EmbarkVR: Outdoor Virtual Reality Experience

Software Requirements Specification

Jake Jeffreys, McKenna Jones, Spike Madden, Sean Marty

November 8, 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 to 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 development 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 amking 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 prject so we need to make sure we can pick up the tools quickly. The three main aspects of our project are construct the environment, integrating Columbia products, and create a fishing experience.

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 immersive 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 ẗFor 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 theyve 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 this 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. According to a study done by Intel and Thug[13], 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. 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. 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. 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 compement 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 concerend 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 [14]. 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[15]. From here you can simply drag and drop to attatch 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 pletiful 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 [16]. Alternatively, one can use Lua to access the TrackingState struct which contains the linear velocity, acceleration, and all other tracking info of each controller [17]. 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 [18]. 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 [**?**]. This means that number of available assets is significatly 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 prefered 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, Lumbeyard 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 suports 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 instalation 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 [ONE]. This then brings up the question of software.

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

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

easy to use for developers familiar with C#. There are extensive built-in libraries and a very intuitive structure. 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. 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. 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[ONE], 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. This 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. 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 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.

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

2.3.2 Import Columbia Gear

2.3.3 Animate Clothing on Avatars

2.3.4 Allow User Interaction with Products

3 Conclusion

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

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

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

[16] 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

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

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