Code Visualization using Virtual Reality

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Anurag Shah (B16CS034) **Chinmay Garg** (B16CS041)

under the guidance of

Dr. Sumit Kalra

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY JODHPUR
JODHPUR - 342001, RAJASTHAN

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Code Visualization using Virtual Reality**" is a bonafide work of **Anurag Shah** (**Roll No. B16CS034**) and **Chinmay Garg** (**Roll No. B16CS041**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Jodhpur under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Sumit Kalra**

Assistant Professor,

November, 2018

Department of Computer Science & Engineering,

Jodhpur.

Indian Institute of Technology Jodhpur, Rajasthan.

Acknowledgements

We wish to sincerely thank our mentor, Dr. Sumit Kalra sir for his unconditional and constant help and support. Whether it be helping us to understand the project, procurement of the VR headset, or guiding us through the process, he was always there.

Contents

List of Figures

Chapter 1

Introduction

1.1 Objective

The objective of this documentation is to provide the specific information on what are the services provided by the project Code Visualization using Virtual Reality, under what constraints it will work, and how it will interact with the users.

This report is primarily for the the end-users but the developers can also refer this document for better insight of the software.

1.2 Scope

The software can be used by any C++ programmer for the debugging of various run-time and logical errors. The software specifically targets the beginners in C++ who develop not so large programs and are not so familiar with the process of debugging using gdb.

1.3 Constraints

1. The software can only debug programs written in the C++ language.

2. No breakpoints can be placed by the user.

3. User have to manually update the program file in the app folder present in the Windows PC device, every time it is updated or changed.

1.4 Assumptions

1. Compile time errors must be resolved prior to uploading it to the software folder present in the Windows PC device.
2. All files part of the software package are downloaded and installed, for both Windows PC device and the Linux PC device.
3. Script which comes along with this software is already present on the Linux PC device.
4. Script is executed and all the required files are obtained and all these files are uploaded to the Windows PC device.

1.5 Dependencies

1. Computer runs on the Linux operating system.
2. Windows version is sufficient to execute VR app.
3. GNU GDB is installed on the system.

Chapter 2

Literature Survey

For this project, we learned the use of the Unity3d software, C# language, GNU GDB software, Android Studio software, Time command utility present in Linux and bash scripting in Linux.

For this we went through the documentations of these various softwares and video tutorials available on the internet regarding the use of these softwares.

2.1 Unity3d: Version 2018.2.7f

Unity is a cross-platform real-time engine developed by Unity Technologies. Unity gives users the ability to create games in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality.

We learned basically everything from scratch for this software. It was our first time using this platform for developing applications. Major things we used in this software were: UI elements, C# scripts, file handling, FPS control etc.

2.2 GDB: GNU Project Debugger

GDB is a portable debugger that runs on many Unix-like systems and works for many programming languages. GDB offers extensive facilities for tracing and altering the execution of computer programs. The user can monitor and modify the values of programs' internal variables, and even call functions independently of the program's normal behavior.

We learned how to use GDB for debugging in Linux, writing scripts for GDB, logging output of GDB, etc.

2.3 Time command in Linux

This is a useful command for developers who want to test the performance of their program or script. It has a large number of formatting options like: elapsed time, number of page faults, CPU usage, etc.

It was our first time coming across such a utility command and using it.

Chapter 3

Product Description

3.1 About

Debugging is the process of identifying and removing bugs from a program. It has now become an inseparable part of the process of programming. Thus, it is taught compulsorily in every engineering course along with the basic programming skills course. Our project intends to implement this debugging procedure in a virtual space using the virtual reality technologies available nowadays.

Through virtual reality, we will be creating a three-dimensional, computer generated environment which can be explored and interacted with by the user. This virtual environment will be having multiple screens displaying various data regarding the code and its debugging.

This will help users to visualize the procedure of debugging effectively and efficiently, hence, increasing the understanding of the procedure and the code itself. A virtual space also provides a much larger view-able screen, when compared to other conventional computer desktop screens. It has been many a times seen that programmers work with multiple desktops to increase the ease and comfort while debugging. A virtual space eliminates such requirements of multiple screens, as it provides a 360-degree view, in which we can place multiple virtual screens or a single 360-degree curved screen. Hence, this project is useful

for both students and programmers.

The software is also useful for those programmers who lack the knowledge and experience in debugging using the GDB software. It also provides a hands-free experience with no-user input intervention, once the process is started in the application.

Our project also aims to provide the users with additional useful and relevant information regarding the code and the debugging procedure that is not generally provided by other debugging softwares.

3.2 Development

The project was developed on two sides: i) Linux and ii) Unity.

3.2.1 Linux Side

On Linux, we wrote a script, named 'script.sh', to compile and debug the C++ program, named 'main.cpp'. This script also implements the time command and retrieves other relevant information regarding the process, storing them in a file named 'time.txt'.

The script also executes a C++ program named 'func_finder.cpp'. This program finds all the functions and stores them in a file named 'func_list.txt'. Later this file will be used to map all these functions inside the Unity software, to display them on one of the screens.

All the GDB commands are stored in a file named 'inputgdb.txt', which is actually a GDB script. This script makes successive calls for the 'step' and 'info locals' functions. This 'info locals' function prints all the local variables present in that current frame, along with their values.

For the GDB process, pagination is set to off, so that irrelevant information is not displayed, and logging is set to on, so that the output is stored in a file named 'gdb.txt'. The breakpoint is set at main().

Files that we will be required to be sent to the Windows PC application will be: 'main.txt', 'gdb.txt', 'func_list.txt', and 'time.txt'.

*script.sh

```
/usr/bin/time -o time.txt -f
"\nCPU seconds used by user = %U
\nCPU seconds used by kernel = %S
\nInputs = %I
\nOutputs = %O
\nPage Faults = %F
\nTotal Memory used (KB) = %K
\nCPU percentage used = %P \n"
/bin/sh -c '>gdb.txt;g++ -g main.cpp;gdb a.out < inputgdb.txt'
#!/bin/bash
g++ func_finder.cpp
./a.out main.cpp > fun_list.txt
cp -- "main.cpp" "main.txt"
```

Fig. 3.1 The script.sh file.

inputgdb.txt

```
set pagination off
set confirm off
set logging on
break main
run
while(1)
info locals
s
end
set logging off
```

Fig. 3.2 The inputgdb.txt file: Stores the GDB Script.

time.txt

```
CPU seconds used by user = 1.50
CPU seconds used by kernel = 2.90
Inputs = 0
Outputs = 0
Page Faults = 0
Total Memory used (KB) = 0
CPU percentage used = 31%
```

Fig. 3.3 The time.txt file: Stores the information obtained from the time command

3.2.2 Unity Side

We have two Game Scenes in unity, named 'menu' and 'main'. In 'menu' Game Scene, we have a single UI Canvas. On this UI Canvas, a UI Panel is placed. Over this UI Panel, two GameObjects are overlayed, which we have named 'MainMenu' and 'AboutScreen'. 'MainMenu' has three UI Buttons, named 'START', 'ABOUT', and 'QUIT'. The 'START' button is linked with the other Game Scene which is named 'main'. The 'ABOUT' button is linked with the 'AboutScreen' GameObject. And the 'QUIT' button exits the user from the app. Inside the 'AboutScreen' GameObject, we have a brief description of the app written on UI Text element and a UI button named 'BACK', which is linked with the 'MainMenu' GameObject and brings us back to the homescreen.

Now lets go to the 'main' Game Scene. It has three UI canvases inside one main UI canvas. The three UI canvases are used to display the whole code, display the present function in execution and the debugging information. Each canvas has a UI Panel in which we have 'ScrollRect' object, 'UI Text', 'Verticle Scrollbar' and 'Horizontal Scrollbar' components. ScrollRect is used to display the code or text in a scrollable way. UI Text is the component which allows us to display a proper formatted C++ code. Vertical Scrollbar and Horizontal Scrollbar provide us a handle which is used to move or scroll different parts of code in the UI Text. The canvases are interlinked with each other using C# scripts. We have a UI Slider which is used to maintain the speed of execution. Drag the slider to top to increase the speed and to the bottom, to stop the execution. We are also showing the time taken for execution, CPU Usage and Memory usage of the code.

We are using FPSController (First Person Controller) to show the Virtual space. It enables the user to view the space as first person and provides with 360 degree view of the virtual space.

We attached a C# script with each object which allows us to make the objects/components communicate with each other. Also all the events like detecting pressing of a button and change in the slider value are detected and handled using C# scripts and appropriate action

is taken to handle the events.

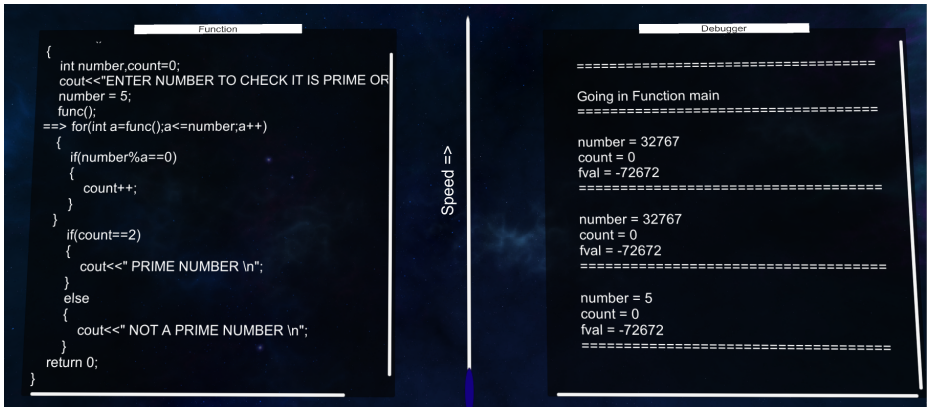

Fig. 3.4 'Function' and 'Debugger' screens with the speed slider between them.

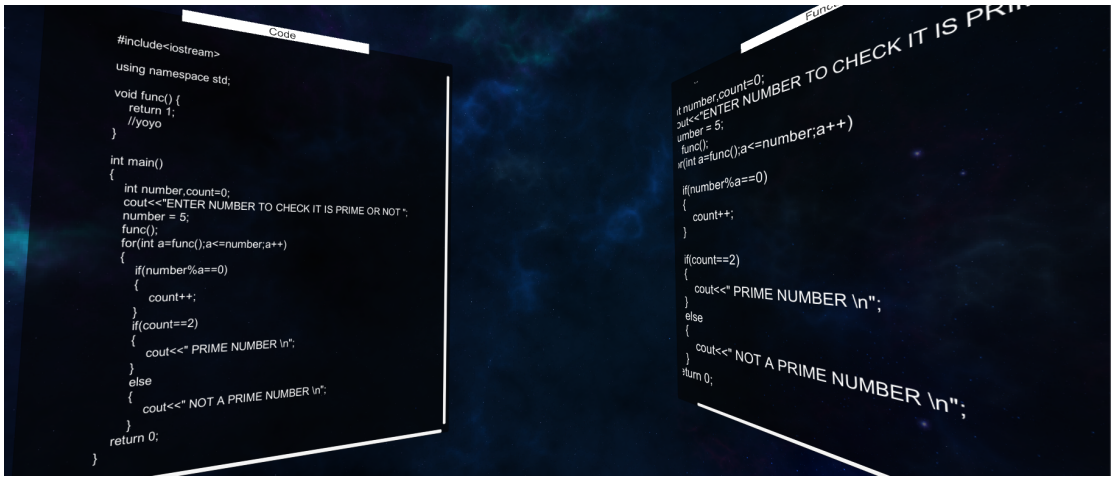

Fig. 3.5 'Code' and 'Function' screens.

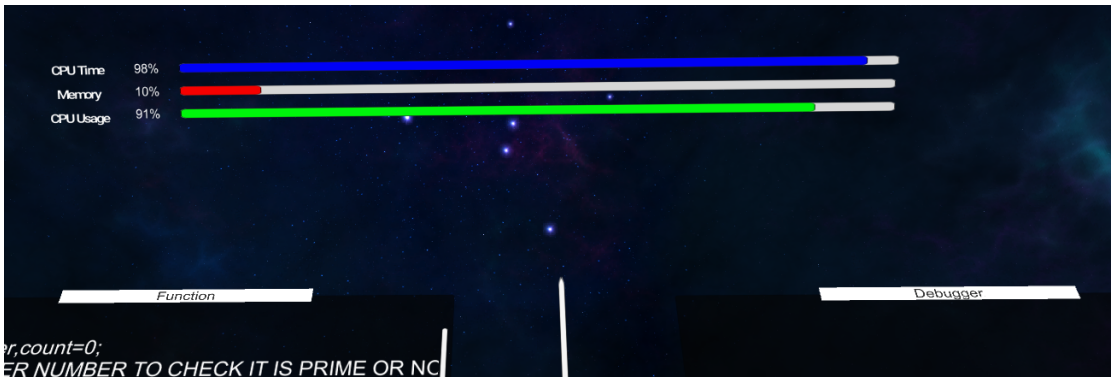

Fig. 3.6 Sliders depicting info obtained from 'time command' placed over the two screens.

3.3 User Guide

Steps to be followed on Linux Device:

1. Rename the C++ program that you want to debug to 'main.cpp'.
2. In the same folder that contains the C++ program, download the files: 'script.sh', 'inputgdb.txt', and 'func_finder.cpp'.
3. In the terminal, execute the command:

   bash script.sh
4. Now move these files to the folder corresponding to our application in the Windows PC device: 'main.txt', 'gdb.txt', 'func_list.txt', and 'time.txt'.

Steps to be followed on the Windows PC Device:

1. Start the application.
2. On the homescreen of the application, select the 'START' button.
3. Now you can see a virtual space, which have three screens and several sliders. The three screens are named: Code, Function and Debugger. There are also three sliders for depicting CPU Time, Memory, and CPU Usage.
4. The debugging process will automatically start. But, you can change the speed of execution suitable to your need using the slider provided in between the 'Function' and the 'Debugger' screens.
5. On the 'Code' screen, you can see the entire code.
6. On the 'Function' screen, the current function, in which the marker is currently at, is visible. There is also an arrow '==>' in front of the current line in execution. Position of this arrow updates as the execution of code proceeds.

7. On the screen named 'Debugger', the local variables of the current frame in execution and their values after the current line is executed are printed. A history is maintained, so that you can see the previous values by scrolling this screen.

8. After the debugging completes, you can exit the app by selecting the 'QUIT' button.

Chapter 4

Challenges Faced

This project was full of challenges. It had not been easy creating this Windows PC application and some of the challenges we faced are:

1. This was the pioneer project done on the Virtual Reality technology, done in our institute. None of our seniors or batchmates had done anything in this field. Even most of us were seeing a Virtual Reality headset for the first time in our life.

2. Although there are many VR based games available on the internet, there are almost no VR based utility apps present. Hence, there are also no tutorials or documentations available guiding the development of a VR based utility app. For every single thing, we had to relate the process with the game development, which is an extremely different field, or had to go with our intuitions.

3. The cloud services provided by Unity Technologies is very poor. Even after trying for multiple times, we were unable to collaborate on a project through Unity. Finally, we had to do the project on one PC, taking turns, reducing our efficiency. This also made using cloud services for our app impossible. Also Unity does not provide a Firebase Assistant, unlike Android Studio.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The android application was working smoothly, increasing the effectiveness and efficiency while debugging a C++ code.

Advantages we receive from this software are:

1. Increased display area and multiple screens on a single device.
2. Hands-free process.
3. No pre-requisite of learning GDB.
4. User suitable speed for the process.

5.2 Future Works

Some Future changes that can be brought to this application are:

1. Use of Augmented Reality technology, instead of Virtual Reality technology.
2. Real-time update of main.cpp code, removing the need of manual moving of files from Linux device to Android.

3. Linking the software to a cloud service, so that multiple codes can be stored in the cloud and then can be selected from the app.

4. Extending it to Android Devices using services like Google Daydream etc.

Chapter 6

References

1. Brackeys(Nov 29, 2017). Start Menu in Unity.
   Retrieved from https://youtu.be/zc8ac_qUXQY

2. Brackeys(Oct 5, 2014). Content Scrolling.
   Retrieved from https://www.youtube.com/watch?v=MWOvwegLDl0

3. xOctoManx(Jun 26, 2015). Sliding in Pause Menu.
   Retrieved from https://www.youtube.com/watch?v=f5Eg3t3glWs

4. Unity Technologies. (Publication: 2018.2-002M. Built: 2018-11-21).
   Retrieved from https://docs.unity3d.com/ScriptReference/

5. User Interface(UI) Unity. (2018 Unity Technologies).
   Retrieved from https://unity3d.com/learn/tutorials/s/user-interface-ui

6. GNU GDB Debugger Command Cheat Sheet
   Retrieved from http://www.yolinux.com/TUTORIALS/GDB-Commands.html

7. Newel, Garry (Aug 18, 2018). Get Return Time Statistics With the Linux Time Command.
   Retrieved from https://www.lifewire.com/command-return-time-command-4054237