Oculus Rift Retina Display

Report of Practical Training

submitted by

Huaijiang Zhu

(03639539)

Supervisor: Prof. Dr. Jörg Conradt, Dipl. Nicolai Waniek

Contents

Chapter 1: Project Background **1**

Chapter 2: Development Tools **2**

2.1 Oculus Rift Development Kit 1 (DK1) 2

2.2 Event Based Dynamic Vision System (eDVS) 2

2.3 OpenGL 3

Chapter 3: Implementation of the Project **4**

3.1 Development Environment Setup 4

3.2 Visualization of the eDVS Data 4

3.3 Integration of eDVS with the Oculus Rift 5

3.4 Recording/Replaying Function and Time Synchronization 6

3.5 Attachment 6

Chapter 4: Experiments **7**

4.1 Stereo Vision 7

4.2 Minimal Requirement of Information Determination 8

Chapter 5: Summary of the Project **9**

Appendix A: Experiment Results for Stereo Vision **10**

Appendix B: Sources for Figures **11**

Bibliography **12**

Chapter 1: Project Background

From July 28, 2014 to September 26, 2014 I did my practical training in the Neuroscientific System Theory Group affiliated to the Chair of Automatic Control Engineering at TUM under the supervision of Professor Dr. Jörg Conradt and Mr. Dipl. Nicolai Waniek. During my internship, I carried out the project of Oculus Rift Retina Display together with my colleague Mr. Zhejun Zhang.

The goal of this project is to determine the minimal requirement of information for a human being to navigate safely and correctly in unknown environments.

This is important in the development of retinal prothesis, because the volume of data that need to be transmitted can affect not only the latency of the whole system but also the amount of the heat generated on the implemented chip. To avoid the possibility of thermal damage to the retina (Piyathaisere et al., 2002), the volume of data should be minimized.

The virtual reality headset Oculus Rift and eDVS (Event Based Dynamic Vision Systems) are used to answer this particular question. The eDVS is well suited for this project because unlike conventional camera it only sends the local pixel-level changes caused by movement instead of redundant data of the whole frame.[1] The data provided by eDVS will be displayed by an interface software on the Oculus Rift headset. Thus, psychophysical experiments can be performed in order to investigate how detailed the information must be so that the navigation is possible.

[1] http://www.nst.ei.tum.de/projekte/edvs/

Chapter 2: Development Tools

The main task of this project is to design a system which can simulate the visual experience of patients implanted with retinal prothesis. To achieve that, following tools were applied.

2.1 Oculus Rift Development Kit 1 (DK1)

The Oculus Rift DK1 is a virtual reality head-mounted display developed by Oculus VR. It provides an immersive experience in an approximately 110° field of view, using two lens in a head-mounted display that combine to form a 1280 x 800 resolution. Each eye sees 640 x 800 pixel.(Lang, 2012)

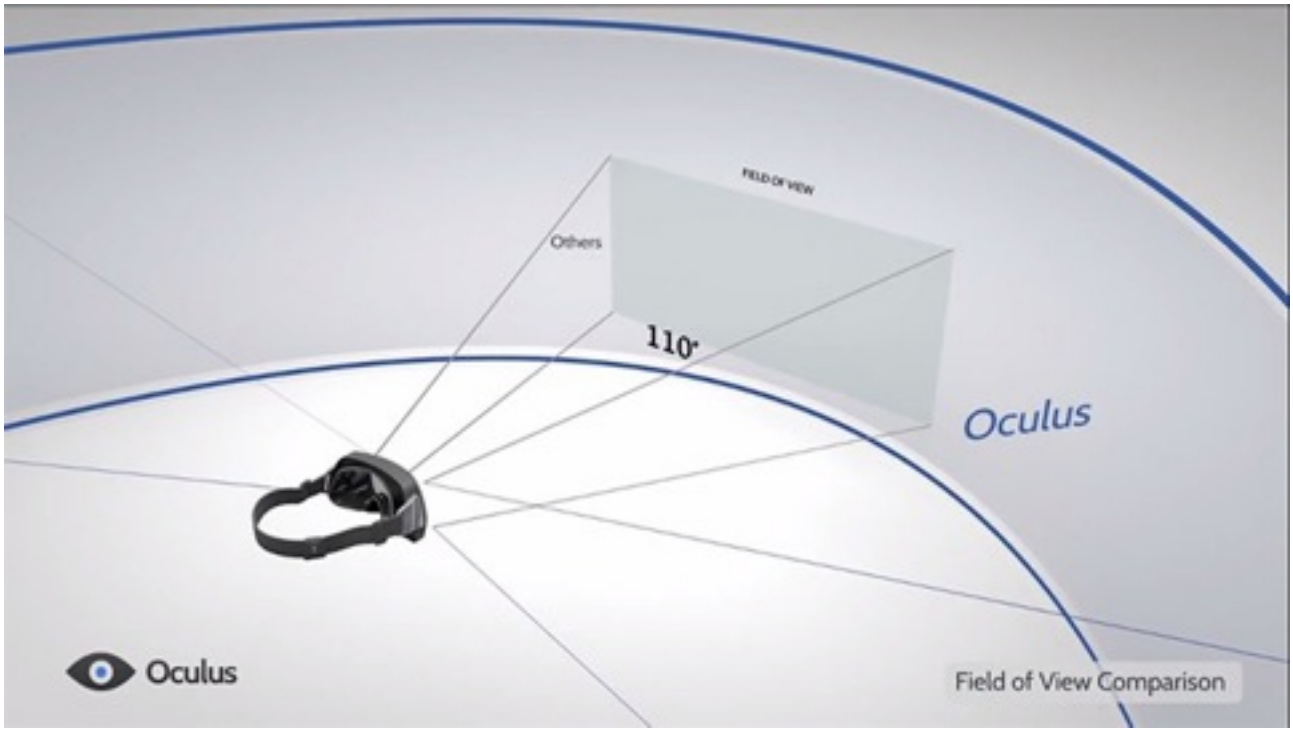

FIGURE 2.1: FIELD OF VIEW OCULUS RIFT

The distortion and the chromatic aberration caused by the lens will be corrected in an approach called SDK Distortion Rendering. While doing so, the Oculus SDK takes care of all necessary calculations when generating the distortion mesh.

2.2 Event Based Dynamic Vision System (eDVS)

The Event Based Dynamic Vision System is a small embedded system consisting of an ARM7 micro controller with a DVS sensor.[2] Instead of wastefully sending entire images at fixed frame rates, only the local pixel-level changes caused by movement in a scene are transmitted – at the time they occur. The result is a stream of events at microsecond time resolution, equivalent to or better than conventional high-speed vision sensors running at thousands of frames per second. Power, data storage and computational requirements are also drastically reduced, and sensor dynamic range is increased by orders of magnitude due to the local processing.[3]

In this project, an existing library written by David Weikersdorfer, an alumni of NST, will be applied to interface with the Oculus Rift. A stream of events packed in a structure consisting

[2] http://siliconretina.ini.uzh.ch/

[3] http://www.nst.ei.tum.de/projekte/edvs/

of abscissa, ordinate, polarity and time can be obtained through the universal asynchronous receiver/transmitter (UART). The resolution of the DVS sensor used in the project is 128 x 128 pixels.

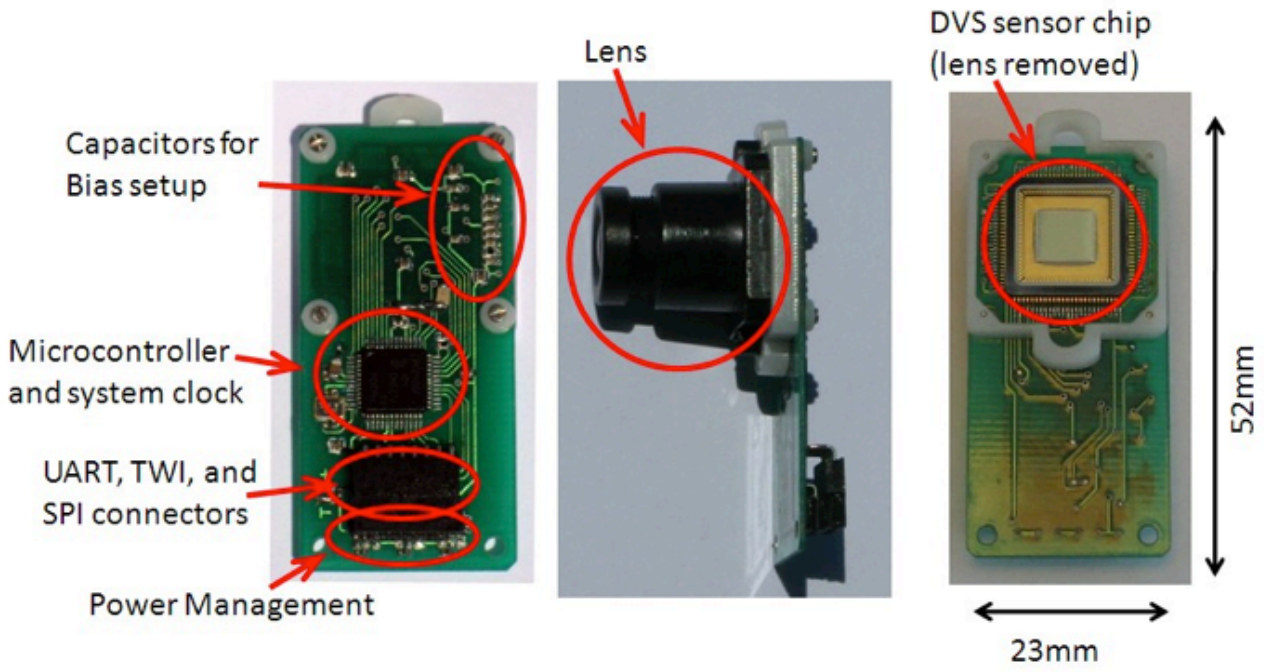

FIGURE 2.2: EVENT BASED DYNAMIC VISION SYSTEM

2.3 OpenGL

OpenGL (Open Graphics Library) is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. To do that, OpenGL takes a sequence of steps as the Rendering Pipeline. The general steps are shown in the Figure 2.3. (Shreiner, Sellers, Kessenich, & Licea-Kane, 2013)

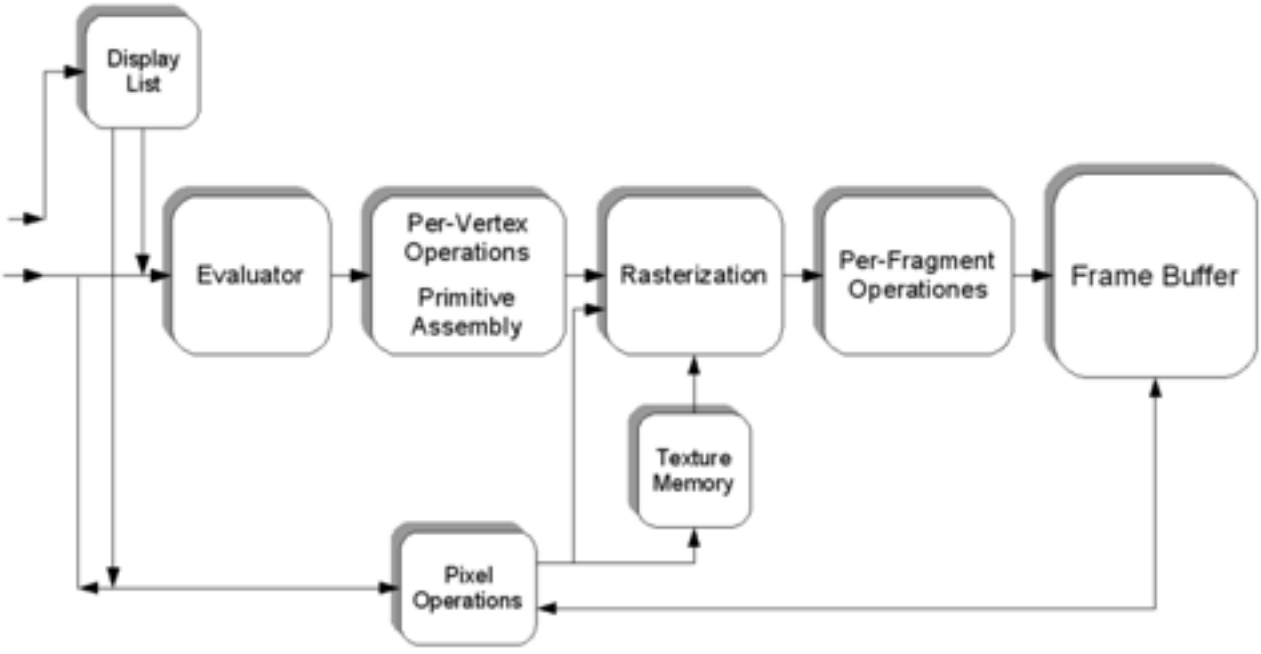

FIGURE 2.3: RENDERING PIPELINE OF OPENGL

Chapter 3: Implementation of the Project

The project is a team work with my colleague Zhejun Zhang, our supervisors and other students who offered help. In this chapter, I will focus mostly on my work and experience during the practical training.

3.1 Development Environment Setup

The project requires integration of various code libraries which have the best support under Linux platform. So after a number of failed attempts to modify the related libraries to work under Windows/Mac OS X, my colleague Zhejun Zhang and I decided to develop the software under Ubuntu 14.04 despite the fact that we had already started to learn and set up OpenGL environment under other platforms for a couple of days.

While Zhejun Zhang was reading and trying to employ the existing eDVS interfacing libraries in our program, I engaged myself in OpenGL and Oculus Rift SDK.

Due to the convenient package management of Linux, it didn't take me much time to install the library dependencies and to set up the development environment of OpenGL. After reading several tutorials on the Internet, I wrote my first OpenGL program—drawing a red triangle with Eclipse 3.8. However, the Oculus Rift SDK was relatively hard for a beginner like me to learn in a short time, since Oculus VR provided a very sophisticated demo-program but a less detailed document along with it.

My colleague Zhejun Zhang did some research on the Internet and found a simple program on the Oculus Developer Forum.[4] This program discarded the complicated modeling as well as the user controlling part of a full functional game. Instead, it focused only on the basic usage of the Oculus Rift SDK by simply demonstrating a rotating cube. Although this program used a less modern way of OpenGL to draw the entities and still needed to be rewritten into OpenGL 3.3 style, it was already a good starting point for the whole project.

3.2 Visualization of the eDVS Data

To visualize the data from eDVS, OpenGL was chosen as the main tool because of its cross-platform compatibility for the Oculus Rift SDK. As in Section 2.2 described, every single eDVS event is packed in a structure consisting of abscissa, ordinate, polarity and time. The task was to map these events to OpenGL vertices and to displaying them in a way that human beings can understand.

The first thing I did is to read the programs that other students wrote before, using Matlab and Qt-Library. Then I found that the polarity of an event was interpreted differently in these two programs. While in the Qt-program the pixel would be set to white if the polarity was 1 and black if it's 0, the Matlab program changed the hue of it. So I asked my supervisor Nicolai Waniek, and he explained that the polarity shows the on/off state of the events, but its time scale is too short for human beings to understand what really happened. Therefore, a fading/washing out effect must be added so that a event lasts longer on the screen. The changing hue in the Matlab program actually did this job and there was an another equivalent function in the Qt-program. Furthermore, since the eDVS data don't contain any information of color, it makes no significant difference in which color the event is displayed.

The other part was to map the 128 x 128 pixels of eDVS data to the OpenGL vertices. In OpenGL the vertex data are usually transmitted to the shader program in form of a one-dimensional vertex array. Therefore, the two-dimensional coordinates of a point from cartesian system must be converted. The existing program gave a simple and intuitive solution: See the pixels as a 128*128 matrix and then connect every row of it. Thus, the pixels would be put into a one-dimensional array and the index of each element would be calculated as the addition of abscissa multiplied by 128 and the ordinate.

---

[4] https://developer.oculusvr.com/forums/viewtopic.php?f=20&t=8680

Since the eDVS sensors were not available at the beginning of the project, I used a text file of recorded data from another project for test. During the test I found that the program displayed the events much slower than it should be. After my supervisor reminded me, I realized that a rendering loop of OpenGL takes normally much longer time than a eDVS event. So it's necessary to accumulate a quantity of events and display them in one rendering loop. But this involved another problem: the recording speed and displaying speed of the events would not be synchronized if the accumulated number of events was not chosen properly. And I decided to deal with it after testing the program with real eDVS sensors, since only by that can I know how the eDVS data would be recorded.

For further use of this program in the project I rewrote it into a C++ Class after the test. To achieve that, I read some tutorials and learned the basics of the implementation of OOP in C++. The class named eDVSGL stores all the variables and can be easily instantiated with the external environment, no matter whether it's a Oculus Rift demonstration or a normal OpenGL program. With the member function in it, the eDVS events data can be converted and stored in OpenGL-specified form for the visualization later on. When other people need to use this program, they don't have to deal with the complicated codes. Instead, 4 lines of initializing, setting up, updating events and drawing will be enough.

3.3 Integration of eDVS with the Oculus Rift

After I succeeded in visualizing the eDVS data with OpenGL, the next step was to integrate it with the Oculus Rift SDK, so that an interface software could allow users to see the visualization through the Oculus Rift Headset.

To add Oculus support to an OpenGL program, following steps are required:

1. Initialize LibOVR.
2. Enumerate Oculus devices, create the ovrHmd object, and start sensor input.
3. Integrate head-tracking into the application's view and movement code.
4. Initialize rendering for the HMD.
5. Modify application frame rendering to integrate HMD support and proper frame timing.
6. Customize UI screens to work well inside of the headset.

(Oculus VR, 2014)

With the demo-program found in the Oculus Developer Forum, I completed the first 4 steps without much trouble. The fifth step was a little bit tricky for the project because we were required to integrate two instead of one eDVS sensors as the source of input, while the demo-program had only one. My intuitive idea was to create another object of the eDVSGL class and run the drawing function separately in two different fields of view. But one field of view showed constantly blank background after the modification, although debugging indicated that the codes I added were executed without problem. Then I started to check the codes line by line and finally found out the problem after a whole day.

The window-creating library GLFW applied in the project uses double-buffering, which means the developer has to swap the front and back buffer when the entire frame has been rendered.[5] Since there were two fields of view in my program, the buffers should be swapped respectively in each eye. This wasn't done in the demo-program since it only rendered from one source.

Another problem in the demo-program was that it modified the model-view and projection matrix according to the data from the sensors of Oculus Rift so that the image changed itself

[5] http://www.glfw.org/docs/3.0/quick.html

when users move their heads, which, in our project, was not required, since the eDVS sensors would be mounted on the Oculus Rift and move with it. This was easily solved by setting the model-view matrix to an identity matrix.

3.4 Recording/Replaying Function and Time Synchronization

In order to conduct the psychophysical experiments required for the project, a recording/ replaying function should be integrated to the program. This was implemented with the file input/output function of C - fscanf and fprintf. In our program, a function will be called to obtain events in every rendering loop under recording mode. These events will be stored simultaneously in two plain text files, for left and right eye respectively. Since the rendering speed varies massively between different CPU/GPUs, a rendering loop can take much longer time on an old PC. This consequently increases the time interval between two requests of events.

As in Section 3.2 described, when the program reads from files of recorded data, a fixed number of events will be accumulated and displayed in every rendering loop. This can cause a problem of time synchronization for reasons above. For example, when a relatively faster computer records the data, the time interval between every 500 events can be very short - for instance, 10 microseconds. But a slower computer might demand 30 microseconds to finish a rendering loop. When it still accumulates 500 events in every loop, the displayed footage will be approximately three times slower. Therefore, an accumulation of fixed number is not practical for this situation.

I solved this problem by accumulating events in a time interval equivalent to the duration of the rendering loop. Although each rendering loop on the same computer also slightly differs in duration for different complexity due to the dynamic scene, we can set it to a fixed time by active waiting.

3.5 Attachment

My colleague Zhejun Zhang designed a simple attachment which can mount the two eDVS into the Oculus Rift Headset. This consists mainly of a polycarbonate plate remodeled by a laser cutter. A sketch of this is shown below.

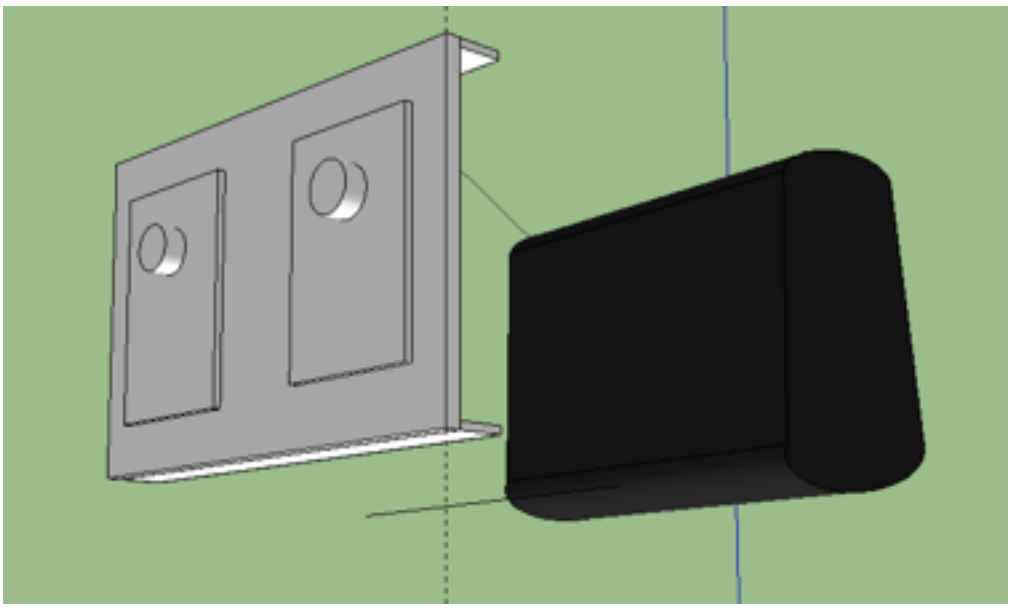

FIGURE 3.1 SKETCH OF THE ATTACHMENT

Chapter 4: Experiments

After a series of debugging, the system of two eDVS working cooperatively with the Oculus Rift could present the scene captured by the sensors so that related experiments could be performed.

4.1 Stereo Vision

At the project status meeting, our supervisor Professor Dr. Jörg Conradt pointed out some problems mainly focusing on the stereo vision.

Stereo vision involves two processes: The fusion of features observed by two (or more) eyes and the reconstruction of their three-dimensional preimage. (Forsyth & Ponce, 2012)

In our project, an algorithmic fusion was not necessary, because the scene would be captured with two eDVS respectively and then directly displayed in front of two eyes. To acquire a good visual experience of stereo fusion, some parameters like inter-pupillary distance and size of the field of view should be adjusted individually. While the human brain will complete the fusion during the process, it is hard to calculate the proper parameters quantitatively. Therefore, a trial-and-error method was applied.

Modification of the related parameters can be easily done by changing the OpenGL viewport and model-view matrix. With the keyboard controlling function developed by Zhejun Zhang, users can adjust the parameters conveniently and save/load their settings later on.

To verify if the stereo vision was achieved after parameter adjustment, an experiment was carried out. For each experimental run, subjects saw two recorded footages successively in which a single line was located in different distances from the shooting point. A sample of the footage is shown below.

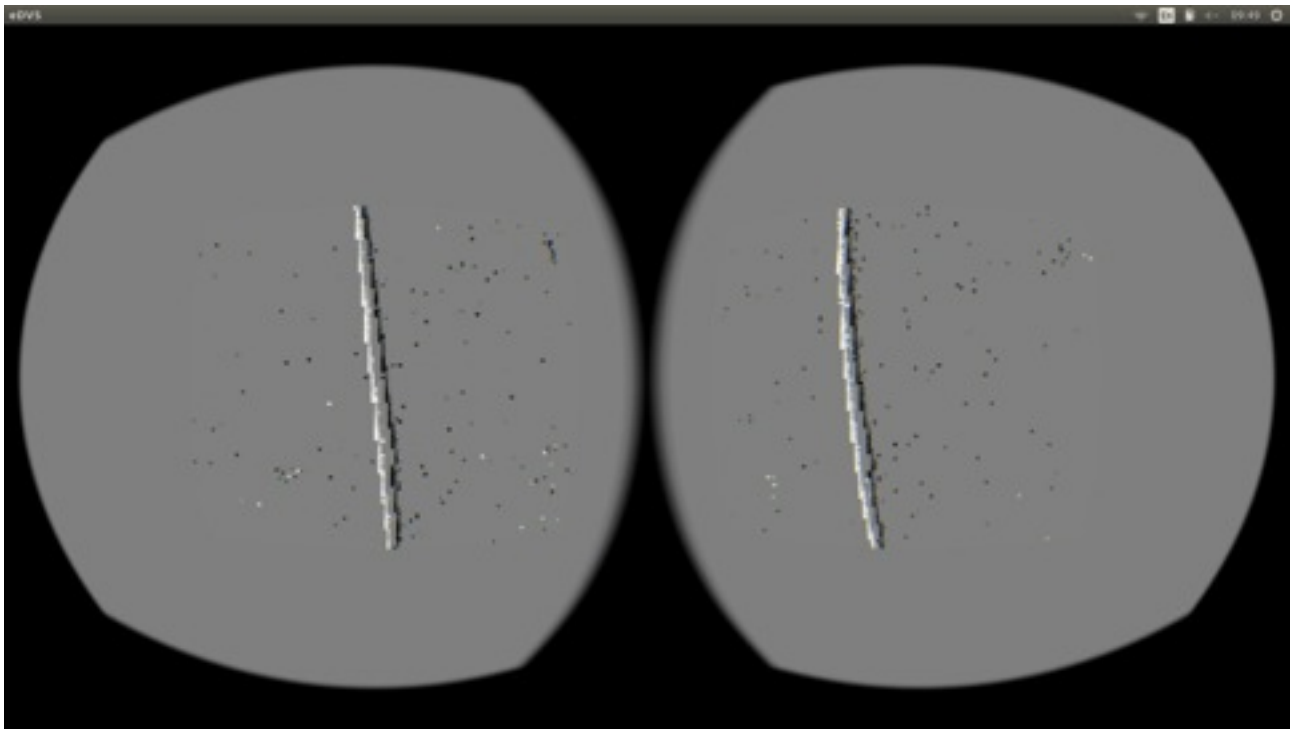

FIGURE 4.1 AN EXAMPLE OF THE EXPERIMENT FOOTAGE

After that, the subjects were asked in which of the footages the line was closer (or at the same distance). Among total 20 trials only 3 of them were misjudged. This occurred when the line located at almost the same distance.

Detailed results of the experiment can be found in Appendix A.

4.2 Minimal Requirement of Information Determination

The initial target of the project was to determine the minimal requirement of the information for a human being to navigate safely and correctly in unknown environments. Given that the portability of the system was limited by the power supply and the PC, a thorough and systematic experiment could not be performed at the moment. Instead, we did some local tests to verify subjects' capability of completing everyday tasks, including picking up a mobile phone on the desk and tossing paper into trash can.

The core parameter that would be continually readjusted during the tests was the update interval of the events. As in Section 2.2 described, eDVS events were caused by movement. In a dynamic scene, the same pixel can be triggered multiple times within several milliseconds. These events might be redundant for the navigation. In our program, events will be filtered by a parameter called update interval first. This parameter controls after how long the already triggered pixel can be renewed again. Repetitive events of the same pixel occurred in this time interval will be ignored.

The optimal value of the update interval turned out to be around 1000 microseconds according to our tests. Under this value, the result of the tests were positive. With 5-10 minutes' customization and adaption of the system, most participants were able to complete these tests flawlessly.

Chapter 5: Summary of the Project

With the practical training, I've learned how to apply OpenGL to visualize data, got familiar with the basics of Oculus Rift and eDVS, and most importantly learned how to work with others as a team.

After nine weeks of work, the major part of the project is accomplished. The Oculus Rift Retina Display system can now present dynamic scenes with adjustable parameters and simulate the visual experience of the patients implanted with retinal prothesis. Without the help of my colleague Mr. Zhejun Zhang, our supervisors and the helping hands from other students, it can't be possible to achieve this in such a short period.

However, there is still much room for improvement.

A portable solution of the whole system should be provided. For instance, with wireless data transmission and portable battery as power supply. Thus, a more thorough and systematic experiment can be performed.

The stereo vision is a major issue in this project. Although the experiment gave positive results, we can still not rule out the possibility that the size of the object played a role in the sense of depth during the experiment. Furthermore, we found that the fusion of stereo vision only worked in a certain range of distance, probably due to the fixed distance between the two eDVS.

Still, I'm convinced that this system can help the neuroscientists in the future development of retinal prothesis with the mentioned improvement above.

Appendix A: Experiment Results for Stereo Vision

Distance A (cm)	Distance B (cm)	Correct Judgement
47.5	60	YES
85	97.5	YES
72.5	72.5	NO
110	85	YES
60	110	YES
72.5	97.5	YES
85	97.5	YES
110	47.5	YES
60	72.5	YES
110	122.5	YES

TABLE A.1: EXPERIMENT RESULTS OF SUBJECT 1

Distance A (cm)	Distance B (cm)	Correct Judgement
60	85	YES
97.5	72.5	YES
72.5	85	YES
122.5	110	YES
60	72.5	YES
110	85	YES
47.5	60	NO
122.5	97.5	YES
85	85	NO
110	97.5	YES

TABLE A.2: EXPERIMENT RESULTS OF SUBJECT 2

Appendix B: Sources for Figures

Figure 2.1: Field of View Oculus Rift:

https://developer.oculusvr.com/forums/viewtopic.php?f=26&t=11505

Figure 2.2: Event Based Dynamic Vision System:

https://wiki.lsr.ei.tum.de/nst/programming/edvsgettingstarted

Figure 2.3: Rendering Pipeline of OpenGL:

http://en.wikipedia.org/wiki/OpenGL

Bibliography

Forsyth, D. A., & Ponce, J. (2012). Stereopsis Computer vision: a modern approach (Second ed., pp. 197-198): Prentice Hall Professional Technical Reference.

Lang, B. (2012). Early Oculus Rift Specifications and Official Site, Confirms $500 Target. 2014, from http://www.roadtovr.com/early-oculus-rift-specifications-and-official-site-confirms-500-target/

Oculus VR, I. (2014). Oculus SDK C API Overview (SDK Version 0.3.1 Preview). 5. http://static.oculusvr.com/sdk-downloads/documents/Oculus_SDK_Overview_0.3.1_Preview.pdf

Piyathaisere, D. V., Margalit, E., Chen, S.-J., Shyu, J.-S., D'Anna, S. A., Weiland, J. D., . . . Kim, S. Y. (2002). Heat effects on the retina. Ophthalmic surgery, lasers & imaging: the official journal of the International Society for Imaging in the Eye, 34(2), 114-120.

Shreiner, D., Sellers, G., Kessenich, J. M., & Licea-Kane, B. M. (2013). Introduction to OpenGL OpenGL programming guide: The official guide to learning OpenGL, version 4.3 (Eighth ed., pp. 1-12): Addison-Wesley Professional.