Multi-Threaded Geometric Rendering

Project Report

of Major Project

Bachelor of Technology
(Computer Science and Engineering)

Submitted by:
Amarjeet Singh Kapoor
D4CSE 2013-17
135005
1311017

Govind Sharma
D4CSE 2013-17
135026
1311053

Guru Nanak Dev Engineering College
Ludhiana 141006

Abstract

OpenSCAD is an open source organization that serves a free software to create solid 3d CAD objects. OpenSCAD has in a way redefined how easy 3D modeling can be. But the Wikipedia article on OpenSCAD says that it is a non-interactive modeler, but rather a 3D compiler based on a textual description language.

In openSCAD when the object is described using the textual description language, the description is first compiled (following intermediate forms) into a node tree. This node tree consists, in geometric terms, the details of every component of the final design. Then follows the rendering process. This process is very CPU intensive and eats up a lot of cycles. This involves traverssing the node tree from the very bottom and evaluating nodes as they are encountered. Each evaluated node is then rendered on the screen.

In its present form, the compile and render processes are done in one thread under the one process. This is of course not the most optimal ways of going about this problem.

Adding multiple processor threading support to the compile and render feature will ensure that the system is run on 100 percentage on all available cores/ht rather than just the one.

This issue may be clear in intent but involves many complex considerations. One cannot just start implementing multithreading on these processes. The libraries which are being used for the rendering process are not thread safe (CGAL). Hence it becomes very difficult to implement this directly. Our project is as much about implementing a solution to this as it is about looking for one. Finding the best possible approach for achieving this is one big milestone that needs to accomplished before begining to realising that approach.

One of the original ideas was to create a dependency graph (as in a makefile) for the tree, and then traverse the graph, to keep the original tree const and to avoid multi-threaded access to the node tree. This seems a promising approach but remains to explored thoroughly.

Acknowledgement

We, students of Guru Nanak Dev Engineering College, Ludhiana, have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

The authors are highly grateful to Dr. M.S. Saini Director, Guru Nanak Dev Engineering College, Ludhiana for providing them with the opportunity to carry out their Major project.

The author would like to wholeheartedly thank Amanpreet Singh Brar, Associate Professor, at Guru Nanak Dev Engineering College, Ludhiana who is a vast sea of knowledge and without whose constant and never ending support and motivation, it would never have been possible to complete the project and other assignments so efficiently and effectively.

Finally, we would like to thank our mentors at OpenSCAD organization Marius Kintel and Torsten Paul. Without their encouragement and Guidence, it would not have been possible to complete this project in such an efficient manner.

Amarjeet Singh Kapoor & Govind Sharma

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

Multi-threaded compile and render for OpenSCAD is our major project. It is under the umbrella organization of BRL-CAD. OpenSCAD is a free and Open-source software application for creating solid 3D CAD objects. It is a script only based modeler, with a specific description language. Parts cannot be selected or modified by mouse in the 3D view. An OpenSCAD script specifies geometric primitives and defines how they are modified and manipulated to render a 3D model. OpenSCAD is available for Windows, Linux and OS X. It does constructive solid geometry (CSG).

OpenSCAD has in a way redefined how easy 3D modeling can be. But the Wikipedia article on OpenSCAD says that it is a non-interactive modeler, but rather a 3D compiler based on a textual description language. Pay attention to the above line, its primarily what Ill be talking about.

Solid 3D modeling. That sounds like some serious business. But its just an awesome tool for making models pertaining to many uses (mostly 3D printing). And 3D printing as we can all agree upon is cool. 3D models can be created by anyone using OpenSCAD. OpenSCAD is as much for designers as it is for you and me. What else can most people agree upon apart from the fact that solid 3D modeling is cool? A graphical interface is simpler and more intuitive to use. There is a general aversion for typing commands in order to get things done. Simply put, more people have an inclination towards GUI. Another concern is speed at which the results are presented before the user. It would be a waste of CPU capacity if not all of it is being used in the compilation and rendering of the models created in OpenSCAD. This can be achieved by spliting the process into multiple threads.

Since the rendering of various models internally done using the CGAL library, it becomes very important to check how the library behaves on a multi-threaded approach. This is something that will be considered before actually starting the implementation.

1.2 Project Category

OpenSCAD is a free and Open-source software application for creating solid 3D CAD objects. It is under the umbrella organisation of BRL-CAD. Since our project comes under OpenSCAD, it can be classified as being an application development project. But as much as it is an application development project, it is also a research project. It took a fair share of observation and study in order to find the right approach the problem. The majority of the time was spent in figuring out how the internals of this huge piece of software work. The study involved learning about various mathematical constructs used in the geometry of the models. Researching the computer graphics

part of the sotware was also very essential in order to form a stronger understanding of how the problem in hand is to be tackled. As mentioned before, the CGAL library (which is used to render the models) is not entirely thread safe. So it bacame very crucial to analyse what parts of it are being used in the render process and how these parts actually work. Upon researching the above mentioned things, the right way was to be decided upon. The project essentially changes the design approach of an important segament of OpenSCAD i.e. rendering. So of course it required great consideration before implementing it.

1.3 Objectives of the Project

Objective of this project are following:

- Explore different OS independent ways of parallizing the evaluation.
- Support for multi-threaded evaluation, possibly with some limitations to handle non-thread-safe library calls
- Thread safe cache infrastructure

1.4 The Existing System

OpenSCAD in its present form applies the compile and render process sequentially in one thread of a process. The whole process begins when the user write (or completes) the textual description of their model and chooses to either just compile it or complie and render it one go (F6). The textual description of the language is taken through the following stages sequentially.

1. The descriptional language is scanned, tokenized and converted into an Abstract Syntax Tree (AST). This is much like what any compiler does initially.
2. The AST is converted into an Abstract Node Tree. This consists of the nodes that are to become the part of the final model.
3. The node tree is fed to the CSGEvaluator which generates a preview of the model abstracting finer details.
4. The final rendering of the model is done by the GeometryEvaluator.

The present system is a bit slow in terms of its usage of the available CPU power. In being a single threaded process, everything has to happen in a sequence. This is not something which is required all the time. For example, during the evaluation of the node tree, various nodes can be

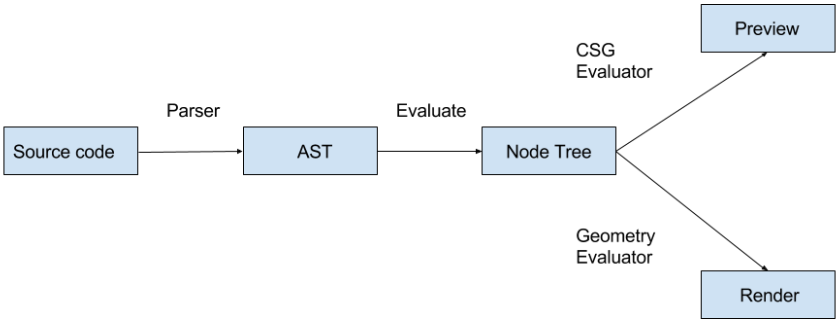

Figure 1.1:

evaluated in parallel in separate threads. Similarly at the time of rendering, this parallelism can be achieved.

Limitations of previous system

- Not utilizes the full potential of the CPU.
- The rendering process is very slow.
- When the user accidentely presses F6, there is no way to cancel that process and the user has to wait for the rendering to finish in order to continue their work.
- The system hangs frequently for more complex models on normal computers.
- It uses large amount of system resources. Their can be a significant improvement in this.

CHAPTER 2

REQUIREMENT ANALYSIS AND SYSTEM SPECIFICATION

2.1 User Requirement Analysis

The main requirement is to some how paralleize the evaluation of nodes in the node tree. This can be achieved by either taking a multithreading approach or having two separate processes for compile and render itself. The major concern is the non thread safety of CGAL library which needs to be accounted for. Another follow up requirement is to add a cancel mechnism to any ongoing render process without loosing context on already evaluated nodes in the tree.

2.1.1 Functional Requirements

- Parallel processing of Nodes.
- Maintaining thread safety of CGAL.
- Sustaining caching of previosuly evaluated nodes to improve speed during repeated renders of the model.
- Introducing a halt mechanism on ongoing render processes.
- Safe exit from compilation.

2.1.2 Non-functional requirements

1. Extensible: It should be able to support future functional requirements
2. Usability: Simple user interfaces that a layman can understand.
3. Modular Structure: The software should have structure. So, that different parts of software would be changed without affecting other parts.
4. Backward compatibility: Addition of new syntax should not forbid script to work correctly on the backward versions of OpenSCAD.

2.2 Feasibility

Feasibility study aims to uncover the strengths and weaknesses of a project. These are some feasibility factors by which we can used to determine that the project is feasible or not:

- **Technical feasibility**: Technological feasibility is carried out to determine whether the project has the capability, in terms of software, hardware, personnel to handle and fulfill the user requirements. This whole project is based on Open Source Environment and is part of an open source software which would be deployed on any OS.

- **Economic feasibility**: OpenSCAD's multi-threaded compile and render is also Economically feasible with as It could be developed and maintain with zero cost as It is supported by Open source community. Plus This project is started with no intention of having any economic gain but still there is an option for donations. Economic Feasibility which can be categorized as follows:

    1. Development costs.
    2. Operating costs.

2.3 Design Approach

Object Oriented design approach has been used for this project. This statement may seem simple and succint but it captures, perfectly, all the complexities and intricacies of the whole OpenSCAD software and our project component in particular. It was not of course a free choice for us as we had to adopt to the design principles already in place for this software. And for a project of this size the answer was always going to be object oriented design. Conventional form of objected orientation has been used in bulk throughout the code base of this software along with the usage of many design patterns. Out of these, we had to encounter with a few in our specific working component. The most notable example being that of the visitor design pattern. We had to study it and observe its usage inside of the software in order to alter it and build upon it. So we did.

2.4 Product Perspective

This product is supposed to be part of an open source project, under the GNU general public license. It is a CAD software for programmers i.e. you code to make models in 3D. Rendering is the process of turning the model created in openSCAD into something the user is able to see on the screen. This is a complex process and is done using special libraries. Since the rendering process is a complex one and often the models created by the users are of great size and detail, the time consumed by the rendering process is not brief to say the least. This may be acceptable when the user actually wants to finally render their model and see the results after they are done with their script and are satisified with the result. But when the user just wants to have a quick look at how the model is turning out or even if the render command is given by accident, the time it takes for the rendering to complete can be very problamatic. That is where the idea of adding multithreading to the whole process came into fruition. It makes sense that for doing such cpu intensive computations, the software is able to use all the cpu power available in the machine on which it is running. This was not the case. And doing so will also ensure that valuable time is saved in the rendering process. If the redering is done via multithreading, it will also allow us to control when the rendering process is to be terminated and how. This will allow us to cancel the process right in its tracks if we do not wish to continue with it. The following are the main features included in the OpenSCAD's Multithreaded Geometric Rendering:

- **Cross platform support: It is able to offer operating support for most of the known and commercial operating systems in form of binaries and also it can be compiled on the platforms.**
- **Backward compatible: OpenSCAD should be backward compatible even after new features have been implemented.**

- **No change in user interaction: There is no change in which the user interacts with the software or the way in which the code is written by user in order to create models.**
- **Increased speed: The speed with which rendering of the model is done will improve based on the number of cores of processors available in the machines.**
- **Increased efficiency: Gain in speed is just a fruit of the actual increase in performance. This is obvious as the cpu idle time is reduced and resources are used to their fullest.**
- **Ability to halt the process: The rendering process can be halted in between by pressing the cancel button.**

2.5 Product Functions

Our project current augments the following features to OpenSCAD:

1. **Multi-threaded geometric rendering: The process of rendering which includes traverssing and then evaluating each node in the abstract node tree is tweaked to work using multiple threads all connected to one another in a hierarchical manner mimicing the original tree. All of this is done in order to ensure that work load is divided between different processor cores in the machine.**
2. **Cancellation of the rendering process: Once the rendering process has been started it should be cancellable from a button in the proggress widget for the rendering process.**

2.6 User Characteristics

We have identified three potential classifications of users of our system:

1. Designers: Designers are the people how the code to create model theirs for their own use or for commercial use.
2. The Client: These are the people which will customzie model to their need made by modelers before ordering or printing model themselves.
3. Developers: These are people who might want to integrate this new feature of OpenSCAD into their systems.

2.6.1 The General User

All users can be assumed to have the following characteristics:

1. Ability to read and understand English.
2. Familiarity with the operation of the basic Graphical User Interface (GUI) components of OpenSCAD.
3. Beyond the above, no further facility with computer technology can be assumed.

2.6.2 Designers

The Designer can be assumed to have the following characteristics:

1. Basic Knowledge of OpenSCAD.
2. Basic Knowledge of Creating models.
3. Basic coding skills.
4. Optional experience of cmd-line.

2.6.3 Developers

The Developers can be assumed to have the following characteristics:

1. Basic Knowledge of programming.
2. Ability to program in cmd-line.

2.7 Flowchart

2.8 Dependency Graph

2.9 Class Diagrams

2.10 Dependencies

CHAPTER 3

IMPLEMENTATION TESTING AND MAINTENANCE

3.1 C++

Figure 3.1: C++ Logo

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications (e.g. telephone switches or space probes). C++ is a compiled language, with implementations of it available on many platforms and provided by various organizations, including the Free Software Foundation (FSF's GCC), LLVM, Microsoft, Intel and IBM.

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2014 as ISO/IEC 14882:2014 (informally known as C++14). The C++ programming language was initially standardized in 1998 as ISO/IEC 14882:1998, which was then amended by the C++03, ISO/IEC 14882:2003, standard. The current C++14 standard supersedes these and C++11, with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Bjarne Stroustrup at Bell Labs since 1979, as an extension of the C language as he wanted an efficient and flexible language similar to C, which also provided high-level features for program organization.

Many other programming languages have been influenced by C++, including C#, D, Java, and newer versions of C (after 1998).

Features of Language:

1. Object storage
    (a) Static storage duration objects
    (b) Thread storage duration objects
    (c) Automatic storage duration objects
    (d) Dynamic storage duration objects
2. Templates
3. Objects
    (a) Encapsulation
    (b) Inheritance
4. Operators and operator overloading
5. Polymorphism
    (a) Static polymorphism
    (b) Dynamic polymorphism
        i. Inheritance
        ii. Virtual member functions
6. Lambda expressions
7. Exception handling

3.2 Qt

Figure 3.2: Qt Logo

Qt is a cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms with little or no change in

the underlying codebase, while still being a native application with native capabilities and speed. Qt is currently being developed both by The Qt Company, a company listed on the Nasdaq Helsinki Stock Exchange and the Qt Project under open-source governance, involving individual developers and firms working to advance Qt. Qt is available with both commercial and open source GPL 2.0, GPL 3.0, and LGPL 3.0 licenses.

Qt is used mainly for developing application software with graphical user interfaces (GUIs); however, programs without a GUI can be developed, such as command-line tools and consoles for servers. An example of a non-GUI program using Qt is the Cutelyst web framework. GUI programs created with Qt can have a native-looking interface, in which case Qt is classified as a widget toolkit.

Qt uses standard C++ with extensions including signals and slots that simplify handling of events, and this helps in development of both GUI and server applications which receive their own set of event information and should process them accordingly. Qt supports many compilers, including the GCC C++ compiler and the Visual Studio suite. Qt also provides Qt Quick, that includes a declarative scripting language called QML that allows using JavaScript to provide the logic. With Qt Quick, rapid application development for mobile devices became possible, although logic can be written with native code as well to achieve the best possible performance. Qt can be used in several other programming languages via language bindings. It runs on the major desktop platforms and some of the mobile platforms. It has extensive internationalization support. Non-GUI features include SQL database access, XML parsing, JSON parsing, thread management and network support.

3.3 Introduction to LaTeX

LaTeX, I had never heard about this term before doing this project, but when I came to know about it's features, found it excellent. LaTeX (pronounced /letk/, /letx/, /ltx/, or /ltk/) is a document markup language and document preparation system for the TeX typesetting program. Within the typesetting system, its name is styled as LaTeX.

Figure 3.3: Logo of LaTeX

Within the typesetting system, its name is styled as LaTeX. The term LaTeX refers only to the language in which documents are written, not to the editor used to write those documents. In order to create a document in LaTeX, a .tex file must be created using some form of text editor.

While most text editors can be used to create a LaTeX document, a number of editors have been created specifically for working with LaTeX.

LaTeX is most widely used by mathematicians, scientists, engineers, philosophers, linguists, economists and other scholars in academia. As a primary or intermediate format, e.g., translating DocBook and other XML-based formats to PDF, LaTeX is used because of the high quality of typesetting achievable by TeX. The typesetting system offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout and bibliographies.

LaTeX is intended to provide a high-level language that accesses the power of TeX. LaTeX essentially comprises a collection of TeX macros and a program to process LaTeXdocuments. Because the TeX formatting commands are very low-level, it is usually much simpler for end-users to use LaTeX.

3.3.1 Typesetting

LaTeX is based on the idea that authors should be able to focus on the content of what they are writing without being distracted by its visual presentation. In preparing a LaTeX document, the author specifies the logical structure using familiar concepts such as chapter, section, table, figure, etc., and lets the LaTeX system worry about the presentation of these structures. It therefore encourages the separation of layout from content while still allowing manual typesetting adjustments where needed.

3.4 Introduction to Doxygen

Figure 3.4: Doxygen Logo

Doxygen is a documentation generator, a tool for writing software reference documentation. The documentation is written within code, and is thus relatively easy to keep up to date. Doxygen can cross reference documentation and code, so that the reader of a document can easily refer to the actual code.

Doxygen supports multiple programming languages, especially C++, C, C#, Objective-C, Java, Python, IDL, VHDL, Fortran and PHP.[2] Doxygen is free software, released under the terms of the GNU General Public License.

3.4.1 Features of Doxygen

- Requires very little overhead from the writer of the documentation. Plain text will do, Markdown is support, and for more fancy or structured output HTML tags and/or some of doxygen's special commands can be used.
- Cross platform: Works on Windows and many Unix flavors (including Linux and Mac OS X).
- Comes with a GUI frontend (Doxywizard) to ease editing the options and run doxygen. The GUI is available on Windows, Linux, and Mac OS X.
- Automatically generates class and collaboration diagrams in HTML (as clickable image maps) and LaTeX (as Encapsulated PostScript images).
- Allows grouping of entities in modules and creating a hierarchy of modules.
- Doxygen can generate a layout which you can use and edit to change the layout of each page.
- Can cope with large projects easily.

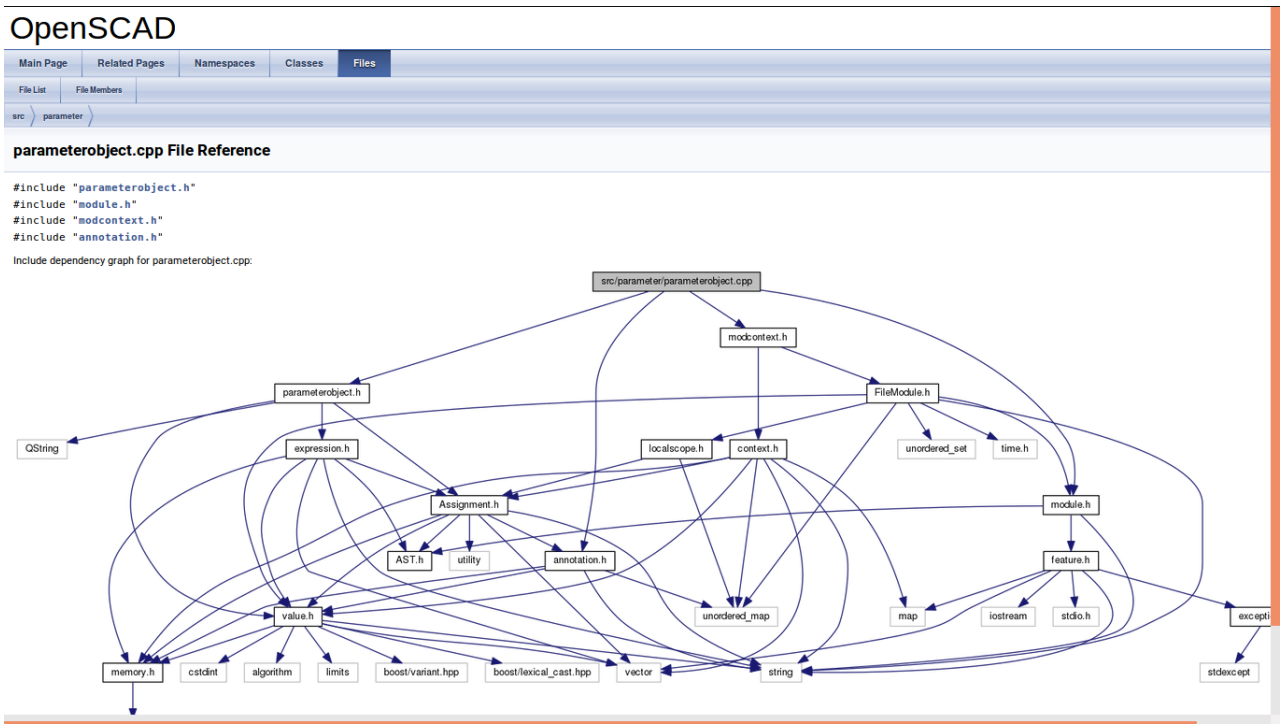

Figure 3.5: Documentation using Doxygen (main page)

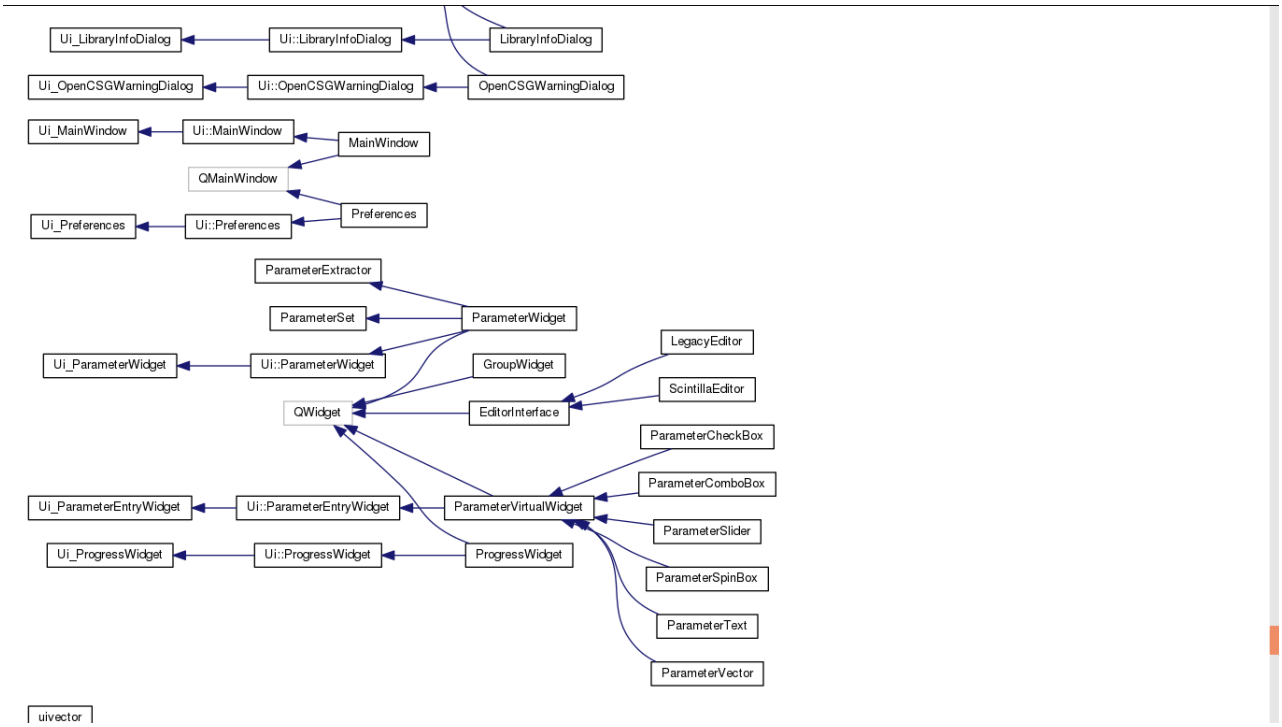

Figure 3.6: Doxygen documentation of a function

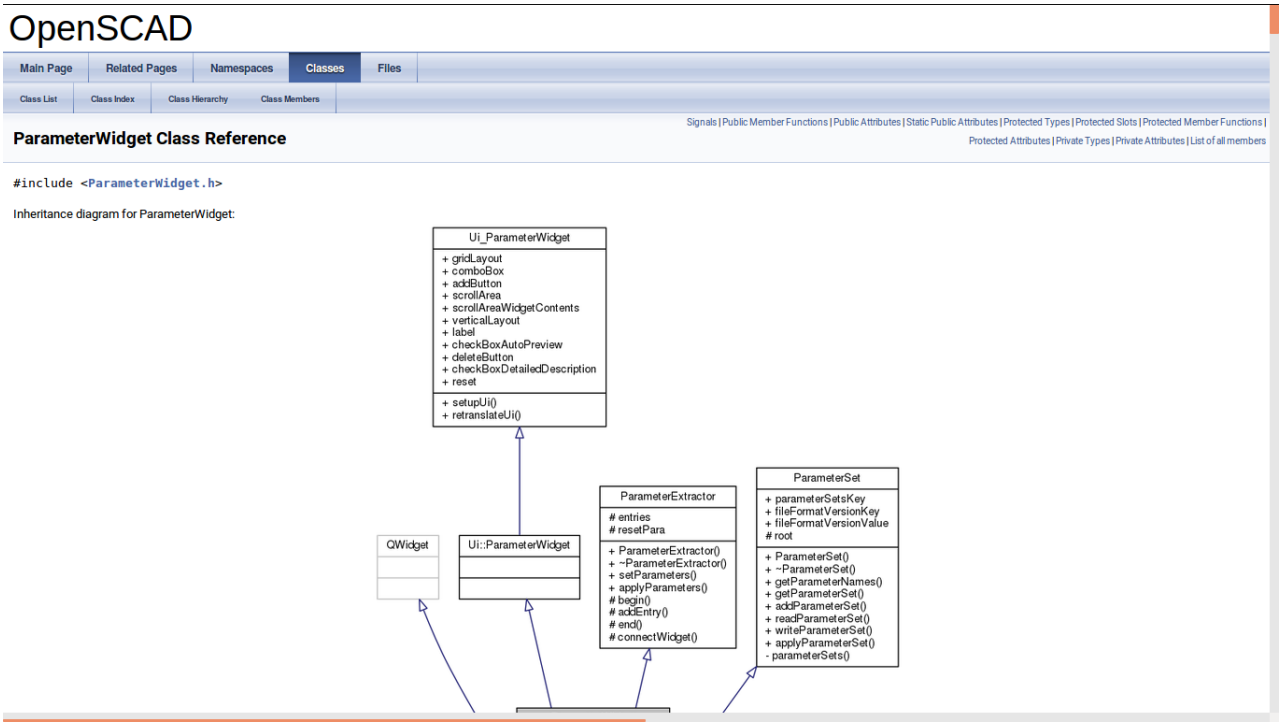

Figure 3.7: Documentation using Doxygen(list of files)

3.5 Introduction to Github

GitHub is a Git repository web-based hosting service which offers all of the functionality of Git as well as adding many of its own features. Unlike Git which is strictly a command-line tool, Github provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project.

This means that you can do things like:

- Frictionless Context Switching.
  Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
- Role-Based Code lines.
  Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
- Feature Based Work flow.
  Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
- Disposable Experimentation.
  Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the workwith nobody else ever seeing it (even if you've pushed other branches in the meantime).

Notably, when you push to a remote repository, you do not have to push all of your branches. You can choose to share just one of your branches, a few of them, or all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others.

3.5.1 What is Git?

Git is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since

Figure 3.8: Git Logo

become the most widely adopted version control system for software development.

As with most other distributed revision control systems, and unlike most clientserver systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free and open source software distributed under the terms of the GNU General Public License version 2 to handle everything from small to very large projects with speed and efficiency.

3.6 Implementation

3.7 Testing

CHAPTER 4

CONCLUSIONS AND FUTURE SCOPE

4.1 Conclusions

This project was something which required in-depth view in analysis of your problems. The main problem in this project was encountered during the research process. The sheer size of the software to be studied was daunting and required intense efforts to even begin to understand. And it was not possible to contribute anything without understanding the working of the whole process. This was because the project was not about adding a new feature but improving an existing central feature by almost completely changing its design. Time is always an eternal challenge. Completing such a task required one's undivided attention and it was certainly challenging to do it with other academic responsibilities. A great deal of things were learned while working on this project. The experience of working with an open source community is always wonderfull. We got to interact and exchange views with some of the technological leaders of this field during this project. The whole experience of working on this project and contributing to a few others has been very rewarding as it has given great opportunities to learn new things and get a firmer grasp on already known technologies. Here is a reiteration of some of the technologies I have encountered, browsed and learned:

- Operating system: Linux
- Language: C++
- Framework: Qt
- Software: Git, cTest, Doxygen
- Building Tools: qmake, CMake, make
- Communication tools: IRC

So during this project, we learnt all the above things. Above all, we got to know how software is developed and how much work and attention to details is required in building even the most basic of components of any project of this size. Apart from above we also learnt things like:

- Planning
- Designing
- Reading code

- Developing code
- Working on a team
- Testing standards
- Licensing constraints
- Working of open source community
- Writing readable code
- Coding standards

4.2 Future Scope

OpenSCAD being an open source project and supported by a large open Source community have a lot of scope for future improvements and additions as other individuals can also contribute in it and add additional functionality. Being an Open Source project there is a constant flow of suggestion and demands by people. Our project is no exception to that. Much of the work that has been done in this project can be scalled to some other components as well. The majority of this project works on top of a nodal tree. And this tree can have millions of nodes at times. Obviously there is immense scope for parallelisation. Of course not everything can be done in parallel, some due to pure algorithimic reasons and others because of liminations of libraries that are being used. OpenSCAD is heavy on CPU intensive work, and as is the case with computers in general, parallelisation is the future for openSCAD as well.

BIBLIOGRAPHY

[1] J. Levine, Flex & bison, 1st ed. Sebastopol, Calif.: O'Reilly Media, 2009.

[2] C. Donnelly and R. Stallman, Bison, 1st ed. Boston, MA: Free Software Foundation, 1999.

[3] "User:Amarjeet Singh Kapoor/GSoC2016/Project - BRL-CAD", Brlcad.org, 2016. [Online]. Available: http://brlcad.org/wiki/User:Amarjeet_Singh_Kapoor/GSoC2016/Project. [Accessed: 27- Nov- 2016].

[4] "User Interface for Customizing Models (Part 3)", amarjeetkapoor1, 2016. [Online]. Available: https://amarjeetkapoor1.wordpress.com/2016/08/17/user-interface-for-customizing-models-part-3/. [Accessed: 27- Nov- 2016].

[5] "OpenSCAD User Manual/WIP - Wikibooks, open books for an open world", En.wikibooks.org, 2016. [Online]. Available: https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/WIP#Customizer. [Accessed: 27- Nov- 2016].

[6] "openscad/openscad", GitHub, 2016. [Online]. Available: https://github.com/openscad/openscad/wiki/Project%3A-Form-based-script-parameterization. [Accessed: 27- Nov- 2016].

[7] "User Interface for Customizing Models (Part 2)", amarjeetkapoor1, 2016. [Online]. Available: https://amarjeetkapoor1.wordpress.com/2016/07/18/user-interface-for-customizing-models-part-2/. [Accessed: 27- Nov- 2016].

[8] "openscad/openscad", GitHub, 2016. [Online]. Available: https://github.com/openscad/openscad. [Accessed: 27- Nov- 2016].

[9] "Doxygen: Main Page", Doxygen.org, 2016. [Online]. Available: http://www.doxygen.org. [Accessed: 27- Nov- 2016].

[10] "OpenSCAD", En.wikipedia.org, 2016. [Online]. Available: https://en.wikipedia.org/wiki/OpenSCAD. [Accessed: 27- Nov- 2016].

[11] http://www.openscad.org/

[12] "MakerBot Customizer", Customizer.makerbot.com, 2016. [Online]. Available: http://customizer.makerbot.com/docs. [Accessed: 27- Nov- 2016].

[13] "User Interface for Customizing Models", amarjeetkapoor1, 2016. [Online]. Available: https://amarjeetkapoor1.wordpress.com/2016/07/04/user-interface-for-customizing-models/. [Accessed: 27- Nov- 2016].

[14] "OpenSCAD - News", Openscad.org, 2016. [Online]. Available: http://www.openscad.org/news.html#20160714. [Accessed: 27- Nov- 2016].

[15] "GNU", En.wikipedia.org, 2016. [Online]. Available:

https://en.wikipedia.org/wiki/GNU#/media/File:Heckert_GNU_white.svg. [Accessed: 27- Nov- 2016].

[16] "Qt (software)", En.wikipedia.org, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Qt_(software)#/media/File:Qt_logo_2015.svg. [Accessed: 27- Nov- 2016].

[17] "",Upload.wikimedia.org, 2016. [Online]. Available:

https://upload.wikimedia.org/wikipedia/commons/c/ce/Doxygen.png. [Accessed: 27- Nov- 2016].

[18] "Git", En.wikipedia.org, 2016. [Online]. Available:

https://en.wikipedia.org/wiki/Git#/media/File:Git-logo.svg. [Accessed: 27- Nov- 2016].

[19] "Qt (software)", En.wikipedia.org, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Qt_(software). [Accessed: 27- Nov- 2016].

[20] "C++", En.wikipedia.org, 2016. [Online]. Available: https://en.wikipedia.org/wiki/C%2B%2B. [Accessed: 27- Nov- 2016].

[21] "Travis CI", En.wikipedia.org, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Travis_CI. [Accessed: 27- Nov- 2016].

[22] "CMake/Testing With CTest - KitwarePublic", Cmake.org, 2016. [Online]. Available: https://cmake.org/Wiki/CMake/Testing_With_CTest. [Accessed: 27- Nov- 2016].