PaintSpace:

Exploring the Concept of Painting on a 3D Canvas Using Virtual Reality and 3D Input

Benjamin Madany

Advisor: Robert Signorile

Boston College

Abstract

3D technology has seen a wide range of innovations, from 3D graphics and modeling to 3D printing. Among the most recent of these innovations are immersive virtual reality and 3D input. These have allowed for the creation of unique, 3D experiences, and they also present the opportunity for a wide variety of applications whose purposes range from entertainment to educational or medical use. One possibility is an extension of 3D modeling that utilizes these recent technologies to present a 3D canvas to an artist. Applications such as Google's Tilt Brush explore this concept of drawing in 3D space. As the ability to draw in such space is novel, development of such a tool presents several challenges. This thesis explores the process of building a 3D painting application. I first present the key challenges encountered during development. Then, I detail various solutions and options related to these challenges. Next, I examine the capabilities and state of my application, and finally, I compare it to other available applications.

1. Development Challenges

In designing PaintSpace, there were several key challenges that needed to be addressed. These included the following major issues: the combined input method and handling, how to convert from the received input to meaningful artifacts, and what sort of user interface and additional controls are required for this application. For each major challenge, there were various possibilities to consider. Additionally, some choices caused entirely different questions to arise.

The question of how to handle input was among the most major problems faced in building the PaintSpace application. The way in which input is received and processed determines the range of capabilities the application can deliver. In addition, the input method for an application is influenced by the application's nature, and must be intuitive. In the case of PaintSpace, it makes sense for an application for drawing in 3D space to have input that maps to 3D space with ease. Along these lines, it was of major importance to determine an input device that would fit the requirements. In this case, it was determined that a physical device would be used, and so it was important to find the correct hardware. Additionally, it was necessary to ensure that the application would be able to receive and interpret the device's data with ease.

The next issue was that of how to "draw" or otherwise create a meaningful 3D representation of the received input. This challenge came in the form of determining how to actually create whatever 3D artifacts were required, and it was an important part of deciding on what platform to build PaintSpace, as different capabilities would be available to different bases. Various possible solutions to this problem, such as utilizing static meshes or particle emitters, came with their own sets of issues. For instance, did the method chosen provide adequate performance

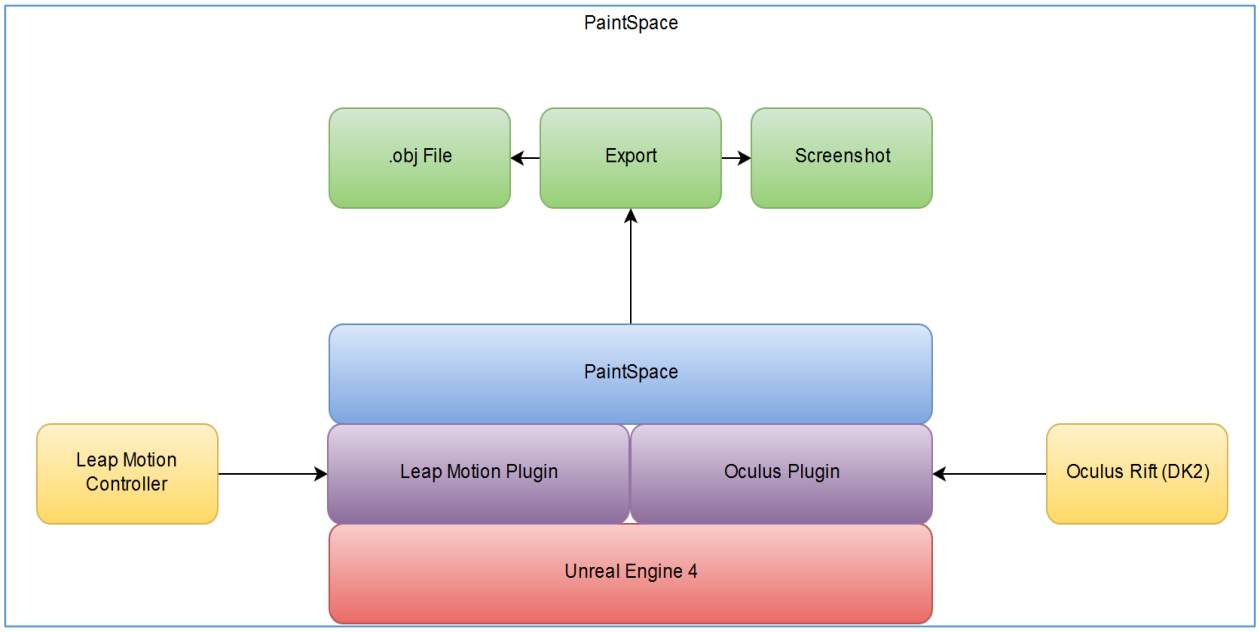
while keeping in mind the relevant frame-rate requirements for virtual reality? Also part of this challenge was the question of what the end goal of "drawing" was supposed to be. Should the resultant 3D artifact be a static mesh object intended for easy export and use in other applications? Or was the purpose to create artistic representations on a canvas, emphasizing aesthetic over practical use? Perhaps pursuing both end goals was also a viable option that would give the application greater depth. This question of the creation of 3D artifacts ties into the third major challenge, what sort of user interface and options are a result of and support the previous challenges?

The application's user interface was primarily determined based on the chosen input device. This further influenced what sort of options would be available and how to design them. Another major influence on this challenge was the question of what sort of tools and options the application needed to support. In an application whose purpose is akin to painting on a canvas, it made sense to include brush color and size options, and other related configurations. Based on the type of artifacts being created, options such as export and import were also necessary to consider.

PaintSpace required solutions to a variety of challenges, and in the next chapter of this thesis, the design choices made to answer the major challenges of developing this application will be detailed.

2. Design Choices

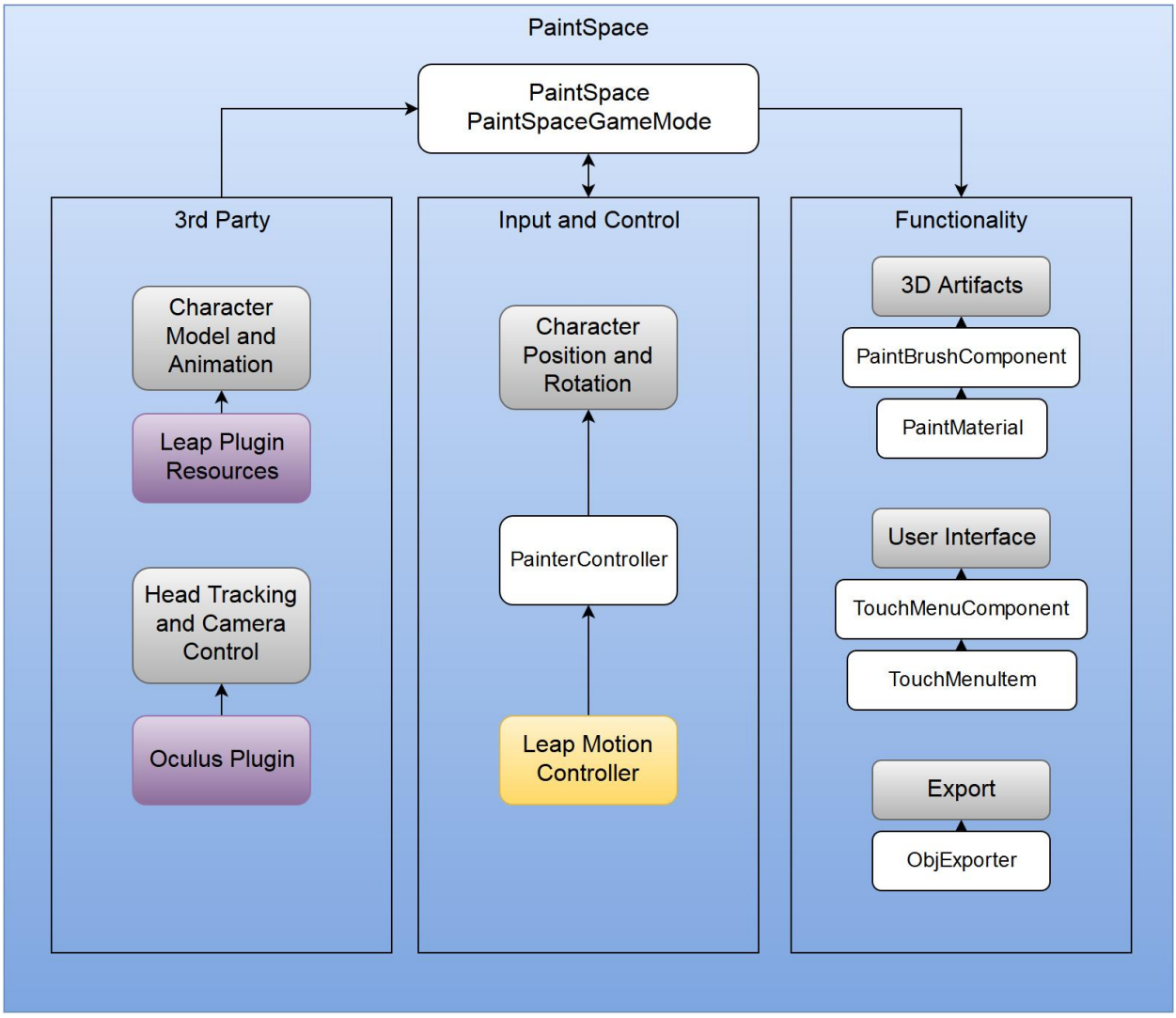
The following section will detail the methods and solutions used to solve the various challenges encountered during development, beginning with an overview of the application structure and design. The following diagram details at a high level the environment in which PaintSpace functions.



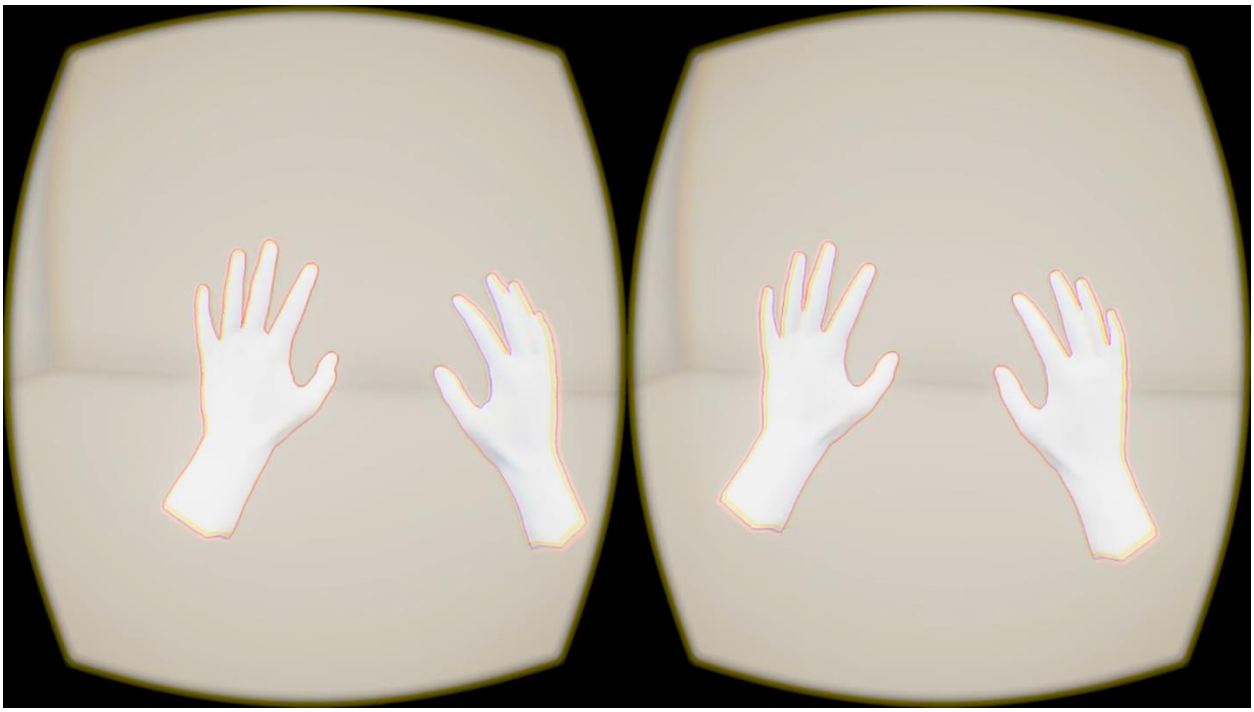
PaintSpace is built on top of Unreal Engine 4, and makes extensive use of two plugins for the engine, the Leap Motion plugin and the Oculus plugin. Each of these are primarily involved in enabling the use of specific hardware within Unreal. The Leap Motion plugin is integral to receiving and handling three dimensional input, which it enables through the use of Frame objects, which contain information such as hand and finger positions, types, and other qualities. These Frame objects enable the use of the user's hand motions as the primary form of input in PaintSpace, and they provide the basis for interpreting gestures made by the user within the

engine. The Oculus plugin handles and controls the viewport of the user. Head tracking and movement data from the Oculus hardware are transmitted and automatically used to control the camera position and give the user an accurate sense of being "in" the application space. For the most part, this plugin and its function requires little intervention to function sufficiently, and so it enables the PaintSpace application to work in virtual reality without any additional effort.

The next diagram is a more detailed view of PaintSpace itself, providing information on the key components of the application and their relationships with each other.



Each of the lightest colored blocks represents a C++ class or pair of classes. As described earlier, the 3rd party plugins provide specific functionality that is integral to the application's function. Additional resources from the Leap Motion plugin are also utilized. These include a set of models and animation rigs to create the properly animated hand static meshes that appear in the application as seen below.



The classes responsible for setup and initialization of the application state are PaintSpace and PaintSpaceGameMode. These determine the "character" type for the user and setup the initial state of the application. From there, input is received and handled by the PainterController class. This information is used to determine the rotation and position of both the character's camera and the "hands" of the character, which are analogous to the hands of the user.

Functionality is built into several component classes, as seen in the above diagram. These classes are subclasses of a built-in engine class called a USceneComponent. In Unreal Engine 4, these components are intended to provide some functionality or visible component that also

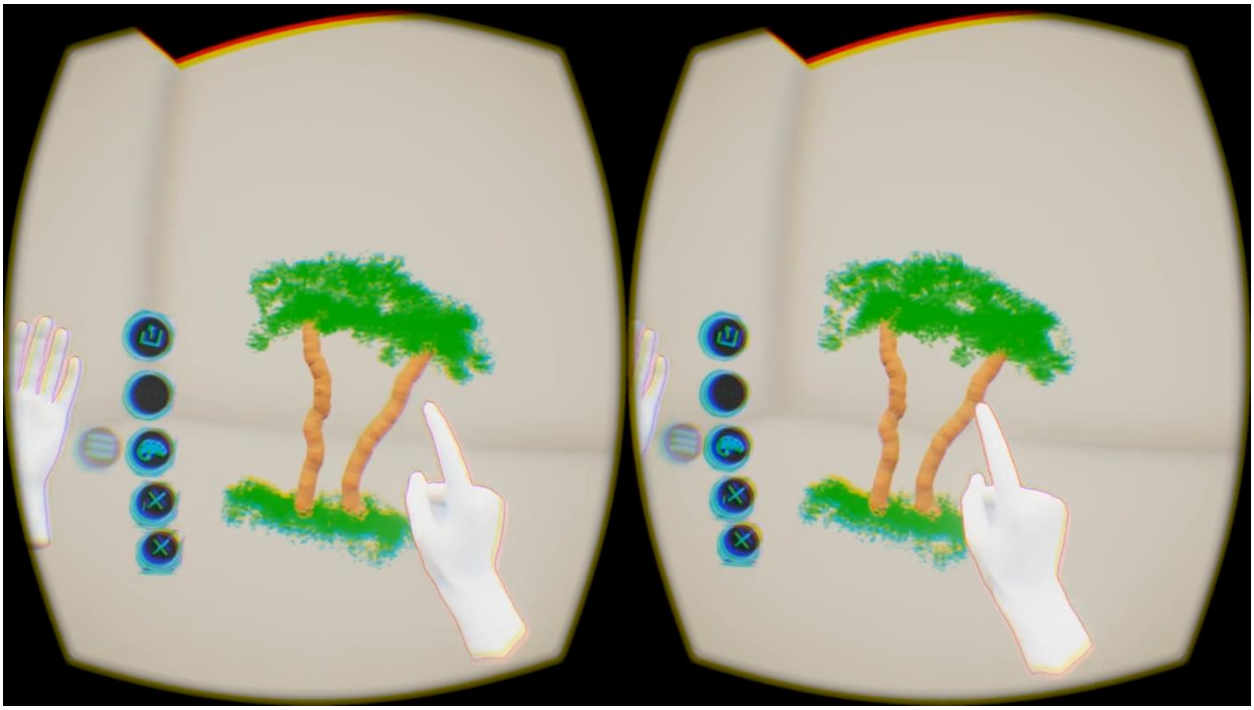
incorporates a transform, allowing it to move about the world space. In PaintSpace, these components are used to provide the major features of the application.

The PaintBrushComponent class is attached to the right hand index fingertip, and uses input data passed to it, primarily Frame data from the Leap Controller, to determine when and what to "draw" at its location in world space. It also maintains a PaintMaterial class that contains the various materials and other pieces of information that are necessary for the creation of the 3D artifacts. For instance, the PaintMaterial class contains a reference to the instanced static mesh and its texture information that is then used to create the more solid, clay-like 3D artifact. It also contains a reference to a particle generator system that is used to create a spray-paint-like cloud in the canvas.

The TouchMenuComponent class is attached to the left hand, and also uses input data to determine when and where to display the TouchMenuItem components it contains. These items are children of the TouchMenuComponent and can be expanded to create a submenu. The TouchMenuItems also maintain the type and texture information for their respective buttons, and determine what function is executed upon being "pressed."

The other major piece of functionality, ObjExporter, is not based on any engine classes, and instead is a basic C++ class that receives the information it needs about the current world state in order to execute a set of functions that write out an Object (.obj) file, which can be loaded in to various 3D modeling applications. The primary piece of information needed by this class is the instanced mesh data and position data from the particle emitter. Both of these are stored in the PaintMaterial instance. This data is used to write the necessary vertices and indices out to the .obj file.

Below is an image example of both the TouchMenuComponent and PaintBrushComponent in action. While the PaintBrushComponent itself is invisible, the results of its functions can be seen in the drawing of a tree below.



The decision to use Unreal Engine 4 was made because it provided significant convenience in terms of implementing the creation of 3D artifacts. It also helped with the implementation of input and control, and so generally made implementing various parts of PaintSpace simple. The major drawback to this approach was the time cost of gaining significant familiarity with the use of the engine and the tools it provided. A significant portion of the first month or so spent working on PaintSpace involved becoming familiar with the inner workings of the engine, especially regarding static mesh rendering data and where to obtain it. Additionally, the use of Unreal Engine necessitated developing some familiarity with C++, though the use of Visual Studio 2015 made this relatively simple.

As is evident in the above discussion of the application overview and details, the question of input was answered using a specific piece of hardware, the Leap Motion Controller. This tool provided an intuitive and simple way to interface in 3D. The user is able to "see" their own hands while in the virtual reality, and is able to make movements outside the application that are replicated accurately within it. This is certainly a "natural" way for humans to interface with many objects, and while it is fairly novel in the realm of software, it is intuitive and simple to pick up the use of one's own hands in a virtual realm. Additionally, gesture based movement was implemented to make up for the sitting experience provided by the Oculus hardware. Because the Oculus is not intended for movement around a room, it was difficult to examine a 3D artifact from all sides. To answer this, the Leap Motion Controller was used to provide specific gestures that would manipulate the character position in the application space, and allow for rotation so that the user could view artifacts they created from all sides. The Leap Motion Controller was also able to provide a significant degree of accuracy and responsiveness that would only have been beaten by specialized hardware built for use in virtual reality.

Also shown above is the end result of the challenge that involved what sorts of artifacts to create. It was decided that, in order to provide a variety of styles and options, two primary methods of creating artifacts would be used. The first is the creation of instanced static meshes, these allow for the creation of thousands of meshes whose data can be grouped together to reduce the number of draw calls made to the GPU. This is a massive increase in efficiency over creating thousands of individual static meshes, and allowed the frame-rate to maintain the necessary level for comfortable use in virtual reality. The second method was to use a GPU particle emitter to create up to millions of particles that gave off a 3D spray-paint sort of feel. This allowed for the creation of a wider variety of "drawings" such as the tree seen above.

The use of these two approaches to "painting" stems from the intuitive nature of sculpting, the most analogous real life form of art. The method of using static meshes to represent 3D artifacts is akin to using material like clay to sculpt an object. At the same time, while creation of clouds is not exactly within an artist's power, the use of particle emitters can be compared to a combination of spray-paint and a cloud-like object such as cotton candy. The result of this concept in PaintSpace is not difficult to work with and understand.

The interface design choices were a result of having a control scheme that was based on the user's hands. Given accurate representations of the hands within the program, the major problem was to provide buttons that were used to manipulate options controls. Since our hands do not have buttons, an interface was developed that was gesture based. Floating buttons in 3D space are fairly intuitive, and resulted in a clearly visible and easy to use interface. This particular challenge ended up being more difficult to implement, though, in comparison to the others. While in the other cases there were engine tools or plugins available that contributed significantly to implementing those features, the concept of this kind of interface is still relatively novel, and there weren't any available implementations to work from in Unreal Engine. Developing these from scratch resulted in various issues, particularly when it came to providing deeper functionality than just switches. While simple buttons were fairly easy, more complex functionality involved a much greater amount of work. One such issue was the creation of recursive sub menus, which for the purposes of demonstration was deemed unnecessary and left out. Another feature, still in development at the time of writing, is the creation of slider style buttons that would allow the user to accurately control something such as size or color. These options benefit from a gradient much more so than specific points on a scale. Problems such as these made this particular design challenge require more interesting considerations that the other

major issues above. With the novel nature of this problem in mind, there is certainly plenty of room in the future for development and innovation in the area of three dimensional interfaces.

3. PaintSpace

At the time of presentation and the conclusion of this thesis as an academic project, the PaintSpace application provided many of the capabilities and features initially planned as part of the design. At the same time, however, there are still many features left to be developed, or issues that remain to be fixed. As it stands, PaintSpace allows for the creation of varied 3D artifacts, and provides an interface that allows for some manipulation of these artifacts. Additional convenience controls are present to facilitate other aspects of creating 3D artifacts, such as the gesture-based movement used to move around the world in the application. PaintSpace allows the user to experience and explore the creation of art on a 3D canvas in one possible approach of many. However, in its current state it is still comparable to a first approach or proof of concept.

Various features will continue to see development as the application continues to be fleshed out and improved. Part of the end goal of PaintSpace itself is to not only demonstrate features such as drawing 3D artifacts in various colors and styles, but allow full manipulation of all aspects of the process of drawing, and incorporate a more robust set of options that contribute to this. Additionally, it is an important part of the goal of PaintSpace that the application serves as a platform for exploring various styles of creation in a 3D space. To this end, additional styles of drawing, such as the creation of wireframes rather than direct static meshes themselves, are part of the development goals for the project.

4. In Comparison and the Future

It is worth noting that there exists a somewhat similar set of applications to PaintSpace already. Some exist in a state similar to that of PaintSpace, while others are much more fully featured applications. All of them are relatively recent innovations. The most notable of these is Google's Tilt Brush. This is a tool designed for use with another virtual reality device, the HTC Vive. This piece of hardware incorporates a pair of motion controllers that allow for powerful tracking and interface design. The Vive also provides a room-scale experience, allowing the user to walk around their creation rather than sit in front of it and move their character around as is the case in PaintSpace. While Tilt Brush is certainly a much more complete application with more depth than PaintSpace, the significance of its mention here lies in the implications that exist regarding the future of this realm of applications. The development of applications such as Tilt Brush, the various others that exist, and even PaintSpace is indicative of an early exploration of one specific aspect of software that the recent improvements to virtual reality have opened up. As the hardware, in every area of these sorts of applications, continues to see innovation, new styles of creation in 3D space will crop up, and old ones will see improvement. Oculus plans to release a set of 3D input devices called Oculus Touch. Leap Motion plans an improved set of hardware to be released. There is plenty of room for exploration in this field, and applications such as PaintSpace or Tilt Brush only explore a few aspects. The future of this area, as with many others related to virtual reality, is bright, and holds many unknowns.

5. References

The following references include various devices, plugins, and other applications mentioned.

Additionally, a link to the demo from the presentation of PaintSpace and the source are included.

- Demo, https://www.youtube.com/watch?v=rARAndXZMKY
- Source, https://github.com/benmadany/PaintSpace
- Oculus Rift, https://www.oculus.com/en-us/rift/
- Leap Motion, https://www.leapmotion.com/product/vr
- Unreal Engine 4, https://www.unrealengine.com/what-is-unreal-engine-4
- Leap UE4 plugin by Getnamo, https://github.com/getnamo/leap-ue4
- Google Tilt Brush, http://www.tiltbrush.com/