

Virtual Reality Visualization with the Oculus Rift

Nicholas Wereszczak
Department of Computer Science, Tagliatela College of Engineering
Dr. Alice Fischer

Abstract

Virtual reality has always been considered a way of the future and was only seen as an application for video games. Today, there has been development of video games that use the virtual reality technology, but it has come to light that this technology can be used for many other purposes, including academically. This project took his new technology and started a foundation for use in new learning applications and new and improved views on existing data. The Oculus Rift was the tool used to view VR worlds in this project along with the use of a MacBook Pro; however, any laptop with a graphics card can be used with the Oculus Rift. Using the Oculus Rift's software development kit (SDK), a computer program was written to read data and present it in a three-dimensional space. In addition to being able to read data and display it in a graphical form, user entered input can be used to create custom graphs and images.

Introduction

In today's world, there are many programs that allow you to view images in three-dimensions, but only on a flat screen. This project allows the user to feel like they are in the same space as the image they are viewing. This will allow for a deeper understanding of data. People that learn visually will be able to have a clear picture of what is trying to be represented.

This research explored the use of virtual reality for learning application. The overall objective of this project was to create a program that could pave the way for new technologies to be used in the academic community and to explore new possibilities using this new technology. Specific goals included being able to learn and understand the Oculus Rift, develop a technique to express data into a series of coordinates in three-dimensional space, as well as to discover, experiment, and pave the way for new and future technologies.

Materials

A few pieces of hardware were necessary for this project; the Oculus Rift device to be able to view the virtual reality that was used as shown in Figure 1, and a MacBook Pro to run the written program and the Oculus Rift SDK.



Figure 1: Oculus Rift Headset

After these devices were ordered, an overall design was made for the project to determine what other materials would be needed. The original design showed that there

would only be one way for the user to observe the data being presented.

The Oculus Rift headset and the Oculus Rift SDK both included tools that would allow the programmer to track head position. This seemed like an issue for users who wanted to be able to rotate and observe the data from different angles. It was then deemed necessary that a connectable controller, shown in Figure 2, should be added to increase ease of use. This created two types of input for movement, which is shown in Figure 3.



Figure 2: Connectable Controller

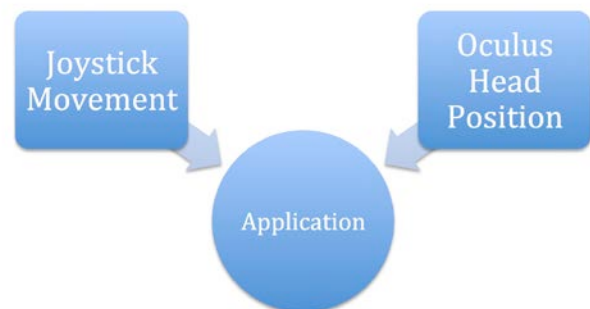


Figure 3: Inputs into Application

After the hardware side of the design was finalized, the software side was considered. The Oculus Rift SDK was analyzed, and it was determined which additional software was compatible with the current SDK.

First, the programming language was decided. There were only a few compatible languages that could have been used, but it was decided that C++ would be a good language to choose. It was chosen because most of the pre-written programs were in C++ and there are many graphical libraries available to help transform the data into graphs.

A few integrated development environments (IDE) were considered. Code::Blocks, NetBeans, and Visual Studios were all considered and are compatible with C++. Since a MacBook Pro was being used, Xcode was also available as an IDE, which meant Visual Studios was not. Xcode was chosen due to its well-known debugging assistance and that many example projects were pre-created for Xcode.

The next step was to look at the available 3rd Party libraries that would help create the virtual reality worlds. Many were available, so an example project was examined to see what libraries were used and what libraries would be useful to the project. A few libraries were decided on. The ones that were used in this project were, OpenGL, glew, glm, and zlib. These 3rd Party resources were open source and can be found online.

The software and hardware were tested with the examples given within the Oculus SDK to check compatibility. After both the hardware and the software that would be used were finalized, the research began.

Research and Input

After the Oculus Rift was ordered and the design was finalized, the first week was spent doing research on the Oculus Rift SDK. A manual was given with the download of the software kit and more research began.

At the start of the project, the demo and example code was studied and learned. This was done in an effort to simplify what was being looked at in the code, and what code would become a useful tool in the project goal.

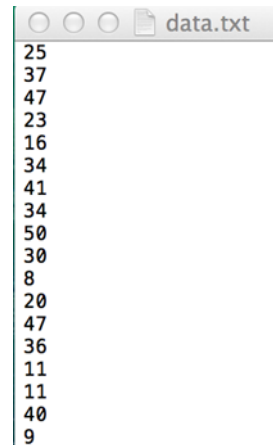
Code to control and interpret a connectable controller was found, studied, and saved so that later on it could be integrated into the design. Most of the given code was in C++, however some of it was in Objective-C, which took longer to decipher.

After an understanding for the Oculus Rift SDK and how to change and control the camera, the early stages of research were completed. It was then determined how the data would be implemented.

Two ways were considered for entering data. The user would be able to enter information directly into the program or information could be read from a text file. It was decided that reading from a text file would serve the application efficiently. For this project, two different types of data were entered. They were stored in different ways.

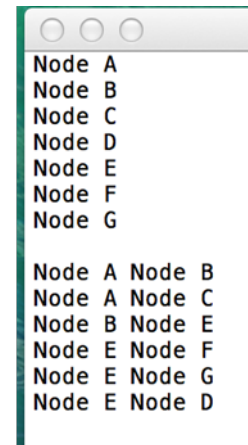
A tree structure was used to store information that was generated randomly in the program and to store large amounts of data if some was input, shown in Figure 4. This seemed like the simplest way to store data and be able to draw it in a unique way. A modified and simpler version of a binary search tree (BST) was used. Data was read in and placed into the modified BST.

The data that was small and custom-entered was sorted in a Flex array, shown in Figure 5. A Flex array is a modified vector data type and was created by Dr. Alice Fischer. It was implemented by the use of her Flex Template, flexT.h.



```
data.txt
25
37
47
23
16
34
41
34
50
30
8
20
47
36
11
11
40
9
```

Figure 4: Tree File (Large)



```
Node A
Node B
Node C
Node D
Node E
Node F
Node G
Node A Node B
Node A Node C
Node B Node E
Node E Node F
Node E Node G
Node E Node D
```

Figure 5: Non-Tree File (Small)

Drawing Algorithms

Two drawing algorithms were created to generate a three-coordinate plane. The algorithm used to draw the smaller graph to three-dimensional space has multiple parts. It starts off with a creation of a new type, node.

Node has three properties. The first is an integer; named index, which holds the number of that node. The index increments every time a new node is appended. The second property is a glm::vec3, named locs. Locs holds the location of the current node within three-dimensional space. The third property is an array of all of the indexes of the nodes that the current node is connected to. This is used to draw the lines in the three-dimensional space.

The algorithm creates a Flex array of nodes. It first reads in each node and appends its name to a list. The algorithm then creates a new node with index 0 and appends this node to the Flex array of nodes. Each time a new node is read from the text file it checks to see if the name has already been read, to prevent duplicates, and then appends it to the Flex array of nodes.

The algorithm then proceeds to the part of the text file where the connections are being shown. It appends the appropriate indexes to the correct node-connects array. Later, when drawing in the three-dimensional space, it uses the locs vector and the connections array to draw the nodes and their “connections”, or “lines”, to the other nodes.

The second algorithm used to draw the larger graph in three-dimensional space was not as complicated. When reading from the tree text file it reads in a number that is inserted into the tree structure using the original BST algorithm. After all the nodes are inserted into the tree, the function preorder() is called and creates and stores locations for each node compared to the last. The first node has the location (0, 0, 0) and then every node after is referenced relative to that. The basic point of the algorithm is to make a two-dimensional structure display in three-dimensions. The idea behind it is that for every 6th node, a -z coordinate

is added to switch the direction of the tree, and every 4th node, a +z coordinate is added to switch the tree back. The code for this algorithm is shown in Figure 6.

```

if (p1=root) {
    spot = prev->index;
    p->index = num;
    if (!left) {
        if (spot==0) {
            locs->append(((*locs)[spot])+glm::vec3(-0.2f, -0.2f, 0));
            p->el = (*locs)[spot+1]; //changes data in tree
            temp.p1 = (*locs)[spot];
            temp.p2 = (*locs)[num] - (*locs)[spot];
        }
        else {
            if (num%5 == 0)
                locs->append(((*locs)[spot])+glm::vec3(-0.1f, -0.1f, -0.2f));
            else if (num%5 == 1)
                locs->append(((*locs)[spot])+glm::vec3(-0.1f, -0.1f, 0.3f));
            else locs->append(((*locs)[spot])+glm::vec3(-0.1f, -0.1f, 0));
            p->el = (*locs)[spot+1]; //changes data in tree
            temp.p1 = (*locs)[spot];
            temp.p2 = (*locs)[num] - (*locs)[spot];
        }
        lines->append(temp);
    }
    else if (!left) {
        if (spot==0) {
            locs->append(((*locs)[spot])+glm::vec3(0.2f, -0.2f, 0));
            p->el = (*locs)[spot+1]; //changes data in tree
            temp.p1 = (*locs)[spot];
            temp.p2 = (*locs)[num] - (*locs)[spot];
        }
        else {
            if (num%5 == 0)
                locs->append(((*locs)[spot])+glm::vec3(0.1f, -0.1f, -0.2f));
            else if (num%5 == 1)
                locs->append(((*locs)[spot])+glm::vec3(0.1f, -0.1f, 0.3f));
            else locs->append(((*locs)[spot])+glm::vec3(0.1f, -0.1f, 0));
            p->el = (*locs)[spot+1]; //changes data in tree
            temp.p1 = (*locs)[spot];
            temp.p2 = (*locs)[num] - (*locs)[spot];
        }
        lines->append(temp);
    }
    count++;
}
else { //root
    p->index = 0;
    count = 0;
    locs->append(glm::vec3(0,0,0));
    p->el = (*locs)[0];
}
return count;

```

Figure 6: Store Location Algorithm

Results and Discussion

In the end, a basic foundation was created for interpreting data and transferring it into three-dimensional space. The output was impressive. It accomplished the goal to transfer data and to be able to view data in a new way. In Figure 7, a small custom tree is drawn from the given input from Figure 5. In Figure 8, a binary search tree is shown after being drawn in three-dimensional space using the algorithm in Figure 6.

The data was clear, and when used with the Oculus Rift headset and controller, it created a new experience for the user. Wearing the headset was a different experience for many and showed the potential for future applications. The specific goals were met and the project can now be easily improved with the help of the mentor and researcher.

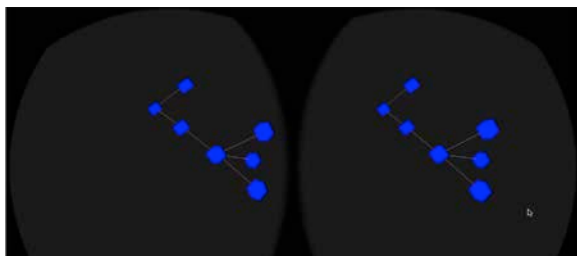


Figure 7: Small Simple Custom Graph

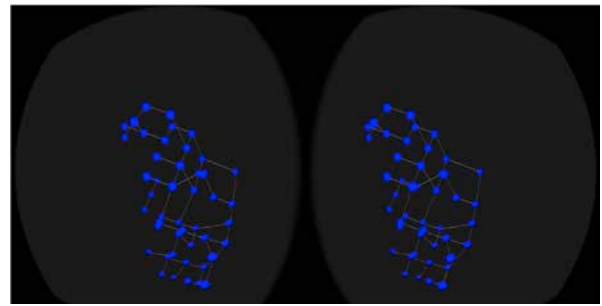


Figure 8: Large Complex Tree Graph

New researchers in this field will be able to save time by examining this project's simple application that has been pieced together out of the Oculus Rift SDK. Some new technologies that can be created from this are, for example, chemistry and biology molecule representation and three-dimensional building projects that convey the feeling of walking around a building before it has been built. Another example would be to depict cells division, or osmosis, as a mini clip and watch the process as if it was right in front of you. There are many more applications.

The project however, was not perfect. A few improvements could be made by our successors. Work would be needed on the joystick controls on the software level. This would help the user-interface immensely. Work would also be done to have multiple graphs drawn in the same three-dimensional world.

One problem that came up during the design was that real data was way too big to process on a small laptop. If more processing power and faster graphics were available, then a more detailed and larger graph could have been created. This could also lead to better performance and response time when the application was being used. Another benefit would be that the mini clips, as mentioned before, could have more detail and run more smoothly. With enough processing power users might even be able to manipulate and change variables in real time.

Conclusion

Overall the project was a success. The project used the Oculus Rift headset and the Oculus Rift SDK. A great understanding of virtual reality was developed, and of how it can be implemented. OpenGL along with other graphical software libraries were learned, which could be used with other languages in the future. One specific goal that was accomplished was the ability to have users enter personal data to see their own graphs and change the shape and color of the nodes. The most important goal of creating a foundation for the new and upcoming applications in this field was completed in hopes someone will take this project to the next level.

References

1. Oculus Rift [Web Photo]
http://upload.wikimedia.org/wikipedia/commons/a/ae/Oculus_Rift_-_Developer_Version_-_Front.jpg
2. Logitech Controller [Web Photo]
<http://gaming.logitech.com/assets/47832/f310-gaming-gamepad-images.png>
3. Oculus SDK - Version 0.3.2 Preview 2
<https://developer.oculusvr.com/?action=hist&ver=16>
4. OpenGL SDK – GLEW/GLM Library
<http://www.opengl.org/sdk/libs/>
5. zlib SDK – zlib Library
<http://www.zlib.net>
6. Project Setup and Reference from sample project
<https://github.com/OculusRiftInAction/OculusRiftInAction>

Acknowledgments

I would like to thank both my research advisors and mentors, Dr. Christopher Martinez and Dr. Alice Fischer, for their knowledge and support throughout the research. I would also like to thank Mark Morton, who helped find certain hardware that was necessary for compatibility between devices. I would also like to thank UNH's SURF program for making my research a reality. I would also like to thank the Carrubbas for their generous donation and for keeping the SURF program running every year. I would also like to thank everyone else involved in the SURF program that helped me with my research. A special thanks to everyone who keeps this program running: you help students realize their potential and get the experience they need.

Biography

Nicholas Wereszczak is majoring in Computer Science and is a junior at the University of New Haven. Nicholas is also a teaching assistant for Intro to C Programming at the University of New Haven and a member of the UNH programming team. In addition to his love for computer science, he also enjoys swimming, IEEE club meetings and hanging out with his close friends.

The research from this project helped Nicholas improve his programming skills immensely and he is glad that he has been given this opportunity. It also gave Nicholas helpful experience to use in graduate school. Nicholas is interested in graduate school, but not yet sure where he would like to study or what type of Computer Science he would like to concentrate in. However, he did greatly enjoy using visual and graphical SDK's.

