one I was looking forward to the most. Hollow is an Oculus Rift and Leap Motion compatible experience that allows players to ride a horse through a haunted forest. It is only a brief demo, but is a great example of what can be achieved using the Leap Motion and Unreal Engine 4. It is clear that creating a control scheme that felt natural was high on the developer’s list of priorities. In fact, from the early moments of the game when you are seated on your virtual horse, it does not take long to master the use of the controls allowing you to immerse yourself within the experience.

Your horse has reigns and interacting with these using your hands will cause the horse to speed up, slow down, jump and steer. The beauty of these controls is that they do not require precise hand gestures (pinch, drag etc.) but instead you close your fists (as if grabbing the reins) and then direct both hands up/downwards, left/right to change your direction of travel. Hollow takes a unique approach to navigating its UI. In theory, the user need only point with their finger after which a brightly coloured line is traced from the end of their finger allowing them to choose what they wish to select. However, in practice this method is slightly imprecise and slow as a timer is required to confirm the user’s selections. This is OK for a game such as Hollow which does not require the player to spend a lot of time navigating menus, however for the purposes of my project this approach would not be sufficient.

Figure 10: In Hollow, the Leap Motion controller allows the player to simple grab the reigns when they wish to control their horse
Design & innovation

VR headsets are “headphones for your eyes” in that they replace your sense of sight in the same way that headphones are designed to replace your hearing. Because of this, VR creates an amazing new opportunity to immerse people in the virtual world and create a real sense of presence. Unfortunately, traditional input methods such as keyboard/mouse or gamepads (Xbox controllers etc.) tend to shatter this immersion and quickly become a crutch for the whole experience. Furthermore, 2D UIs such as web browser pages are completely unsuitable due to cluttering of information and failing to use the 3D space around the user efficiently. What’s needed is a large scale shift in how Interface design is done and this process has already begun in the industry, driven by giants such as Facebook (who now own Oculus VR) and Google’s VR design department.

The need for a redesign of the UI for EcoVR stemmed from the difficulties involved with interacting with a traditional UI while using a VR headset. There are significant limitations to using this old fashioned approach to designing interfaces for VR applications. These limitations are never more apparent than when attempting to use an application like EcoVR while wearing the Oculus Rift DK2. The interface feels slow, unintuitive and without an experienced user already with you the learning curve is quite steep. All these factors make for a frustrating user experience and make the program difficult to use for significant periods of time. This is an especially critical issue as the EcoVR project was always intended to be VR compatible, in fact this was touted as one of its major features. Unfortunately, Unreal Engine 4 and its currently very basic support for 3D interfaces has been mostly to blame for these issues.

![Figure 11: A 2D plane menu featuring a title and 3 buttons which requires mouse input – UE4’s VR support is currently hamstrung by its very limited 3D widgets](Project screenshot)

As it is clear from Figure 11 above, Unreal Engine 4 only allows designers to layer traditional UIs (buttons, sliders, checkboxes, text labels etc.) on a 2D plane that can then be placed in the 3D game world. This crude implementation is only effective for applications that do not rely on VR headsets. To compound the issue further, the current UE4 implementation will only accept input from keyboard and mouse, with game controller input requiring your own code to wait for controller input and then manually trigger widgets on the menu (a complicated and haphazard approach). Therefore, to overcome these issues a new type of UI was created-
which featured a brand new and more flexible 3D menu layout and new 3D button and slider widgets.

**Timeline menu**

*Figure 12: The (new) Timeline menu: the menu which allows the user to change the play-back settings for the data. The Timeline menu features buttons for play/pause, a timeline slider, a timeline range slider and the speed slider used for adjusting play-back speed (Project screenshot)*

**Challenges**

**Buttons**

The original timeline menu featured 6 buttons, 4 of which behaved as toggle/check buttons (*Loop, Relative, Average* and the *Play/Pause* buttons) and 2 buttons which behaved normally (*Fast forward to start* and *Fast forward to end*). Initially the plan was to implement a system similar to the Unity toggle button, whereby the buttons were made of a series of 2D layers that would compress into one another with the player’s touch. Unfortunately, this was not possible as it would require constraining the movement of the button along a specified axis, which was not something that could be achieved smoothly in Unreal Engine 4.

Therefore, the approach that was taken was to make each button a 3D object (a rectangular prism) that would float in a fixed location and respond to the user’s touch. When the user’s hands pressed the flat surface of the button it would move slightly and when released, would snap back into position. The limitation of this approach would only be apparent if the button was moved too far off centre, in which case it would attempt to re-centre itself regardless of the player’s hand being in the way, causing it to jitter.
Timeline sliders

To understand why slider bars are so critical to this project, we must first appreciate that the main functionality of the EcoVR program was to perform time lapses on a 3D model of data gathered on forests. This dictated that slider bars be used to represent the time-lapse data and allow the user to navigate through it. For this reason, it was critical to implement sliders bars when designing the Leap Motion/VR compatible interface for this project. This would require designing a slider widget that could be placed in 3D space and interacted with using touch from the user’s hands as tracked by the Leap Motion controller, a feature unheard of in Unreal Engine 4 prior to this project.

![Diagram of distance calculation](image)

**Figure 13:** Distance $x$ is calculated using the **centre** of the slider and the **touch location** (the point where the user’s fingers make contact). This same calculation is used across all sliders. **Note:** the black area behind the slider represents the interactive area (not visible in game) which has been deliberately made larger to make it easier to touch (Project screenshot)

The original timeline menu featured 3 sliders: the **timeline slider**, **time range slider** and the **speed slider**. The **timeline** and **time range** sliders work together to allow the user to efficiently navigate through the time-lapse data. The user can first select the period of time within the **whole** data that they would like to view using the **time range slider**. Then they can make fine adjustments using the **timeline slider** to select a specific point of time within that range. The **speed slider** is included simply to allow the user to adjust the playback speed of the time-lapse.

The **timeline slider** and **speed slider** operate in the exact same way as a traditional slider bar. The challenge here was adapting the existing Unreal Engine 4 slider to accept input from the user’s in-game hands. The solution was found involved representing the slider bar as a 2D plane that was rendered in the 3D space of the game world and presented to the player. When the player’s hands made contact anywhere on the 2D plane containing the slider bar, the point of collision was converted to the corresponding value on the slider bar.
The time range slider is a special case as it is unique to the EcoVR project, with nothing similar in Unreal Engine 4. This was by far and away the most challenging widget to implement in this project. The first solution involved using a 3D object: a rectangular prism to represent the time range bar and allowing the user to grab the sides of the object to stretch or squeeze it to achieve their desired range. However, halfway through the development of a prototype for this idea it was decided that this approach was not only difficult to implement in Unreal Engine 4, but also clunky and imprecise from the user’s point of view. Instead a new solution was found in which a 2D plane was used to represent the time range bar (much like in the design of the timeline and speed sliders described above) and the player would interact using touch input. The only possible issue with the current implementation is that the user must use both hands when interacting with this widget. A touch with the left hand will set the start point of the time range and a touch with the right hand will set the end point.

Hand gestures

So far in this section we have only discussed the issues involved in implementing the functionality of the Timeline menu, when in fact creating a reliable method for actually toggling the Timeline menu was another challenge altogether. The issue was that I was limited with the number of gestures available for triggering events due to the inconsistencies with Leap Motion tracking and still relatively limited support for gesture control in Unreal Engine 4. I had at my disposal several types of gesture input events including: swipe, grab, pinch and circle (similar to the motion of wiping a window). During my testing I found only one of these gestures behaved consistently: the grab gesture. I quickly discovered that the swipe and pinch gestures were easily misinterpreted by the Leap API and required a lot of attention on the user’s part with regards to hand placement/movement in order to not trigger them unintentionally.
Sensor data toggle menu

Figure 15: The (new) Sensor data toggle menu (SDT). The menu that allows the user to choose which sensor data layers they would like to overlay. The SDT menu features a series of buttons that allow the user to toggle each layer (Project screenshot)

Challenges

The implementation of the SDT menu was fairly straightforward as it built upon some of the design from the Timeline menu. A lot of the lessons learned while creating the new Timeline menu were also directly applicable to the SDT menu, such as the placement and scale of buttons. The main challenge here was creating an intuitive and user-friendly method for opening the SDT as well as switching between it and the Timeline menu.

Just like the Timeline menu, hand gestures are used to activate the SDT menu, but due to the inconsistent nature of the circle gesture in the current version of Unreal Engine 4 a compromise was necessary. Therefore the both hand grab gesture, which is already used to toggle the Timeline menu, is used again here to toggle the SDT menu. This works by only allowing the user to cycle through the 2 menus rather than select one or the other individually. The first gesture opens the Timeline menu, the second time switches to the SDT menu and the third and final time will close all menus and return the player to normal movement mode.
References


Appendix 1 – Project description

The task for this project will be to design a novel interface for the ECOVR program, utilising Leap Motion hand tracking technology. The Leap Motion controller, developed by Leap Motion Inc., is a USB peripheral device that uses infrared cameras to track the user’s hands in a 3D space at a distance of about 1 meter. The device is able to track user’s arms, hands and finger movements and therefore provides an excellent opportunity for VR applications. Using this technology, it is possible to create applications where 3D virtual arms follow the user’s precise movements, therefore allowing more natural interaction with virtual objects and a heightened sense of immersion.

Background info: ECOVR is a project born from a Techlauncher project led by Dr. Tim Brown in the beginning of 2015. The project is a “proof of concept to explore new methods of visualising complex time-series environmental data, on the landscape.” To realise this goal, a spatially accurate 3D interactive model of the ANU Research site forest, at the Australian National Arboretum, was created using the Unreal 4 gaming engine with native support for the Oculus Rift VR headset. The virtual landscape was created using real data such as: a Digital Elevation Model (DEM) and point cloud data from 3D LiDAR scans. Time-series environmental data such as: soil moisture and air temperature, gathered by a mesh network of environmental sensors, can also be overlaid and played-back by the user.

The existing User Interface for ECOVR is made up of 2D planes (menus) that are presented to the player in the 3D space and house all the widgets necessary for interacting with the virtual world. Player input is handled by Keyboard/mouse or Xbox controller, allowing the player to control their movement, interact with the UI and toggle which menu is currently being viewed. The UI was designed with VR headset users in mind, but has significant limitations that disrupt the experience. This project proposes using Leap motion technology to create a more intuitive and immersive experience for users of VR headsets. The Leap Motion controller will allow the Keyboard/Mouse and Controller to be substituted for the more intuitive hand gesture control. Using specific hand movements the user will be able toggle menus on/off, press button widgets, adjust slider widgets and then smoothly transition back to traditional input mode.

The areas of the UI that will be redesigned will include (and are not limited too):

- The Sensor Data Toggling menu. The menu that allows the user to choose which sensor data layers they would like to overlay. Currently, this menu features a series of radio buttons that allow the user toggle each layer.
- The Timeline menu. The menu that allows the user to change the play-back settings for the data. Features buttons for play/pause, a timeline slider and a timeline range slider.
Appendix 2

<table>
<thead>
<tr>
<th>Assessed project components</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Report:</strong> Documentation, Research and Reflection report in the form of an online wiki.</td>
<td>45</td>
<td></td>
<td>(Please see contract)</td>
</tr>
<tr>
<td><strong>Artefact:</strong> Interactive UI for Leap Motion controller, to be used with ‘ECOVR’ program (features ‘Timeline menu’ and ‘Sensor Data Toggling menu’).</td>
<td>45</td>
<td></td>
<td>Dr. Tim Brown</td>
</tr>
<tr>
<td><strong>Presentation:</strong> Demo new UI, discuss design principles, reflect on what was learned about Software Engineering process.</td>
<td>10</td>
<td></td>
<td>(Please see contract)</td>
</tr>
</tbody>
</table>

Appendix 3 – READ ME

**How to open this project?**

To open the project in the Unreal Engine 4 editor simply double-click the file called "LeapMotionUI_EcoVR.uproject". Once here you can run the project by clicking on the play symbol at the top of the screen (make sure you have the Oculus Rift and Leap Motion plugged in before opening the editor).

**How to view code?**

Much of the work done with User Interfaces in Unreal Engine 4 is done using blueprints:

"Blueprints Visual Scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. This system is extremely flexible and powerful as it provides the ability for designers to use virtually the full range of concepts and tools generally only available to programmers." (Unreal Engine Documentation - Blueprints Visual Scripting 2016)

To view the blueprints for this project please follow these steps:

1) Open the project.
2) In the "Content browser" window at the bottom of the screen, click the "Contents" tab.
3) Now double click on the folder titled "LeapMotion".
4) All the files you see here contain my own work. To view the blueprint graphs double-click on any of the assets here (except SliderHandleImage.uasset and LPgamemode.uasset) and when a new window opens use the tabs near the top of the screen to view the blueprint graphs.
List of program files submitted

The following files are my own work:

1) LeapMotionUI_EcoVR\Content\LeapMotion\Custom_LM_Hands_Character.uasset
2) LeapMotionUI_EcoVR\Content\LeapMotion\LM_Gamemode.uasset
3) LeapMotionUI_EcoVR\Content\LeapMotion\LM_MenuActor.uasset
4) LeapMotionUI_EcoVR\Content\LeapMotion\LM_RangeSlider.uasset
5) LeapMotionUI_EcoVR\Content\LeapMotion\LM_Slider.uasset
6) LeapMotionUI_EcoVR\Content\LeapMotion\SliderHandleImage.uasset

File number 3) (LM_MenuActor.uasset) is the class that contains all the functions and logic that make the UI work. This class inherits from the UE4 Blueprint "Actor" which means it can be placed in the game world. This class is responsible for displaying the 3D menus to the user and also handling input from the Leap Motion. This class was designed to be completely separate from the back-end implementation and also to integrate easily with other projects (highly portable). For instance, if you have the latest version of UE4 (4.11 and up) you can simply drag this class into your project and begin using it without making changes to code in other classes.

Testing

Due to the nature of this project, testing required a more hands on approach. Several hours of testing have been put into this project to improve the quality of the UI and interactions. This includes testing by myself and by my supervisor Tim Brown. Feedback has been critical and several improvements have been made throughout the semester. It is worth noting that the LeapMotion has caused issues such as jerky tracking and brief moments of freezing, although there hasn't been enough time to explore these problems further.