EmbarkVR: Outdoor Virtual Reality Experience

CS Senior Capstone

Final Report

Jake Jeffreys, McKenna Jones, Spike Madden, Sean Marty

May 31, 2017

Abstract

1 ORIGINAL REQUIREMENTS DOCUMENT

EmbarkVR: Outdoor Virtual Reality Experience
CS Senior Capstone
Software Requirements Specification

Jake Jeffreys, McKenna Jones, Spike Madden, Sean Marty

November 4, 2016

Abstract

Many outdoor activities these days initially require a large mental and economic investment to get started. This makes people less likely to try new outdoor activities. The goal of the project is to develop an interactive product demonstration with virtual reality to combat this issue. This project has the potential to inspire people to get outdoors and try new things by first getting them comfortable in new environments or performing new movements. It strives to make outdoor activities accessible to everyone no matter their experience level. This project will not only inspire but also improve the retail experience by making it more immersive, interactive, and informative while being entertaining. The main tool being used is Unity Gaming Engine in tandem with an HTC Vive Virtual Reality System. The objective of this project is to create a functional VR outdoor experience ready to be piloted in a Columbia retail store by May of 2017.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The main goal of the project is to make customers feel more inclined to purchase Columbia gear through the use of an immersive, outdoor Virtual Reality experience. This document exists both for development of the project and to provide a detailed description of the technical requirements for the clients.

1.2 Scope

We want to create an outdoor virtual reality experience for customers at a Columbia retail store. The application will consist mainly of visual, audio, and tactile experiences to create an outdoor world in which the user can navigate. The main activity available will involve fly fishing in one of the rivers within the environment. Users will also have the ability to interact with Columbia products while in the experience and gain specific product information.

1.3 Definitions

Virtual Reality (VR)	Artificial environment that is created with software
HTC Vive	A virtual reality headset produced by HTC
Unity Game Engine	The Unity Game Engine, developed by Unity Technologies is used in this project to develop the virtual reality simulation.
Avatar	An icon or figure representing a particular person.
Wands	Controllers that are used with the HTC headset.
Base Stations	These allow the Vive to track the movement and location of the wands and headset.

1.4 References

[1] S. Michalak and E. Lind, "Virtual reality heuristics, results from user testing for prioritization and development," Sep 2016.

[2] ——, "Virtual reality heuristics, results from user testing round 2," Oct 2016.

[3] H. Corporation, "Vive ready computers," 2016.

[4] C. Hall, "Sony to devs: If you drop below 60 fps in vr we will not certify your game," 2016. [Online]. Available: http://www.polygon.com

1.5 Overview

The next chapter of the document will give an overview of the basic functionality of the Virtual Reality application. It contains informal requirements to provide background for section three, Specific Requirements. Section three will provide more detailed requirements and is intended for a more technical audience such as developers.

2 Overall Description

2.1 Product Perspective

This VR product may be new to Columbia Sportswear but will still have ties to existing products. Within the experience, users will have the ability to view and interact with Columbia gear. Users will also be given the opportunity to wear Columbia gear while participating to learn how the clothes feel while executing certain movements.

The product will rely heavily on Virtual Reality Hardware. Specifically, the HTC Vive System. This system consists of the the headset, two wands, and two base stations. Additionaly a Virtual Reality compatible computer is needed to actually run the software. In terms of software the product will rely on the Unity Game Engine. Unity will do the heavy lifting when it comes to rendering the virtual environment and making it look as realistic as possible.

2.2 Product Functions

The VR setup will allow the user to simulate outdoor experiences. Specifically, the user will be able to virtually see Columbia Sportswear gear in the environments they are intended to be used in. This will be done using 3D renderings of Columbia items which will then be placed within the environment. We will then be using Unity to design VR interaction capabilities on top of the renderings. The final product will allow the user to test a variety of clothes and equipment. This product will also give customers the ability to save the gear they liked in the VR, and access that information after the experience is over.

2.3 User Characteristics

The general type of user of this project will be a customer at a Columbia Sportswear retail store. Under this umbrella lies a few different types of customers. First there are customers who are inexperienced in the outdoor activity they are buying gear for. This target audience will benefit most from the VR expreience as it will allow them to experience the activity without a lot of economic or time commitment. Secondly, there are customers who are experienced in the activity they are buying gear for. This audience will benefit from the VR experience becauseit will allow them to view themselves actually using and testing out new gear.

2.4 Constraints

A virtual reality headset like the HTC Vive has some inherent restrictions. The first one is space. The HTC Vive tracks how much space you have set the system up in and creates virtual barriers. This limitation can be alleviated by using the controllers to move the users within VR environment. Besides physical space, space in the virtual display is also a concern. Information needs to be supplied to the user without obstructing the VR experience. A second limitation are the graphics within the VR environemnt are not entirely realistic. According to studies done by Intel and Thug Design [1][2], the categories most important to the feeling of immersion are realistic interactions, responsiveness, graphic clarity, and smooth transitions in that order. Because we are trying to promote Columbia Sportswear Gear in a realistic environment it needs to be as authentic as possible.

2.5 Assumptions and Dependencies

An important assumption made in this requirements document is that the virtual reality experience will be run on a computer system that can run the HTC Vive software. The following are the minimum specifications to run Vive, as found on the HTC Vive website [3]:

- Processor: Intel Core i5-4590 or AMD FX 8350, or better
- Graphics: NVIDIA GeForce GTX 1060 or AMD Radeon RX480, or better
- Memory: 4GB RAM or better
- Operating system: Windows 7 SP1 or better

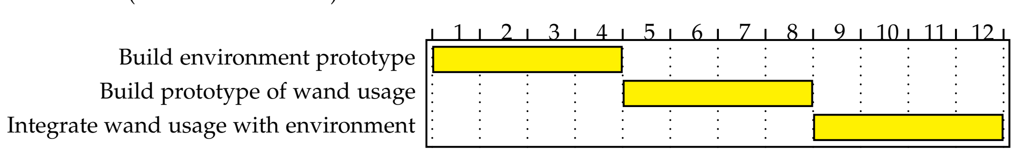
Design decisions and optimizations will be made so that a computer with the above specifications can run the experience with little noticable lag, but if the machine drops below the minimum capabilites, the requirements regarding responsiveness will have to change.

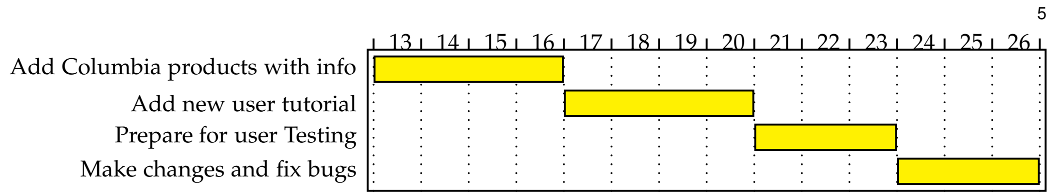
Also, the requirements often depend on the availability of a set of Columbia and Unity 3D assets. If either of those sources of assets is not available, the requirements about being able to see Columbia Sportswear gear in a realistic environment will have to change.

2.6 Apportioning of Requirements

One part of the project that will likely be delayed until later versions is the social aspect. Ideally the user would be able to share their VR experience on social media sites like Facebook or Youtube. This could be in the form or 360 degree images or videos. At the moment this requirement is not a high priority.

Gant Chart (measured in weeks)





3 Specific Requirements

3.1 External Interfaces

- 360 degree view of outdoor scenario within VR experience using HTC Vive headset. This will contain optional user guidance (visual) and offer Columbia product information (visual).
  - Input: Movement of headset
  - Output: Visual data
- Immersive noises from outdoor VR experience. This includes audio from the optional user guidance.
  - Output: Audio through speakers and/or headphones.
- Ability for other users not using headset to see user's current view.
  - Output: Visual data on external monitor.
- Controller available to be held by user to interact within VR experience.
  - Input: HTC Wand movement

3.2 Functions

- Ability for users to interact with fly fishing equipment.
- Ability to see Columbia fishing apparel in use.

3.3 Performance Requirements

- Must maintain at least 60fps throughout experience.[4]

3.4 Software System Attributes

Our unity environment should be portable and work on all HTC Vive systems. Correctness can be evaluated by how authentic the real-world experience we're trying to replicate is. We should be able to adjust the scenes easily and accommodate for any changes that the client wants.

3.4.1 Reliability

The system will be considered reliable if it can provide the virtual reality experience consistently without failure. Failure can be defined as any technical issue that breaks immersion. This could include noticeable lag between the user and avatar's actions, distortion of the environment, or a software defect that causes the system to crash.

3.4.2 Availability

The system should be available to customers whenever the virtual reality station in the store is set up. There should be no downtime when the customer puts on the headset and the customer should almost immediately be placed in the immersive environment.

Many of the attributes of our system (such as availability and reliability) will be handled through the Unity game engine.

3.4.3 Security

The environment will be designed to not require a system with internet access, and will not need any user information. The expected implementation will be on a closed system in a Columbia store, where physical and remote access will be controlled by Columbia employees.

3.4.4 Maintainability

As time goes on it should be easy to adapt the product to easily include more Columbia Sportswear products and virtual reality environments to accompany them. Changes to the environment through the addition of assets (new Columbia gear) should not be difficult and an updated scene should be accomplished with a small patch.

3.4.5 Portability

The environment should work on all HTC Vive systems with the minimum requirements to be Vive Ready [3].

4 Signatures

Document: Software Requirements Specification (EmbarkVR)

Intel Sponsor	Date
Columbia Sponsor	Date
Student Team Members	Date

2 How Have Requirements Changed?

3 Original Design Document

EmbarkVR: Outdoor Virtual Reality Experience
CS Senior Capstone
Software Requirements Specification

Jake Jeffreys, McKenna Jones, Spike Madden, Sean Marty

November 4, 2016

Abstract

Many outdoor activities these days initially require a large mental and economic investment to get started. This makes people less likely to try new outdoor activities. The goal of the project is to develop an interactive product demonstration with virtual reality to combat this issue. This project has the potential to inspire people to get outdoors and try new things by first getting them comfortable in new environments or performing new movements. It strives to make outdoor activities accessible to everyone no matter their experience level. This project will not only inspire but also improve the retail experience by making it more immersive, interactive, and informative while being entertaining. The main tool being used is Unity Gaming Engine in tandem with an HTC Vive Virtual Reality System. The objective of this project is to create a functional VR outdoor experience ready to be piloted in a Columbia retail store by May of 2017.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The main goal of the project is to make customers feel more inclined to purchase Columbia gear through the use of an immersive, outdoor Virtual Reality experience. This document exists both for development of the project and to provide a detailed description of the technical requirements for the clients.

1.2 Scope

We want to create an outdoor virtual reality experience for customers at a Columbia retail store. The application will consist mainly of visual, audio, and tactile experiences to create an outdoor world in which the user can navigate. The main activity available will involve fly fishing in one of the rivers within the environment. Users will also have the ability to interact with Columbia products while in the experience and gain specific product information.

1.3 Definitions

Virtual Reality (VR)	Artificial environment that is created with software
HTC Vive	A virtual reality headset produced by HTC
Unity Game Engine	The Unity Game Engine, developed by Unity Technologies is used in this project to develop the virtual reality simulation.
Avatar	An icon or figure representing a particular person.
Wands	Controllers that are used with the HTC headset.
Base Stations	These allow the Vive to track the movement and location of the wands and headset.

1.4 References

[1] S. Michalak and E. Lind, "Virtual reality heuristics, results from user testing for prioritization and development," Sep 2016.

1.5 Overview

The next chapter of the document will give an overview of the basic functionality of the Virtual Reality application. It contains informal requirements to provide background for section three, Specific Requirements. Section three will provide more detailed requirements and is intended for a more technical audience such as developers.

2 OVERALL DESCRIPTION

2.1 Product Perspective

This VR product may be new to Columbia Sportswear but will still have ties to existing products. Within the experience, users will have the ability to view and interact with Columbia gear. Users will also be given the opportunity to wear Columbia gear while participating to learn how the clothes feel while executing certain movements.

The product will rely heavily on Virtual Reality Hardware. Specifically, the HTC Vive System. This system consists of the the headset, two wands, and two base stations. Additionaly a Virtual Reality compatible computer is needed to actually run the software. In terms of software the product will rely on the Unity Game Engine. Unity will do the heavy lifting when it comes to rendering the virtual environment and making it look as realistic as possible.

2.2 Product Functions

The VR setup will allow the user to simulate outdoor experiences. Specifically, the user will be able to virtually see Columbia Sportswear gear in the environments they are intended to be used in. This will be done using 3D renderings of Columbia items which will then be placed within the environment. We will then be using Unity to design VR interaction capabilities on top of the renderings. The final product will allow the user to test a variety of clothes and equipment. This product will also give customers the ability to save the gear they liked in the VR, and access that information after the experience is over.

2.3 User Characteristics

The general type of user of this project will be a customer at a Columbia Sportswear retail store. Under this umbrella lies a few different types of customers. First there are customers who are inexperienced in the outdoor activity they are buying gear for. This target audience will benefit most from the VR expreience as it will allow them to experience the activity without a lot of economic or time commitment. Secondly, there are customers who are experienced in the activity they are buying gear for. This audience will benefit from the VR experience becauseit will allow them to view themselves actually using and testing out new gear.

2.4 Constraints

A virtual reality headset like the HTC Vive has some inherent restrictions. The first one is space. The HTC Vive tracks how much space you have set the system up in and creates virtual barriers. This limitation can be alleviated by using the controllers to move the users within VR environment. Besides physical space, space in the virtual display is also a concern. Information needs to be supplied to the user without obstructing the VR experience. A second limitation are the graphics within the VR environemnt are not entirely realistic. According to studies done by Intel and Thug Design [1][**?**], the categories most important to the feeling of immersion are realistic interactions, responsiveness, graphic clarity, and smooth transitions in that order. Because we are trying to promote Columbia Sportswear Gear in a realistic environment it needs to be as authentic as possible.

2.5 Assumptions and Dependencies

An important assumption made in this requirements document is that the virtual reality experience will be run on a computer system that can run the HTC Vive software. The following are the minimum specifications to run Vive, as found on the HTC Vive website [**?**]:

- Processor: Intel Core i5-4590 or AMD FX 8350, or better
- Graphics: NVIDIA GeForce GTX 1060 or AMD Radeon RX480, or better
- Memory: 4GB RAM or better
- Operating system: Windows 7 SP1 or better

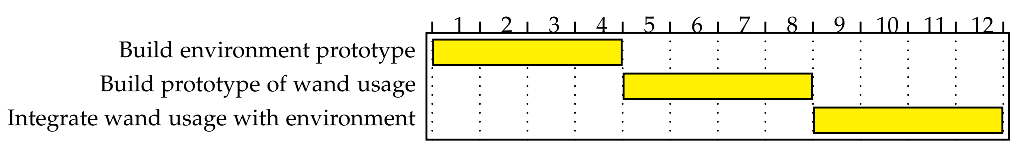
Design decisions and optimizations will be made so that a computer with the above specifications can run the experience with little noticable lag, but if the machine drops below the minimum capabilites, the requirements regarding responsiveness will have to change.

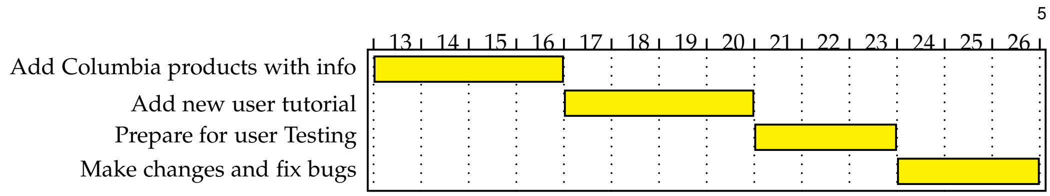
Also, the requirements often depend on the availability of a set of Columbia and Unity 3D assets. If either of those sources of assets is not available, the requirements about being able to see Columbia Sportswear gear in a realistic environment will have to change.

2.6 Apportioning of Requirements

One part of the project that will likely be delayed until later versions is the social aspect. Ideally the user would be able to share their VR experience on social media sites like Facebook or Youtube. This could be in the form or 360 degree images or videos. At the moment this requirement is not a high priority.

Gant Chart (measured in weeks)





3 Specific Requirements

3.1 External Interfaces

- 360 degree view of outdoor scenario within VR experience using HTC Vive headset. This will contain optional user guidance (visual) and offer Columbia product information (visual).
  - Input: Movement of headset
  - Output: Visual data
- Immersive noises from outdoor VR experience. This includes audio from the optional user guidance.
  - Output: Audio through speakers and/or headphones.
- Ability for other users not using headset to see user's current view.
  - Output: Visual data on external monitor.
- Controller available to be held by user to interact within VR experience.
  - Input: HTC Wand movement

3.2 Functions

- Ability for users to interact with fly fishing equipment.
- Ability to see Columbia fishing apparel in use.

3.3 Performance Requirements

- Must maintain at least 60fps throughout experience.[**?**]

3.4 Software System Attributes

Our unity environment should be portable and work on all HTC Vive systems. Correctness can be evaluated by how authentic the real-world experience we're trying to replicate is. We should be able to adjust the scenes easily and accommodate for any changes that the client wants.

3.4.1 Reliability

The system will be considered reliable if it can provide the virtual reality experience consistently without failure. Failure can be defined as any technical issue that breaks immersion. This could include noticeable lag between the user and avatar's actions, distortion of the environment, or a software defect that causes the system to crash.

3.4.2 Availability

The system should be available to customers whenever the virtual reality station in the store is set up. There should be no downtime when the customer puts on the headset and the customer should almost immediately be placed in the immersive environment.

Many of the attributes of our system (such as availability and reliability) will be handled through the Unity game engine.

3.4.3 Security

The environment will be designed to not require a system with internet access, and will not need any user information. The expected implementation will be on a closed system in a Columbia store, where physical and remote access will be controlled by Columbia employees.

3.4.4 Maintainability

As time goes on it should be easy to adapt the product to easily include more Columbia Sportswear products and virtual reality environments to accompany them. Changes to the environment through the addition of assets (new Columbia gear) should not be difficult and an updated scene should be accomplished with a small patch.

3.4.5 Portability

The environment should work on all HTC Vive systems with the minimum requirements to be Vive Ready [**?**].

4 SIGNATURES

Document: Software Requirements Specification (EmbarkVR)

Intel Sponsor	Date
Columbia Sponsor	Date
Student Team Members	Date

4 ORIGINAL TECH REVIEW

EmbarkVR: Outdoor Virtual Reality Experience
CS Senior Capstone
Technology Review and Implementation Plan

Jake Jeffreys, McKenna Jones, Spike Madden, Sean Marty

November 14th, 2016

Abstract

Virtual reality is just starting to enter all kinds of markets. Everyone is trying to figure out the best way to implement their VR applications. Each VR project is going to be looking for different things when it comes to hardware and development environments. For our project we have been given an HTC Vive which handles the hardware but we still need to choose appropriate software for use in development. There are a variety of options and we'll be looking for one that is inexpensive, easy to learn, powerful enough to develop a realistic VR application, and easily configurable with SteamVR. In this document we break down the different aspects of our project and compare possible technology against specific requirements related to each piece.

CONTENTS

1 Introduction

The development experience in Virtual Reality is still in its infancy. However, gaming engines have been around for a while and fortunately some of them support SteamVR integration. There are currently multiple powerful and free VR gaming engines on the market to choose from. For our project we need to develop a realistic application that users are able to navigate and interact with a variety of objects and natural elements. When making a decision, the first thing we will be looking at is whether or not the engine is powerful enough to create a VR application. Next will be its integration with SteamVR and how easy it is to learn. We only have six months for our project so we need to make sure we can pick up the tools quickly. The three main aspects of our project are constructing the environment, integrating Columbia products, and create a fishing experience.

Authorship of Sections

- 2.1.1, 2.1.2, and 2.1.3: Sean Marty
- 2.1.4, 2.2.3, and 2.2.4: Jake Jeffreys
- 2.2.1, 2.2.2, and 2.3.4: McKenna Jones
- 2.3.1, 2.3.2, and 2.3.3: Spike Madden

2 Project Sections

2.1 Environment

2.1.1 Create Static Environment

To begin any virtual reality development, one must first create a basic static environment to work in. This generally involves a couple major steps, and every legitimate game development platform can do these basic things. That means that in order to research which platform is best, we need to look not just at whether basic environment creation is possible, but how easy and smooth it is. There are many game engines out there, and many facets involved in creating a static environment, but we had to narrow down our research to just three engines and a handful of the most important features. In the simplest sense, there are three steps to creating a static environment in most game engines. First, one needs to be able to explore an empty landscape through a virtual reality headset. In our case, we already have been given an HTC Vive so that will be our headset of choice. Second, one needs to place static objects in the empty landscape. Third, it is important to look at how easy and intuitive it is to work with the camera because that is very important to any virtual reality experience. The three game engines we will look at are the Unity game engine, the Unreal Engine, and Amazon Web Services' Lumberyard game engine.

To create a static environment in Lumberyard, one starts with creating a new level. These levels are the basis for game development in Lumberyard. [1] On creation, a developer is asked to fill out information about heightmaps, terrain texture, and color multipliers. Once a level has been created, the next step is to populate the environment with static objects. Lumberyard breaks objects into a couple categories. These can include, among other things, brushes and entities. Brushes are objects that the user cannot interact with, while entities have the ability to be interacted with dynamically. [2] Lumberyard handles camera creation, views, and movement like the developer is shooting a cinematic scene (which sometimes they are). A camera is an object itself, and can be selected and moved about in the Track View editor. To move a camera, simply unlock it and use the mouse and keyboard to move it as it records. [2]

If one instead wants to go about creating an environment in the Unity game engine, simply create a new project and a basic empty scene will be generated. Adding objects to the scene can be done through the Object menu, where there is a list of 3D objects to choose from. [3] In order to interact with a basic camera object in Unity, one simply makes a parent object from the basic object class and moves that object, bringing the camera with it. This way, all the same movement and animation processes that apply to normal objects apply to cameras as well. [3]

Finally, if one wants to create a static environment in the Unreal Engine, they create a project and, like Lumberyard, add a new level. There is a specific empty level for virtual reality development, which adds certain basic settings and capabilities automatically. [4] To add static objects to the recently created level, one should add actors. Actors are any objects that can be placed into a level. Specifically, geometry brush actors are the simplest way to add geometry to a level. [4] Cameras are just a type of actor that can be added to a scene in Unreal Engine. Once added, cameras have their own set of attributes and methods that allow interaction.

All three game engines have certain appealing qualities for our project. However, the scene that we are creating is relatively simple in the sense that there will not be significant character movement and most objects won't need to be interacted with. Also, we will most likely only need one scene, so the ability to easily work with multiple levels does not have much appeal. With those constraints in mind, the Unity game engine would be best for simple static object creation because of its simplicity and the universal handling of 3D objects.

2.1.2 Animate Environment

Animating objects and characters in an environment is a key aspect of virtual reality development, just as with any game development. Unresponsive objects or unrealistic animations can negatively effect how real a virtual reality experience feels. Again looking at different game engines, we wanted to figure out which engines are the most intuitive and powerful. We chose two areas to focus on in regards to animation, so that we could narrow our comparisons. The first is linear animation, which is preset, and runs like a movie. User interaction does not effect linear animation. We will use some linear animation for the parts of our scene that the user cannot affect, such as swaying trees or distant water movement. Second, we want to look at interactive animation, which is a set of animations that are performed based on user input. This is a huge section for virtual reality development, because the basis for a real experience is whether the world feels interactive.

For each engine listed in the table below, we will compare how the engine handles both kinds of animation.

Engine Name	Linear Animation	Interactive Animation
Unity	Animation Curves	Animation Events
Unreal Engine	AnimMontages	EventGraphs
Lumberyard	Track View and LimbIK Technical	AnimationEvent and XML file

When looking to animate objects in Lumberyard, there is the clearest split between linear and interactive animation. For linear animation, developers use the Track View to work through animations frame by frame. When animating people, Lumberyard developers can utilize Limb IK to set an endpoint for a hand or foot and have the engine calculate limb movement that achieves that end goal. [5] For interactive animation, Lumberyard has two categories to handle dynamic animated responses. Avatar controls deal with the character that a user is going to control. This means that movement commands come mostly from outside the game code. AI controls deal with non-user controlled characters. Every movement and decision made by these controls is housed in game code. Both of these are part of the bigger picture for interactive animation in Lumberyard that attempts to implement automatic motion synthesis. [5]

Animation in Unity is based on a collection of clips that can be combined in complex ways, controlled using animation curves, and triggered by animation events. For linear animation, Unity provides a flowchart-style window to manage animation clips and their interaction with each other. [6] Once a set of animations has been combined, Unity developers can use animation curves to connect other game content and parameters with an object in motion. [7] When an object needs to respond to user interaction, animation events come into play. These are simple functions with triggers that set up a specific animated response for an input action. [6]

The Unreal Engine handles animation with a combination of skeletal and vertex morphs, which are controlled through a visual graph of animation events. Linear and interactive animation events are handled in somewhat the same way, by adding interactions between events to the EventGraph. Developers can manage events themselves, their effects on the overall animation blueprint, and interaction between events all in one place. [8]

For our project, the animation needs in the general environment are broken into these two categories equally. We will have linear animation used to make the environment around the user come to life. Also, we will utilize interactive animation both to make close objects react to the user's presence, as well as handle item selection and manipulation by the user. After learning about all three game engines and how they handle animation, it seems that the Unity game engine fits our needs the best. The other two engines are focused heavily on complex human animation and interaction, and we do not need all that for our game. The Unity engine breaks every animation down into manageable and reusable clips that we can populate our entire environment with.

2.1.3 Add Audio

Another key aspect of creating an immersion virtual reality experience is adding audio effects. A static soundtrack works well for 2D video, but for a virtual reality experience the audio has to be much more complex and dynamic. As Mona Lalwani mentions in her article F̈or VR to be truly immersive, it needs convincing sound to match,̈ the biggest keys to realistic 3D sound with the technology we currently have are sound cues and three-dimensional sound. [9] Sound cues are audio events that react to specific triggers. For example, if I move my foot through water that should elicit some sort of sound response. 3D audio can be created in a couple different ways, including sound attention and occlusion. Basically, this just means changing the audio in an experience based on user location and the other objects in the area. Finally, another key to developing audio in a game engine is how sound data is managed. Most engines have various ways to manage their sound data, each with benefits and drawbacks.

We will compare the usual three game engines: Unity, Unreal Engine, and Lumberyard.

Engine Name	Sound Cues	3D Sound	Managing Sound Data
Unity	AudioSource Effects	AudioSource Matrices	Audio Spacializer SDK
Unreal Engine	Visual Sound Cue Nodes	Attenuation Shapes	Audio Node Graph
Lumberyard	Audio Play/Stop Triggers	Raycasting	Audiokinetic Wwise LTX

In Lumberyard, 3D audio is handled by using raycasting through the Audiokinetic Wwise LTX audio system. Attenuation and occlusion are calculated by tracking a sound vector. This means that the path from a sound source to the user is calculated, and objects and distances in between affect the final sound. In order to implement responsive sounds, Lumberyard has Audio PlayTrigger and StopTrigger methods. This allows sounds with all of their properties to be triggered dynamically. [10]

Audio in the Unreal Engine is handled like many other things in the engine, with a node-based visual map. This map allows audio clips to play in any order, and interact with each other in complex ways. Each individual sound is a node on the chart that represents an audio clip bound to a Sound Cue Node. These sounds can then be made realistic by adding attenuation through either a simple distance algorithm or attenuation shapes. Attenuation shapes help create consistent, realistic soundscapes in an environment by calculating attenuation based on a geometric shape that might fit a certain location better than a simple distance calculation. [11]

Unity audio is a system of audio clips that are sent from an Audio Source to and Audio Listener. Both are attached to objects, the latter usually to the main camera object. Audio Sources generate sounds when audio clips are triggered through AudioSource Effects, allowing for dynamic sound cues. To make those sounds more realistic, Unity uses the Audio Spacializer SDK to handle attenuation and other audio effects like echoing and the Doppler Effect. A system of matrices handles how the listener and source handle complex audio, although the spacializer performs good portions of the calculations. [12]

In the end, our choice in game engine for audio comes down to choosing a simple system and choosing a system that will most help bring realism to the soundscape of our environment. This is a hard balance, but it seems like the Unreal Engine finds the best middle ground. In the Unreal Engine there is no added outside software to handle complex sound manipulation. Also, the attenuation shapes feature of the Unreal Engine might be extremely useful to relatively quickly create a real sounding landscape.

2.1.4 Improve Realism and Immersion

The goal of our project is create a realistic, virtual, outdoor experience that makes users feel like they have been transported to a different location. After weve created the environment, animated objects, and added audio, the next step is to improve the visual realism.

The first thing that came to our mind was to increase the detail of the objects in the environment but this may actually make the experience less realistic from the perspective of the user. It turns out that detail is only one aspect of creating a visually successful game. The other two are frames per second and resolution. These three concepts make up what is called the graphical fidelity triangle[13]. According to a study done by Intel and Thug[14], immersion needs graphical fidelity, not realism. They found that what was important was that graphics were crisp and clean at all times. They found that a smooth experience, one without glitches and lost frames, was the most important aspect of immersion. They even went a step further to argue that photo-realism is often times worse because it makes inaccuracies more obvious.

In order to find out which of these tools will be the most effective we will be looking at simple particle systems, texture libraries, and clean texture mapping. Its important that which game engine we use that the asset store contains a wide variety of simple textures and materials so that we are able to create realistic environment objects. There are also going to be a lot of moving parts such as animals and river water. These need to be realistic enough to create immersion but not too detailed as to create graphical lag. The three tools I will be looking at are Unity, Unreal Engine, and Lumberyard.

Engine Name	Particle Systems	Texture Libraries	Clean Texture Mapping
Unity	Yes	Yes	Yes
Unreal Engine	Yes	Yes	Yes
Lumberyard	Yes	Minimal	Yes

Unity is an incredibly common tool for building in virtual reality, especially for the HTC Vive. It has managed to find a good balance of realism throughout the asset library[15]. This asset store offers a wide range of particle effects, textures, and materials. Within the Unity development environment, they have made it easy to map texture on to objects to create a truly crisp experience. They also offer a variety of lighting options to add even more outdoor realism. Unreal Engine also offers a wide range of particle animations and texture packages[4]. Unreal engine looks like it would be a much more effective tool for creating high end virtual reality games but for our usage it may be overkill. Overall it has a lot of the same capabilities if not more but they also come at a price. Most of the asset store costs money which is not what we are looking for. Lumberyard has a good variety of particle effects but struggles when it comes to textures and texture mapping[1]. Developer freedom within the environment is easy to learn but limited.

The best tool to create an immerse, realistic environment would be Unity. It offers a wide variety of free assets that can be used throughout projects to add that extra bit of realism. The Unity community also strives to create simple, crisp designs that dont have unnecessary details. They have recognized that these details can actual hinder the immersive experience instead of help it.

2.2 Fishing Activity

2.2.1 Process User Wand Movement

The first step to creating an immersive fishing experience will be collecting the users's movement from the virtual reality controllers. Since this is an essential component of the project, we will be examining how three different gaming engines go about doing this. For this specific project we will be using the HTC Vive so we are concerned with capturing the input from the two HTC Vive Wands. This is a core component of our system that other parts of the product will rely on so it should be simple and reliable at the same time.

We will consider three different gaming engines to accomplish this task, Unity, Unreal Engine, and Amazon's Lumberyard gaming engine. Two different aspects of each engine will be compared in order to reach a final verdict on which is the most useful for the task. First there is the consideration of which language, and tools are used to complete the task. Second, since this is a core function of our system we will consider if the gaming engine has native virtual reality support or not.

Engine Name	Language	Native VR Support
Unity	C#	Yes
Unreal Engine	Blueprint Visual	Yes
Lumberyard	Lua	No

First we will consider the Unity engine. Scripting in Unity is primarily done in C#. Most of the team working on this project has experience in C#, so that is an easy benefit of using Unity. Even for members of the team who do not have experience, C# is a fairly easy language to pick up. In order to handle controller input in Unity one can make use of built in SteamVR calls. For example there are SteamVR callls to access the controller itself, as well as various buttons on the controller [16]. This abstracts the process of retrieving controller input greatly, which is a good thing in this case. The generally accepted approach to capture the controller input in Unity is to create a controller class that makes use of these calls. Unity is officially supported for development for the Vive. Therefore much documentation exists on the subject of capturing controller movement.

The next engine to consider for this task is the Unreal Engine. Unlike Unity, Unreal does not make use of a traditional programming language. Instead, as a developer you use Blueprint Visual. Similar to C#, it is used to define object oriented classes. The only difference is that it is done visually instead of with code. The process for working with Motion Controllers in Unreal is much different than Unity. The developer picks from a list of Motion Controller-specific inputs in the Pallete panel of the Blueprint Editor[17]. From here you can simply drag and drop to attach a certain controller action to an action within the game. This simplifies the process greatly, but also has the drawback of not being as customizable as if we were writing this in code. The Unreal Engine has support for SteamVR, and therefore the Vive. That being said, the documentation available for VR related topics is not plentiful as with Unity.

The final option for this task is Amazon's Lumberyard Engine. What makes Lumberyard unique is that you can choose to use the Flow Graph System for visual scripting on the Lua language for code based scripting. As far as capturing controller input, it is possible with both. Using the Flow Graph one can create a VR:ControllerTracking node which provides up to date info regarding the controller's current position and status [18]. Alternatively, one can use Lua to access the TrackingState struct which contains the linear velocity, acceleration, and all other tracking info of each controller [19]. Lumberyard has this benefit of being flexible in the implementation. However, Lumberyard's VR support is currently still in beta and the documentation is lacking.

After considering the above options, Unity stands out as the best option to capture the controller movement. First of all, the language, C#, that Unity uses for scripting is ideal. Most of the team members are familiar with the language so there will be no time wasted learning a new language. Also, compared to Blueprint Visual and the Flow Graph of Unreal and Lumberyard, respectively, scripting in C#, offers more flexibility for capturing controller movement. Unity has the best support for the HTC Vive. This is important for the project specifically because as it will be in a retail setting, it needs to be as reliable as possible. Finally, Unity is the best option in terms of documentation which is important for development.

2.2.2 Import Fishing Assets

As none of the members of this team are artists, the project will rely heavily on downloadable assets. These assets will be used to create nearly all aspects of the virtual reality environment. Therefore the assets used play an essential role in creating the most realistic environment possible. A wide variety of assets will be used to create the environment. There is not a defined list of different medias that will be used but a few likely ones are still images (textures), animations, and audio.

As in other sections, the ability to handle assets in three different gaming engines, Unity, Unreal, and Amazon's Lumberyard engine will be compared. Three main categories will be considered in detail. Firstly, the access to a native asset store. This is preferable to searching for assets elsewhere. Secondly, the types of files that are supported for import. Since we are not certain of which assets we would like to use, the engine which has the widest support for file types is preferred here. Finally, we will examine each engine's method of organizing and importing assets. This project will make use of many assets so an organized system is crucial.

Engine Name	Asset Store	File Support	Asset Organization and Import
Unity	Yes	Images, 3D models, Animations, Audio	Robust
Unreal Engine	Yes(new)	Images, 3D models, Animations, Audio	Supported
Lumberyard	No	FBX files	Minimal

First we will consider Unity. One of Unity's selling points is its Asset Store. The Unity Asset store launched in 2010, making the oldest and most mature of the three gaming engines in discussion [15]. The asset store currently has over 15000 free and paid 3D assets to choose from. While browsing the store it becomes apparent that there has been an asset created for just about everything that the mind can imagine. Importing an asset into Unity from outside of the asset store is as simple as drag and dropping the asset into the Unity project window. Assets are organized in Unity in the Assets folder of the Project windows. Within the Assets folder assets are organized into subgroups of materials, textures, etc. This makes finding assets a painless process. Unity supports all major types of Images, 3D models, Animations, and Audio files.

Second we will consider the Unreal Engine. Like Unity, the Unreal engine also has an asset store, called the Unreal Engine 4 Marketplace. The Marketplace, however, is much younger than the Asset Store of Unity. The Marketplace opened to developers during 2014 [20]. This means that number of available assets is significantly smaller than the Unity Asset Store. As far as importing assets, the process in Unreal is straightforward, and guided by a GUI. Organizing assets in Unreal is a bit more manual than Unity. One must manual manage the folders where assets are stored, and there is not an assets folder by default. Unreal supports all major types of Images, 3D models, animations, and Audio files. The preferred file format for importing assets FBX files, which is slightly limiting, as not all assets can be found in this format.

Finally we have Amazon's Lumberyard Engine. Unlike Unity and Unreal, Lumberyard does not have a place to download assets from within the engine. This means that you must look to other sources for assets, such as Unity's Asset Store, Unreal's Marketplace, or other websites. Like other parts of Lumberyard, the Asset Importer is still in preview release [**?**]. Currently it only supports FBX by default. If you would like to import other files you need to need to manually implement a new importer that will generate a SceneGraph for that particular file type [**?**]. Importing files requires navigating the installation location of Lumberyard and manually copying and pasting the files to the correct location.

After considering the three engines, Unity seems like the clear choice for importing and managing the assets needed to create our Virtual Reality environment. While both Unity and Unreal have places to download assets, Unity's Asset Store has the most assets by a long shot. Unity also has the benefit of having simple workflow for importing assets compared to Lumberyard. Unity has native support for importing many different file types, which is beneficial as it is not clear which assets we will be using yet. Finally, the process of importing and managing assets is organized and stable, which cannot be said about Lumberyard.

2.2.3 Fishing Rod Interaction and Mechanics

Within the outdoor virtual reality experience there will be the opportunity for users to go fly fishing in the virtual river. In order to achieve this capability, we will need to allow the user to first interact with a fishing rod. The mechanics of this process can get quite complicated and therefore it is important for us to decide on the correct tool to build this functionality. Not only will the user need to be able to pick up the fishing rod but they will also need the ability to cast and real the line back in. These are the basic functionality of the fishing rod and will need to be as realistic as possible to create the illusion they are actually participating in the activity. In this document I will be comparing the virtual object mechanics within different, free gaming engines: Unity3D, Unreal Engine, and Lumberyard.

In order to find out which of these tools will be the most effective we will be looking at the ease of scripting mechanics and programming haptic feedback. Fly fishing is all about the smooth motion of the rod. In order to create a similar experience in virtual reality there needs to be some kind of haptic feedback (controller vibration). Fortunately, my team has been given an HTC Vive setup which comes with two wireless controllers. These controllers offer HD haptic feedback with 24 sensors to ensure accurate movement tracking[21]. This then brings up the question of software.

The first engine to discuss in Unity which is one of the most common tools for beginners developing virtual reality applications. Unity tools related to the physics engine and haptic feedback are incredibly

Engine Name	Language	Physics Engine	Haptic Scripting	Documentation
Unity	C#	Yes	Yes	Yes
Unreal Engine	Blueprint Visual	Yes	Yes	Yes
Lumberyard	Lua	Minimal	Minimal	Yes

easy to use for developers familiar with C#. There are extensive built-in libraries and a very intuitive structure[3]. Documentation and support is also strong for Unity programming which explains why Unity is a great choice for people new to physics engines and virtual object mechanics. The next tool to discuss in Unreal Engine which uses Blueprint Visual. This is an incredibly powerful engine and therefore offers an extensive physics engine[4]. The Blueprint Visual interface for developers makes development easy and visually clear. This could make the process of implementing physics simple and straightforward. Haptic feedback is implemented in a similar fashion with check boxes and drop down menus instead of having to write a single line of code. This would be great for those unfamiliar with programming fundamentals. The last engine is Lumberyard which uses the Lua scripting language. The physics engine and haptic feedback programming are still in their infancy but do offer some capabilities[2]. For a virtual reality project with a lot of moving pieces, the simplicity of lumberyard may be a hindrance and not offer enough freedom to developers.

For our project, the best tool to use is going to be easy to learn yet still have extensive physics functionality to give us enough freedom while developing. Based purely on the ease of use and capabilities of the physics engine and haptic feedback control, the best tool would be Unity. The main reason for this is that Unity is easier to get started on and given the timeframe of our project, it is important we are able to create a fundamental application as quickly as possible. The physics engine is clean and should give us enough freedom to create a realistic fly fishing experience.

2.2.4 Integrate Usage with Environment

Creating a realistic fishing rod that users are able to pick up and move around is one thing but to integrate these movements with the environment is incredibly important to creating a realistic experience. According to user studies done by Intel and Thug[14], realistic interaction is the most important heuristic when it comes to overall enjoyment and feeling of immersion with correlation coefficients of .49 and .57 respectively (1.0 is a perfect correlation). If these are the two standards we are looking at, in order to have a successful application we will need to allow users to easily interact with all aspects of the fishing environment. While fly fishing, people stand either in the water or on the bank. These locations have specific characteristics such as certain insects, fish, plants, water movement, and sounds. The user will then need to be able to interact with these objects as well as the other way around. In this document I will be comparing the virtual object mechanics within different, free gaming engines: Unity3D, Unreal Engine, and Lumberyard.

In order to find out which of these tools will be the most effective we will be looking at the ease of importing animals, of triggering sounds, and of animating these objects. The animation is the most important as it needs to not only give the illusion of realism but also react to user movements. For example, if the user steps into the water then not only will sounds need to occur but also certain fish animations may need to get triggered such as swimming away from the user. Sounds have an incredible power of creating immersion and therefore need to be comprehensive yet subtle. Subtlety is a big part of immersion as users should never feel overwhelmed by noises or animations.

Engine Name	Animal Assets	Animation Assets	Sound Assets	Object Assets
Unity	Yes	Yes	Yes	Yes
Unreal Engine	Yes(Paid)	Yes(Paid)	Yes(Paid)	Yes(Paid)
Lumberyard	No	Minimal	Minimal	Minimal

One of the biggest areas to look at here is the availability of free assets. The scope of this project is quite small so it is important to join a game engine community that supports this. Upon looking at Unity we found that there is a wide range of support for creating interactive games[3]. Objects are easily importable and interaction is easily programmable. Objects interactive well and are able to demonstrate accurate collision mechanics. There is also a lot of flexibility when it comes to detected user location and movements. Unreal engine is equally as powerful[8] but it doesnt provide nearly as many free assets. Realistic interactions require a lot of subtle objects which wont be possible in our time frame if we have to create everything from scratch. Lumberyard is an incredibly simple piece of software and offers very little when it comes generating realistic user interactions[2].

The amount of support behind Unity makes this tool much more effective for our needs. The community is made up of more enthusiasts and hobbyists which generates more free, quality assets. The flexibility and simplicity when programming user interaction will also give us a lot of freedom and the ability to create rapid activity prototypes.

2.3 Columbia Products

2.3.1 Create Avatars

Animated characters, or avatars, are going to be essential in building an immersive and realistic virtual reality environment for the Columbia/Intel Outdoor Gear project. Avatars are going to be especially useful in this project as they gave us a means of displaying various Columbia gear to the VR user. The user themselves will be presented as an avatar wearing Columbia gear, and they’ll also be able to see other avatars wearing Columbia gear. This variation of perspective will help the consumer get a better feel and understanding of the various gear choices available to them.

Three common game engines that are compatible with the HTC Vive are Valve's Unity game engine, Epic Games' Unreal Engine 4 and Amazon's Lumberyard. This section will examine each game engine's capabilities when it comes to avatars and character assets. All three game engines have support for character models and are all very similar in their implementation; they require a rigged and skinned model from a third party digital content creation package for character setup. The exported model is a collection of polygons in a 3D package. The process of rigging defines the skeleton and joint hierarchy of the model and is what determines the range of possible movements. Skinning is the process of connecting skin or mesh to the previously defined joint hierarchy[22]. Examples of these DCC packages include Autodesk 3ds Max or Maya, Poser, Makehuman, and Mixamo.

Engine Name	Asset Store Quality and Variety	Import assets from DCC
Unity	Robust	Yes
Unreal Engine 4	Moderate	Yes
Lumberyard	Does not exist	Yes

Unity's animation system, known as Mecanim, handles all humanoid creation and animation. In addition to being able to import models created by third party DCC packages, Unity also allows for downloading of character content from the Unity Asset Store. The Unity Asset Store has a wide variety of both free and purchasable character models. At the time of writing this document, the Asset Store has 3885 3D character models available for use. These include animals, humanoids and robots. Unity also supplies a service to create an animated humanoid character from scratch through the process of modelling, rigging and skinning within the client. The best case scenario for this project would be for Columbia to already have these complete models for their products, and to have them be easily imported into the Unity environment.

Unreal Engine 4 also supports the importing of art assets and animations from third party DCC packages[23]. It also has a Marketplace where developers can download various character assets for use in their environments. The selection is limited compared to Unity's Asset Store, and a lot of the models are paid; there are very few if any free ones.

Lumberyard also supports the importing of art assets and skeletal meshes from third party DCC packages[24]. There are several asset packages on their website but no designated asset store of any kind.

In conclusion, our team will likely be using Valve's Unity Game Engine. All three of the game engines allow for importing of assets from third party DCC packages, but the quality and quantity of available 3D avatars in Unity's Asset Store is unrivaled.

2.3.2 Import Columbia Gear

Arguably, the central focus of this outdoor virtual reality experience is to promote Columbia gear and lead the customer to a sale of a product. With that in mind, it's crucial to have Columbia gear presented as realistically and prominently as possible within the virtual environment. Columbia gear includes various apparel items such as coats, jackets, boots and other products from their fishing line.

Once again, the three game engines that will be compared are Unity, Unreal Engine 4 and Lumberyard. Each of these platforms will be evaluated on their asset stores, supported import file formats and ease of use. While it is unlikely that any of these platforms' asset stores have accurate and high quality models of Columbia gear, a strong selection of similar assets could be helpful in testing and development. This leads to the second point of supported file formats. The most favorable situation, and most likely, would be for Columbia to provide us with assets and animations of their products. In this situation, we would need the game engine to import external files; the game engine with the most supported file formats would be preferable. Lastly, since these assets are such a big part of the virtual reality experience, it's vital for us as the developers to be able to interact with these assets as easily as possible.

Engine Name	Asset Store Quality and Variety	File Support	User Friendly
Unity	Robust	3D models and animations	Yes
Unreal Engine 4	Moderate	3D models and animations	Yes
Lumberyard	Does not exist	FBX	No

Unity's Asset Store has over 15000 free and purchasable 3D assets available to users. The simple 5 star rating system of the store allows developers, at a quick glance, to find reputable asset packages that will any project's' needs. If nothing in the store stands out, which may be the case with Columbia gear, Unity supports all major types of 3D models and animations. Models and animations can be created in third party digital content creation packages and imported into a Unity project. Unity also allows for the creation and animation of assets, within the client, if no suitable assets can be found anywhere. Asset access within the project is simple, and is organized clearly in an Assets folder in the Project window. Specific components within the Assets folder can also be browsed.

Unreal Engine 4 also has an asset store, named the Marketplace, but is not as diverse as Unity's Asset Store. The Marketplace is much newer[20] as Unity's Asset Store came out 4 years earlier, and as a result, has not built up the collection of quality and quantity that Unity offers. It also seems like a lot of the asset packages are paid; there are very few, quality, free asset packages. But once again, it is highly unlikely that any of the premade assets in the Marketplace will suffice for this project that wants to present Columbia gear as accurately as possible. Unreal Engine 4 offers support for all major file types as well, and the organization of assets within the menus is reasonable.

Amazon's Lumberyard is still in beta and is lacking a lot of features as expected. It does not have a native asset store and only supports FBX files by default. The Asset Importer requires the user to manually find the path of the project and paste the required files[25].

Since Columbia gear is a key part of this project, we want to be able to work with a gaming engine that allows us to effortlessly work with these assets. Due to Unity's large selection with the Asset Store, ability to support a majority of popular file formats and ease of use within the client, Unity seems like the best choice over Unreal Engine 4 and Lumberyard.

2.3.3 Animate Avatars

Proper animation of the avatars and environmental components in the virtual reality space is going to be key in creating a believable, immersive fly fishing experience. We need a game engine that is capable of animating the avatars, Columbia gear, and outdoor environment. As per our requirements document, the experience must maintain at least 60 frames per second.

Engine Name	Animation Assets from Store	Import Animations from DCC packages
Unity	Robust quantity and free	Yes
Unreal Engine 4	Moderate quantity and paid	Yes
Lumberyard	Does not exist	Limited

Unity's animation system, Mecanim, includes services that allow for retargetable animations, full control of animation weights at runtime, event calling from within the animation playback, sophisticated state machine hierarchies and transitions, and blending of shapes for facial animations[6]. In particular, retargetable animations for humanoid animations is going to extremely useful in this project. With retargetable animations, it is possible to apply the same set of animations on various character models using the humanoid models' bone structure. The animation workflow in its entirety is discussed in the next section.

Mecanim uses Animation Clips which store information on how objects should adjust their position and other properties over time. These clips can come from third party digital content creation packages, motion capture studios or can be created within Unity. Animation Clips can be considered the building blocks for all animation sequences in Unity. These building blocks are organized into Mecanim's Animator Controller to chain various animations together. The Controller keeps track of the Animation Clips in a flowchart system and behaves like a state machine that keeps track of current Animation Clips and when they should change. The Controller is also capable of blending multiple clips to provide smooth transitions between animations. The Animation Clips and Animator Controller are aggregated into a GameObject through the Animator Component[6].

Unreal Engine 4 also has a complete animation system and is similar in structure to that of Unity's. Unreal Engine 4's animation editing toolset, Persona, allows for editing of Skeletons and their meshes, Animation Sequences and Animation Blueprints. Animation Sequences are similar to Unity's Animation Clips and is defined as a single animation asset that is associated with a Skeletal Mesh. Blueprints in Unreal Engine 4 are a gameplay scripting system. Object oriented classes are defined as nodes and developers can define interactions between the nodes[8]. The Animation Blueprints specifies the movement of bones and also allows for blending of animation. Unreal Engine 4 also has a Controller that moves or alters an avatar's properties due to a trigger.

Lumberyard's documentation on animation is limited, which is expected as it's still in beta. Lumberyard allows for the creation of both linear animation and interactive animations. Linear animations are predefined animations that do not interact any outside influences. Interactive animations are influenced by the environment and are responsive[5].

Both Unity and Unreal Engine 4 have animation systems that are capable of handling the task of creating realistic animations that will help to build an immersive virtual reality experience. Both Unity and Unreal Engine 4 have very similar technologies that implement the animation workflow. Since Lumberyard is still in beta, Unity and Unreal Engine 4 seem like suitable choices for this project when it comes to animation.

2.3.4 Allow User Interaction with Products

In order for the user to consume information related to the Columbia Sportswear gear it will need to be displayed in some fashion within the VR environment. This will likely be done with a mixture of menus and dialogs. Different gaming engines go about doing this in different ways. The following section will compare support for different menu and dialog systems in Unity, Unreal, and Amazon's Lumberyard Engine.

There are a couple things to consider when creating menus and dialogs in VR. In traditional games, user interfaces are generally non-digetic, meaning that it does not actually exist in the world. Think of a traditional heads-up display. Unfortunately this approach does not work well in Virtual Reality. It has the possibility of causing nausea and straining the user's eyes. Two other alternatives are diegetic UI and spatial UI. Diegetic UI elements are attached to something within the environment. Spacial UI elements are positioned in the environment. Another thing to consider is how the user will interact with the UI elements. In our case, using the Vive, it is possible for the user to interact with the UI either with the controllers or the headset. Therefore, we will consider which different types of UI and user interaction each gaming engine supports.

Engine Name	Supported types of UI	Supported types of user interaction
Unity	Diegetic, Spatial	Headset and Controller interaction
Unreal Engine	Widgets	Controller and Headset interaction
Lumberyard	No native VR user interfaces	No support

The first gaming engine to consider is Unity. Unity has very good support for VR related user interfaces. The main tool for creating VR user interfaces in Unity is the World Space UI. This UI system allows the developer to create UI elements that are either spatial or diegetic. The World Space UI allows you to create a virtual canvas where they can then place GUI interfaces [26]. As far as user interaction with user interfaces, Unity has virtual reality support for this built in and well documented. There are three main classes that can be used for interaction: VREyeRaycaster, VRInput, and VRInteractiveItem [27]. VREyeRaycatster casts a virtual ray in the environment and determines if it hits any colliders. This could be used for either the headset or the controllers. VRInput is a class that retrieves the button presses from the controllers of the VR system. Lastly, you can use VRInteractiveItem with any GameObject that you would like to be able to interact with in the VR environment.

The second gaming engine to consider is Unreal. Unreal does not have an default way to create VR user interfaces. However, the most commonly accepted way is to use Unreal's Motion Graphics UI Designer. This tool can be used to create 3D widgets. While these widgets are not necessarily designed to be used as menus or dialogs, it is possible to do so, but it is a bit of a workaround. It is possible to interact with these widgets with both the headset and the controllers, however, it is no easy task. Unreal does that have any classes or libraries to make this process easier. The developer will need to implement the user interaction from the ground up, using the Blueprint Editor.

The last engine to consider for this task is Lumberyard. As Lumberyard is still in beta, it is lacking when it comes to both user interfaces and user interaction with said interfaces. Lumberyard contains a UI editor which can be used to create most 2D interfaces. Interfaces that resemble heads up displays can easily be created in Lumberyard, but as discussed above this is not useful for VR. The other problem is that it has no support for 3D interfaces, and no native support for VR specific user interfaces. Because of this, Lumberyard also has no support for interaction with user interfaces within the VR environment.

After considering these three engines for the task of interacting with information regarding Columbia Sportswear gear found in the environment, Unity seems like a good pick for the issue. First of all in Unity, it is easy to create both digetic and spatial interfaces using the World Space UI. On the other hand, neither Unreal or Lumberyard contain native support for VR specific user interfaces. Secondly, Unity has a set of classes which can be used to simplify the process of creating an interactive user interface. This process is complex in Unreal, and non existent in Lumberyard. Finally, Unity supports interaction with both the headset and the controllers. These factors make it a clear choice for displaying the information related to the Columbia Sportswear gear.

3 Conclusion

After considering the three different gaming engines discussed in this document, the best pick is clearly Unity, for a number of reasons. The fact that we are working with the HTC Vive VR system greatly influences this decision. Unity is the accepted game engine for SteamVR development and therefore the Vive aswell. Unity is also the engine that has built in support for various VR related functions, whereas in Unreal and Lumberyard, VR support is minimal and cumbersome. Finally, there is the added benefit of Unity using C# as its scripting language, a language that our team is experienced in. Overall, Unity will allow us to hit the ground running and develop our project as efficiently as possible.

References

[1] A. W. Services, “Lumberyard levels and environment,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/userguide/level-intro.html

[2] ——, “Lumberyard object and entity system,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/userguide/entities-intro.html

[3] U. Technologies, “Unity getting started with vr development,” 2016. [Online]. Available: https://unity3d.com/learn/tutorials/topics/virtual-reality/getting-started-vr-development

[4] E. Games, “Unreal engine editor manual,” 2016. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Editor/index.html

[5] A. W. Services, “Lumberyard animation,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/developerguide/animation-intro.html

[6] U. Technologies, “Unity animation,” 2016. [Online]. Available: https://docs.unity3d.com/Manual/AnimationSection.html

[7] ——, “Unity animation tutorial,” 2016. [Online]. Available: https://unity3d.com/learn/tutorials/topics/animation

[8] E. Games, “Unreal engine skeletal mesh animation,” 2016. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Animation/index.html

[9] M. Lalwani, “For vr to be truly immersive, it needs convincing sound to match,” January 2016. [Online]. Available: https://www.engadget.com/2016/01/22/vr-needs-3d-audio/

[10] A. W. Services, “Lumberyard audio system,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/userguide/audio-intro.html

[11] E. Games, “Unreal engine audio and sound,” 2016. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Audio/index.html

[12] U. Technologies, “Unity audio,” 2016. [Online]. Available: https://docs.unity3d.com/Manual/Audio.html

[13] T. Wilde, “Pc graphics options explained,” Oct 2014. [Online]. Available: http://www.pcgamer.com/pc-graphics-options-explained/

[14] S. Michalak and E. Lind, “Virtual reality heuristics, results from user testing for prioritization and development,” Sep 2016.

[15] W. Freeman, “Unity launches unity asset store,” November 2010. [Online]. Available: http://www.develop-online.net/news/unity-launches-unity-asset-store/0108154

[16] R. Reddy, “Steamvr: Handling vive controller input in unity,” 2016. [Online]. Available: http://www.leadingones.com/articles/vive-dev-unity-4.html

[17] I. Epic Games, “Motion controller component setup.” [Online]. Available: https://docs.unrealengine.com/latest/INT/Platforms/VR/MotionController/

[18] A. W. Services, “Setting up a basic virtual reality flow graph,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/userguide/virtual-reality-flowgraph-setup.html

[19] ——, “Vr lua functions,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/developerguide/lua-scripting-ref-vr.html

[20] D. Water, “Marketplace: Now open for business!” 2014. [Online]. Available: https://www.unrealengine.com/blog/marketplace-now-open-for-business

[21] H. Corporation, “Buy vive hardware.” [Online]. Available: http://www.vive.com/us/product/

[22] U. Technologies, “Using humanoid characters,” 2016. [Online]. Available: https://docs.unity3d.com/Manual/UsingHumanoidChars.html

[23] E. Games, “Setting up a characters,” 2016. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Animation/CharacterSetupOverview/

[24] A. W. Services, “Setting up a characters,” 2016. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Animation/CharacterSetupOverview/

[25] ——, “Asset importer (preview) technical overview,” 2016. [Online]. Available: http://docs.aws.amazon.com/lumberyard/latest/developerguide/asset-importer-intro.html

[26] U. Technologies, “Creating a world space ui,” 2016. [Online]. Available: https://docs.unity3d.com/Manual/HOWTO-UIWorldSpace.html

[27] ——, “Interaction in vr,” 2016. [Online]. Available: https://unity3d.com/learn/tutorials/topics/virtual-reality/interaction-vr

5 Weekly Blog Posts

6 Final Expo Poster

7 Project Documentation

7.1 Technical Overview

7.2 How to Install

7.3 Running the Experience

7.4 Hardware Requirements

8 How We Learned

9 What We Learned