TU WIEN Informatics

A Holistic and Interactive 3D Visualization of Version-Controlled Source Code Repositories

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Maximilian Johannes Zenz, BSc
Matrikelnummer 01633058

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Thomas Grechenig

Wien, 21. August 2024

Unterschrift Verfasser

Unterschrift Betreuung

TU WIEN Informatics

A Holistic and Interactive 3D Visualization of Version-Controlled Source Code Repositories

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Maximilian Johannes Zenz, BSc
Registration Number 01633058

to the Faculty of Informatics

at the TU Wien

Advisor: Thomas Grechenig

Vienna, 21st August, 2024

Signature Author

Signature Advisor

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

A Holistic and Interactive 3D Visualization of Version-Controlled Source Code Repositories

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Maximilian Johannes Zenz, BSc
Matrikelnummer 01633058

ausgeführt am
Institut für Information Systems Engineering
Forschungsbereich Business Informatics
Forschungsgruppe Industrielle Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Thomas Grechenig

Wien, 21. August 2024

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Maximilian Johannes Zenz, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. August 2024

Maximilian Johannes Zenz

Danksagung

An dieser Stelle möchte ich mich bei meinem Professor für die gute Zusammenarbeit und Unterstützung bei der Erstellung dieser Arbeit bedanken. Ich werde für immer dankbar sein für die Möglichkeit, an einem so interessanten Thema arbeiten zu können.

Zusätzlich möchte ich meinen Kollegen beim INSO für ihre großartige Unterstützung bei meiner Arbeit danken. Ohne eure Hilfe und Unterstützung wäre dieser Weg nicht derselbe gewesen.

Außerdem möchte ich mich bei allen Teilnehmern bedanken, die an den Interviews teilgenommen haben, die im Rahmen dieser Arbeit durchgeführt wurden, um meine Forschungsfragen zu belegen. Ohne sie wäre diese Arbeit nicht möglich gewesen, wofür ich ihnen sehr dankbar bin.

Acknowledgements

Here I want to thank my professor for the great work together and support in writing this Thesis. I will be always grateful for the chance to work on such an interesting topic.

I also want to thank my colleagues at INSO for their great support with my thesis. Without your help and support this path would not have been the same.

Additionally, I want to thank all the participants who took part in the interviews that were conducted during this thesis needed to prove my research questions. Without them, this work would not have been possible which I am deeply thankful for.

Kurzfassung

Moderne Software wird immer umfangreicher und damit auch die Daten, die Versionskontrollsysteme erzeugen. Dies führt zu hohen Eintrittsschwellen, wenn man versucht, ein Repository zu analysieren oder wenn man einem neuen Projekt beitritt. Daher werden in einigen Fällen, um mit den riesigen Datenmengen umzugehen, Visualisierungen verwendet. Diese helfen die Analyse von großen Datenmengen zu vereinfachen. Im Rahmen dieser Arbeit wurde eine dreidimensionale Visualisierung von Versionskontrollsystemen vorgeschlagen und prototypisch umgesetzt, um zu zeigen, dass eine solche Visualisierung die Analyse eines Repositories verbessern kann.

Bevor mit der Entwicklung des Prototyps begonnen wurde, wurde ein Entwurf erstellt und mit drei Experten evaluiert. Im Rahmen dieser Evaluierung wurden verschiedene vorgeschlagene Funktionen vorgestellt und erläutert. Dies half den Experten, sich ein besseres Bild von der vorgeschlagenen Visualisierung zu machen, da ähnliche Konzepte wie die in dieser Arbeit entwickelte nicht sehr verbreitet sind. Zusätzlich zu dem positiven Resultat der Evaluierung konnte unter den Teilnehmern der Evaluierung auch bereits Interesse geweckt werden, wie eine solche Visualisierung funktionieren könnte.

Der entworfene und entwickelte Prototyp konzentriert sich dabei auf mehrere verschiedene Metriken wie Änderungs-, Entwickler- und Verfasserdaten, um einen umfassenden Überblick über ein Software Repository zu schaffen. Er zeigt alle Commits eines Repositories mit allen geänderten Dateien gleichzeitig an und ermöglicht es außerdem den Quellcode und die Änderungen bestimmter Dateien zu einem bestimmten Zeitpunkt zu betrachten. Der praktische Anteil dieser Arbeit liegt dabei auf dem Visualisierungsteil, der mit der Videospiel-Engine Unity3D erstellt wurde, während die Mining-Daten vom Binocular-Projekt der INSO Research Group zur Verfügung gestellt werden.

Um die angestrebten Vorteile der Visualisierung zu überprüfen, wurden mehrere Interviews mit Experten geführt, die bereits mit der Analyse von versionskontrollierten Repositories vertraut sind. Diese Interviews kamen zu dem Schluss, dass das Konzept zwar vielversprechend ist und tatsächlich bei der Analyse von versionskontrollierten Repositories helfen kann, dass aber noch ein langer Weg vor uns liegt, um einen Prototyp wie den in dieser Arbeit entwickelten praxistauglich zu machen.

Keywords: *Visualisierung, Versionskontrollsystem, 3D, Git, Binocular, Unity3D*

Abstract

Modern software is getting bigger and bigger and thus also the data version control systems produce. This results in major issues when trying to analyze a repository or enter a new project. Therefore, in some cases to deal with the huge amounts of data, visualizations are used to collect and simplify the analysis of such. As part of this thesis, a three-dimensional version control visualization was proposed and prototyped to show that such visualization can improve the analysis workflow of a repository.

Before starting the development of the prototype, a design was created and evaluated with three experts. As part of this evaluation, different proposed features were presented and explained. This helped the experts get a better idea of the proposed visualization as the concept is uncommon. This resulted in a positive outlook and curiosity about how a visualization like this could work.

The designed and developed prototype thereby focused on multiple different metrics like change, developer and ownership data to create a comprehensive overview of the repository. It shows all commits of a repository with all changed files at the same time and even enables one to view the source code and changes of specific files at a specific point in time while navigating in a three-dimensional space. The main aspect of this thesis is thereby on the visualization part created with the game engine Unity3D while the mining data is provided by the Binocular Project from the INSO Research Group.

To evaluate those claims multiple interviews were conducted with experts who were already familiar with analyzing version-controlled repositories. Those interviews concluded that while the concept is promising and can indeed help people to analyze version-controlled repositories, there is still a long way to go to make a prototype like the one developed in this thesis production-ready.

Keywords: *Visualization, Version Control System, 3D, Git, Binocular, Unity3D*

Contents

Kurzfassung **xiii**

Abstract **xv**

Contents **xvii**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Motivation . . . 2
1.3 Expected Results . . . 3
1.4 Structure . . . 3

2 Foundations **5**
2.1 Version Control Systems . . . 5
2.2 Visualizations of Version Control Systems . . . 6
2.3 3D Graphics . . . 7
2.4 3D Version Control Visualizations . . . 8
2.5 High-Level Shader Language . . . 8
2.6 Unity Shader Graph . . . 11

3 State of the Art **13**
3.1 Scientific Visualizations . . . 13
3.2 Commercial Visualizations . . . 21
3.3 Independent Open-Source Visualizations . . . 26

4 Requirements **29**
4.1 Current Limitations . . . 29
4.2 Research Questions . . . 30
4.3 Methodology . . . 30
4.4 Design . . . 31

5 Prototype **43**
5.1 Technologies Used . . . 43
5.2 Data Used . . . 44

5.3 Iteration 1 - Basic Implementation 46
5.4 Iteration 2 - Refined Implementation 51
5.5 Iteration 3 - Data Highlighting 57
5.6 Iteration 4 - Final Prototype 65

6 Evaluation **69**
6.1 Interviews 69
6.2 Evaluation Results 78

7 Discussion **81**
7.1 Research Questions 81
7.2 Results 83

8 Conclusion **85**
8.1 Retrospection 85
8.2 Future Work 86
8.3 Summary 87

List of Figures **89**

Listings **91**

Acronyms **93**

Bibliography **95**

Book References **101**

Online References **103**

Appendices **107**
Appendix A 108
Appendix B 122

CHAPTER 1

Introduction

Software repository visualizations are a widespread topic. Many researchers are trying to make it easier to understand the vast amount of data a version-controlled repository produces but visualizations of such are still rarely used in the field. Additionally, those visualizations rarely explore topics beyond classic statistics. Therefore, it was chosen that as part of this thesis, a new kind of version control repository visualization should be developed to demonstrate that there are still different ways to explore a repository. In the next sections is described more in-depth what the precise problem is that this thesis attempts to solve and the motivation behind it. Additionally, the expected results and the structure of the thesis will be described.

1.1 Problem Statement

Currently, in the field of visualizations for version-controlled software repositories most of the different options available are aggregated or detailed views that do not show the whole picture of a version-controlled source code repository. To resolve those limitations a visualization will be designed and prototyped that holistically includes and displays all the data from a version-controlled repository.

Due to the complexity a visualization like this would yield it was decided to design it in a three-dimensional space. This makes it possible to display more data at once. It also helps the user to get a better overview of the whole repository at once. Additionally, there is currently a lack of 3D repository visualizations that use state-of-the-art computer graphic methods. Most are either outdated or use web-based technologies. This hinders the scope and capabilities of visualizations possible. Due to the huge interest of the entertainment industry computer graphics and rendering engines are making large advancements year after year. Those could improve 3D visualizations by a huge amount so that it becomes feasible to create a state-of-the-art holistic visualization of a version-controlled project. Also, due to the underrepresentation of 3D visualizations, most of the current 3D

implementations of version-controlled repositories are code cities [1] or a variation of code cities [2]. Those only show an aggregated view of a repository and do not show the complete history of a repository at the same time. Although code cities have their place in the field of three-dimensional version control visualizations, there is a huge potential for improvements by different concepts. Previous attempts at a visualization like this are old and outdated and do not show a complete holistic picture of the version-controlled source code repository [3]. Although not part of this thesis, the introduction of newer technologies like virtual, augmented and mixed reality can have an impact on how such visualization will be designed and used in the future. A few of those visualizations like CodeHouse or VR-Git will be described later in Section 3.1.

1.2 Motivation

The main motivation for this master thesis is to prove that a large-scale 3D holistic version control visualization is feasible and can give additional insights into a version-controlled repository. This thesis builds hereby on the assumption that such a visualization could improve the workflow of analyzing the history and trends of a repository. There are already many visualizations that do specific aspects of version-controlled repositories in-depth and there are many approaches like „Visualizing GitHub Issues“ [4], „The City Metaphor in Software Visualization“ [1], „Exploring Developer Preferences for Visualizing External Information Within Source Code Editors“ [5] and „Can Git Repository Visualization Support Educators in Assessing Group Projects?“ [6] that provide an aggregated overview of a repository. None of them can give a holistic view of a version-controlled source code repository. Therefore, this thesis proposes a holistic visualization that helps to gather additional insights for a repository more efficiently and makes it possible to analyze trends from the repository to the file level. Additionally, there is not much development in the space of three-dimensional version control visualization which could vastly improve the information density.

A visualization like this helps developers gain a deeper understanding of the project they are working on by gaining insight into how parts of the project evolved and which developers were involved in. In addition to that it supports team and project leaders to understand better how the project they lead is structured and who contributed to what part of the project and when. Insights into what parts of the project are outdated and how to distribute resources more efficiently can also possibly be extracted from a holistic visualization like the one proposed. Additionally, it can also give a benefit to university professors or tutors to improve how they analyze a student's work. Especially in the educational context, it is difficult but necessary to gather information about what and how much each student contributed to properly grade students. In detail, the visualization will provide the possibility to analyze code ownership on a large scale and even analyze how the code expertise is distributed over the history of a repository.

1.3 Expected Results

The result of this thesis is the creation of an interactive holistic three-dimensional helix visualization. The data for the visualization is provided by the Binocular research project [7]. Binocular already provides the functionality can mine multiple version control-related data sources at once and export the data accordingly. Binocular was also chosen because of the past and current involvement of the author of this thesis within the project. The proposed visualization will display the whole history of a repository with all its files, folders, and developer data at once and the user will be able to overlay different kinds of data like change, developer, or hotspot data. It will also have a focus on ownership data and how ownership changed throughout the development history. This visualization prototype is the basis for a scenario-based expert evaluation at the end of the study. These interviews will investigate if and how such a visualization can improve the workflow and data gathering of developers. From a scientific perspective, the following research questions will be answered by this thesis.

- **RQ1a:** How do developers analyze the history and trends of a repository?
- **RQ1b:** How do developers rate the benefits of the proposed visualization?
- **RQ2:** What visualization and interaction methods are needed to show a holistic view of a code repository?
- **RQ3:** How suitable is the visualization to analyze the proposed benefits regarding source code evolution, code hotspots, code ownership and code expertise?

Those research questions are set to be answered by conducting a scenario-based evaluation with six experts. Each of them will be guided separately through a presentation of the final prototype and then they will be questioned about the functionality, feasibility and usefulness of a visualization like this. They will also be guided through multiple scenarios where they should rate how suitable the visualization is to accomplish the scenario. With this evaluation, it should be possible to answer the before-defined research questions.

1.4 Structure

At first, this thesis will go through the concepts needed in the chapter Foundations (Chapter 2). This will include the basics of version control systems, version control system visualization, 3D graphics and 3D version control visualizations. Having a basic understanding of that topic will help the reader of this thesis to understand the chapters following.

After that, current and past projects will be described to show the current state of the art (Chapter 3) of version control visualizations both in 3D and relevant non-3D ones. Those will include both scientific and commercial visualizations as well as general open-source projects.

The next chapter is the requirements chapter (Chapter 4). It includes the current limitations, the research questions and the methodology used to answer the previously defined research questions. This chapter also includes the design section where the general design of the prototype will be described. Additionally, there will be a section about the evaluation of the proposed features and the design of the visualization. The evaluation is designed as an expert evaluation with three different experts.

The following chapter is about the development of the prototype (Chapter 5) divided into four development iterations. Here the full development process will be described in depth. This includes how the prototype was developed, what technologies were used and what decisions and compromises had to be made during the development. This is important to understand the whole process of how the final prototype was created.

The second to last chapter (Chapter 6) is about evaluating the developed prototype. This will evaluate the prototype against the previously defined research questions to show that those can be answered by a visualization like this. The evaluation is done through six scenario-based interviews with experts.

The last chapter is the conclusion (Chapter 8) where the whole project gets revised, summarized and a conclusion gets drawn. This also includes a section about future work and how the project can be improved and extended in future iterations.

CHAPTER 2

Foundations

The sections of this chapter will give an overview of the foundational knowledge needed for understanding this thesis. The first section (Section 2.1) will be about version control systems in general. Here the basic functionality is described and how these tools improve the work of a developer. As part of this section, there is also a short description of what version control systems exist and the most used version control system today, Git.

After that, it is important to understand why visualizations (Section 2.2) are important and how they can improve the workflow of gathering data. The last two chapters will focus more on 3D Graphics (Section 2.3) and how 3D Graphics can be used in version control visualizations (Section 2.4).

As part of this chapter, there are also two sections about how shaders are written in Unity 3D. The first will be about shaders written in the High-Level Shader Language (Section 2.5). The second will be about the graphical node-based shader editor built into Unity3D called Shader Graph (Section 2.6)

2.1 Version Control Systems

A Version Control System or short VCS is a piece of software that helps developers manage the different versions of files within a project [8], [9]. A Version Control System can be implemented in both a centralized or distributed way [10]. Today, distributed version control systems are more popular (Figure 2.1) with the most prominent representative being Git.

Through time many different version control systems existed [12]. Some of the most

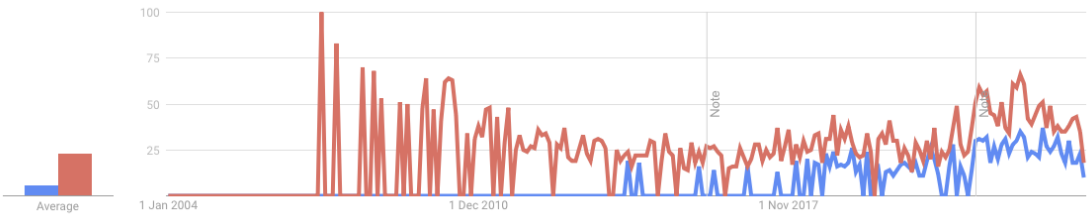


Figure 2.1: Google Trends [11] of the popularity in Google Search of centralized version control (blue) against distributed version control (red).

prominent representatives RCS[1] [13], CVS[2] [14], SVN[3] [15] and Git[4] [16]. Today Git is the most used version control system with a market share of over 93 percent[5] [17] that was developed by Linus Torvalds in 2005 [18], [19].

In its core functionality, a Version Control System tracks all the changes that happen from one version to the next of every file added to the VCS within the repository. Those versions are called commits and can contain additional information like a commit message that describes what happened within a commit, the author of the commit, what commits are the parents of the commit or the commit date and time. This information is necessary to help the user as well as the version control system itself to track and find specific commits that contain specific changes.

A major reason why version control systems are used is that they support when multiple developers work on the same project or file. When there are two different versions of the same file available to the version control system at the same time it supports the user to combine those. The process to combine those two is called a merge. To further improve parallel development most version control systems support branches as part of how they manage parallel working developers. Branches help developers work on different features over multiple commits without interfering with each other and only having to merge one time after a feature is finished.

2.2 Visualizations of Version Control Systems

Version control systems store a lot of data that can quickly become too much for a user to comprehend. Therefore, as with other big data sources, visualizations can be a useful tool to get insights into it. Packing large amounts of data within graphical representations

[1]Revision Control System: `https://www.gnu.org/software/rcs/` (Last accessed 07.06.2024)
[2]Concurrent Versions System: `https://cvs.nongnu.org` (Last accessed 07.06.2024)
[3]Apache Subversion: `https://subversion.apache.org` (Last accessed 07.06.2024)
[4]Git: `https://git-scm.com` (Last accessed 07.06.2024)
[5]Stackoverflow Git Usage: `https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/` (Last accessed 07.06.2024)

is necessary because it is easier and faster for the human brain to analyze visual data compared to abstract one as described in [20].

All this also applies to version control system visualizations as a subcategory of big data visualizations. The version control system visualizations most developers are most likely familiar with are the repository statistics available in GitHub (Section 3.2.1) or GitLab. Those are used to get a quick overview of the most common statistics of a repository, like the commits per week or similar. However, many standalone visualizations can give even deeper insights into a version-controlled repository. Those are further described in Chapter 3.

Most of the currently available and used version control system visualizations are two-dimensional and focus more on general statistics. Three-dimensional visualizations as the one designed (Section 4.4) and developed (Chapter 5) as part of this thesis are rather rare (Chapter 3) as they are more complex and thus harder to develop and use.

2.3 3D Graphics

As the prototype developed as part of this thesis harnesses a three-dimensional environment 3D graphics are an important part of the work. At its core 3D graphics consist of drawing points called vertices through a so-called vertex shader in a 3D space. The vertex shader transforms the vertex coordinates from the world or object space to the screen space. After the transformation, the vertexes need to be connected to form objects. The most common way to connect them is through triangles [21], which are the simplest polygons that can define a surface in three-dimensional space.

After that, those surfaces that result from the last step need to be rasterized and filled with a color to make them visible in the 3D space. This step is called shading and is done by the fragment shader. The shader also controls how light is interpreted and can manipulate the surface. This process is described more closely in the Graphics Compendium[6] [22] and can be seen in Figure 2.2.

As part of the shading process, texturing is a vital technique used to enhance the visual quality of 3D meshes. A 3D Mesh herby describes a collection of vertices that are connected to form a 3D object. Texturing involves mapping a 2D image texture onto the surface of a 3D object. This is achieved by assigning UV coordinates to each vertex of the 3D model, which determines how the texture is wrapped around the object. UV hereby stands for the two axes, u and v, of the texture. The letters u and v were chosen as the texture coordinates because X, Y and Z were already used for the coordinates in the 3D space and W is commonly used for calculating quaternion rotations. The UV coordinates essentially provide a correspondence between the texture map and the 3D geometry, ensuring that the texture is accurately applied to the model.

[6] Graphics Compendium: `https://graphicscompendium.com/intro/01-graphics-pipeline` (Last accessed 01.07.2024)

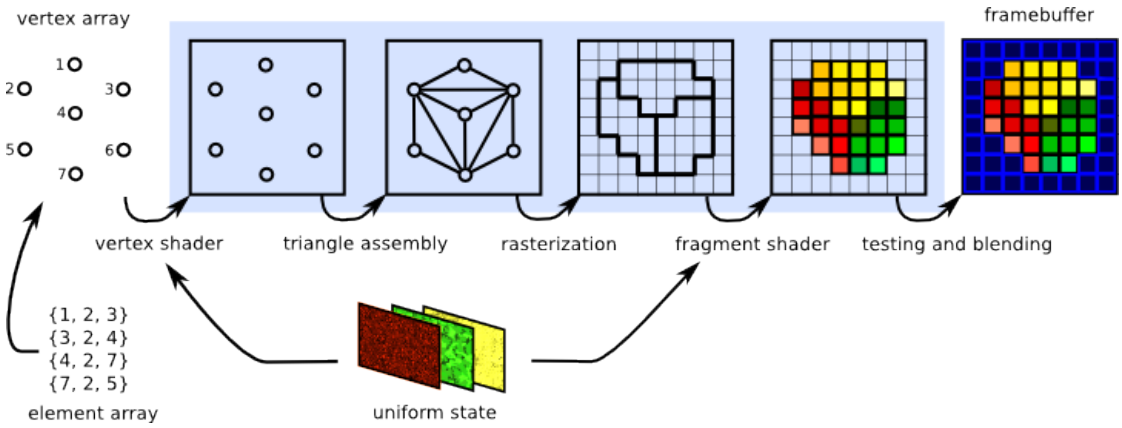


Figure 2.2: Basic 3D Graphics Pipeline [22] of OpenGL.

All that information gets transferred to the GPU which produces the final image and pushes it to the displaying device connected like a screen or VR glasses.

To streamline the complex process of rendering 3D objects, 3D engines exist to simplify and automate repetitive tasks. Those engines are mostly used in creating games but can also be used to create other types of three-dimensional real-time renderings. A common representative of 3D engines is Unity3D as described further in Section 5.1.

2.4 3D Version Control Visualizations

Whereas two-dimensional version-control visualizations are widely used to analyze version-controlled repositories their three-dimensional counterparts are still rarely used. In „Past, Present, and Future of 3D Software Visualization - A Systematic Literature Analysis“ [23] 631 papers about software visualizations were examined whereas only 115 were about 3D visualizations. Common two-dimensional visualizations include for example the built-in visualizations in GitHub as described in Subsection 3.2.1. Three-dimensional version control systems combine the huge amount of data of a version-controlled repository with the capabilities of 3D graphics. This can improve the data density of the visualization. Some notable examples of 3D version control system visualizations can be found in Chapter 3. Even if 3D version control system visualizations have a greater data density in general, they are more complex to understand and harder to use and implement thus they are developed less frequently.

2.5 High-Level Shader Language

HLSL is a shader language developed by Microsoft for the Direct3D 9 API. It is also the shader language that is used in Unity3D. HLSL is analogous to GLSL which is the

shading language of OpenGL[7] [24]. HLSL is further described in the Microsoft[8] [25] and Unity[9] [26] reference. An example of a simple single color HLSL shader for unity can be seen in Listing 2.1 from the Unity Shader Reference[10] [27].

[7] OpenGL: https://www.opengl.org (Last accessed 25.05.2024)

[8] High-level shader language (HLSL) - Win32 apps | Microsoft Learn: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl (Last accessed 01.07.2024)

[9] Unity - Manual: Writing shaders overview: https://docs.unity3d.com/Manual/SL-ShadingLanguage.html (Last accessed 01.07.2024)

[10] Unity Shader Reference Examples: https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html (Last accessed 27.07.2024)

```
1   Shader "Unlit/SingleColor"
2   {
3       Properties
4       {
5           // Color property for material inspector, default to
                ↪ white
6           _Color ("Main Color", Color) = (1,1,1,1)
7       }
8       SubShader
9       {
10          Pass
11          {
12              CGPROGRAM
13              #pragma vertex vert
14              #pragma fragment frag
15
16              // vertex shader
17              // this time instead of using "appdata" struct, just
                    ↪ spell inputs manually,
18              // and instead of returning v2f struct, also just
                    ↪ return a single output
19              // float4 clip position
20              float4 vert (float4 vertex : POSITION) : SV_POSITION
21              {
22                  return mul(UNITY_MATRIX_MVP, vertex);
23              }
24
25              // color from the material
26              fixed4 _Color;
27
28              // pixel shader, no inputs needed
29              fixed4 frag () : SV_Target
30              {
31                  return _Color; // just return it
32              }
33              ENDCG
34          }
35      }
36  }
```

Listing 2.1: A simple single color Unity HLSL Shader.

2.6 Unity Shader Graph

Besides the ability to write shaders in HLSL unity also offers a proprietary graphical shader editor called Shader Graph[11] [28] to simplify the development of comprehensive shaders. Shader Graph is a fully node-based visual editor and does not require deeper knowledge of shader programming like HLSL. An example of a checker pattern shader with a gradient can be seen in Figure 2.3 from Unity Learn[12] [29]. Shader Graph also provides a live preview of the designed shader and how it changes when a parameter gets changed.

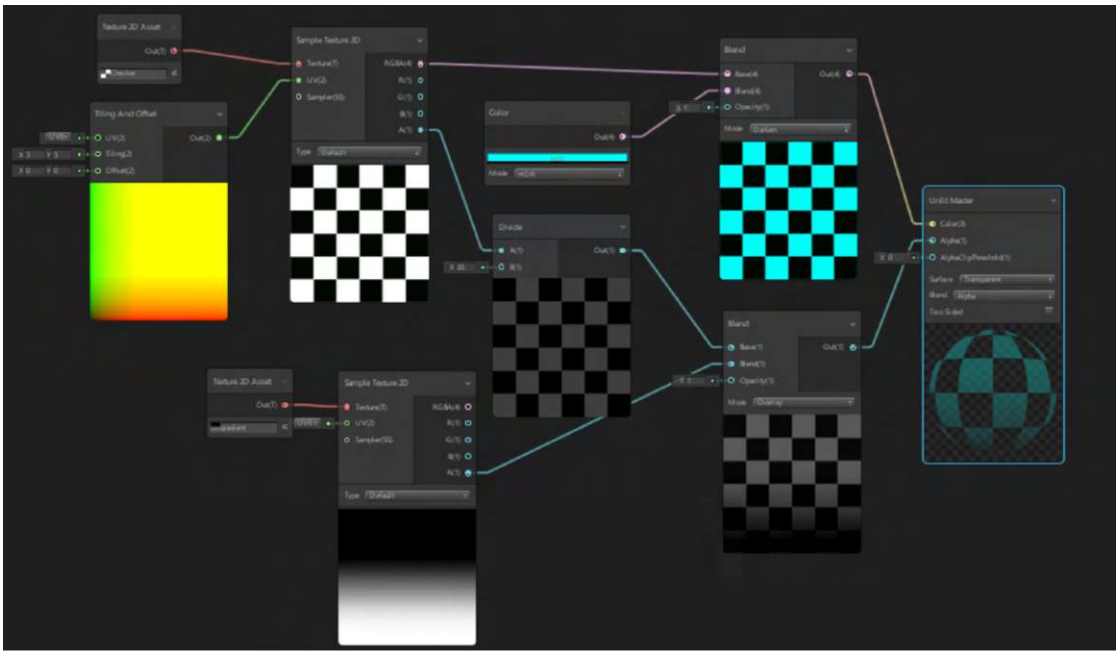


Figure 2.3: An example [29] for a checker pattern shader with a gradient made in Unity Shader Graph.

[11] About Shader Graph: `https://docs.unity3d.com/Packages/com.unity.shadergraph@17.0/manual/index.html` (Last accessed 01.07.2024)

[12] Unity Learn - Introduction to ShaderGraph: `https://learn.unity.com/tutorial/introduction-to-shader-graph` (Last accessed 27.07.2024)

CHAPTER 3

State of the Art

This chapter will focus on the current state of version control visualization and 3D version control visualizations. First, Section 3.1 gives an overview of current scientific solutions followed by Section 3.2 about current commercial visualizations. At last Section 3.3 is about some independent open-source visualizations that do not fit in either previous category.

3.1 Scientific Visualizations

There are already many scientific attempts to visualize version-controlled repositories. Most of them visualize those repositories in two-dimensional visualizations and use some sort of aggregated data. Still, there are attempts to create three-dimensional visualizations that can display more data at once. The following sections give an overview of some relevant scientific visualizations of version-controlled repositories. Additionally, an older overview of visualizations that harness three-dimensional graphics can be found in „An Overview of 3D Software Visualization“ from 2009 [30].

3.1.1 Code Cities

The first and most common type of three-dimensional visualizations for version-controlled software repositories are code cities. Here the data extracted from a repository gets aggregated and displayed in a city-like visualization as seen in Figure 3.1. Most of the time the data gets grouped in some way or the other for example into modules or classes. Those groups then in general get drawn as areas on a plane which then get extruded to encode additional data for each group. The pillars resulting out of that often are displayed or look like buildings and skyscrapers hence the name code city. There are many different variants of this type of three-dimensional visualization as seen in Figure 3.2 but at its core, they all cover the same style of visualization. This is further described

in „The City Metaphor in Software Visualization“ [1], „Visualizing Data in Software Cities“ [31] or „Visualizing Evolving Software Cities“ [32]. Some examples of Code Cities can be found in „Code Park: A New 3D Code Visualization Tool“ [33], „Visualization of Codes and Code Smells with 3D Mushroom Farm Metaphor“ [2], „IslandViz: A Tool for Visualizing Modular Software Systems in Virtual Reality“ [34], „UrbanIt: Visualizing repositories everywhere“ [35], „GoCity: Code City for Go“ [36] and „Softization-3D visualization of a version control repository“ [37].

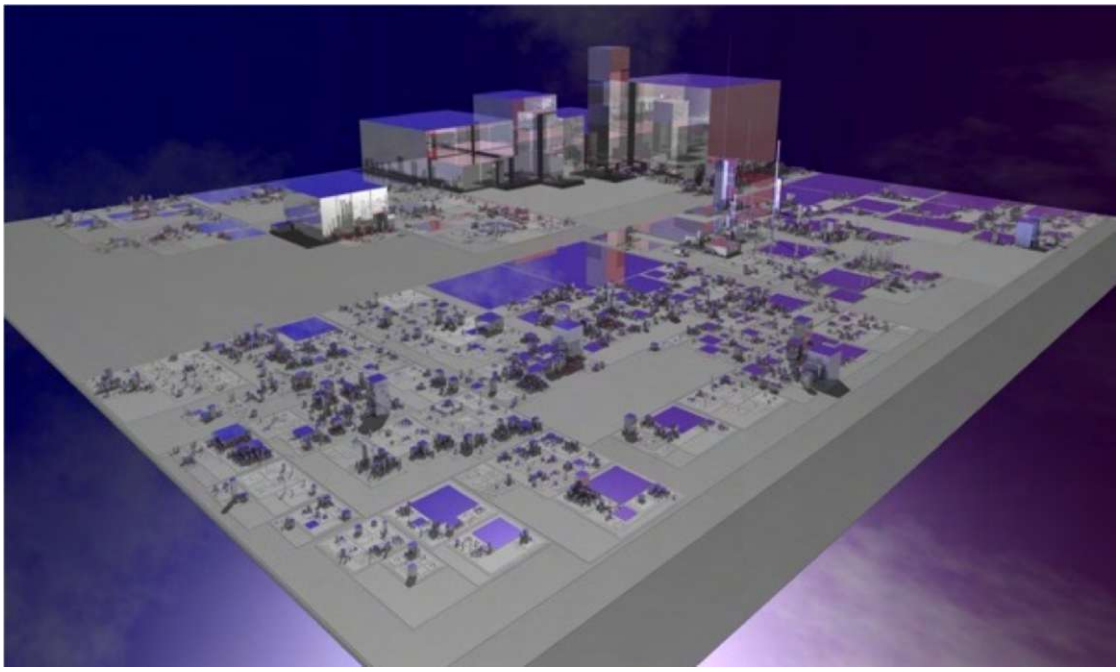
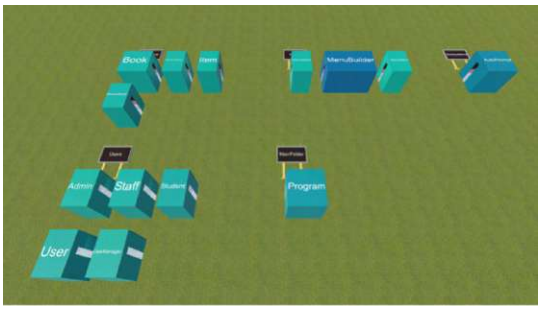
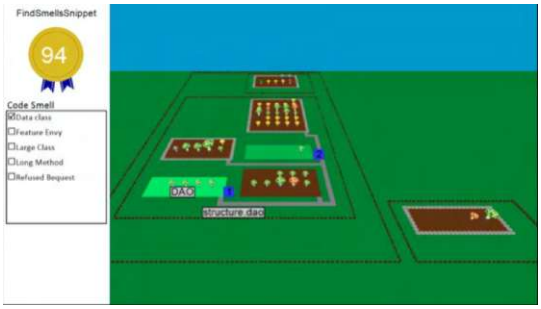


Figure 3.1: Example code city as described in [1] how it is currently most common in research.

In addition to the more common visualizations that use an average screen setup, there are also attempts to build a code city with new technologies in mind like virtual reality as described in „VR-GitCity: Immersively Visualizing Git Repository Evolution Using a City Metaphor in Virtual Reality“ [38]. This can be seen in Figure 3.3. New interaction methods like VR, MR and AR can improve the usability of three-dimensional visualizations as such.

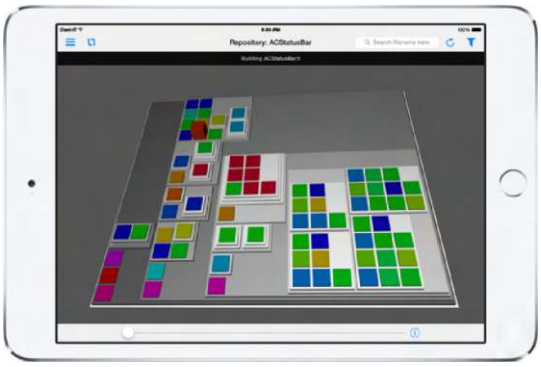


(a) A code city-like visualization as described in [33] that displays a project as Rooms with its content inside the rooms.



(b) The Visualization shown in [2] displays a code city as mushrooms and where the "smell" of the mushrooms conveys additional information.

(c) The code city described in [34] displays the code as islands where bundles get displayed as islands, packages as regions and class types as buildings.



(d) UrbanIt [35] is an attempt to design a code city that is accessible through mobile devices so that it is available to a developer from everywhere.

Figure 3.2: Variations of code cities that try to show different aspects of a version-controlled repository.

3.1.2 CodeHouse

Code House [39] is an interesting attempt at visualizing version-controlled repositories. It displays the repository in a three-dimensional space that the user can explore through a virtual reality headset. As the name already suggests it visualizes the repository as a house with floors and rooms where each floor represents a layer that the user can define and each room represents a module. It is also possible to select a room and enter the room in virtual reality and you can explore all the elements within the module like Functions within the module as seen in Figure 3.4.

3.1.3 VR-Git

VR-Git [40] is one of the most interesting attempts at a 3D source code version control system visualization. It is the closest 3D VCS visualization to the visualization developed

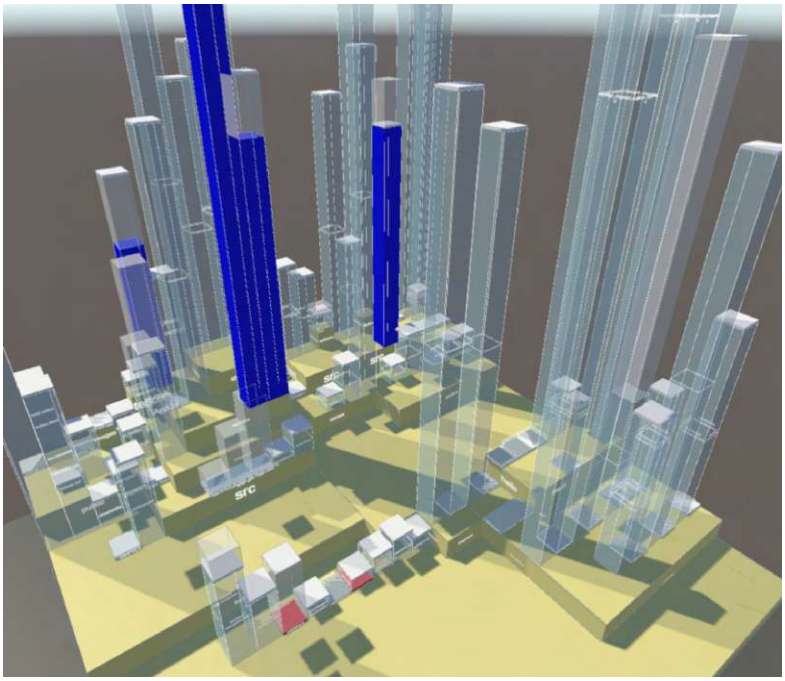


Figure 3.3: In this figure an example of a VR code city is shown from VR-GitCity [38].

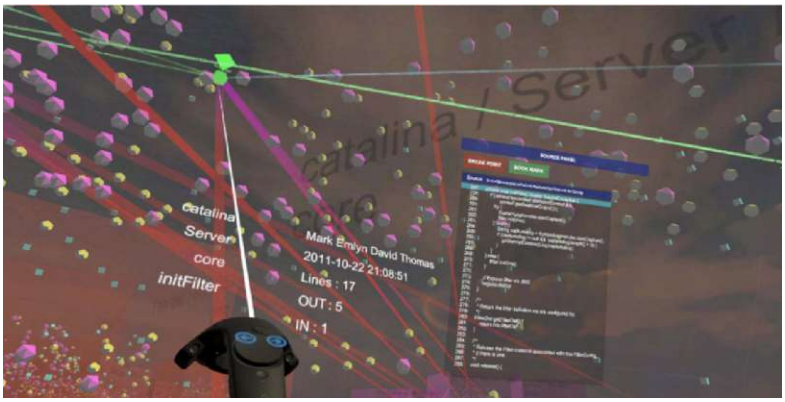


Figure 3.4: Example visualization from within a room in a house where a function gets selected [39].

as part of this thesis. At its core, VR-Git visualizes the whole commit tree with all changed files for each commit. It also adds additional information through color coding. This especially shows in how the individual changed files are displayed with green marked files receiving additions, red ones deletions and blue ones receiving just changes. It also gives the option to view the code for each changed file. In Figure 3.5 you can see an example visualization created with VR-Git that shows multiple branches and commits and shows the files that are changed in the commit.

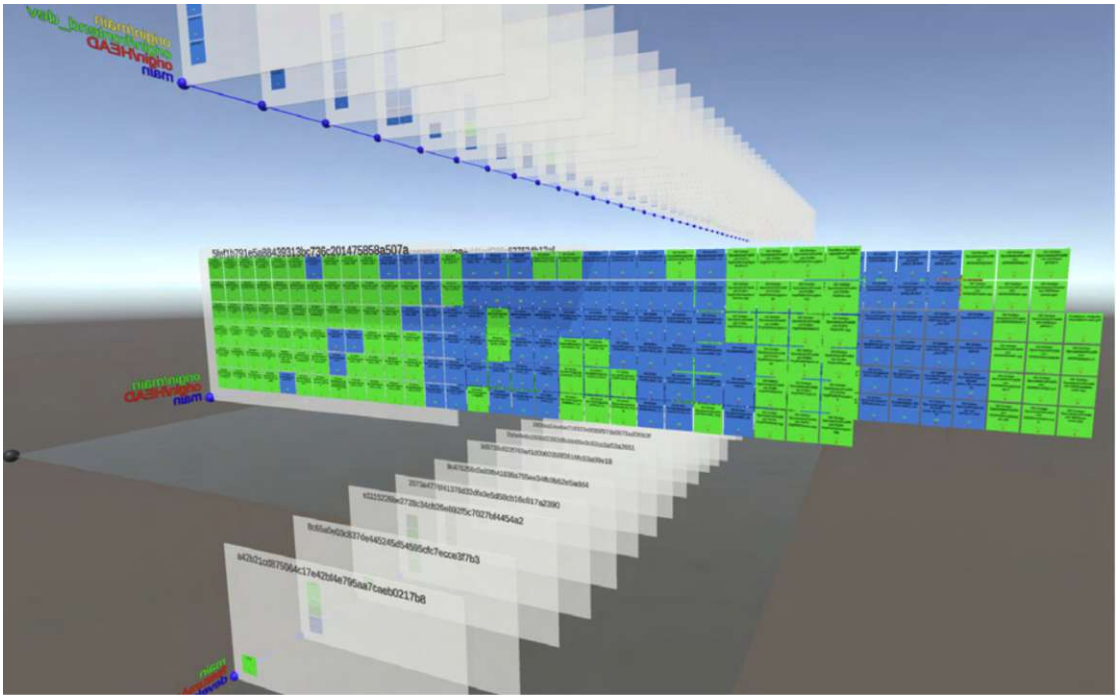


Figure 3.5: Example visualization made with VR-Git that shows multiple branches with commits and the changed files for each commit. [40]

3.1.4 VRCS

VRCS or Visual RCS is the oldest three-dimensional version control system visualization found. It was developed in 1997 and uses data from the version control system RCS as Git did not exist in 1997. VRCS is close to the prototype developed as part of this thesis but rather focuses more on module management than encoding version control system metrics. Some of those differences are also explainable by the differences between RCS which VRCS uses and Git which most newer visualizations and also this thesis build on. The basic structure consists of a strand of versions of the files within the projects nested along a central structure. Hereby, VRCS places the different releases of the software in the center as big spheres. Impressive for its time VRCS still had some limitations which can be resolved with modern visualization technologies and computer hardware. As described in the paper *VRCS: Integrating version control and module management using interactive three-dimensional graphics* by Koike et. al. [3] performance was a huge concern back then. Another problem was that it could only be used by one user.

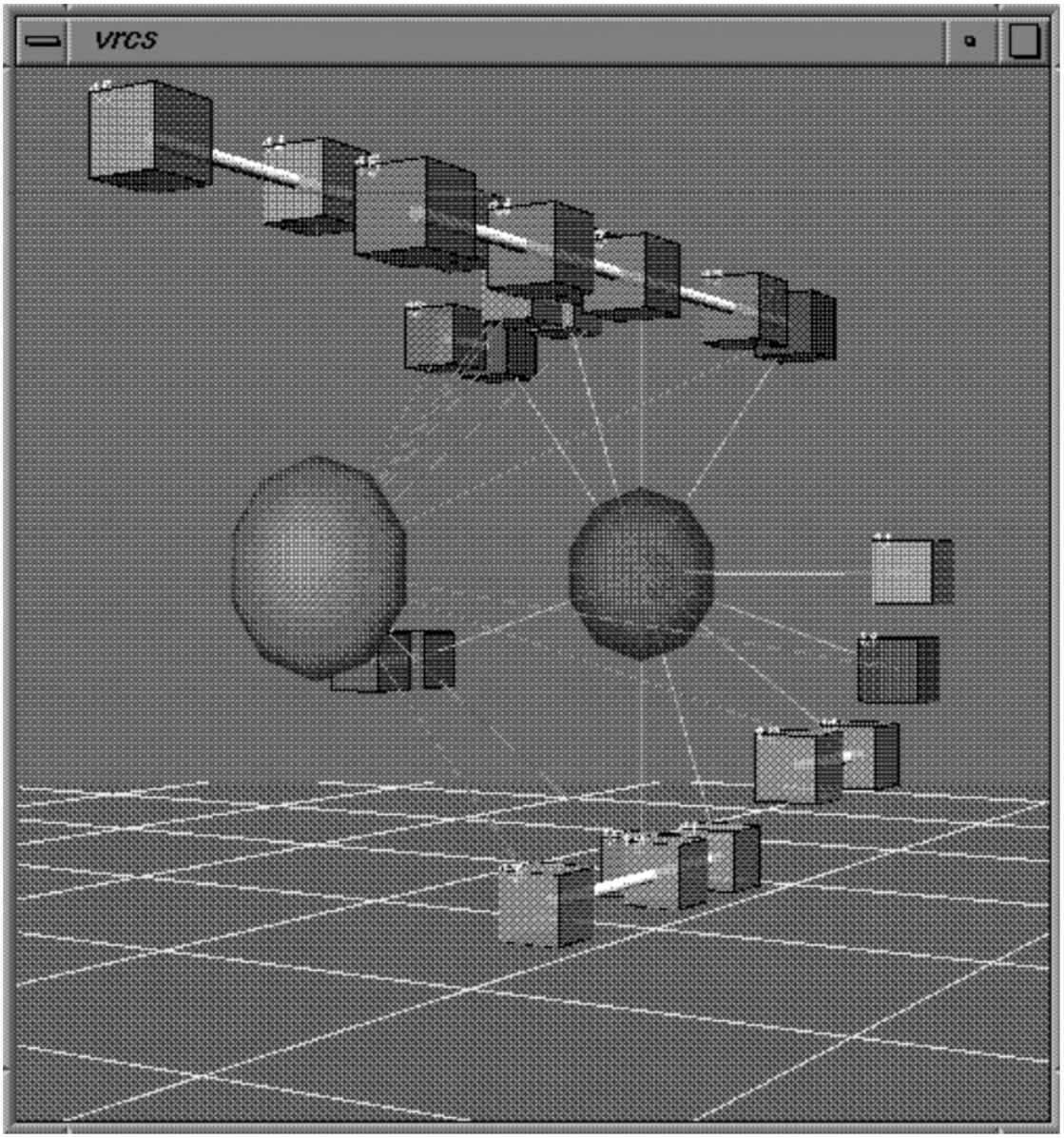


Figure 3.6: VCRS [3] as one of the first 3D Version Control System Visualization on the base of RCS in 1997. In this example, it visualizes a project with 2 releases (Spheres in the center) and multiple files (Cubes on the outside) with multiple versions per file.

3.1.5 Githru

Githru is a Git analysis software project that tries to visualize as much of the information Git provides in a two-dimensional space by combining multiple different diagram types into one as seen in Figure 3.7. It also compresses the information to make it better understandable for the user. This helps the user get a better overview of the project

without the need to analyze a huge number of commits. This is further described in [41].

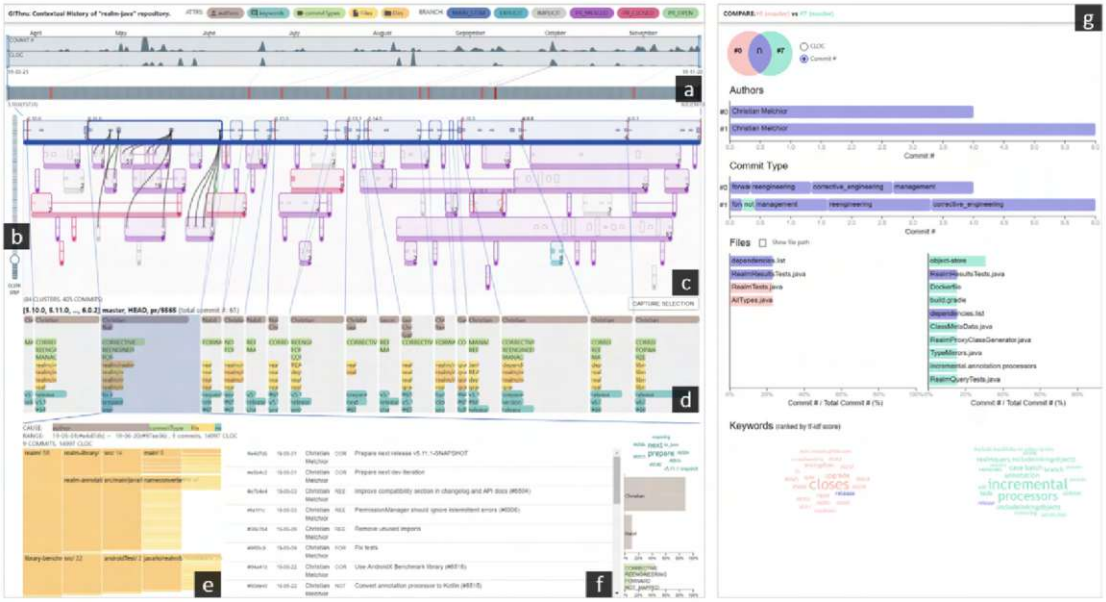


Figure 3.7: In the picture above an example visualization of Githru is shown as described in [41].

3.1.6 VisGi

VisGi [42] is a visualization that tries to help the user get a deeper insight into the evolution of the Project. It thereby offers the ability to visualize a commit in a radial tree visualization. Additionally, it offers to compare multiple commits against each other which helps to understand how the file structure evolved from one commit to the next as seen in Figure 3.8. This is similar to how the structure described in this thesis is generated but in a reduced two-dimensional space.

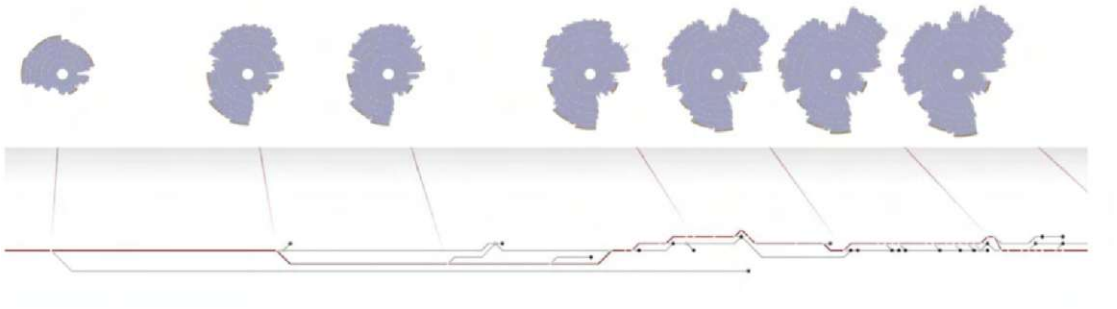


Figure 3.8: Here an example visualization is shown created with VisGi [42]. It consists of a git branch view on the bottom and multiple radial tree views of the file structure at different points in the development history. This helps to get a better understanding of the evolution of the Project structure during the development of the repository.

3.1.7 GETAVIZ

Although GETAVIZ [43], [44] in its core follows the concept of code cities it has some interesting approaches to visualizing the behavior of a repository over time and the different versions of a repository. This can be seen in the concepts for its Behavioral View (Figure 3.9) and Evolutionary View (Figure 3.10).

Figure 3.9: In this Figure the Behavioral View of GETAVIZ [43] is visible. It visualizes the call duration in the three-dimensional space for each method.

3.1.8 VRIDE

VRIDE [45] is an attempt to visualize, explore and interact with a whole repository in a virtual space. It also offers the option to change the code in a file like with any other

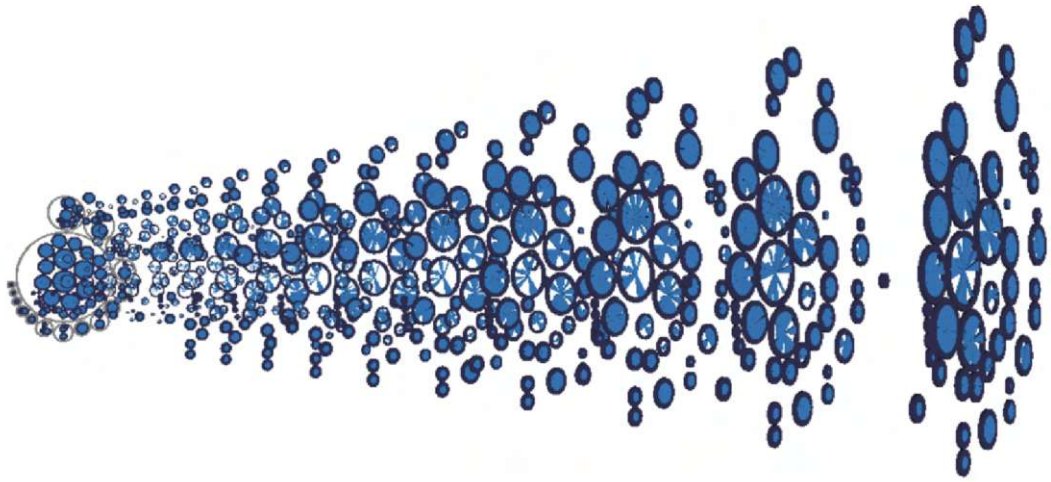
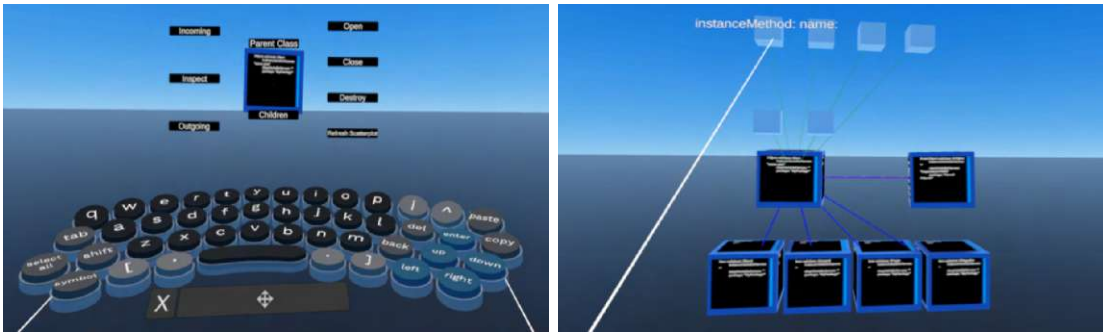


Figure 3.10: The Evolutionary view of GETAVIZ [43] shows all the different versions of a repository as slices in the three-dimensional space.

IDE but in VR. An example of VRIDE can be seen in Figure 3.11. At the time when the paper about VRIDE was written only Python and Pharo were supported as programming languages for VRIDE.



(a) Editing Mode (b) Inspection Mode

Figure 3.11: Here two examples from VRIDE [45] are shown. **(a):** *Editing Mode* - Here it is shown how code editing in VR can look like. **(b):** *Inspection Mode* - In this Figure it is shown how VRIDE envisions the inspection of a repository.

3.2 Commercial Visualizations

Even if the major number of tools to analyze version-controlled repositories are research projects there are still some visualizations out that could be classified as provided or used in a commercial context. Even more uncommon to non-existent are commercial

visualizations that use 3D graphics. Through the following sections some notable mentions of commercially used or provided visualizations and tools to analyze git repositories.

3.2.1 GitHub built-in Visualizations

GitHub is the biggest provider of Git services besides self-hosted Git servers[1] [46]. Part of the GitHub are also some useful tools and visualizations. Those tools and visualizations can give you a deeper and faster insight into a repository when you use GitHub to host your repository. GitHub[2] [47] offers a lot of built-in visualizations and insights into repositories on their own but they mostly only give a rough overview of the repository. They show data like the number of changes over time as seen in Figure 3.12a or the network graph of a repository as seen in Figure 3.12b but deeper insights are lacking.

GitHub also offers an additional visualization that visualizes the activity of a user over a year called the GitHub Skyline[3] [48] as seen in Figure 3.13. This visualization is mainly designed to view and share the activity of a GitHub user with other social media networks in a visually appealing way.

(a) GitHub Insights of the network tree. (b) GitHub Insights of the code frequency

Figure 3.12: Some of the insights GitHub provides. In **(a)** the git network graph that GitHub generates [49] and **(b)** shows the code frequency graph [50] that gives an overview of the additions and deletions over time.

3.2.2 Apache Kibble

Apache Kibble[4] [51] is an interactive Git visualization collection maintained by the Apache Software Foundation[5] [52]. Kibble is one of many projects the Apache Foundation maintains [53]. Kibble is a collection of multiple tools to visualize software repositories. It can give many insights into the development history of a repository from source code analysis to commit statistics. It is also able to track multiple repositories at once and

[1] Github Market Share: https://www.datanyze.com/market-share/source-code-management--315 (Last accessed 02.07.2024)
[2] GitHub: https://www.github.com (Last accessed 07.06.2024)
[3] GitHub Skyline: https://skyline.github.com (Last accessed 07.06.2024)
[4] Apache Kibble: https://kibble.apache.org (Last accessed 07.06.2024)
[5] Apache Foundation: https://apache.org (Last accessed 07.06.2024)

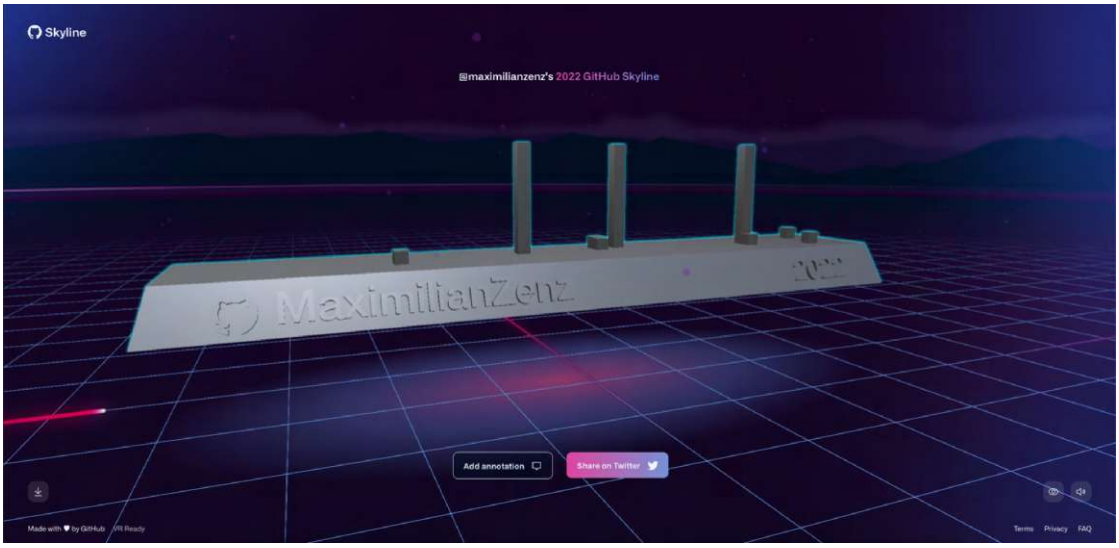


Figure 3.13: The GitHub Skyline [48] gives an overview of the contributions a GitHub user gives over a year.

other connected data services like issue trackers. This should help the user to get a comprehensive overview of all repositories of an organization as seen in Figure 3.14.

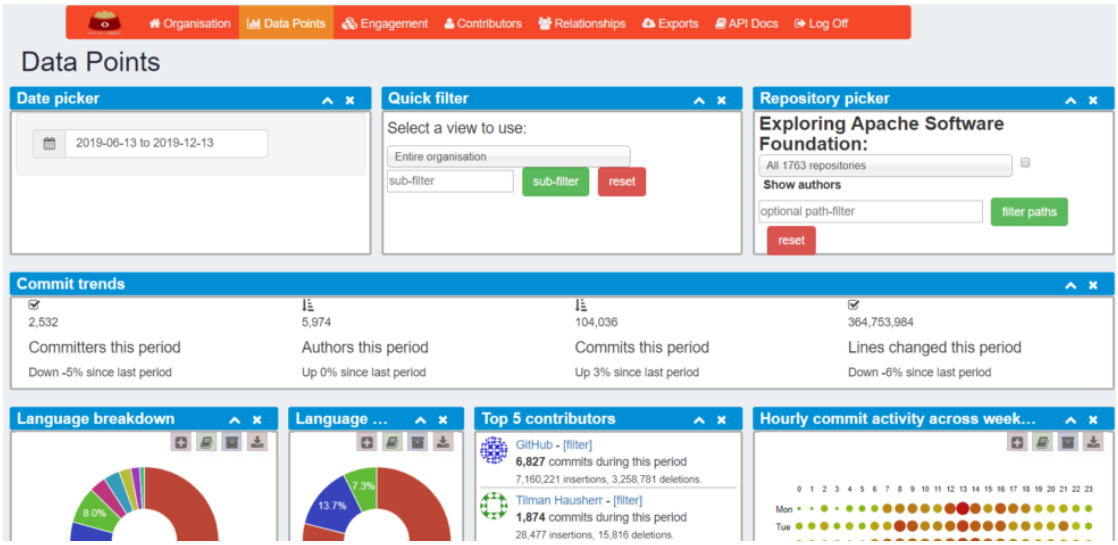


Figure 3.14: This screenshot shows the demo visualization of Apache Kibble [54].

3.2.3 Autodesk Hubble

Autodesk Hubble[6] [55] is a Git repository statistics visualization collection developed by Autodesk as an open-source project. Autodesk Hubble is mainly designed to analyze different metrics over multiple repositories and organizations. It also gives insights into metrics like pull requests, API requests or contributors. Autodesk Hubble is a comprehensive tool to quickly analyze the activity of multiple repositories within an organization or multiple organizations.

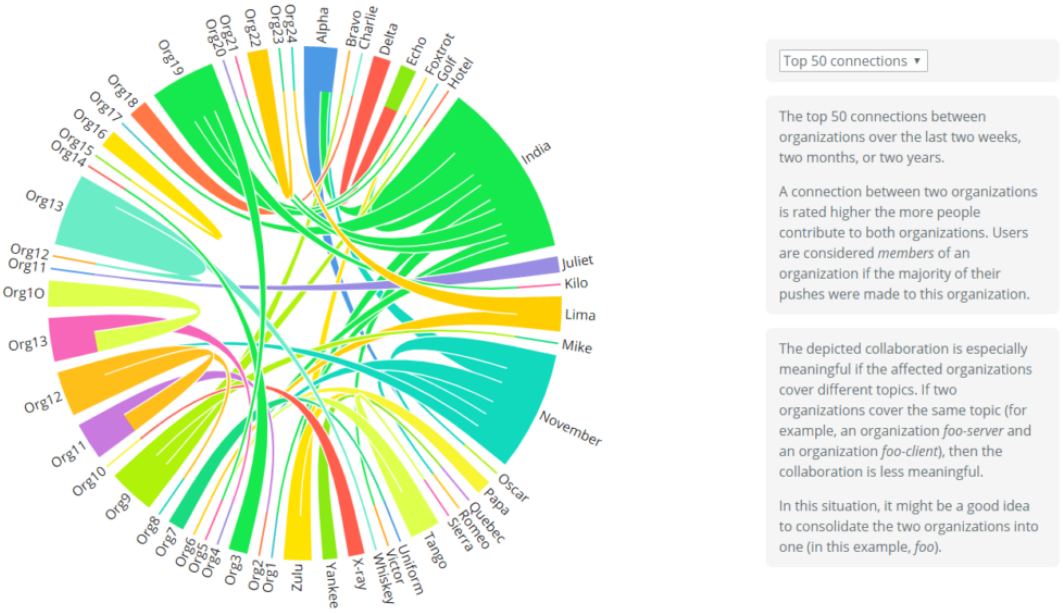


Figure 3.15: The collaboration view of Autodesk Hubble from the demo page [56]. In this case it visualizes connections between multiple organizations.

3.2.4 GitClear

GitClear[7] [57] is a comprehensive visualization collection for analyzing Git repositories. In Figure 3.16 an example is shown that shows the last commits color grouped by issue. It is one of the few commercial solutions that have a real commercial interest and requires a monthly subscription to use. They thereby praise themselves as „The first Developer-Friendly Analytics tool“[8]. The core functionality of it is to help with code reviews and „helps developers self-optimize and cultivates team collaboration“[9]. The main view of GitClear offers an overview of the last commits with their size depending on the commit size and their color depending on the issue linked. An example of this

[6]Autodesk Hubble: `https://autodesk.github.io/hubble/` (Last accessed 07.06.2024)
[7]GitClear: `https://www.gitclear.com` (Last accessed 07.06.2024)
[8]Slogan of GitClear found on their website
[9]Qoute from GitClear found on their website

view can be seen in Figure 3.16. Additionally, it is possible to interact with every commit to get a more detailed overview of what happened. GitClear also gives insight metrics about the quality of an issue and an overview of current and past pull requests. Similar tools on the market provide a similar set of features like Pluralsight Flow[10] [58], Code Climate[11] [59], Waydev[12] [60] and LinearB[13] [61]. All those are visualization collections targeted to be used in a commercial setting.

Figure 3.16: The main view of GitClear shows all last commits grouped and color-coded by issue. It also scales the size of each commit according to the number of changes. This example shows a visualization of the Binocular repository with the free demo version of GitClear [62].

3.2.5 CHAOSS

CHAOSS[14] [63] is a project by the Linux Foundation. It is designed to create metrics to better understand the open-source community. On the base of those metrics, there are multiple visualizations and dashboards built on top of it. The first notable visualization on top of CHAOSS is the CHAOSS Dashboard from GrimoireLab[15] [64]. In Figure 3.17 an example is shown of the Chaoss Dashboard by GrimoireLab. An example of this dashboard can be seen in Figure 3.17. Another dashboard built on top of CHAOSS is 8Knot[16] [65]. 8Knot was not fully functional when this thesis was written but it promises useful insights and visualizations for the future.

[10] Pluralsight Flow: `https://www.pluralsight.com/product/flow` (Last accessed 07.06.2024)
[11] Code Climate: `https://codeclimate.com` (Last accessed 07.06.2024)
[12] Waydev: `https://waydev.co` (Last accessed 07.06.2024)
[13] LinearB: `https://linearb.io` (Last accessed 07.06.2024)
[14] CHAOSS: `https://chaoss.community` (Last accessed 27.07.2024)
[15] GrimoireLab: `https://chaoss.github.io/grimoirelab/` (Last accessed 27.07.2024)
[16] 8Knot: `https://metrix.chaoss.io` (Last accessed 27.07.2024)

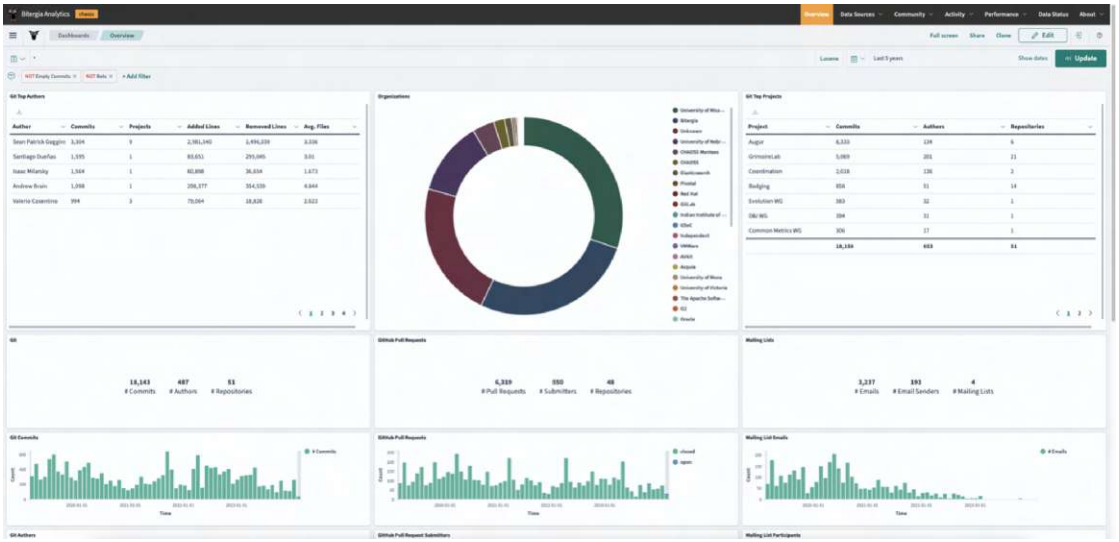


Figure 3.17: The CHAOSS dashboard from GrimoirLab and Bitergia Analytics [66] shows an overview of multiple open-source git projects.

3.3 Independent Open-Source Visualizations

There are also some independent open-source visualizations not developed by a commercial institution but rather by small independent teams. As with other open-source projects [67], [68] their full source code is available on GitHub or a similar platform. Open-source projects give the user the option to view and manipulate the source code itself. They don't rely on promises like closed-source projects and can be improved by anybody.

3.3.1 Gitinspector

Gitinspector[17] [69] is one of the more well-known Git analysis tools. In Figure 3.18 an example of gitinspector is shown. It was developed in Python but due to the abandoned state of the project since 2016 it is outdated. After mining, gitinspector can export its results to many different output formats such as creating an interactive HTML page. This makes it easy to integrate into an automated workflow to analyze a project and host the results as a webpage. Gitinspector mainly shows aggregated views and statistics about a single repository. It is especially good to compare different authors within a single project against each other.

3.3.2 Gource

Gource[18] [70] is a tool to display Git repositories as a force-directed animated tree [71]. Its main goal is to visualize the development story of a repository. Gource displays every

[17] Gitinspector: `https://github.com/ejwa/gitinspector` (Last accessed 02.08.2024)
[18] Gource: `https://gource.io` (Last accessed 07.06.2024)

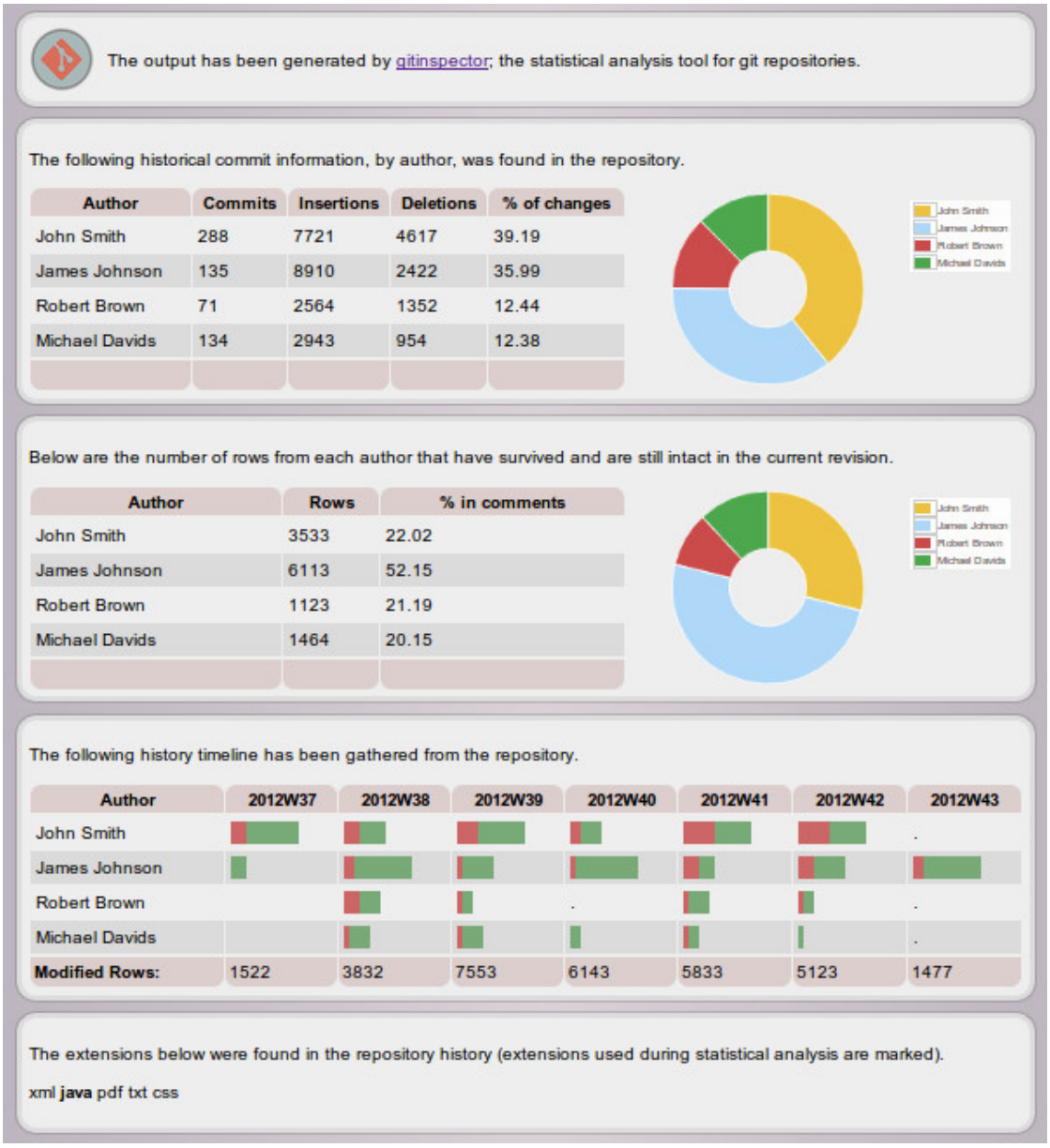


Figure 3.18: Gitinspector [69] is used to quickly analyze basic metrics of a git repository.

file in the file structure as a node in the tree and the branches of the tree visualize the folder structure. Additionally, it shows the authors working on a project flying around from file to file when they worked on it during the development history. As the output, Gource creates a video file of the animation that could be used in presentations or similar. A similar scientific approach to Gource can be found in „code_swarm: A Design Study

in Organic Software Visualization“ [72] and CodeFlower[19] [73]. Figure 3.19 shows a snapshot of such a video created by Gource that visualizes a version-controlled repository.

Figure 3.19: Gource [70] is used to create animations of how a repository evolved by visualizing the project structure and what authors are working on it over time.

[19] CodeFlower: `http://www.redotheweb.com/CodeFlower/` (Last accessed 07.06.2024)

CHAPTER 4

Requirements

The next few sections will go through all the requirements needed before the creation of the prototype in Chapter 5. At first the Current Limitations (Section 4.1) will be described followed by the Research Questions (Section 4.2) this thesis is set out to answer. After that, a short section will follow explaining the Methodology used to answer the Research questions (Section 4.3). As the last part of this chapter the design process of the prototype (Section 4.4) will be described and evaluated (Section 4.4.3) with a group of three experts.

4.1 Current Limitations

Currently, most of the version-control visualization in existence use 2D graphics and most of the time they only show aggregated views or statistics. To obtain a comprehensive project overview, developers frequently need to depend on various visualizations and manually integrate the data. Additionally, they need a deep knowledge of how version control systems work and the project itself to interpret the visualizations correctly. This uses a lot of time that could be used otherwise.

Some visualizations try to show a broader overview of a repository with all its facets. But visualizations like those still often have one glaring issue. They often leave out necessary data when the person analyzing the repository wants to go more in-depth. For example, if the user is interested in both the broader overview and file-level information. This results as already described above in a need to use multiple visualizations at once. One for the broader overview and one for the details at file level. Still, most VCS visualizations today only show statistical and aggregated insights into the repository as described in Chapter 3.

Another problem is that visualizations that harness the capabilities of 3D graphics often use quite outdated visualization methods. That is caused because those visualizations

are not getting developed that frequently. Also, most of the time they get developed in a research setting where state-of-the-art graphics are not that important. For that kind of project showing feasibility is far more important than state-of-the-art graphics. Computer graphics currently take huge leaps forward year after year in their fidelity and performance so a state-of-the-art implementation could show more and more data in a more refined overview. Therefore, a visualization like this was chosen as part of this thesis.

4.2 Research Questions

To guide the research and development conducted as part of this thesis a set of research questions were designed. Those questions define the core aspects of this thesis that are set out to be answered through the next chapters. As a recall, they were already defined in Section 1.3 and are as following:

- **RQ1a:** How do developers analyze the history and trends of a repository?
- **RQ1b:** How do developers rate the benefits of the proposed visualization?
- **RQ2:** What visualization and interaction methods are needed to show a holistic view of a code repository?
- **RQ3:** How suitable is the visualization to analyze the proposed benefits regarding source code evolution, code hotspots, code ownership and code expertise?

With those research questions in mind, it is possible to show a coherent picture of the feasibility of a holistic three-dimensional version control system visualization. In the following section, the methodology will be described which is used to give answers to those questions.

4.3 Methodology

To answer the Research Questions defined in Section 4.2 a multi-step process was chosen. At first, a prototype was designed (Section 4.4) for how a holistic three-dimensional version control system visualization could look. This not only includes the basic structure of such visualization but also how and what data can be encoded. That design was then evaluated by a group of three experts through interviews (Section 4.4.3). It is necessary for design evaluation to create visual designs as close as possible to the prototype because most of the interviewees were not familiar with a concept like the one designed. Following the interviews the prototype was implemented in Unity3D and based on the already existing mining data from Binocular (Chapter 5). It was chosen to not implement a new mining component as part of the prototype because of the already complex implementation of such visualization and the redundancy of such mining implementation.

The last step of the process was the evaluation of the prototype through a scenario-based expert interview (Chapter 6). Here the experts are introduced to the developed prototype and asked questions about the usefulness and features implemented in the prototype and if they fulfill the defined needs. How this process is structured and which method applies to which research question can be seen in Figure 4.1. The expert interviews were chosen over other evaluation methods for both the design evaluation as well as the end evaluation because of the deeper insights experts could give to the topic. Due to the scale of the research in this thesis and the expected limitations, it was deemed that a total of six experts was sufficient for this type of evaluation. The main focus hereby was the evaluation of the functionality and usefulness of the prototype in extracting and displaying the required data instead of actual usability, which could be part of future iterations.

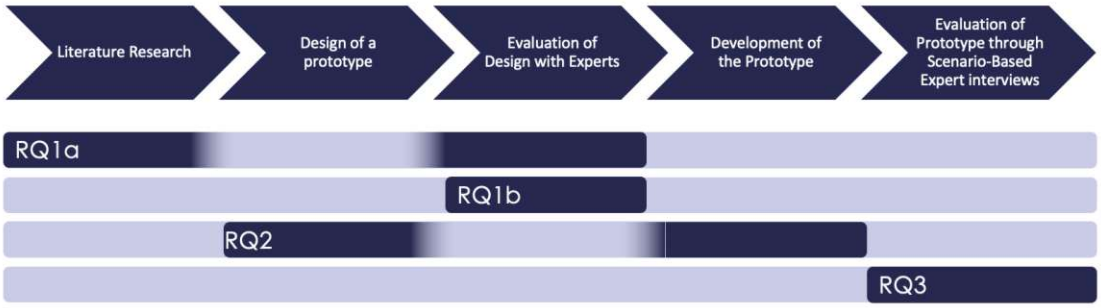


Figure 4.1: Methodology applied as part of this thesis with indicators of which research questions are getting answered by which method.

4.4 Design

The following chapters are about the process of designing the visualization from the first thoughts to the final design. During the design process, it was always clear that the visualization would harness 3D graphics because of the huge amount of data it should display. Also due to prior experience and to simplify the design work in the three-dimensional space all designs through the next sections were done in Blender. The following three sections will go through the main aspects of the design process. The first section (Section 4.4.1) will be about the first ideas and how they were created. The second section (Section 4.4.2) is all about refining the design and adding additional data and functionality. At last, the third section (Section 4.4.3) will be all about the evaluation of the design with three experts.

Blender

Blender[1] [74] is a comprehensive, free and open-source 3D modeling software. It is primarily used to create 3D models, animations, and visual effects and can even be used to create complete animated movies. In Figure 4.2 the interface of Blender is shown with the first design of the prototype loaded.

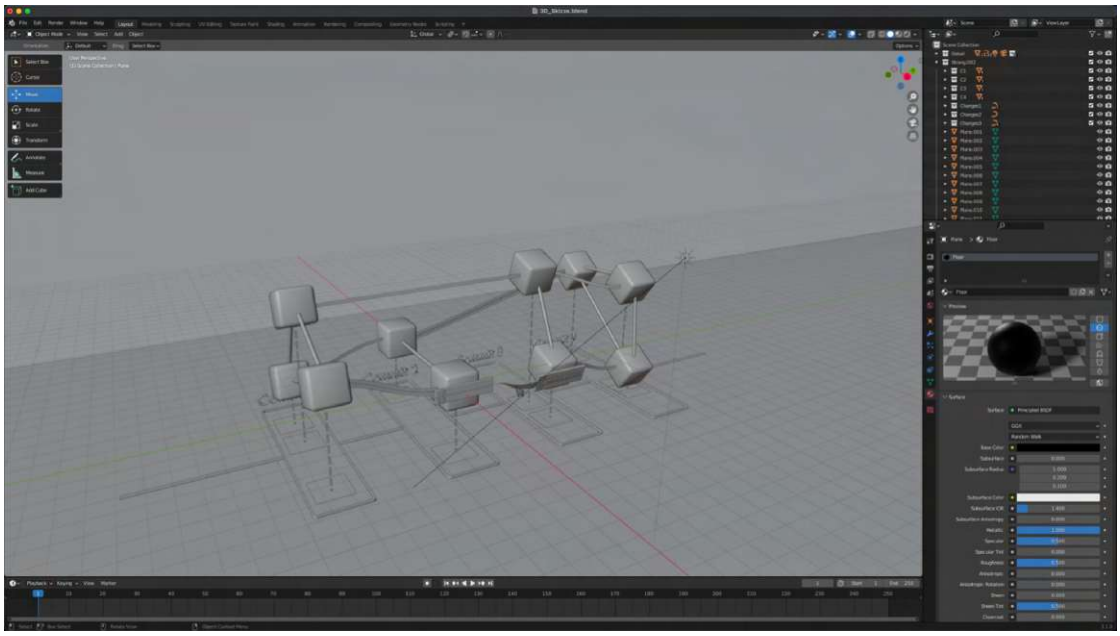


Figure 4.2: Interface of Blender with the first design loaded.

In the context of this thesis, Blender plays a crucial role in the development process by facilitating the creation of three-dimensional concepts for the final prototype. The large toolset Blender provides to create and manipulate 3D objects vastly accelerates the creation of such designs.

One of the biggest advantages of Blender is because it is free and open source a large community has been built around it. This helps immensely in finding tutorials, examples, plugins and other kinds of helping resources.

4.4.1 First Design

The most important thing about designing the visualization is how to create a view where all branches and all versions of a file can be displayed at the same time in the same space. This is a problem since in a huge repository there can be a lot of branches and a lot of files with changes on every branch. To make the structure more understandable for the user it should also include the file and folder structure. A Comprehensive structure like this makes it easier to understand when and how files changed and where the file

[1]Blender `https://www.blender.org` (Last accessed 07.06.2024)

tree activity happened. For the first sketch, the main focus was on creating a simplified version with just one folder with three files in it. To visualize it the idea is to display it like the common representation of a DNA double-helix structure first proposed by James Watson and Francis Circk in 1953 [75]. The idea behind the double helix approach was to create a radial display where files that belong to one folder are visually connected. Also, at this point of the design, it was envisioned to twist the branch so that each file has the equal possibility to get displayed in front of the structure through the development cycle.

The idea hereby was to display the files as nodes and connect them like base pairs in a DNA helix-like structure. One plane hereby displays one commit. Those planes contain file nodes for all files changed within the commit. The connections between files within a commit and from one commit to another can be used to encode different information. For example, Figure 4.3 shows how such a structure can look like displaying four commits with three files whereas not every file got modified in each commit. The connections between commits show if additions or deletions happened. The thickness of those connections can also encode additional information like the number of additions and deletions rather than just if additions or deletions happened. Additionally, it is shown how an exemplary interaction could look like where, if you hover over a connection, a detailed view of the changed hunks is shown including the source code that was changed. Even if not part of this design there can even be additional information encoded in this structure by color coding the file-to-file connection within a commit. At this point, those connections are used to make it easier for the user to recognize which files belong to the same commit.

4.4.2 Refined Design

With the basic structure defined in the previous section (Section 4.4.1) the next challenge was to design a way to structure folders and files within the helix file-commit structure. To accomplish that, it was chosen that each folder would be displayed as an individual strand. Those individual folder strands then get nested within each other to form bigger strands. This results in a structure similar to a radial tree where the whole file tree is encoded within nested strands. All the files and folders within the same level are also connected with each other to make it easier for the user to find files that belong to one file tree within the three-dimensional structure. In figure 4.4 a cross-section is shown of how a nested file structure can look like. Every gray rectangle stands for a file and every green ring is a folder. In figure 4.5 it is shown how a structure like this could look with two commits that are connected like mentioned in Section 4.4.1. Additionally, in this figure each strand twists itself and all nested branches. At this point in the design the intertwined twisting of the strands was still part of the project which was later dropped because of time constraints and because the view got more confusing when visualizing bigger file systems with many commits.

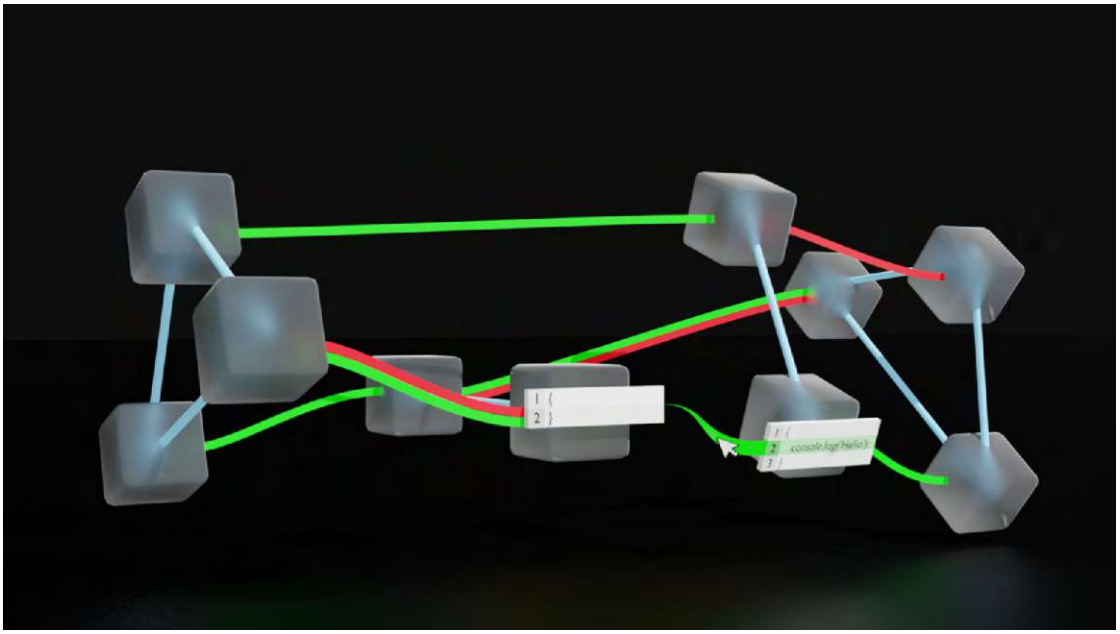


Figure 4.3: First 3D Design with a simple DNA Helix-like structure and 4 commits is with just 3 files. The connections show how many additions and deletions happened from one commit to the next and the interaction shows an exemplary hover where you can see a detailed change overview with the changed source code.

Additionally, as part of this design step, it was proposed to also encode additional data corresponding to each file within the file nodes. This can include addition/deletion, ownership and commit data as shown in Figure 4.6. In theory, coding different metrics may result in patterns that can be analyzed by the person using the visualization. This is comparable to other forms of Big Data [76], [77]. It can also help the user to get a quick overview of a project and find hotspots more easily.

As an example, by color coding the file nodes according to the person who committed them and filtering to only one committer, it can be easy to get an overview of which parts of a project a developer was involved in and when in the development timeline developers were active. Color coding according to the current ownership at the time of the commit can help to analyze how the ownership changed over the development of a repository as seen in Figure 4.6c.

Additionally, when coloring the file nodes according to the number of changes they received within a commit it may be possible to find hotspots within the development cycle that can point to hard-to-implement features or big changes in the project. Those hotspots may result in bugs in the future. This can be seen in Figure 4.6b which shows an example of how such Code Hotspots could look in the context of the designed prototype. Features like this can vastly accelerate the analysis of version-controlled software repositories which reduces the time people need to spend on such tasks.

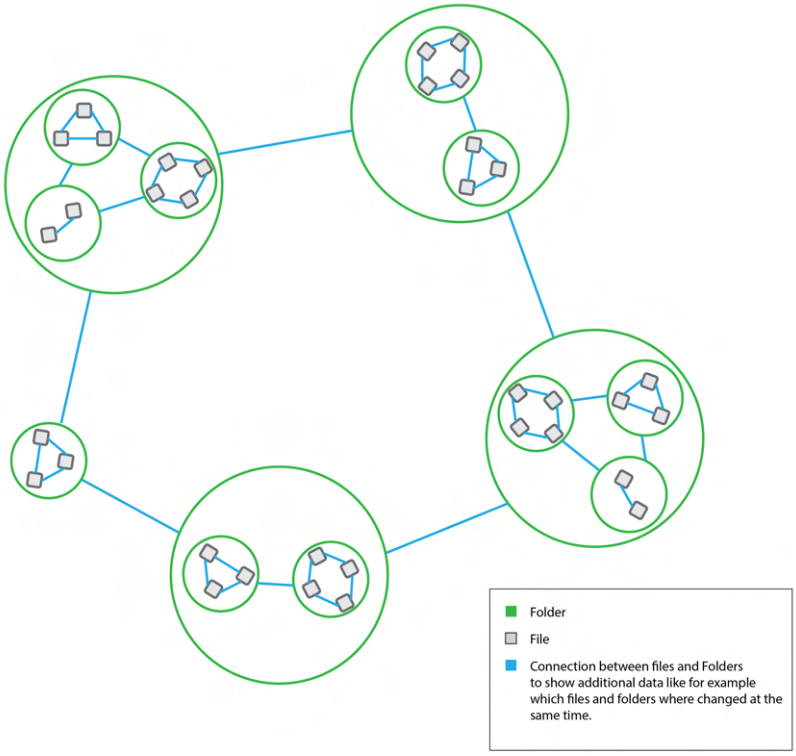


Figure 4.4: Design of how a cross-section could look with multiple folders nested within each other. The individual files in this design are colored gray and the folders are displayed as green rings.

4.4.3 Design Evaluation

To evaluate the design described in the section above a small group of three experts were interviewed. To gather all the information necessary, the interviews were sectioned into six distinct parts. Those range from classifying the experience level of the experts to thoughts about precise features that could be part of the prototype. As part of those sections, multiple questions were formulated so that every interview follows the same overarching structure. All of those sections and the questions asked as part of them are

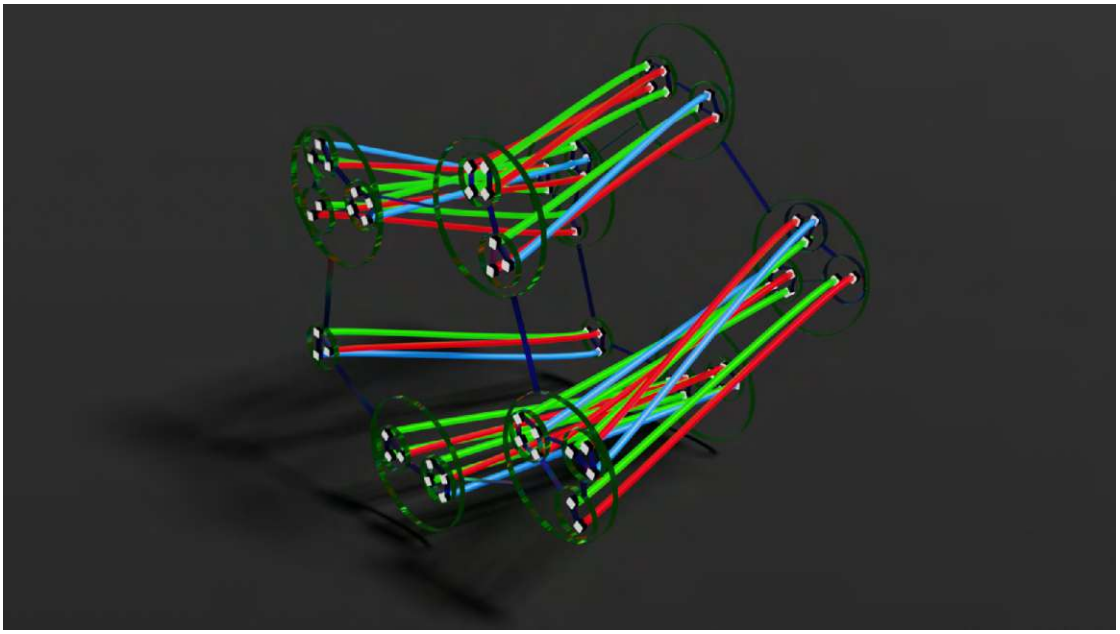


Figure 4.5: Here you can see how the nested structure from figure 4.4 could look like in 3D. In the picture, you can see one example commit where additions (green), deletions (red) and changes (blue) were made for the files in the project

necessary to draw a whole picture of the feasibility of the proposed design, a requirement for the development of the prototype in Chapter 5. The complete interview guide for the design evaluation can be found in Appendix A.

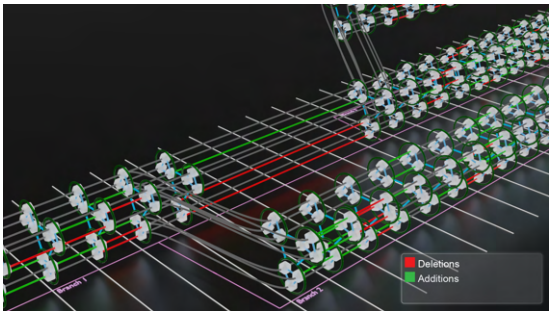
Demographic & Experience

The demographic of the interviews consisted of three experts who all rated themselves in the interview as experienced with version control systems with an average rating of four point three out of five. Also, all of them have already used some kind of tool to analyze version-controlled repositories and all three of them rated their experience with such tools three out of five. According to them, they all use such tools regularly to analyze repositories such as gitinspector[2] [69] or even Binocular[3] [78] which this thesis builds on and uses as its mining component.

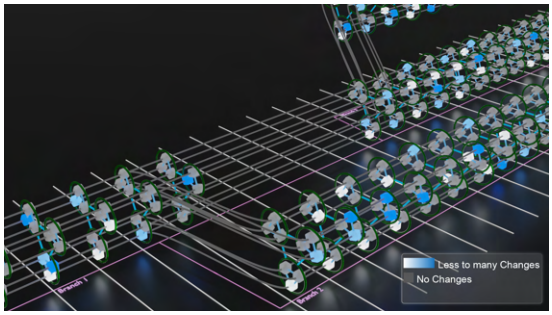
3D Visualizations of Version-Controlled Repositories

In this part of the interview, the participants were asked about their experience with three-dimensional visualizations of version-controlled repositories in particular. Two out of three were already familiar with the existence of three-dimensional VCS visualization.

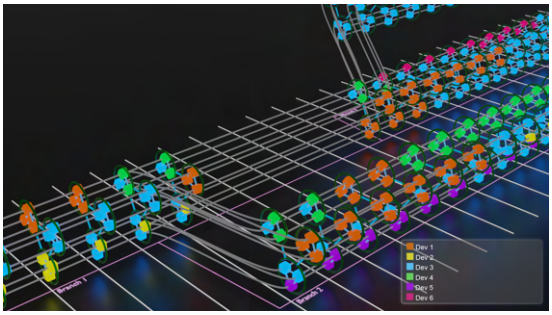
[2]Gitinspector: `https://github.com/ejwa/gitinspector` (Last accessed 07.06.2024)
[3]Binocular: `https://github.com/INSO-TUWien/Binocular` (Last accessed 07.06.2024)



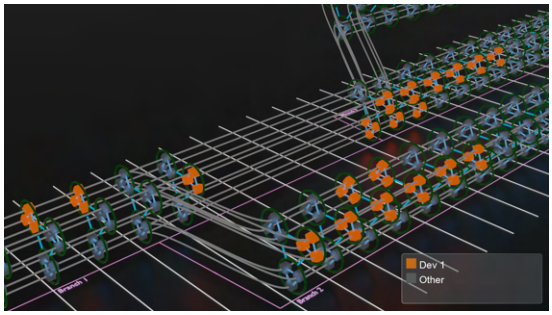
(a) Code Evolution



(b) Code Hotspots



(c) Code Ownership



(d) Code Expertise

Figure 4.6: In the four sketches above it is possible to see how the different visualization modes could look like with three different folders with three files each. **(a):** *Code Evolution* - In this visualization mode, the connections between the commits show if additions or deletions happened. **(b):** *Code Hotspots* - In this visualization mode, the files are colored depending on how many changes happened in this commit. **(c):** *Code Ownership* - In this visualization mode, the files are colored depending on who held the ownership of the file at the point of commit. **(d):** *Code Expertise* - In this visualization mode, only the files are colored that a specific developer owns the code ownership.

All three participants stated that they think that a three-dimensional visualization can give additional insights into a repository. The kind of three-dimensional visualization most of them are familiar with were code cities as already described in Section 3.1.1. The two participants who already were familiar with three-dimensional VCS visualizations also stated that they already had used such visualization. The next part of the interview was all about the possible advantages of three-dimensional visualizations over two-dimensional ones. Four advantages were provided to the participants and they were asked to sort after their importance to them (Figure 4.7). The highest rated advantage according to the participants of the interview is that more data can be displayed at once. Following that the next biggest advantage was that relations could be displayed easier. As an additional advantage of three-dimensional visualizations over two-dimensional ones one participant also mentioned that in a three-dimensional space, it can be easier to overlap data in the visualization if necessary compared to their two-dimensional counterpart.

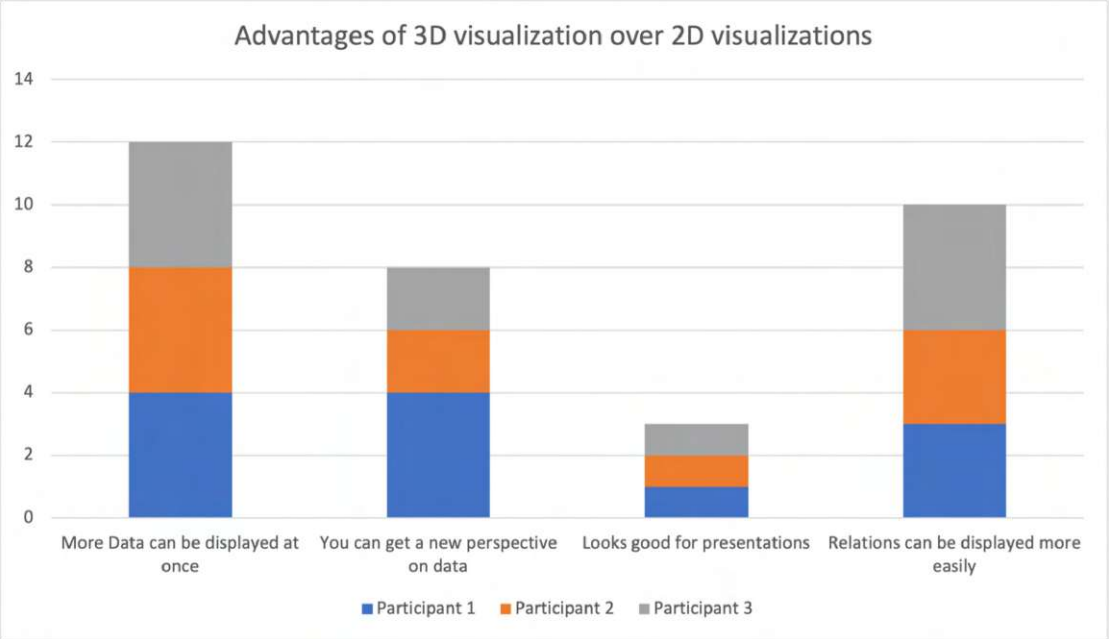


Figure 4.7: This chart shows the four different proposed advantages that 3D visualization may have over 2D visualizations and how the participants of the interview rated them. Every color visualizes one interviewee and the rating they gave each filtering option. (higher is better)

Evaluation of Benefits

After the introduction to my design, the participants were shown four different possibilities which they should rate according to how useful they think they are. Additionally, they were asked about what they think about the different ways of encoding data within the proposed visualization (Figure 4.8). The highest-rated option of encoding data was the Code Hotspots Visualization as seen in Figure 4.6b followed by Source Code Evolution (Figure 4.6a) and Code Expertise (Figure 4.6d) with the same rating on the second place. The Code Ownership (Figure 4.6c) received the lowest rating of usefulness from the participants. In general, the ratings for the four provided possibilities were all close to each other. When asked about the specifics all of the participants preferred the hotspots view where the files are colored in different shades above the evolution view where the connections are colored according to the number of additions or deletions. They stated that the evolution could be overwhelming and that the hotspot variation is much simpler to understand. Especially when projects get bigger than the ones shown in the designs. They think that the connections could end up being too much. When asked about the ownership view compared to the expertise view the consensus was that similar to the comparison before. The view with all developers could be overwhelming, especially with bigger projects so they preferred the variation where only the ownership of one developer is highlighted. It was also mentioned by one participant that displaying code ownership

as presented in the evaluation could become problematic. This is because people could use it to analyze the performance of developers and who is necessary for a project or who is not.

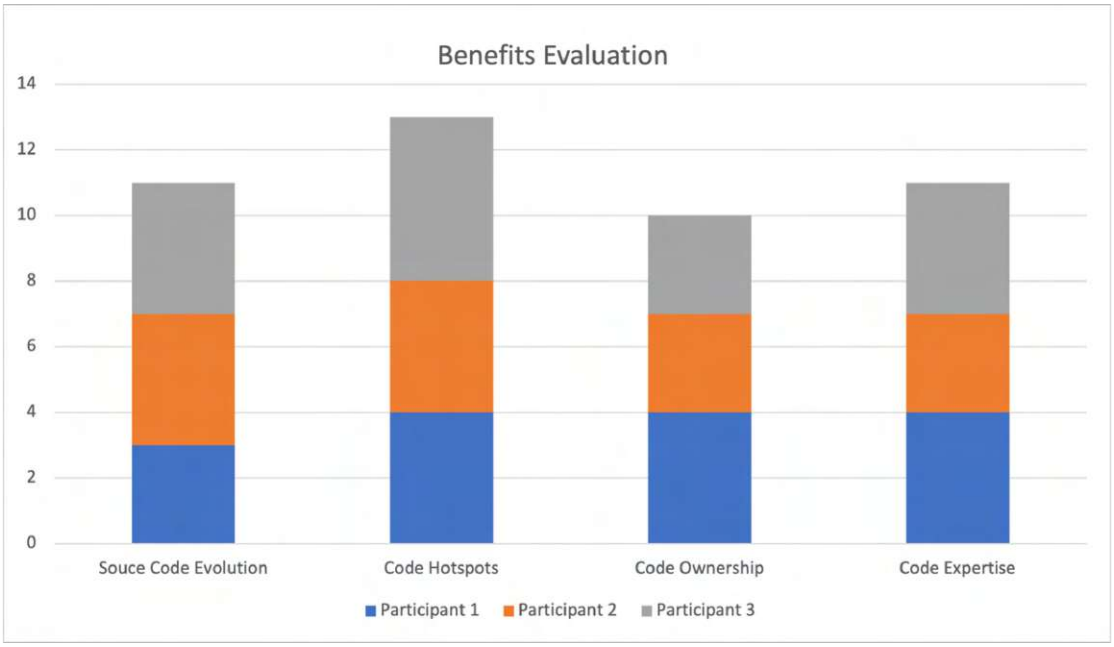


Figure 4.8: This chart shows the four different benefits the proposed visualization should bring and how the participants of the interview rate their usefulness. Every color visualizes one interviewee and the rating they gave each filtering option. (higher is better)

Data Filtering & Configuration

The next part of the interviews focused on filtering and configuration options for the proposed visualization. At first, the participants were presented with a list of five possible filtering options that they should rate after what they think is most important (Figure 4.9). The highest-rated option in this ranking was to filter after „branches“ followed by „specific files“, „developers“ and „date and time“. The lowest rated option by far was „developer contribution in percent“. It was described as a filter that shows only developers that, for example, contributed more than 20 percent to the project which can help to find active contributors. Next, the interviewees were asked if they would prefer if the commits were distributed equally or on a date-time scale. Two out of three stated that they would prefer them to be equally distributed while all of them mentioned that it would be good if both options were available. Additionally, the participants were asked if they could think about other filtering options that would be fitting. Two out of the three participants mentioned that a setting to filter where code ownership has changed to identify ownership transfers and refactorings would be interesting. Other ideas the participants mentioned are as follows:

- Number of Code Lines or Files Changed/Significant Changes: This method of filtering can be used for identifying refactorings or phases within the history of the project where a lot of changes happened.
- Historic Evolution of Code Blocks: This method can be used for tracking the evolution of a specific code block and to gather more information about its historic changes.

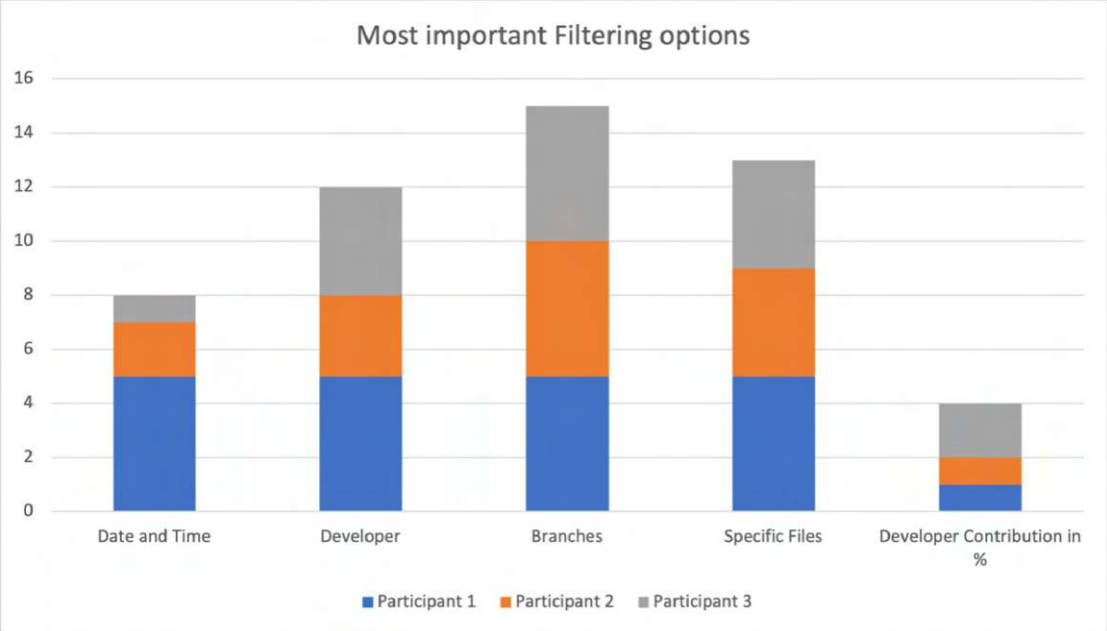


Figure 4.9: This chart shows the five different proposed filtering options the participants should rate in their interviews. Every color visualizes one interviewee and the rating they gave each filtering option. (higher is better)

Final Thoughts

The last part of the interview was about what the participants thought in general about the proposed visualization after all the information they got through the previous sections. All three liked the concept but mentioned that there can be some problems that could surface when developing visualization. The biggest problems they mentioned were:

- False assumptions could be made if it is not completely clear what data is currently displayed. This can be a problem, especially with code ownership data. This can result in a steep learning curve for a visualization like the one proposed. With such a complex visualization it is very important to make exactly clear which information is currently displayed.

- Performance can become a problem with bigger repositories. Advanced 3D graphics need a lot of performance which can stack up fast with huge version-controlled repositories with thousands of files, branches and commits. Especially if the visualization tries to display everything at once like the visualization proposed as part of this thesis.

At last, they were asked what additional data they would like to see within such a visualization. One of the participants mentioned that it would be good to include issue data while another participant mentioned file renaming/moving data would be a good addition.

At last, they were asked if developers would profit from a visualization like that or if it is more interesting for project leaders and similar. All three stated that project leaders and people doing forensics on software projects will profit more from a tool like the one proposed than the average developer.

CHAPTER 5

Prototype

The focus of this chapter will be the implementation of the prototype named Holocular resulting from the combination of the word holistic (all including) and Binocular which is the base of the project and mining data. First, in Section 5.1, there will be an overview of the technologies used to develop the prototype and why those were chosen. Another important aspect that is needed before talking about the development of the prototype is the data that was used as a basis as described in Section 5.2. After that, there will be a detailed description of the development of the prototype separated into four iterations. The first iteration (Section 5.3) will hereby focus on the basic implementation of the prototype. This includes the creation of the basic radial structure. In the second iteration (Section 5.4) the prototype will be refined mainly by improving the way the branch and folder structure gets rendered and created. Another huge point of this iteration is to improve the addition/deletion connection. After that the third iteration (Section 5.5) will introduce methods to highlight and brush data. In the fourth iteration (Section 5.6) everything will come together in a coherent prototype and the final result of this work.

5.1 Technologies Used

To implement the prototype Unity3D (Section 5.1) is used to visualize and transform the data. As the backend, the already available implementation from Binocular[1] [78] is used. Those will give a solid foundation to build the prototype as described previously in Section 4.4.

Binocular is a project of INSO[2] [79] and is used because it already gives a solid base as a mining tool. It already offers a lot of data and visualizations. Binocular is hereby able to mine both local version control data as well as GitHub and GitLab data with

[1] Binocular: `https://github.com/INSO-TUWien/Binocular` (Last accessed 07.06.2024)
[2] INSO: `https://www.inso.tuwien.ac.at` (Last accessed 27.07.2024)

possible more data sources to come in the future. Binocular is an active project that permanently gets improved which in turn also can improve the data quality of the proposed prototype. It also can export all of its mined data as JSON files that can be used in other visualizations. This is the way used in this prototype to import real data into the prototype. For future iterations, it would also be possible to directly access the binocular backend through GraphQL [80] request which Binocular also provides or implement a standalone C# mining component.

For the frontend and main prototype Unity 3D is used. While it is primarily a game engine it can be used for more than that as described in Section 5.1. A major reason to use Unity3D for the visualization is that the author of this thesis was already familiar with developing applications in Unity3D. It also has a huge existing community that offers a lot of help and insights into the development of Unity 3D applications.

Unity3D

Unity3D[3] [81] is a comprehensive 3D game engine that is used nowadays for all sorts of 3D applications. From making movies[4] [82] to industrial visualizations[5] [83] everything that involves 3D graphics can be done with it. It is one of the most widely used 3D engines currently on the market[6] [84] which offers great support for multiple platforms from desktop and game consoles to even the possibility to export HTML5 and JavaScript code for web-based 3D applications. Unity3D also offers the option to write custom shaders as described in „Building Quality Shaders for Unity“ [85] and „Practical Shader Development: Vertex and Fragment Shaders for Game Developers“ [86]. Unity3D herby offers both code-based shaders that have to be written in HLSL (Section 2.5) as well as its proprietary graphical shader editor called Shader Graph (Section 2.6) to simplify the development of comprehensive shaders. In Figure 5.1 the interface of Unity3D is shown with the project of this thesis loaded.

5.2 Data Used

As its base, the prototype developed as part of this thesis uses the data exported from Binocular. Binocular uses an ArangoDB[7] [87] database and can export all its data as JSON Files This includes all kinds of mined data from version control over issue tracking to build data. Binocular provides this data exporting both as parts of its backend as

[3]Unity 3D: `https://unity.com` (Last accessed 02.07.2024)

[4]Unity Virtual Production: `https://web.archive.org/web/20240304204219/https://unity.com/roadmap/virtual-production` (Last accessed 02.07.2024)

[5]Unity Industrial Visualizations: `https://unity.com/solutions/design-visualization` (Last accessed 02.07.2024)

[6]Usage Statistic Itch.io: `https://itch.io/game-development/engines/most-projects` (Last accessed 02.07.2024)

[7]ArangoDB: `https://arangodb.com` (Last accessed 30.07.2024)

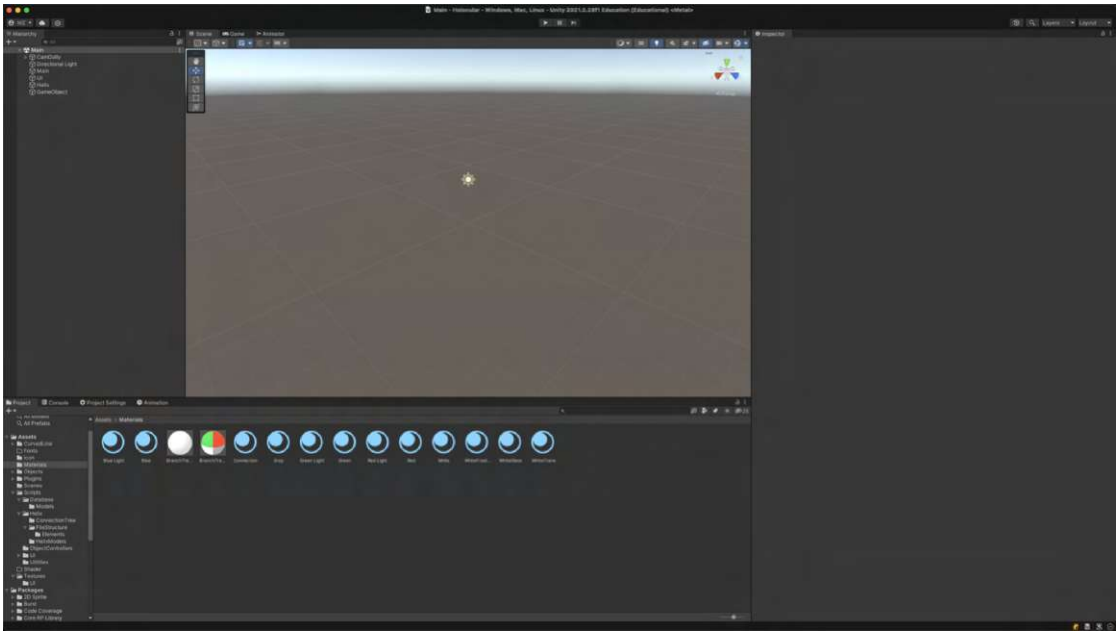


Figure 5.1: Interface of Unity with Holocular loaded.

well as through the frontend. Additionally, it is also possible to use arangoexport[8] [88] to export the ArangoDB database directly. As the focus of this visualization mostly lies on version control systems only the corresponding collections were imported into the prototype. The database collections needed for this visualization and imported from the Binocular export into the prototype are:

- **Commits:** The commits collection consists of all the commit data including among others the sha, commit date, commit message and the number of additions and deletions.
- **Branches:** The branches collection includes the branch data like branch name and if the branch is currently active.
- **Files:** The file collection provides all the data that correspond to a file including the file path.
- **Stakeholder:** The stakeholder collection includes a list of all stakeholders available in the project with their git signature.

Additionally, to the main collections, there were also some relation collections imported into the Prototype. Those are important for connecting the different elements of the

[8]arangoexport: `https://docs.arangodb.com/stable/components/tools/arangoexport/` (Last accessed 30.07.2024)

main collections accordingly. Some of those relation collections also include additional data that belongs to the specific combination of two collections. The relation collections needed for this prototype are:

- **Commits-Commits:** The commits-commits relation connects parent and child commits with each other to create the Git commit tree.
- **Commits-Files:** The commits-files relation provides a connection for every commit to all the files that were changed as part of this commit. This relation additionally includes data like the line count at the time of the commit, the number of additions and deletions and a list of change hunks. A change hunk gives insight into which lines have changed.
- **Commits-Files-Stakeholders:** The commit-files-stakeholder relation provides a connection between the commit-files relation and the stakeholder collection. Those connections include the ownership data at the point of a commit. Additionally, it includes for every developer that partly owns a file at a commit data like the number of owned lines and a list of ownership hunks. Those ownership hunks give deeper insights into which lines were owned by the stakeholder at a specific commit.
- **Commits-Stakeholders:** The commit-stakeholder relation connects every commit with the corresponding stakeholder that did the commit.

5.3 Iteration 1 - Basic Implementation

The first step after setting up the basic Unity3D project import the JSON files that Binocular produces. As Unity uses the C# programming language it cannot natively work with JSON files like for example JavaScript. To import JSON data into C# objects it is necessary to create serializable classes that mirror the structure of the JSON file. After that, the file can be converted into an object of that class that can be used like any other C# object with the data of the JSON file inside of it. Below there is an example of how this may work. In Listing 5.1 an example JSON code can be seen that represents a simple JSON array of persons with a name and age. In Listing 5.2 an equivalent C# class can be seen. The class is marked as serializable so that persisted data like in this case JSON files can be imported or exported from objects of this class. This is further explained in the .NET Documentation[9] [89]. The JSON files must be read as strings and finally, in Listing 5.3, a code snipped is shown that converts the JSON string into full C# objects.

[9]Microsoft Learn Serialization: `https://learn.microsoft.com/en-us/dotnet/standard/serialization/` (Last accessed 01.06.2024)

```
[
    {"name":"name1","age":25},
    {"name":"name2","age":30},
    ...
]
```

Listing 5.1: Example JSON file of a list of persons with a key for the name and the age of the person.

```
[System.Serializable]
public class Persons
{
    public Person[] persons;
}

[System.Serializable]
public class Person
{
    public string name;
    public int age;
}
```

Listing 5.2: Example C# class that mimics the structure of a JSON file.

```
...
Persons personsObject = JsonUtility
    .FromJson<Persons>("{\"persons\":" + JSONCode + "}");
...
```

Listing 5.3: Example C# class that converts a JSON string into a C# object.

After the data is available as a C# object the first step is to generate a file node for every change as seen in Figure 5.2. This was mainly done to check if the data got read correctly. All commits are displayed on a single axis and intented depending on which branch they belong to. On the other axis, all files that changed within the commit are getting displayed. The order of the files on the axis is dependent on when they got added to the project which in itself already creates an interesting shape. At its core, this creates a sort of heatmap where analyzable patterns already emerge. At this point, it was also experimented if the files that haven't changed within the commit also get displayed but grayed out. It was decided against it because it resulted in too much noise which distracts from the important parts of the commits. In addition, the performance drop was too high to justify the change.

Also at this point, the layout of the GUI was already decided. On the right side of the visualization, a retractable debug panel was designed that shows information about the state of the visualization. This includes hints of how to change the mouse sensibility and shows the current frames per second. Additionally, this first iteration of the debug panel

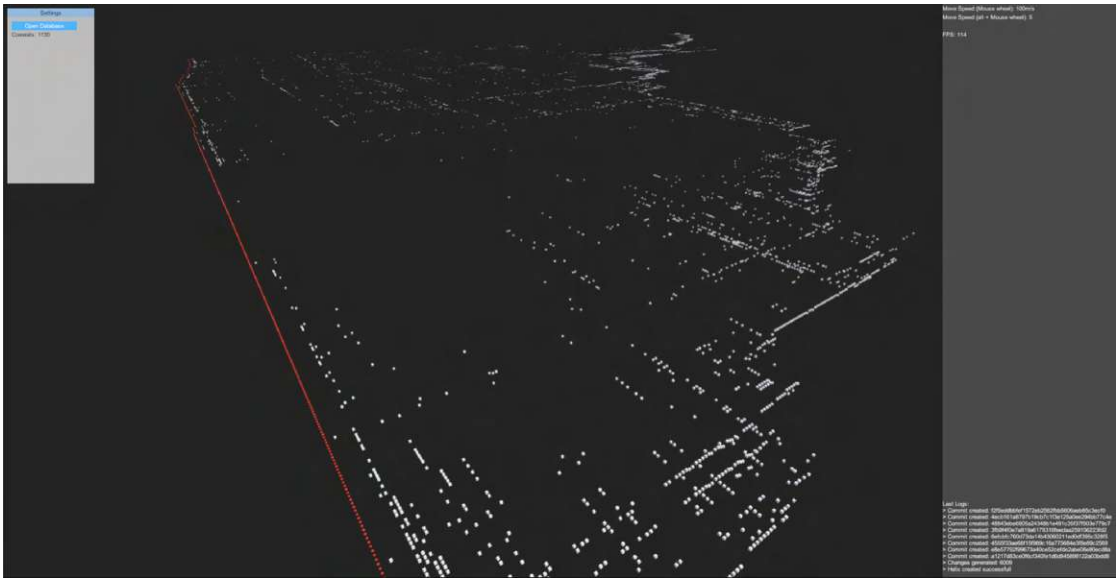


Figure 5.2: In this figure an early display of data is shown where only changed files are displayed as a cube on a flat plane where the z-axis represents the commit timeline and the x-axis all the files that changed within the commit are displayed.

already included a custom log viewer on the bottom which shows the activity of the prototype. The rest of the interface is designed as small windows within the application. This was designed to make the GUI more user-customizable. First, only a basic settings window was implemented with a simple button that opens a dialog to select an exported binocular database folder to visualize. For the file dialog, the Runtime File Browser Package[10] [90] from the Unity Asset Store was used. The settings window also already showed how many commits are included in the loaded data in the loaded data which will be extended to more data points in future iterations.

At this point, the decision against the graphical GUI editor[11] [91] from Unity3D in favor of a code-based GUI[12] [92] that gets generated at runtime to be more flexible and so that it is easier to reuse components. Unity3D provides as part of its GUI library a method to simplify the generation and handling of within the application. An example of the code of a basic window can be seen in Listing 5.4 as described in the Unity Documentation[13] [93].

[10]Runtime File Browser: `https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006` (Last accessed 02.07.2024)

[11]Unity UI: `https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/index.html` (Last accessed 26.06.2024)

[12]Unity GUI Reference: `https://docs.unity3d.com/ScriptReference/GUI.html` (Last accessed 26.06.2024)

[13]Unity GUI Window Documentation: `https://docs.unity3d.com/ScriptReference/GUI.Window.html` (Last accessed 26.06.2024)

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class ExampleClass : MonoBehaviour
5   {
6       public Rect windowRect = new Rect(20, 20, 120, 50);
7
8       void OnGUI()
9       {
10          // Register the window. Notice the 3rd parameter
11          windowRect = GUI.Window(0, windowRect, DoMyWindow, "My␣
                ↪ Window");
12      }
13
14      // Make the contents of the window
15      void DoMyWindow(int windowID)
16      {
17          if (GUI.Button(new Rect(10, 20, 100, 20), "Hello␣World"))
18          {
19              print("Got␣a␣click");
20          }
21      }
22  }
```

Listing 5.4: Example C# class that creates a simple movable window with a button inside it.

After the data was made available as C# objects and the design for the GUI was settled the next step was to generate the basic nested ring structure that visualizes the folder structure as seen in Figure 5.3. Although in the design-phase (Section 4.4) the structure was designed twisted to give every file the possibility to be on the front, it was eventually decided to not twist the structure in favor of an easier-to-understand structure. In addition, the nested twisting of the branches would also greatly increase the development complexity. If Holocular gets revised at some point in the future it could be picked up again as it can still yield advantages over the untwisted strand that was developed as part of this work.

At this point, major performance problems surfaced because of a problem of how the individual rings were drawn. To resolve the problems the generation of the rings was improved. In the first implementation, the rings were built out of multiple small lines that connected each node on the ring. Generating and rendering all those small lines was inefficient as all those lines had to be rendered individually as individual objects. This was improved by rendering every folder ring as a single polygon that is rendered through the unity built-in line renderer component. This vastly improved the performance from around 25 to over 200 frames per second on the tested PC with 100 commits loaded.

After the generation of the basic structure was sorted out in the next step, branch and change data were added as seen in Figure 5.4. Here all the branches are spread out

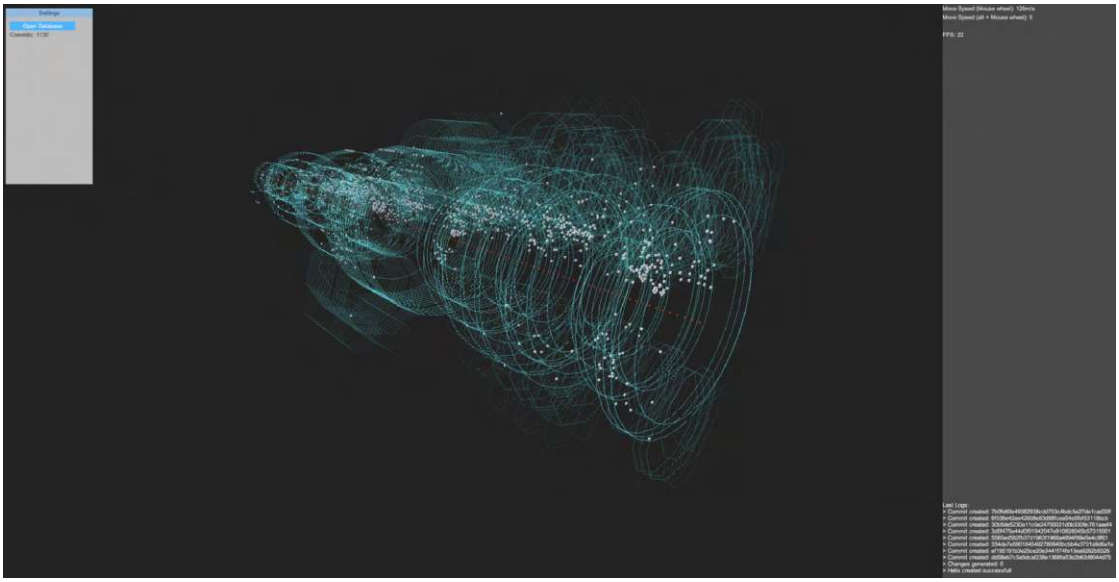


Figure 5.3: First implementation of the nested radial structure that displays folders in blue and files that changed within the commit as white cubes. Every layer is equivalent to one commit.

and run in parallel to the main branch. The resulting structure is similar to a common Git commit tree but in three dimensions to make it more compact. This approach was already described in the design of the prototype in Section 4.4.2.

To offset the branches so that they do not overlap they were placed on invisible rings with equal distribution with always three branches per ring as seen in Figure 5.5. Currently, only three branches are placed on each invisible ring but this could be changed in future iterations if necessary. It would possibly make sense to gradually increase the number of branches of rings that are further from the center main branch. The different rings can also be offset from each other to generate a twisted look as seen in Figure 5.6. This twisted branch distribution was eventually chosen to be used in the prototype because it was easier to distinguish different branches. The way branches are getting distributed is still quite rudimentary in this prototype and can be revisited in a future iteration.

As part of the iteration, a basic implementation of the change data display was also added as part of this iteration step also seen in Figure 5.7. Here, you can already see all the inter-commit file connections from one file version to another. During this development stage, additions and deletions were ignored. Only the number of changes were encoded into a gradient from red (least changes) to green (most changes). At this point, the important thing was to create the basic implementation of the file-to-file inter-commit connections. This view had more common with the hotspot file coloring mode introduced later in Subsection 5.5.1 than with the addition and deletion indicator those connections should resemble.

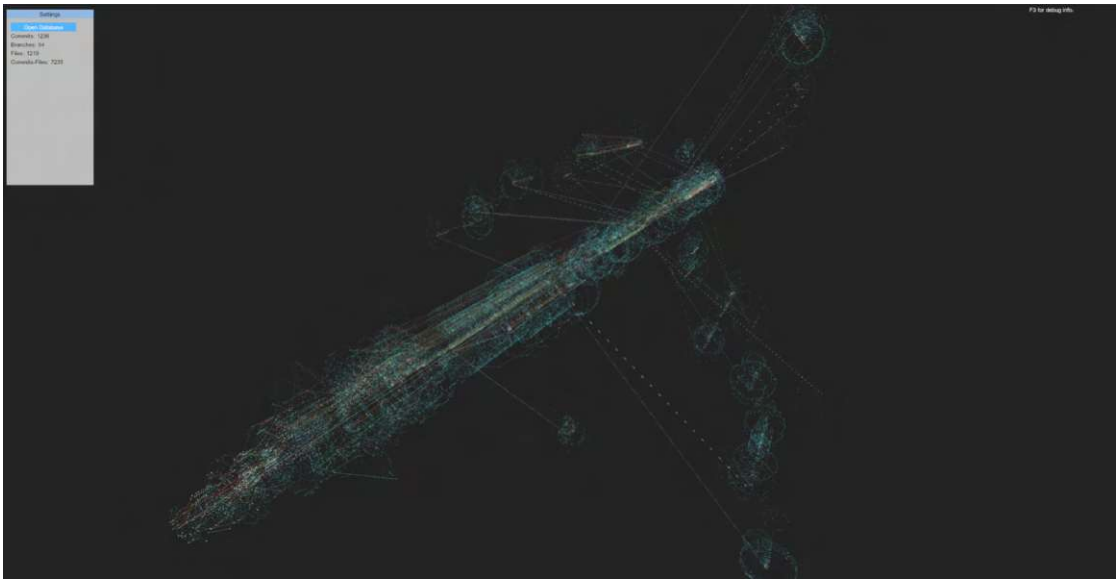


Figure 5.4: First iteration of the full structure if all branches split out. It also already includes the file-to-file connections between commits that are either red or green colored depending on if more additions or deletions happened from one version to the next.

A rudimentary file interaction was also implemented when if a file gets selected a debug message gets shown with the name of the selected file and zooming the camera to the selected file. This was accomplished by detaching the camera from the camera dolly and moving it to a position close to the file node in the three-dimensional space. The camera dolly itself stays on position in the 3D space so that when the camera gets moved again after selecting a file Holocular can automatically reset the camera to the previous position it was located before the zoom. This was done to make the interaction more intuitive.

5.4 Iteration 2 - Refined Implementation

The goal of the second iteration is to refine the already implemented code. This includes mainly the at this point quite rudimentary implementation of the inter-commit change data visualization and the interaction of files. To improve the change connections a custom line and split line renderer was developed. This was done because the Unity intern line renderer is quite limited. Especially when aligning two lines with different thicknesses it can be challenging with the default implementation.

It was also decided that instead of a gradient from green to red the changes would be displayed as a split green and red line with a changing thickness depending on how many additions and deletions were recorded within a commit. In Figure 5.8 a closeup is shown of how the split change lines look like. Additionally in Figure 5.9 a view from further

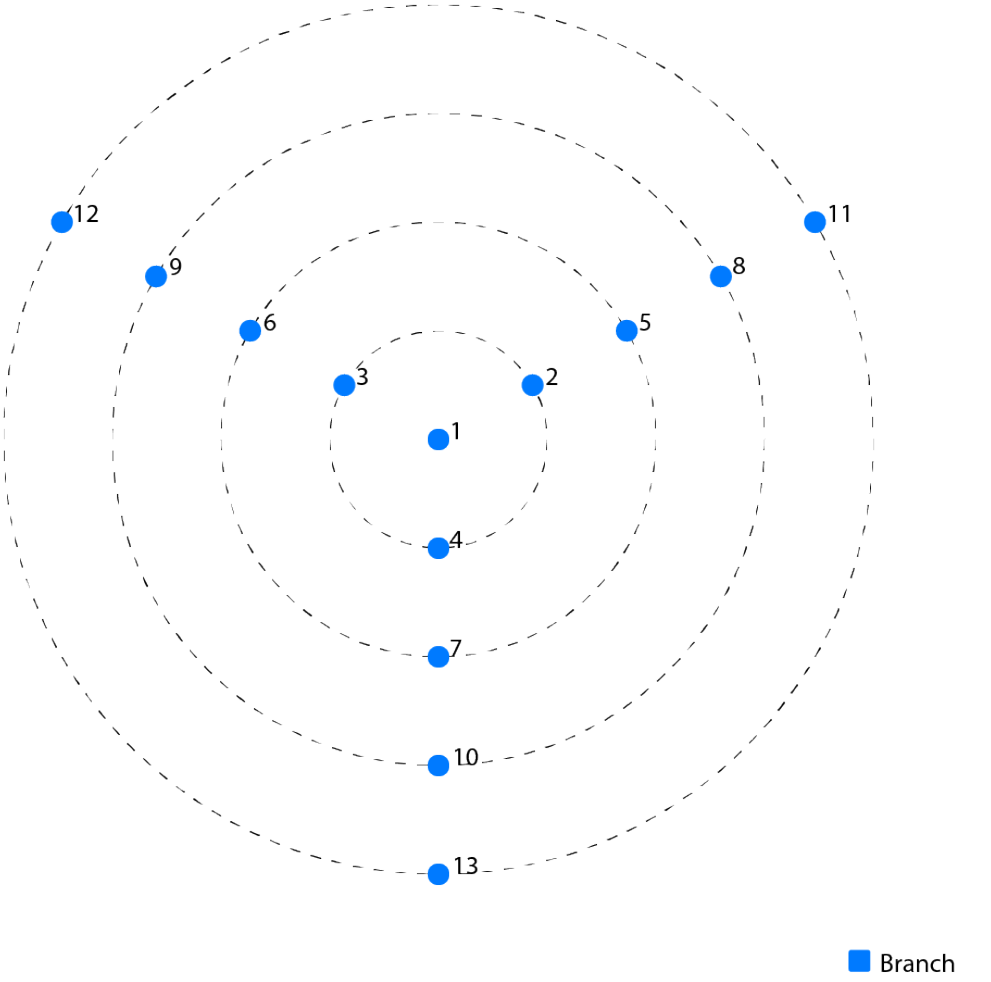
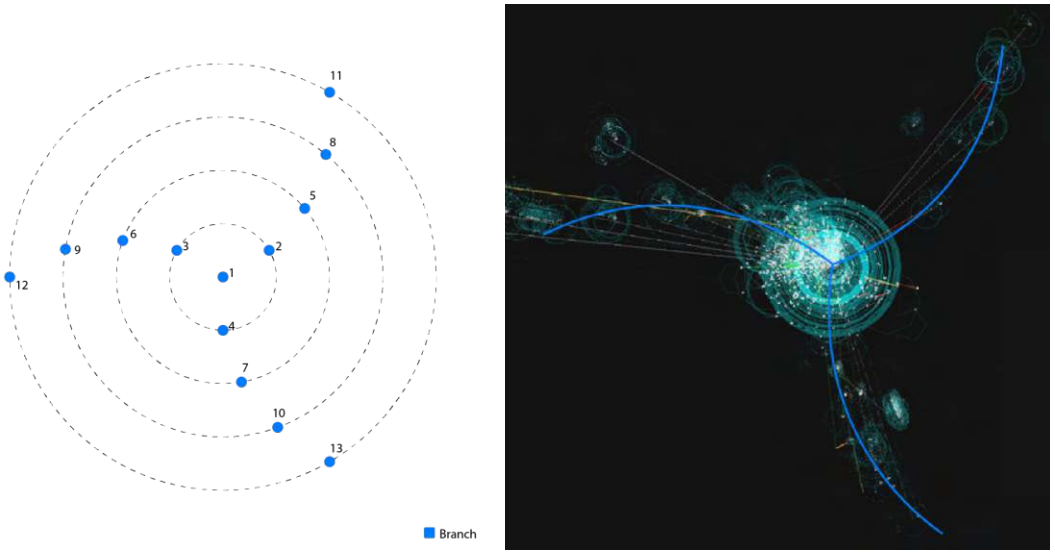


Figure 5.5: Branches get distributed on a slice with blue points representing the branches. The branches get distributed along invisible rings shown as dashed lines. The numbers on the side of each dot represent the order in which the branches get distributed.

away is shown where it is already better visible where changes were made compared to how it was implemented in the first iteration.

Also during this step of development, the project was updated from Unity's basic built-in core 3D renderer to the newer Unity Universal Render Pipeline for more flexibility with shaders and better performance. Some additional experimentation was also done during this development iteration with more advanced „Unity HLSL“ and „Shader Graph“ shader to get closer to the look as shown in the design. Those were ultimately dropped



(a) Twisted Branch Distribution (b) Overlay over Prototype

Figure 5.6: **(a)** shows an example of a twisted offset branch distribution; **(b)** shows how this looks in the prototype

for the sake of performance and replaced by the default unity built-in shaders. Mainly the „Universal Render Pipeline/Simple Lit“ shader was used from this point on.

This was accomplished by creating a custom mesh for each file where all the vertices for each connection are getting added and connected to a line with varying thicknesses that connect all versions of a file. To create a custom mesh Unity needs an array of 3D vectors for the individual vertex positions combined with a corresponding 2D vector UV coordinate array for the texture coordinates and a triangle array that defines the indexes of vertices building the mesh from the previously defined vertices array. This process is further described in general in Section 2.3 and an Example of such implementation can be seen in Listing 5.5 taken from the Unity Mesh Documentation[14] [94]. In the prototype, all those coordinates and indexes are calculated during runtime and can be recalculated whenever settings or data changes.

[14]Unity Mesh Documentation: `https://docs.unity3d.com/ScriptReference/Mesh.html` (Last accessed 27.06.2024)

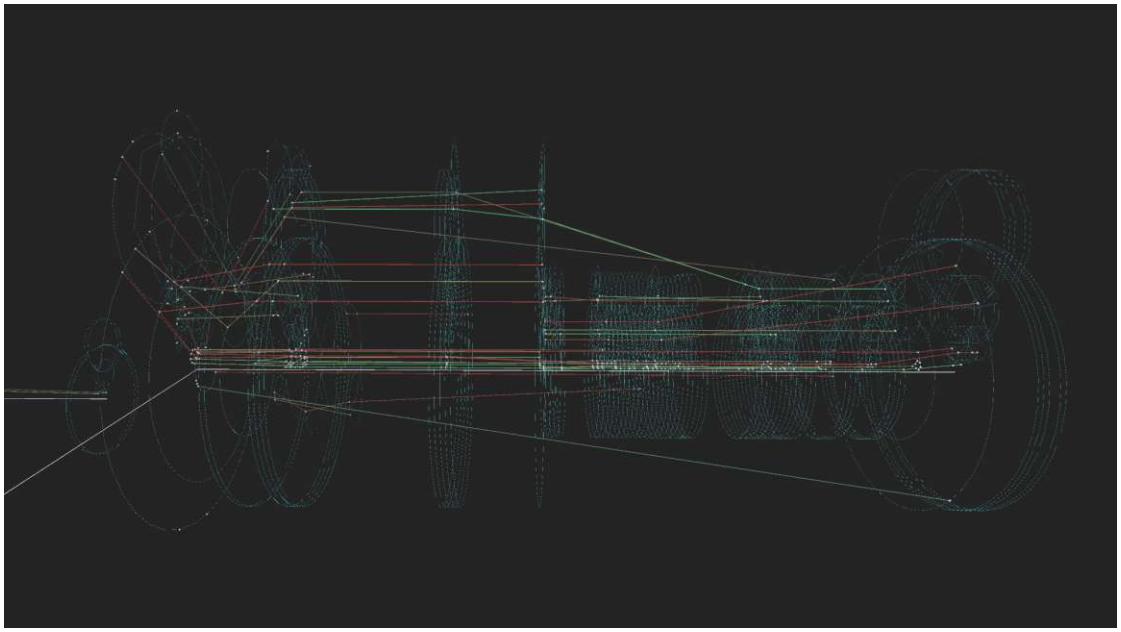


Figure 5.7: This figure shows the first implementation of the change data connection with a gradient from red to green depending on the number of changes.

```
1   using UnityEngine;
2
3   public class Example : MonoBehaviour
4   {
5       Vector3[] newVertices;
6       Vector2[] newUV;
7       int[] newTriangles;
8
9       void Start()
10      {
11          Mesh mesh = new Mesh();
12          GetComponent<MeshFilter>().mesh = mesh;
13          mesh.vertices = newVertices;
14          mesh.uv = newUV;
15          mesh.triangles = newTriangles;
16      }
17  }
```

Listing 5.5: Example C# class that creates a custom mesh.

The reason for creating one single line per file instead of one line per connection is the same as with the creation of the folder rings in Iteration 1 (Section 5.3). That huge number of single lines would have become an issue for the performance whereas less but more complex connected lines are easier to render for Unity3D.To create the split

line that shows both the number of additions and deletions simply two of the described custom mesh lines were created. The line that represents additions then extends its thickness in the up direction (positive y-axis) whereas the deletions line extends down (negative y-axis). The thickness of the addition and deletion lines is calculated as a relative value where 0 represents no changes and 1 represents the maximum number of changes. All other values get calculated by a linear function between those two points but could be changed in the future to a logarithmic function or configurable in the UI.

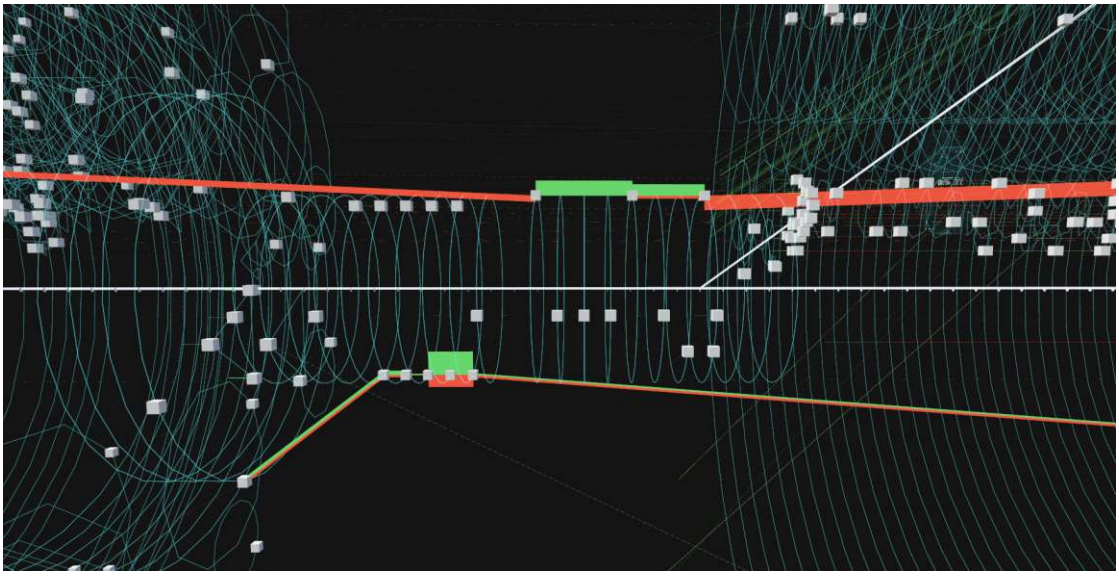


Figure 5.8: This Figure shows a close-up of how the split change lines look like. Each inter-commit change connection consists of a green and red line whereas always the top of the line is green and the bottom red. The thickness of each line shows how many additions (green) and deletions (red) were made from one file version to the next.

The other big refinement implemented in this iteration is the file interaction. Whereas previously when a file was selected, Holocular only zoomed to the file and showed a message about which file was selected in the debug console. As part of this iteration, it was updated to show a window with additional information about the selected file as seen in Figure 5.10. The window automatically opens once a file is selected and stays open as long it is not closed, even if a file is not selected anymore. When a new file gets selected the content of the File Info window changes. Some of the information displayed in this window are:

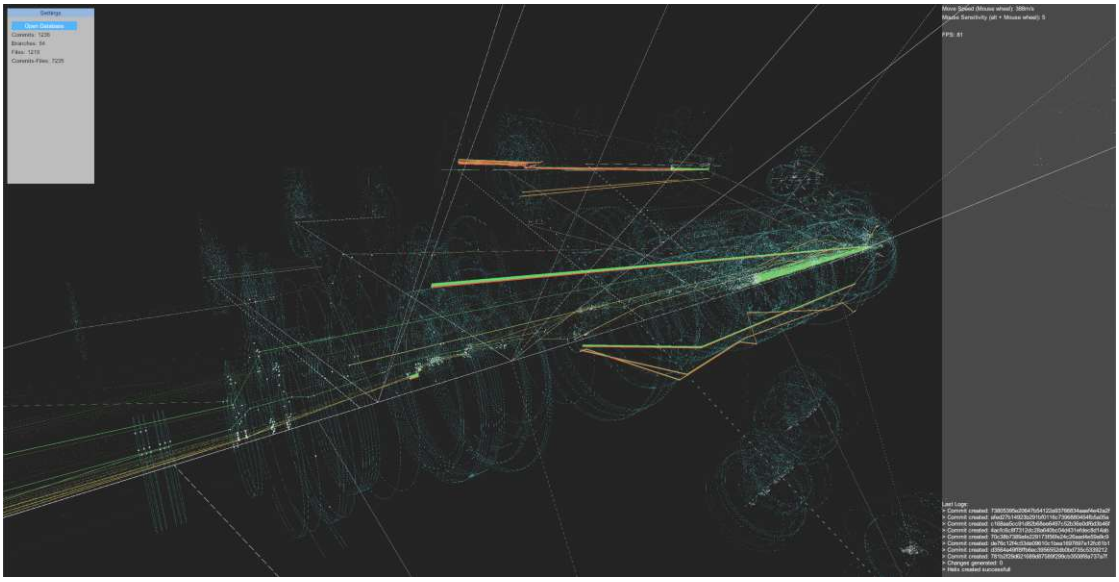


Figure 5.9: Refined implementation of the change connection from a further away perspective where it is easier to recognize phases with a large number of changes in the history of the repository.

- Filename
- Path
- Owner at this commit and how many lines are owned
- Commit information like the commit sha, parent shas, author, commit date, Branch and the web URL to the commit
- Addition and deletions stats
- Change hunks
- Ownership stats (which author owns how many lines)

After the basic file info window, a file compare window was implemented as seen in Figure 5.11. With this dialog, it is possible to compare the selected file version with its parent. This can be very helpful if the user wants to know what happened to a specific file within a commit. To open this comparison-view a new button in the file info window was introduced to open the window. Additionally, if the user keeps the window open and selects a new file it will automatically switch its content to the newly selected file.

All windows and the debug panel can be closed manually through a click on the corresponding button on the newly introduced dock. It is located in the bottom left corner and graphically designed like the docks of popular operating systems. An indicator below the window icons also shows if a window is open or not.

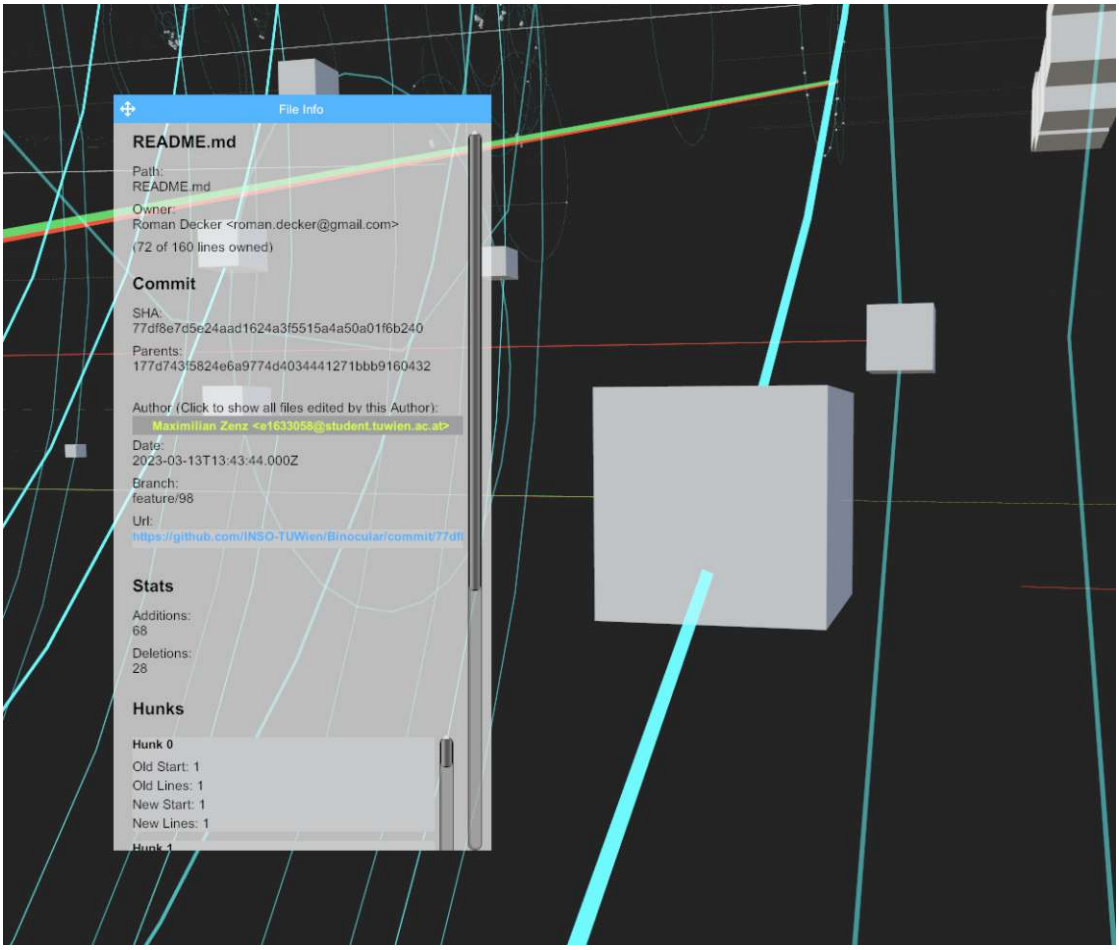


Figure 5.10: This Figure shows an example of how the information to a `README.md` file is displayed including commit owner and changes data.

5.5 Iteration 3 - Data Highlighting

The third iteration of development is all about adding more functionality to the visualization by adding the ability to color code the individual file nodes according to different data sources as described in Section 5.5.1. This concept was already introduced in the design phase in Section 4.4.2. At its core, every file node should be able to get colored according to metrics. Additionally, multiple possibilities were added to highlight or filter specific data as described in Section 5.5.2. This is important to give the person who uses the visualization the ability to get a better overview of the project and help to find specific parts of the project. Those highlighting and filtering options mostly correlate with the coloring modes.

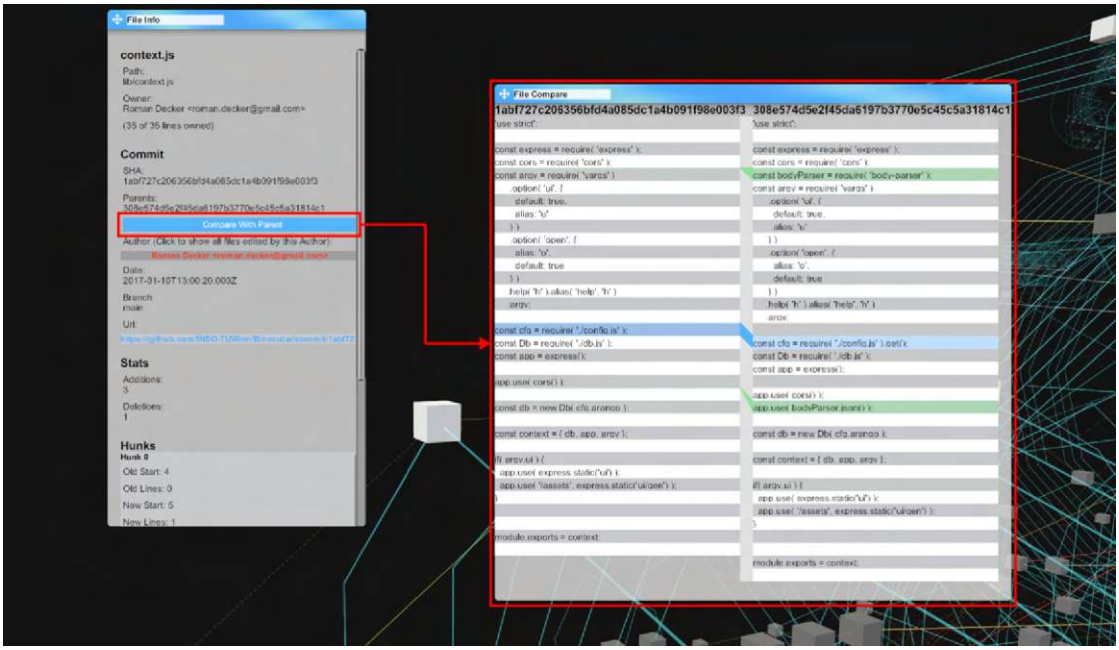


Figure 5.11: Here an example of how the file compare dialog could look like when comparing a version of the `context.js` file with its parent is shown. The file compare dialog is hereby triggered by pressing the compare file button in the file info view. Codelines that changed from one version to another get marked blue, additions green and deletions red.

5.5.1 Coloring Modes

The first part of this iteration is about the four different coloring modes the visualization offers. Each of the four modes hereby offers insights into different aspects of the project. All of the coloring modes can be enabled and disabled through newly introduced switches in the settings window. The four different coloring modes as described in the following sections are Committer, Owner, Hotspots and Branches.

Committer

The first coloring mode implemented is the highlight by committer mode as seen in Figure 5.12. This metric is important to get an overview of which author(s) are more active within the project than others. It can also help to find expertise at a specific point in the development history when getting an overview of which developer worked when on what. In addition to the ability to color files by its committer an author list window was implemented which shows an interactive author palette. This palette is necessary to show the user of the visualization which color belongs to which author. Without it, it would be necessary to always click on the color corresponding file nodes and view the author in the file info window to know which author belongs to the color. The author

palette also has interactive features by selecting a specific author. A closer look at how the interactive author palette works will be given in Section 5.5.2.

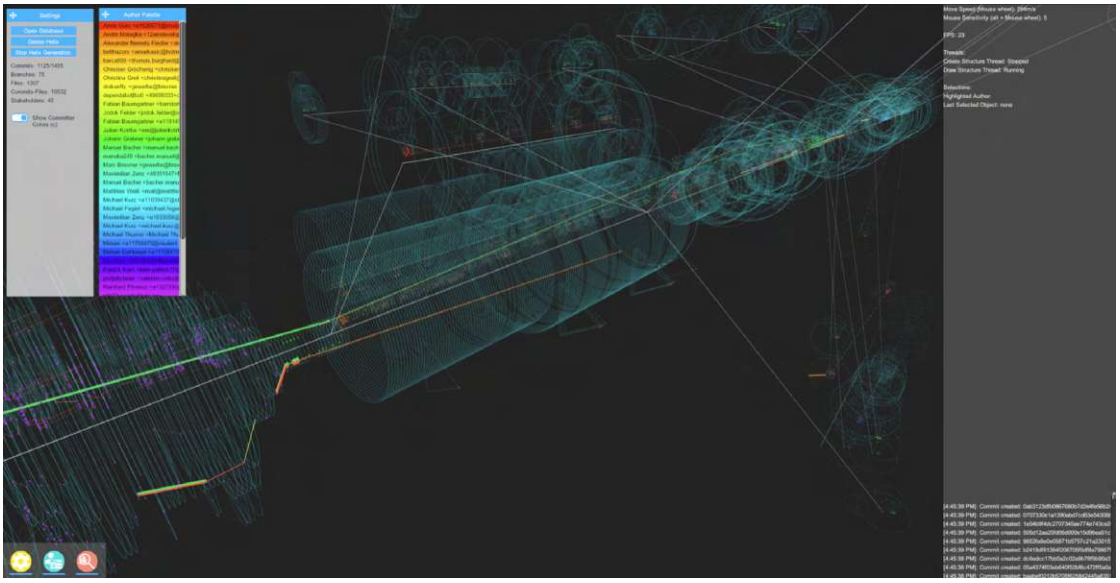


Figure 5.12: Here a part of a project is shown where the individual file nodes are colored by their committer. It is possible to see on the screenshot where one author stopped working on the project and the other one started.

Owner

The next coloring mode implemented is the ownership mode as seen in Figure 5.13. Ownership is hereby defined by which developer last edited most lines within the file at the moment of a specific commit. To implement this feature into Holocular more data from the Binocular database had to be read and converted into C# object. Precisely the Commit-Files-Stakeholder connects every connection from the Commit-File connection with all the stakeholder that have ownership and also tracks how many lines they own. Additionally, this collection also contains the ownership hunks which are also displayed in the File Info window. This is necessary to give an accurate and detailed overview of the ownership of each file over the development of the project.

Binocular calculates the ownership data by reading the results of a git blame command that returns the current owner for each line at a specific commit for a specific file as seen in Listing 5.6. After reading the data binocular transforms it into individual ownership hunks per author per commit and file which include the start and end line of the hunk, the signature of the committer and the commit sha. This is the data that gets written into the Commits-Files-Stakeholder database collection which is later used by Holocular.

```
# Command:
git blame -p [commit] -- [filename]

# Output Format:
[commit] [filename] ([Author] [timestamp] 1) [Line 1 Content]
[commit] [filename] ([Author] [timestamp] 2) [Line 2 Content]
[commit] [filename] ([Author] [timestamp] 3) [Line 3 Content]
[commit] [filename] ([Author] [timestamp] 4) [Line 4 Content]
...
```

Listing 5.6: Git blame command to get the Ownership for each line of a file at a specific commit with an example how the output is formatted.

The colors for each stakeholder are hereby the same as with the committer colors to minimize confusion and so that the already implemented author list is valid for both the color by committer and color by owner mode. Code ownership and the transfer of it can be a very important metric for the quality of the code as indicated by papers like „Code Ownership and Software Quality: A Replication Study“ [95] and „Don’t touch my code!“ [96]. Therefore, as part of this implementation, a detailed ownership breakdown was also introduced in the file info window.

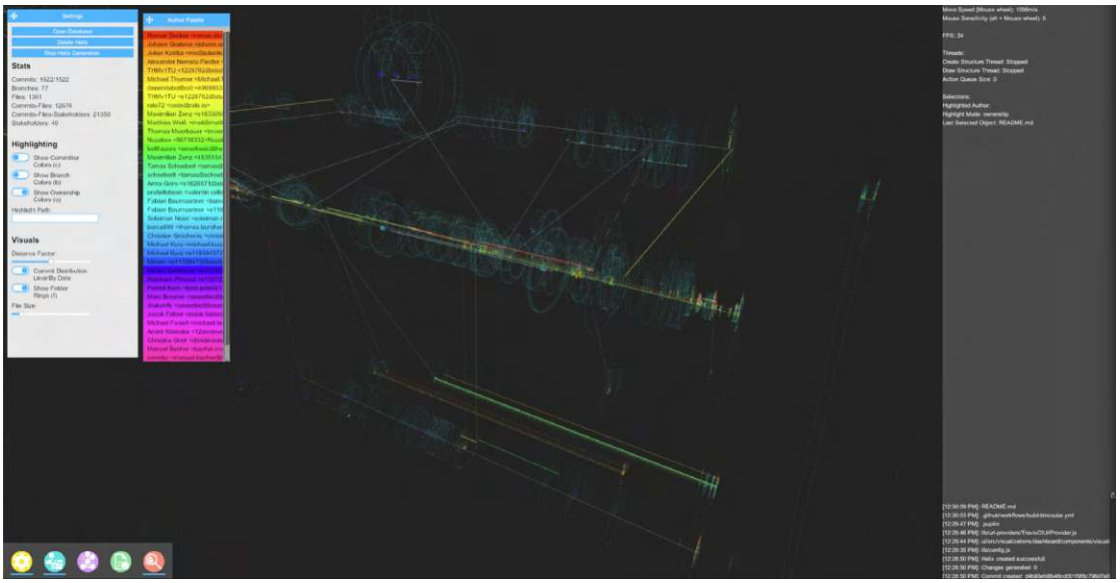


Figure 5.13: In this screenshot an example visualization is shown with the file nodes colored with ownership colors. The ownership is calculated for each file within every commit.

The ownership of a file at a specific Commit and date can be quite different from the committer as seen in figure 5.14. This is due to that when an author creates a file, he or she owns the whole file. After that, multiple authors may work on the file but only change a few lines each which doesn’t necessarily transfer the ownership of a file. Ownership is

only transferred once an author owns more lines than the previous owner. Depending on the size of a file it can take many changes till the ownership gets transferred to a different author.

(a) Colored by Owner (b) Colored by Committer

Figure 5.14: Here you can see an example where it is visible that although the orange committer did a lot of commits to the branch the ownership barely changed. This could indicate that even if changes were made all of those changes were on the minor side.

Hotspots

The third color highlighting mode implemented is the hotspots mode as seen in figure 5.15. When activated, the file nodes get colored along a gradient from white (no changes) to blue (maximum number of changes). This is similar to the red-to-green gradient of the first Implementation of the inter-commit file connections. Additionally, a slider was added to the settings window where it is possible to limit the number of changes for the upper end of the gradient. This is necessary since automatically generated files like the `package-lock.json` can have thousands of changes within a single commit. By just scaling the gradient from 0 to one of those max values all other changes done by the developer would vanish into shades of white. The here introduced limit option on the other hand gives the option to set the upper end of the gradient to your preferred value. For the prototype, the default value of the limit option was chosen to be 100 changes so that every file that had more the 100 lines changed within a single commit gets colored blue and every file with fewer changes gets colored depending on the position on the linear gradient. For future iterations, it can also be thought of scaling this gradient optionally with an exponential function or making it possible to exclude certain files.

Branches

The fourth and final coloring mode implemented is the branch coloring mode as seen in Figure 5.16. Although the different branches are already distinguishable through the separation in the 3D space this coloring mode still has its purpose because it helps to find specific branches more easily. Without the added colors it can still be difficult to find a specific branch. This is also due to the newly implemented interactive branch palette further explained in Section 5.5.2. Conceptionally the branch palette works the same as

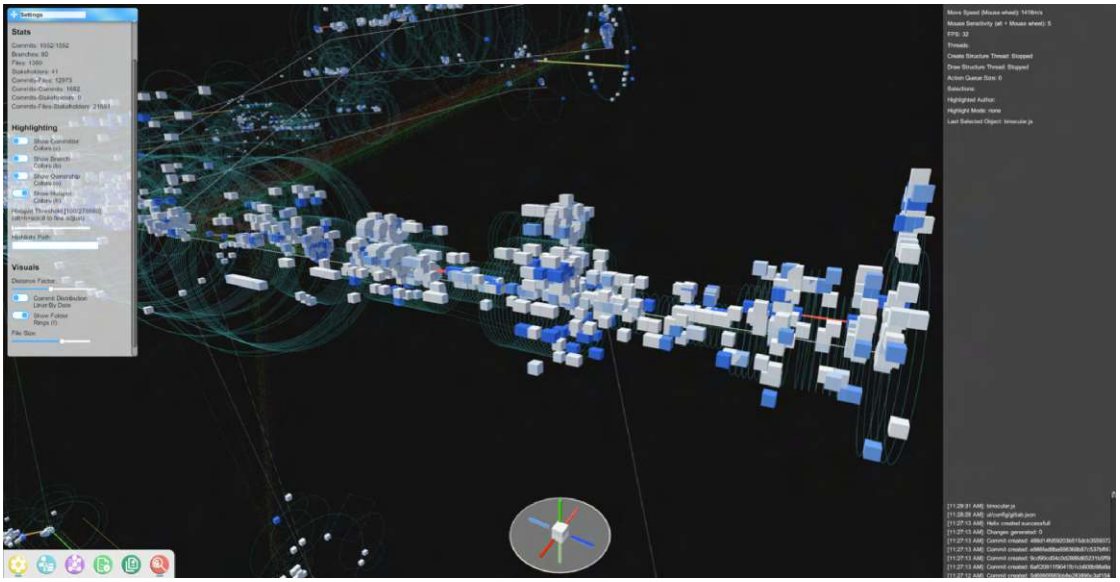


Figure 5.15: In this screenshot a part of an example project is visible with the hotspots coloring mode active and the default scaling from 0 (white) to 100 (blue) changes. In the bottom center a hotspot is visible which could point to a phase in the development with a lot of activity.

the author palette. It shows a list of branches with the corresponding colors and when a branch gets selected all other branches are getting grayed out when the branch coloring mode is selected.

5.5.2 Settings and Highlighting Modes

To make Holocular more usable multiple filtering and highlighting modes were introduced. This helps the user to find the data needed. In the following paragraphs, the four implemented settings (Commit Distribution, Author Highlighting, Branch Highlighting, Module and File Filtering) and highlighting modes are further described.

Commit Distribution

Depending on the situation and the data the user wants to analyze, the linear distribution of commits as implemented in Section 5.3 may not be the right tool to find the data wanted. Therefore, an alternate commit distribution mode was introduced that places the commits depending on their date and time of commit instead of a fixed distance. This can be seen in Figure 5.17 and gives a better overview of how and when the repository developed. It also helps to analyze breaks in the development cycle.

Additionally, to give the user even more control over the placement of commits a slider was introduced in the settings window to adjust the distance between commits. Moving the slider either brings the commits closer together or further apart depending on the

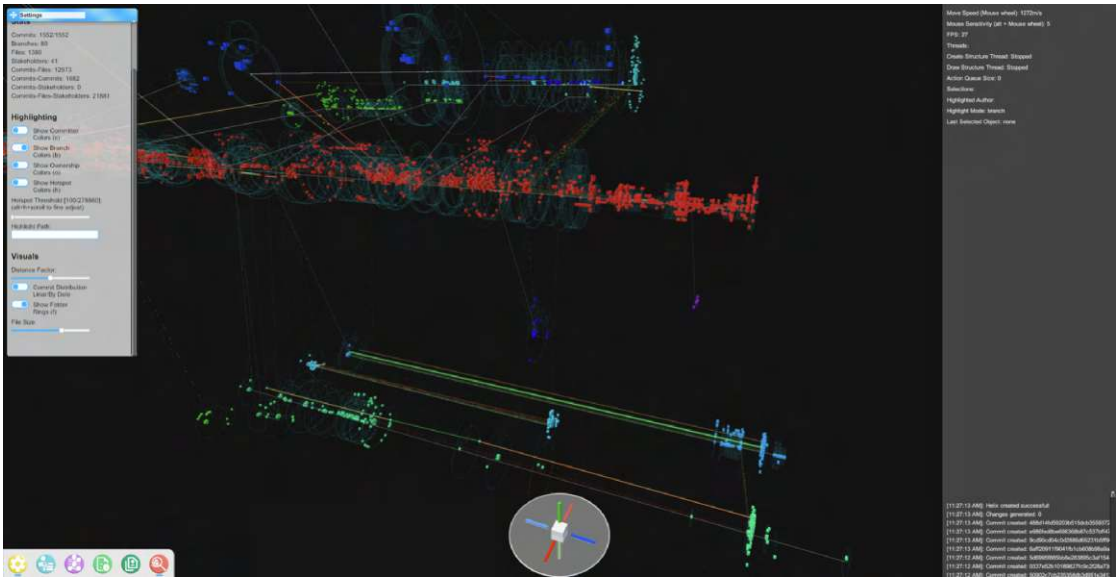
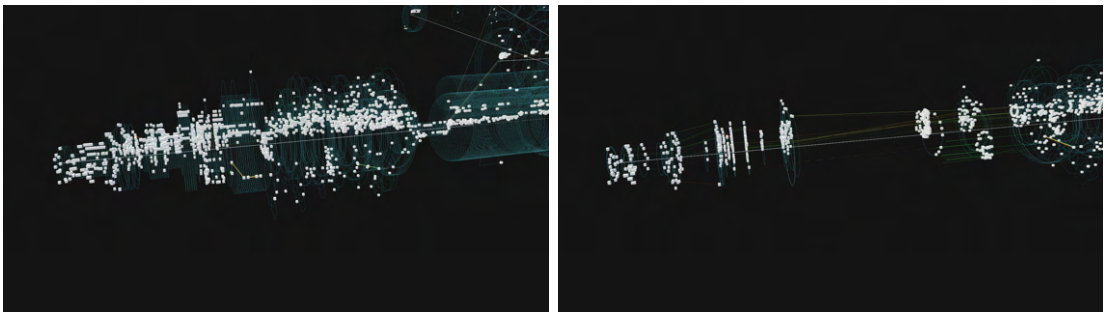


Figure 5.16: In this screenshot a part of a visualized example repository is visible with the branch color highlighting enabled. It helps to distinguish and find branches in the 3D space.

needs of the user. Depending on the number of commits in the analyzed repository and if the user uses the „linear“ or „by date“ commit distribution mode both can be very important.



(a) Commits distributed linear with a constant distance between each other

(b) Commits distributed by their commit date and time

Figure 5.17: Here a comparison is shown between the linear (5.17a) and date based (5.17b) distribution. The user of Holocular can choose which of those distribution modes fits their needs better.

Author Highlighting

To help the user find files that were committed or owned at any point in the development history by a specific author, highlighting a specific author was introduced. This high-

lighting mode is implemented through the interactive author palette introduced with the committer highlighting mode. When either the color file node by committer or author mode is activated, the user can select an author in the author palette by clicking on it. This will result in solely showing the color for files that are committed or owned by the selected author. This depends on the coloring mode the user has currently selected. Every other file node that got committed or is owned by a different author will be grayed out. Additionally, if the user wants to fast switch between the committed and owned files of an author it is possible to click a second time on the author in the interactive author palette. This will cycle through highlighting committed files, highlighting owned files and no highlighting. To further expand the functionality of this feature it is also possible to select an author to highlight through the author field in the file info window. Clicking it will instantly highlight all other file nodes that are connected to the committer and gray out all others. Figure 5.18 shows how a visualization with the owned file nodes of one author highlighted could look like.

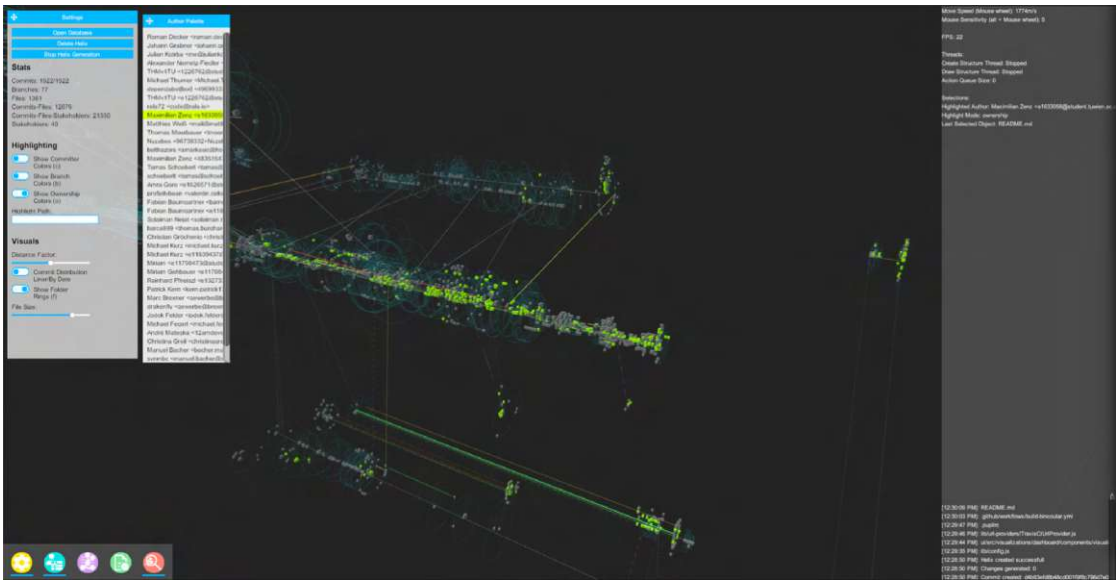


Figure 5.18: In this screenshot an example repository is loaded in the visualization with the ownership coloring mode selected and only one author highlighted. This helps to find ownership changes.

Branch Highlighting

Similar to the author highlighting described in the previous section it is also possible to highlight specific branches. This is done by changing the file node coloring mode to branches and selecting a branch in the interactive branch palette. When a branch gets selected only that one branch will be colored and all other branches will be grayed out. The branch highlighting helps the user to find a specific branch more easily. Finding a specific branch in a fast manner can be important when trying to figure out which

developer(s) worked on specific parts or features of a project. Figure 5.19 shows what an example visualization looks like with one branch selected which then is highlighted as the only colored branch in the helix structure.

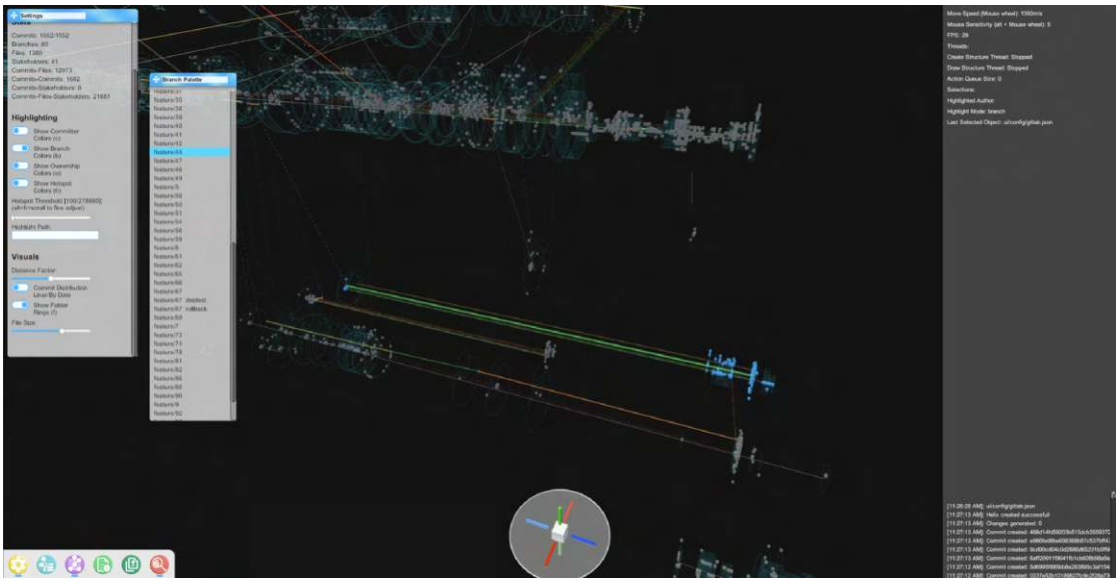


Figure 5.19: Here an example visualization is shown where only one branch is highlighted in light blue. This helps to better find a specific branch/part of the development history of interest.

Module and File Filtering

The final implemented filtering option is the module and file filtering. Here an input field was implemented in the settings window that enables it to either enter a path to a module or file. When doing so, only file nodes and folder rings will be shown that match the input. The search updates the helix structure in real-time so that already during the search it is possible to see immediate results. This helps the user to limit the visualization to a specific part of the project or file. The Module and File Filtering enables the user to analyze specific parts of the repository more precisely without obstructing other parts of the repository. Especially in a three-dimensional structure like the one implemented obstruction can be a huge issue. The module and file filtering can also be combined with all other filtering options and the different coloring modes as shown in figure 5.20. This example shows how to extract specific information like the ownership evolution for a specific part of the project.

5.6 Iteration 4 - Final Prototype

All those features described in the sections above now result in the final prototype of Holocular, which can visualize repositories mined with Binocular in a three-dimensional

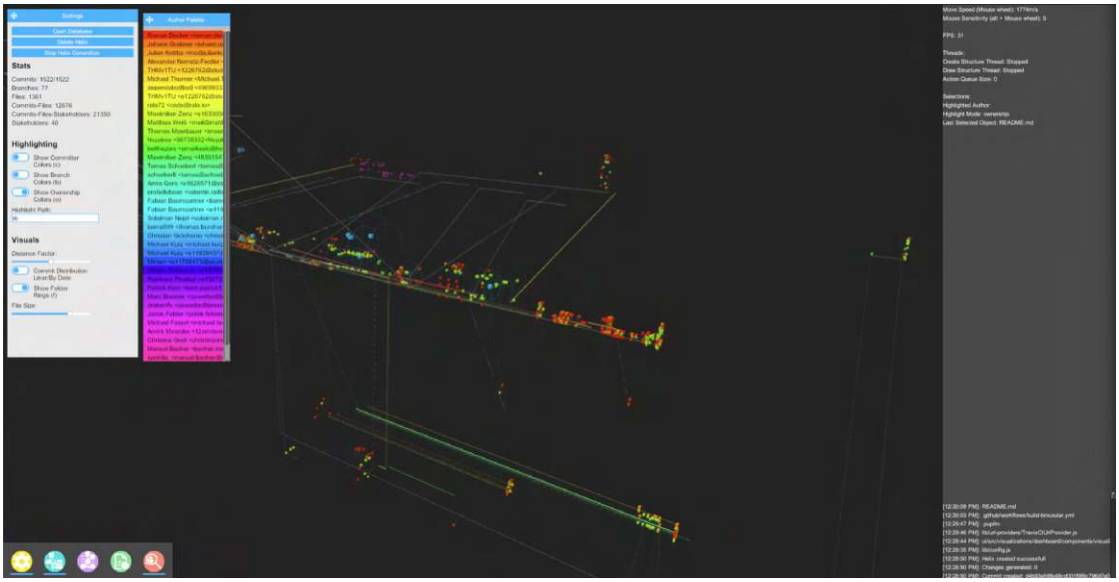


Figure 5.20: Here an example visualization is shown with a filter for the lib module. Only file nodes and folder rings that belong to that folder are visible.

space as seen in Figure 5.23. It is also able to encode additional information through visual indicators like color coding the file nodes according to different metrics. At this point of the development, all major features chosen to be part of the prototype were implemented and only small refinements and bug fixes were done to improve the usability and stability. In Figure 5.21 an overview of the final GUI is shown.

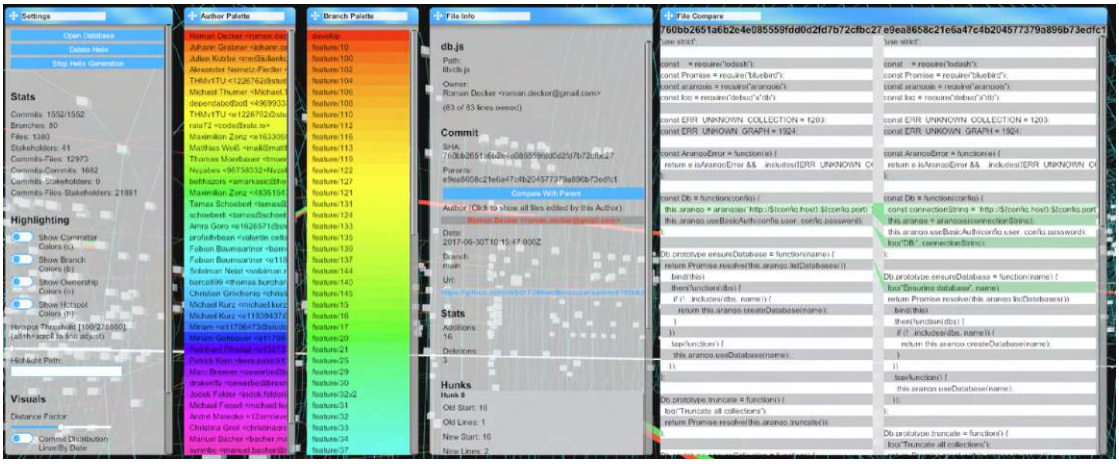


Figure 5.21: This Figure shows all the final GUI elements from left to right: Settings, Interactive Author List, Interactive Branch List, Detailed File Info and File Version Comparison

The biggest refinement implemented in this step of the development of the prototype was

the rework of the 3D movement system as interaction and movement in the 3D space is a huge hurdle for the usability of 3D visualizations. Additional elements were introduced to help the user navigate the 3D space. This is mainly manifested in the interactive compass (Figure 5.22) at the bottom of the viewport. Due to the complexity of the compass and its need to display all three axes it was done as a fixed 3D Element at the bottom middle of the visualization. Every axis and the middle of the compass is interactable and improves the navigation through the helix. In addition to the interaction, the 3D compass also dynamically rotates with the movement of the camera. This helps the user to get a better orientation in the three-dimensional space. When one of the three axes gets selected depending on the direction of the axis the camera will automatically rotate in that direction. This functionality is similar to how comparable features work in Unity itself or other software like Blender or other CAD software that feature a 3D viewport. Clicking the middle cube of the compass will result in focusing the camera on the helix so that the full helix will be visible within the viewport. To accomplish those movements in a smooth way the camera got fully detached from the camera dolly and was programmed to smoothly follow the movements of the dolly. This enables that the position and rotation of the dolly can be changed instantly without the camera jumping to the position but rather smoothly moving to the desired position and rotation.

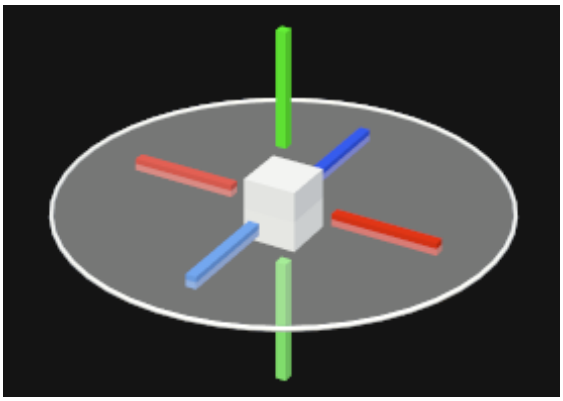


Figure 5.22: This Figure shows the interactive compass as implemented as Part of Holocular.

Additionally, multithreading was introduced so that Holocular doesn't freeze during the creation of the helix and multiple graphical UI improvements were done. From all the in the design phase proposed features only two were dropped. On the one hand, the twisting of branches was dropped which was present in the early design stages. This was done because it resulted in more confusion about which files belong to each other than it had advantages. The other dropped feature was the filtering after developer contribution in percent which was deemed by the design evaluation (Section 4.4.3) as least useful. All other planned features were developed for the final prototype. All this resulted in a comprehensive prototype to analyze a version-controlled repository as a whole in a three-dimensional space.

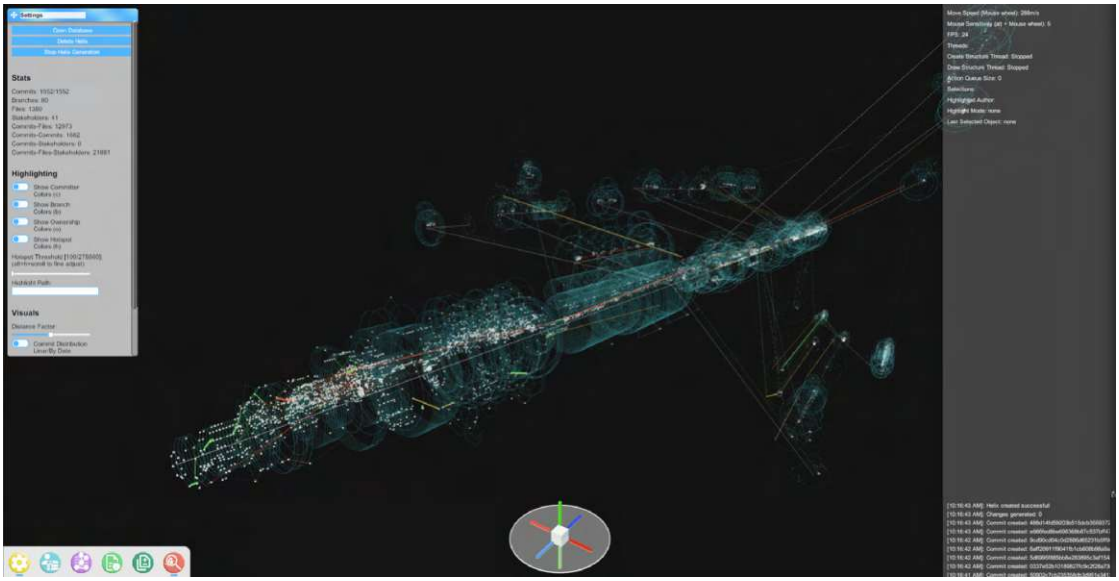


Figure 5.23: Final iteration of the prototype with all features chosen to be part of the prototype.

CHAPTER 6

Evaluation

To evaluate the prototype against the previously defined Research Questions (Section 1.3) it was chosen to conduct multiple interviews with experts. The three experts from the design evaluation returned for this interview in addition to three new participants. As part of the interviews, the prototype was presented (Section 6.1.3) and multiple scenarios were shown (Section 6.1.4). The scenarios are hereby designed to show the interviewee how the prototype can be used and what data can be extracted from it. With all that information it is then possible to get a conclusion (Section 6.2) about the functionality of the prototype.

6.1 Interviews

The conducted expert interviews were designed to follow a defined structure. At first basic questions about their demographic and experience (Section 6.1.1) were asked as well as what their previous experience with three-dimensional version control visualizations was (Section 6.1.2). After that a small demonstration (Section 6.1.3) of the prototype was conducted to give the participant insights into the core functionality of Holocular. Additionally multiple scenarios (Section 6.1.4) were shown to the participant to show possible use cases the prototype could be used for. Every participant had to answer two questions for each of the four scenarios. In addition to that some more general questions (Section 6.1.5) about the features of the prototype and additional thoughts were asked. The complete interview guide for the evaluation can be found in Appendix B.

6.1.1 Demographic & Experience

The first part of the scenario-based expert interview was about the demographics of the participants. Half of the experts were in the age range between 26 and 30 while the other half were older (Figure 6.1). In terms of their experience level with Version Control

Systems and repositories, most of them answered with four to five out of five which results in an average experience level of 4.33. When asked about their familiarity with analysis and visualization tools of version-controlled repositories all but one answered that they are familiar with that kind of tools. In Figure 6.2 an overview is shown of the experience of the six participants. Four out of the six participating experts also stated that they use such tools often in their workflow. In general, the participants who are familiar with those tools stated that their experience level with such tools was between three and five out of five with an average of 3.6. This ensures a qualified group of experts to take part in the scenario-based expert evaluation.

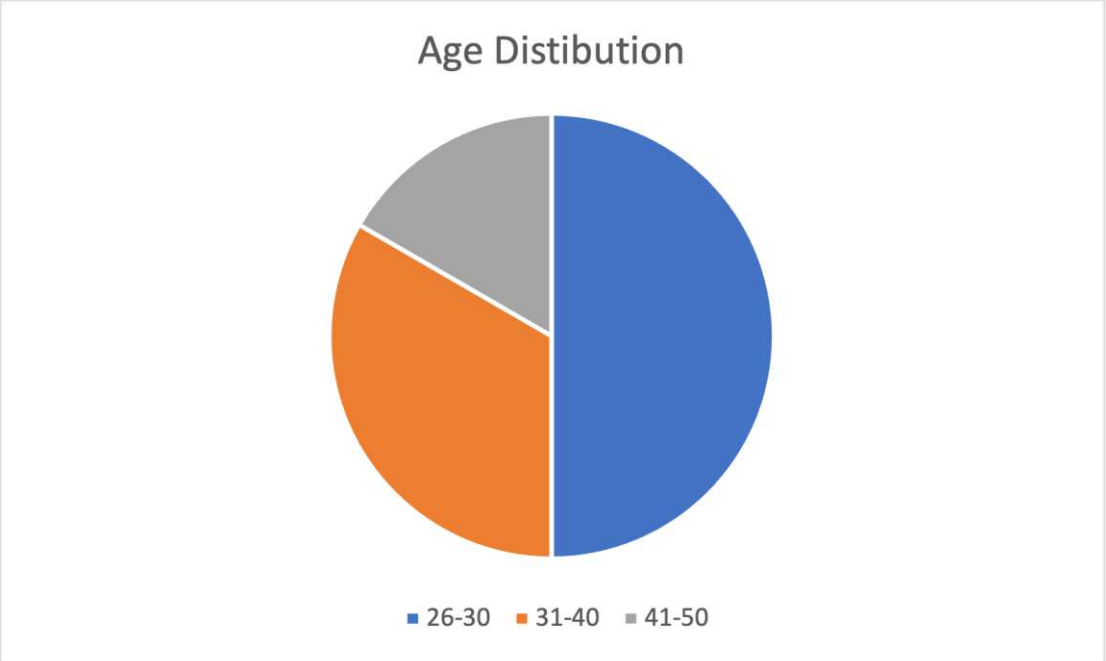


Figure 6.1: In this figure the age distribution of the six scenario-based expert interviews is shown.

6.1.2 Experience with 3D Version Control System Visualizations

The next section of the interviews was about the experience the participants already had with 3D version control visualizations. While not important for the interview that a participant already has deep knowledge about three-dimensional version control visualization it is still a useful metric for interpreting the data harnessed in the later sections. Of the six participants, four already knew some kind of three-dimensional version control system visualization. This was in most cases „code cities“ but in addition to that two of the participants had a deeper experience because of their master thesis on the topic.

They were also asked why they think that three-dimensional version control visualizations

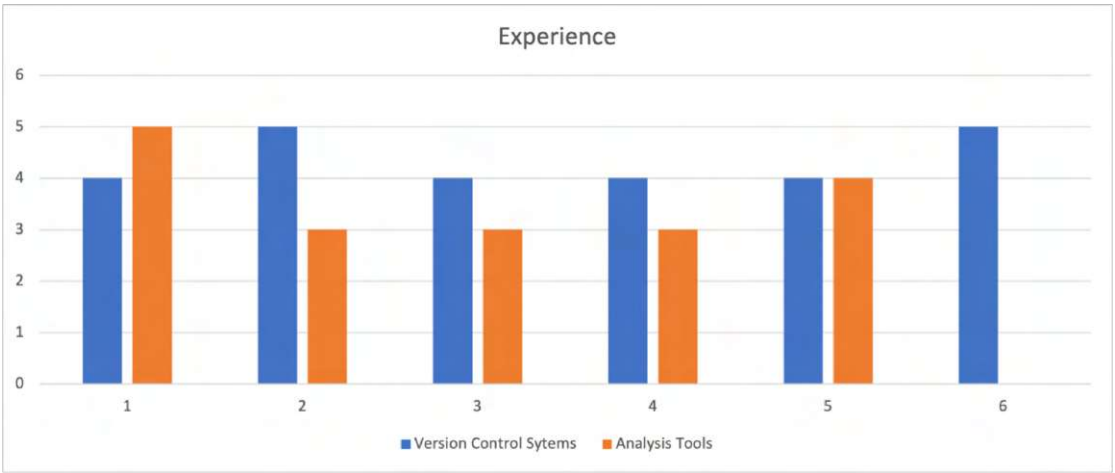


Figure 6.2: This figure shows how the six participants rated their experience with version-controlled repositories and analysis tools of such.

are not used as much as their two-dimensional counterparts. The most common answer was that a three-dimensional visualization is too complex for the average person to use with too much information at once and that it is too hard to use. Especially the interaction in the three-dimensional space is harder according to the participants. Another point made by the participants during the interview was that it requires a very special use case to require such a visualization. One participant also mentioned that most repositories are not complex enough to require the additional data density. Another interesting point one participant mentioned was that the lack of education in that direction and version control system visualizations, in general, could also be a reason why three-dimensional version control visualizations are not that common.

Lastly in this section, the participants were also asked if they think that more cutting-edge technologies like virtual, augmented, or mixed reality can lead to broader adoption of three-dimensional versions. The participants were asked to rate the likelihood from unlikely (1) to likely (5) whereas most answered in the middle with an all-over-average result of exactly three.

6.1.3 Live Demo

The next section of the interviews was all about the prototype (Chapter 5) developed as part of this thesis. The participants were introduced to its core functionality, how it is structured and what parameters can be changed to customize the experience. Getting the interviewees familiar with Holocular is very important for the next sections of the evaluation. In general, even if none of the participants ever saw a three-dimensional version control visualization like this before all of them reacted interested and open.

The main consent between all of them after first seeing it was that it visually looks very appealing. One participant also mentioned that it could be an alternative for complex

projects for a classic tree view found in applications like for example IntelliJ IDEA. It was also mentioned that the addition and deletion connections between the different file versions can be hard to see in the current state of the prototype and that it could be interesting to filter or search specific commits. Other comments were that the data density is quite high which can make reading the data hard. It was also mentioned the folder structure can be confusing and that it would be good if there were additional on-hover information when hovering over folder rings or the addition and deletion connections. All in all, most of them mentioned that it's hard to find specific things because of some missing filtering and search options.

6.1.4 Scenarios

To evaluate the prototype against the research questions, four scenarios were presented to the interviewees and shown how to solve them. Each participant then had to answer two questions for each scenario. At first, they were asked how suitable they think Holocular is to accomplish the scenario with a grade from one (not suitable) to five (really suitable). The second question was if they recognized any issues during the solving of the scenario. All four scenarios started from the base state with a blank freshly opened application and by selecting the example database mined from the Binocular project. At the time of mining binocular contained 1552 commits over 80 branches with 1380 tracked files and 41 authors who participated. The four scenarios were as follows:

- **Evolution:** Open a database and analyze the connections between each file/commit. Find a connection with many changes (for example `lib/git.js`) and compare the source code of the two versions of the file.
- **Hotspots:** Open a database and try to get an overview of where in the history of the project a lot of files got changed within one commit which could point to a refactoring or merge request and open the GitHub commit URL of one of those to check if it is a refactoring.
- **Ownership:** Open a database and check who did the most commits and who has had the most ownership during the development of the provided repository. Did ownership change?
- **Expertise:** Open a database and try to find the branch `feature/137` and find out which developer did most of the commits on this feature.

When combining all the suitability ratings (Figure 6.3) the scenario with the most consistent good ratings was the scenario Hotspots with a rating of 4.83. Scenario Evolution and Ownership had the same average rating of 4.17. From those two the ratings on the Ownership Scenario were further split with three out of six interviewees rated a five and two rated a three. The Evolution scenario on the other hand was rated more uniformly with only two participants rating a five but three rating a four. This

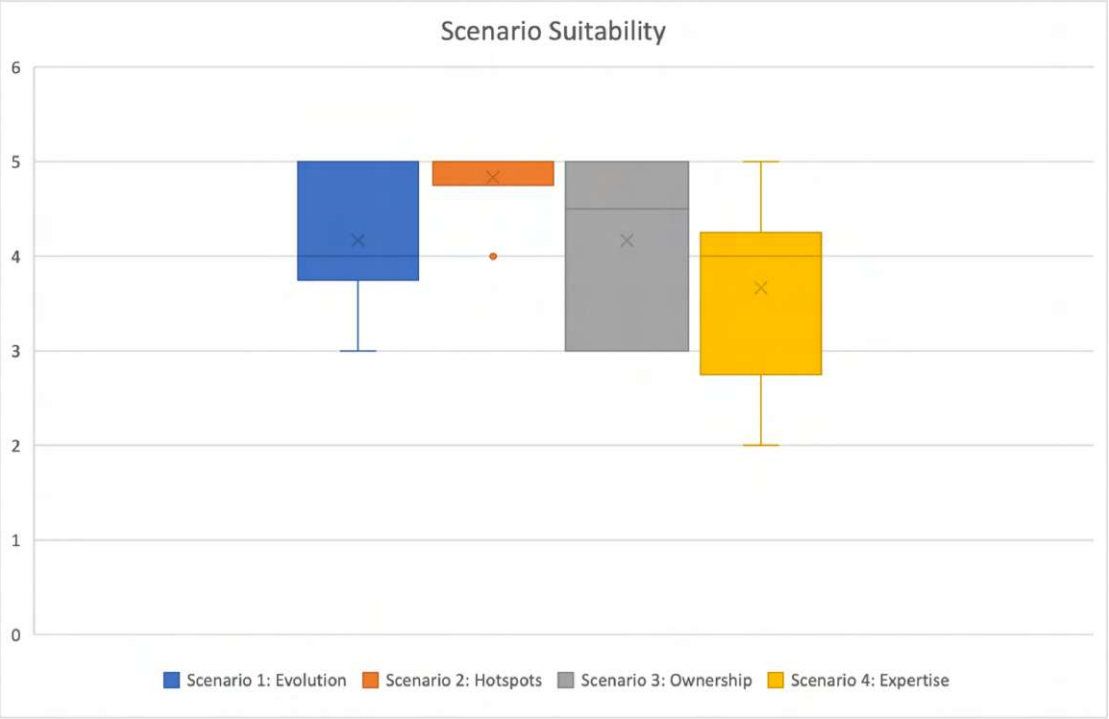


Figure 6.3: In this chart the usability rating of the four scenarios is shown as boxplots.

indicates that the participants were more split in their thinking about the suitability of Holocular for recognizing ownership than for recognizing the evolution data. The lowest-rated scenario out of the four was the Expertise scenario with a rating of 3.7. It is the only one that was rated with an average score below four.

Even if the suitability rating the experts gave was on average good, they still recognized some major issues. In the following paragraphs, the issues will be grouped depending on which scenario they were mentioned.

The main problem that appeared during the Evolution scenario was that when filtered for the specific file the addition/deletion connections for other files were not filtered out as seen in Figure 6.4. This happened because of a limitation in how the line renderer was coded during development and a whole rewrite was not feasible as part of this thesis but would be interesting as a future advancement. Another issue recognized by multiple interview participants was that due to the absolute linear scaling of the thickness of the addition/deletion connections, they tend to be thin and hard to distinguish. This happens because it is always scaled according to the thickest possible connection which in the case of the tested repository is the `package-lock.json` file. Here, many participants stated that a different kind of scaling for the line thickness would be appreciated like logarithmic scaling or manual scaling like the one used for the Hotspots highlighting mode. One Participant also mentioned that it could be confusing to find the connections wanted in the 3D space.

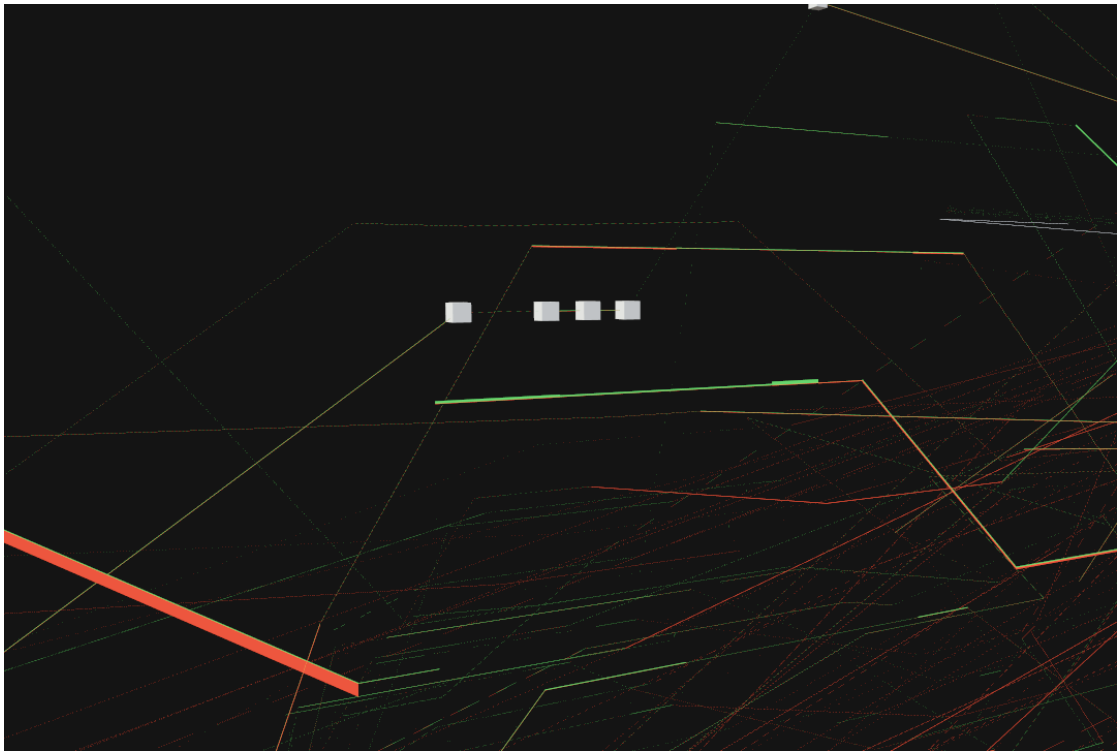


Figure 6.4: In the screenshot a closeup is shown with a filter for `lib/git.js`. The addition/deletion connections don't filter accordingly to the file filter due to the rendering limitation in the current implementation.

For the second scenario problems were mentioned by the participants. The main problem here most of the experts stated that due to the free perspective in the three-dimensional space, it can be hard to differentiate different point clouds. Especially when multiple huge commits are close to each other. Another issue mentioned by one expert was that the current way of tracking file renames could lead to misinterpretation. Currently, if a file is renamed or moved it gets displayed as if the previous file is deleted and the new file is created. This is caused because the version of Binocular this prototype was based on didn't track file renaming. This can result in the creation of some additional point clouds that don't yield the information searched for. This is also an issue for the scenario about extracting the ownership.

The third issue that had the extraction of ownership in mind also had some issues stated by the interviewees. Mainly when there are many authors part of the repository it can be hard to distinguish the different colors as seen in Figure 6.5. That also makes it hard to compare the ownership of different authors with each other. As a possible solution, some of the experts mentioned that it would be good to exclude or gray authors that are not necessary or to merge multiple authors into a single color. A feature like this would also improve the usability when one author commits with multiple signatures. Another

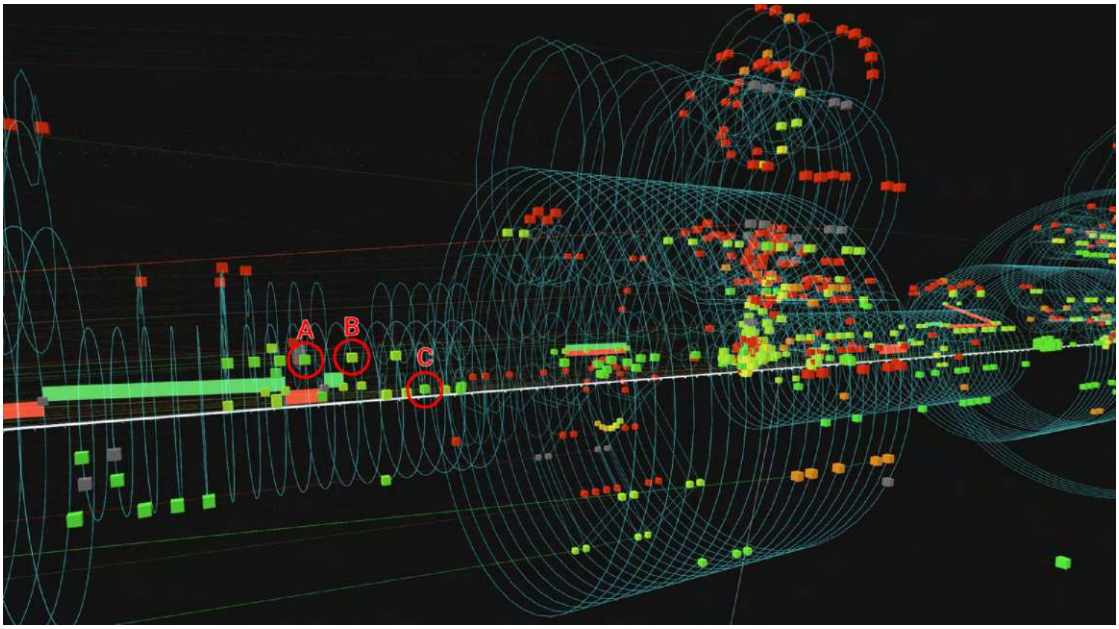


Figure 6.5: In this screenshot a closeup is visible where it can be difficult to distinguish some Authors. For example, the files marked with A, B and C in the figure all have different ownership but due to the large number of committers in the repository, the different hues of green are too close to each other to properly distinguish.

issue recognized is that because only files displayed that got changed within a commit, it is impossible to say how the absolute ownership changed during the development history. Also mentioned was that it is hard to see where ownership changed. One participant suggested some sort of indicator to visualize ownership change could help a lot. One interviewee also stated that the perspective in the three-dimensional space could also result in issues similar to those discussed before with the Hotspots scenario.

For the fourth and last scenario, some of the issues were the same as with the previous ones. The main issue unique to this scenario was that the way to find a branch was too complicated. Currently, the only way to find a branch is through the Branch coloring mode and finding the branch through the branch palette. Here many of the participants stated that some kind of search or fast way to jump directly to the branch in question would be appreciated. Additionally, some participants also mentioned that similar to the ownership scenario it can be quite difficult when there are multiple authors with very similar colors working on the same branch.

6.1.5 General Features & Final Thoughts

The last section of the interview was all about the final thoughts of the participants about the prototype, the scenarios and the general features. This also includes what potential features the experts would want from a visualization like this in the future.

The first question was about whether their perception changed after seeing the four scenarios in comparison to right after the short live demonstration. For most of the participants, the perception of the tool stayed the same as before. One of them stated that with a tool like this, it could be fun to dive into a new repository they are not familiar with, but it would also be appreciated if it could connect to other already established tools. This would also help to bridge the gap and make it easier for new people to explore the tool. A different expert stated that it could be hard to keep an overview of the folder structure because of the way the rings are nested within each other.

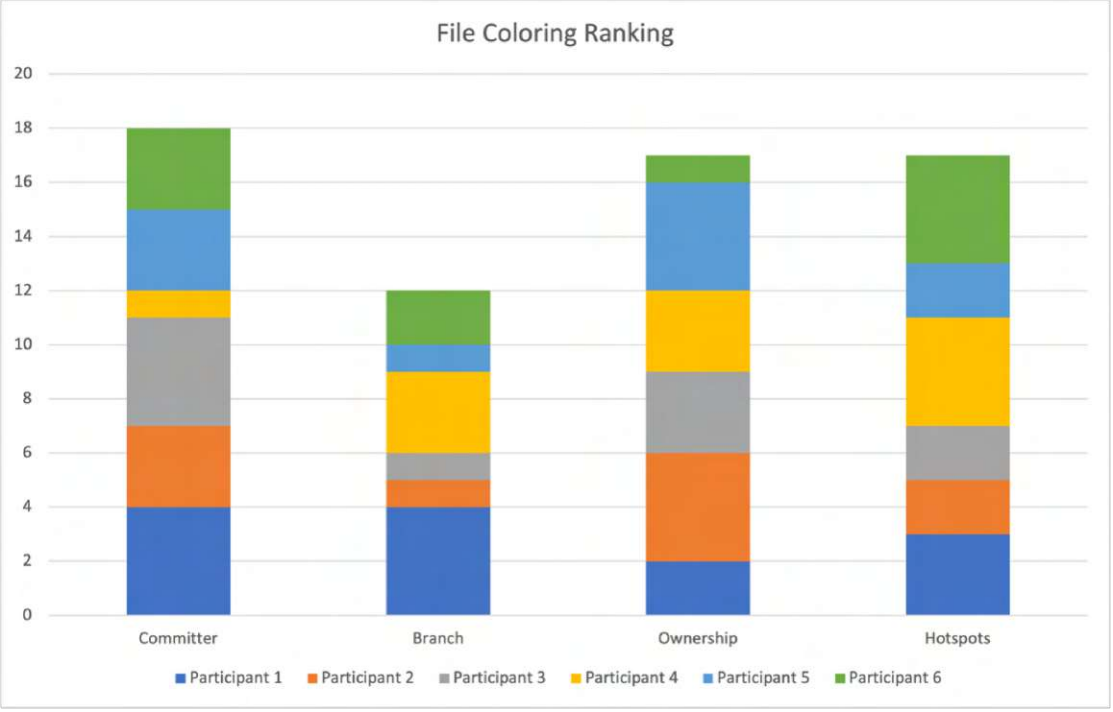


Figure 6.6: This bar chart shows how the five participants rated the four different file color highlighting modes. For simplicity, the ranking was converted into a rating of one (least useful) to four points (most useful). The highest-rated mode was the committer mode whereas the branch coloring mode was the worst-ranked. Both the Ownership and Hotspots coloring mode received a strong second place.

The second question of this section asked the participants to rank each of the four different file coloring modes against each other according to their usefulness. They also had the option to rank two of the coloring modes in the same place if they could not decide. This can be seen in Figure 6.6. The coloring mode that was ranked most useful by the participants was the committer mode. Here each file node gets colored by the color of the person who committed it. As a close second place in the ranking both, the ownership as well as the hotspots mode was placed. The ownership mode refers hereby to the mode where each file node gets colored according to the person that last edited the most

number of lines of the specific file at the moment of the commit. The last place took the branch highlighting mode with a lot fewer good votes than the other modes. This is the mode where every file node is colored depending on the branch it was committed to. When asked most participants stated that they would prefer a better and faster implementation to find a branch for example by using a search bar.

The next two questions were about how the visualization can display commits and which way the participants prefer. At first, they were asked if they preferred the linear or date-based distribution of the commits. Here all of the participants voted for the date-based distribution. Most of them stated that it is possible to extract more data from the date-based approach and that is why they prefer it. The second related question was about the way files and folders are displayed. In particular, they were asked if they preferred the structure with the blue folder rings displayed or hidden. Here the answers of the experts were split with three of the six experts voting for visible rings and three voting for invisible rings. In Figure 6.7 pie charts show how the answers of the participants were distributed.

(a) Commits distributed linear with a constant distance between each other

(b) Commits distributed by their commit date and time

Figure 6.7: **(a)** shows the preference if folder rings should be enabled or disabled; **(b)** shows the preference for the commit distribution.

The final section of the interview was designed as an open discussion about additional features and data the experts would want to see from a visualization like this. In the case of additional data, two of the participants mentioned that some indicator of the development of the file size could give additional insights. One participant recommended that file size could be encoded in the size of the individual file node cubes. Another new data source one of the participants mentioned is the CI pipeline status. That means that with each commit additional information should be shown depending on the pipelines triggered by that commit. This includes data like pipeline and job results. Additionally, one participant also requested that some kind of statistical data could also improve the visualization. This could include that it is possible to display statistical evaluations of a selected commit. Another additional metric one of the experts mentioned that could improve the data density is to mine and display git cherry-pick information.

In addition to the new data, the experts were also asked about general missing features they would want with a visualization like this. The feature most of the participants mentioned that the search for branches would be good and could function similarly to the module and file search. According to the participants, this would vastly improve the speed specific branches could be found. One participant also mentioned that a commit search would be great and could also function similarly.

Another big critique nearly all of the participants had was that when file nodes get filtered the green and red addition and deletion lines do not get filtered which can result in a confusing view that shows a lot of unnecessary data.

Another feature that was mentioned by two of the participants was the ability to filter according to the file type. This could be implemented as part of the file and module search that already exists. Additionally, some of the participants also mentioned that additional hover dialogs would be good. Currently, the only hover dialog is available when hovering your mouse over a file node. This could be expanded to also be able to hover over commits and the addition/deletion connection.

In general, it was also mentioned that there may be a benefit in displaying the individual commits differently as an individual object. According to one of the experts, this would make it easier for the user to distinguish different commits. It would also give the user a target to trigger the hover dialog and make it possible to color code commits without having to color each file within the commit. Additionally, in terms of displaying commits differently one of the participants mentioned that it would be beneficial if it is possible to collapse blocks of commits.

One participant also mentioned that the sheer number of lines and objects can be overwhelming so it would be good if it is possible to have the option to change the opacity of different elements like the folder rings or the opacity of filtered-out file nodes.

Another problem one of the experts recognized was that currently file moves are getting ignored which can result in misleading data. Here it would be preferable according to the interviewee to also track the file renames properly and to give indicators when files are moved.

Also mentioned by some of the participants in combination with the ownership view that it would be good to have some kind of indicator where ownership changed. This would make it easier to analyze how ownership moved within a project or branch.

6.2 Evaluation Results

In general, the experts interviewed reacted positively to the prototype shown. They all stated an optimistic opinion about the future of visualization and that with some work and polishing it could be used to analyze data in real projects. From the presented scenarios the highest-rated scenario was the one about finding hotspots in the development history which could point to refactorings. This scenario was almost exclusively rated

with the highest score by the experts. Despite the good sentiment, there were also some issues recognized by the interviewees. The biggest ones that were mentioned by all of the experts were the missing filtering of the addition and deletion connections when filtering modules and files and the way how the user can find a specific branch. For the filtering of the addition and deletion connections, the reason that this was not implemented was due to the implementation of the line renderer for those lines. With the current implementation, it is not that easily doable, but this can be improved and fixed in the future. For the problem with searching and jumping to branches, it would need some deeper changes in how branches are stored so that it is possible to look up the location of the branch in the three-dimensional space. This would make it possible to search for a specific branch and also highlight and jump to the position of the branch within the three-dimensional space. In general, all those issues should be solvable in future iterations of the prototype.

CHAPTER 7

Discussion

This chapter is about the discussion of the results of this thesis. In Section 7.1 the research questions that guided this thesis will be answered. After that Section 7.2 will be about the general results of this thesis.

7.1 Research Questions

To guide this thesis and the design, development and evaluation of the prototype in Section 4.2 four research questions were designed. In this section, those research questions get revisited and answered with the collected results of the design evaluation (Section 4.4.3) and the scenario-based expert interview (Chapter 6). The four previously defined research questions were as follows:

- **RQ1a:** How do developers analyze the history and trends of a repository?
- **RQ1b:** How do developers rate the benefits of the proposed visualization?
- **RQ2:** What visualization and interaction methods are needed to show a holistic view of a code repository?
- **RQ3:** How suitable is the visualization to analyze the proposed benefits regarding source code evolution, code hotspots, code ownership and code expertise?

RQ1a: How do developers analyze the history and trends of a repository?

According to the conducted interviews in both the design evaluation and the final evaluation, experts stated that they currently mostly use tools that give statistical overviews like gitinspector (Section 3.3.1) or the built-in tools that are provided by

GitLab, GitHub. They also use the tools provided by their IDE like for example the tools built into IntelliJ IDEA. The git terminal with commands like „git log“ or „git blame“ is also a common way to analyze version control data at the moment. Although some of the experts have already seen and used more complex or even three-dimensional version control system visualizations none of the interviewees used them in their normal workflow.

RQ1b: How do developers rate the benefits of the proposed visualization?

During the evaluation of the design the three participants were also asked about the four proposed benefits the visualization should offer. Those were defined as source code evolution, code hotspots, code ownership and code expertise. On average those benefits and how it was presented in the proposed visualization received a rating of 2.75 on a scale from 1 to 5 (higher is better). The highest-rated benefit was the code hotspots mode with an average grade of 3.3. The lowest rated benefit the code ownership with a rating of 2.3. This was mostly due to the uncertainty of how ownership data gets calculated and interpreted as stated by the interviewees as this can be a sensitive topic. In general, the three participants of the design evaluation were more skeptical about the project at this point because they had never used something similar before but looked forward to how the fully implemented prototype would work.

RQ2: What visualization and interaction methods are needed to show a holistic view of a code repository?

To create a holistic visualization like the one prototyped as part of this project, both a strong mining component and high-performance 3D graphics are required to accomplish the huge amount of data. For the mining component, the already available Binocular project was chosen due to the familiarity of the author of this thesis with it and its capability to mine many different sources and export them as structured JSON files. For 3D graphics, the game engine Unity3D was chosen for its focus on high-performance real-time 3D graphics.

As part of this thesis a prototype was designed (Section 4.4) and developed (Chapter 5) to show such visualization and interaction methods and how those work together to create an interactive holistic three-dimensional version control system visualization. This was done by providing a suite of filtering and color-coding options. Due to the complexity of such visualization, a huge focus also has to be placed on the interaction. Interacting with different GUI and visualization elements should give proper feedback and show additional information to the user according to what was selected.

RQ3: How suitable is the visualization to analyze the proposed benefits regarding source code evolution, code hotspots, code ownership and code expertise?

In general, the four main benefits of the prototype described in the evaluation, Code Evolution, Hotspots, Ownership and Expertise were rated quite well by the experts with an average score of 4.2 on a scale from 1 to 5(higher is better). The highest-rated benefit was in the scenario hotspots with an average rating of 4.8. The lowest rated was the scenario Expertise with a rating of 3.7 which can still be considered an above-average rating. In conclusion, all the participants of the scenario-based expert interview gave a mostly positive assessment of the developed prototype with some critique about missing features for better usability which can be added in future iterations to create a better experience.

7.2 Results

The main focus of this thesis was to show the feasibility of a holistic three-dimensional version control visualization and that a visualization like this can give deeper insights into a version-controlled repository. This was accomplished by designing and developing such visualization as a functioning prototype. After the development, the prototype was evaluated with scenario-based interviews with a group of six experts who used version control visualizations or tools to analyze such regularly. All of the experts were interested in the prototype and recognized ways where such visualization could help to gather data. Despite the good results they also recognized some issues and missing features that the prototype didn't offer which they would prefer in order to make the visualization more usable. This mainly included the need for more filtering and parametrization options as well as additional improvements to the visualization itself.

CHAPTER 8

Conclusion

This chapter is about the results of this thesis and what advancements could be made in future iterations to improve on the already existing prototype. At first, there will be a retrospection of the whole project and the research done in Section 8.1. After that, in Section 8.2 the possibilities of Holocular will be discussed, where the project could head next and what improvements can be made according to the previously conducted expert interviews (Chapter 6). At last a small summary will be given in Section 8.3

8.1 Retrospection

All in all, this thesis shows that large-scale holistic three-dimensional version control visualizations are feasible and that such visualizations can give deeper insights into a version-controlled repository. During the research process of this thesis, already at the point of the design evaluation (Section 4.4.3) the participants showed interest in a concept like the one proposed. Many of them already stated that they were looking forward to the finished prototype and how it would work. This interest in the project early on during the research already suggested a promising future for the prototype. The concluding scenario-based expert interviews (Chapter 6) also showed that there is interest from the site of experts who use such visualizations. Especially the results of the scenario-based expert interviews were interesting for the result of this thesis and the future of the prototype as they were optimistic for its future.

In short, all those aspects were necessary for answering the three research questions that this thesis set out to answer. The first research question was hereby split into two parts whereas the first part was about how developers currently analyze the history and trends of a version-controlled repository. Here mostly basic tools were used and no complex or three-dimensional visualization. The second part of the first research question followed up with the design of a visualization that can help with such analysis tasks and the question of how developers are rating the benefits of such visualization. Here three

developers were asked who were already familiar with the analysis of version-controlled repositories. They rated the benefits with an average rating of 3.75 out of 5. With all the requirements sorted as part of research question two the prototype was developed and refined to answer the question of what visualization and interaction methods are shown to create a holistic view of a repository. At last research question three was all about the suitability of the developed prototype to analyze the proposed benefits. This was accomplished by interviewing six experts and showing them four scenarios that each focus on one specific benefit. Here the average rating of the experts was 4.2 out of 5. Those results are discussed more in-depth in Chapter 7.

Although the prototype visualization developed as part of this thesis (Chapter 5) proves that such a visualization is feasible and can help to give people in need of deeper insights into version-controlled repositories a better overview of a project, there is still a long way to go. As discussed in both Chapter 6 and Section 8.2 there are still a lot of features missing that would make a visualization like this ready for usage by everybody. This includes mainly additional filtering and parametrization options to access the searched data more efficiently. In future iterations, the prototype can also be improved with more focus on usability which may result in a usable tool for developers, researchers, teachers and project managers.

Still the goal of this thesis to show a working prototype as proof of concept of such visualization and that a visualization like this can extract useful data for its users was accomplished. The prototype and the following evaluation show that a visualization like this is feasible. It also sparked interest in such visualizations among the experts asked in the evaluation. With a little more work and improvements of the prototype, this can result in a real tool people want to use to analyze their software repositories.

8.2 Future Work

The prototype developed as part of this thesis already showed a promising state how a three-dimensional version control visualization could look like but there are still many things that can be improved and researched from this point. One of the main points would most likely be to implement a native C# based mining component into the project to be more flexible and independent of Binocular. Already during the development, there were issues with the compatibility of the database because of the independent development of Binocular. It would also increase the usability for new users because they would need only one software instead of first mining and exporting the data with Binocular and then visualizing it with Holocular. It would most likely also improve the performance of the mining component because of the better performance of C# compared to JavaScript.

Another improvement that could be made in future iterations would be a complete rewrite of the 3D visualization to improve how file nodes, folder rings and the addition/deletion connections are rendered. This can fix the issue most of the participants had during the scenario-based evaluation that the addition and deletion connections did not get filtered

out during the module/file search. A more performant implementation of the file nodes and folder rings would also result in the possibility of displaying not just the changed files but all files. This would vastly improve the usability of the ownership mode. It would make it possible to see the absolute ownership at each point in time and how it transferred from one version to another.

An improvement in the way the visualization renders the commits and file structure of a project can also help to render branches and the file structure more dynamically. This can create new ways of interacting with the structure like making folders or branches collapsable. This could also result in improved performance in large repositories if parts of the strand that aren't necessary to the user can be collapsed.

A different option is to expand Holocular in the future to add more data sources. This could include issue-tracking data from different issue-tracking systems like GitHub or GitLab. Knowing which commit connects to which issue can help in grouping or coloring commits according to the issue they were mentioned. Another data source that can be added is build/workflow/action data to each commit that triggered such. This can also result in an additional color highlighting mode in which the commits could be colored. Many more additional data sources can be added depending on the use case and development environment. Additional data sources like this can extend the data density of the visualization.

Some smaller improvements could find their way into future versions of Holocular like more parametrization options, additional tooltips or statistical analysis. Those can help to improve the usability of the visualization. Additional parametrization options can also help to find the data needed by the user more quickly.

To achieve a production-ready state the two most important parts are most likely the rewrite of the generation of the 3D structure including the renderer of the addition/deletion connection and the introduction of a built-in mining component to remove the dependency on Binocular. Both were already discussed in more depth in the paragraphs above. Additionally, a more refined error handly would need to be introduced for usage in a production environment.

8.3 Summary

As part of this thesis, a prototype of a holistic three-dimensional version control system visualization was conceptualized, designed and developed. It was also evaluated with a group of experts familiar with the analysis of version-controlled repositories with good results. A visualization like this can give developers, project managers, researchers and teachers greater insights into version-controlled repositories and create a holistic overview of a repository that is hard to achieve otherwise.

The main takeaway of this research was that even if three-dimensional version-control visualizations are rarely developed and used, there is still a huge potential in the field that should be explored further. With this in mind Holocular marks a starting point into

a new type of visualization that is worthwhile to explore further. New projects can build on the foundation of Holocular to extract even deeper data and present it to the user more excitingly and interactively.

List of Figures

2.1 Google Trends CVCS vs DVCS 6
2.2 Graphics Pipeline 8
2.3 Shader Graph Example 11

3.1 Code City 14
3.2 Other Code City Variants 15
3.3 VR-GitCity 16
3.4 Code House 16
3.5 VR-Git 17
3.6 VRCS 18
3.7 Githru 19
3.8 VisGi 20
3.9 GETAVIZ Behavioral View 20
3.10 GETAVIZ Evolutionary View 21
3.11 VRIDE 21
3.12 GitHub Insights 22
3.13 GitHubSkyline 23
3.14 Apache Kibble 23
3.15 Autodesk Hubble 24
3.16 GitClear 25
3.17 CHAOSS Dashboard 26
3.18 Gitinspector 27
3.19 Gource 28

4.1 Methodology 31
4.2 Blender Interface 32
4.3 3D Sketch 34
4.4 Cross-Section Sketch 35
4.5 Multi Branch Sketch 36
4.6 Multi Branch Sketch with Data 37
4.7 Advantages of 3D Visualizations over 2D Visualizations 38
4.8 Benefits Evaluation of the Design 39
4.9 Most Important Filtering Options 40

5.1 Gitinspector 45
5.2 First 3D Data Visualization 48
5.3 First radial 3D Visualization 50
5.4 Full Project with first change data 51
5.5 Branch Distribution 52
5.6 Branch Distribution Twisted 53
5.7 Full Project with first change data 54
5.8 Full Project with split change data closeup 55
5.9 Full Project with split change data 56
5.10 File Info 57
5.11 File Compare 58
5.12 Full Project with files colored by committer 59
5.13 Full Project with files colored by ownership 60
5.14 Full Project with files colored by ownership 61
5.15 Full Project with files colored by number of changes 62
5.16 Full Project with files colored by branch color 63
5.17 Commit Distribution Linear vs By-Date 63
5.18 Full Project with only the files that belong to a single owner highlighted . 64
5.19 Full Project with only the files that belong to a single branch highlighted 65
5.20 Full Project with only the files that belong to a specific module highlighted 66
5.21 UI Elements 66
5.22 Interactive Compass 67
5.23 Final Implementation 68

6.1 Scenario Based Expert Interviews Age Distribution 70
6.2 Scenario Based Expert Interviews Experience 71
6.3 Scenario Based Expert Interviews Scenario Suitability 73
6.4 Scenario Based Expert Interviews Addition Deletion Connections 74
6.5 Scenario Based Expert Interviews Hard Distinguished Committer Colors . 75
6.6 Scenario Based Expert Interviews File Coloring Ranking 76
6.7 Commit Distribution Linear vs By-Date 77

Listings

2.1 A simple single color Unity HLSL Shader. 10
5.1 Example JSON file of a list of persons with a key for the name and the age of the person. 47
5.2 Example C# class that mimics the structure of a JSON file. 47
5.3 Example C# class that converts a JSON string into a C# object. . . . 47
5.4 Example C# class that creates a simple movable window with a button inside it. 49
5.5 Example C# class that creates a custom mesh. 54
5.6 Git blame command to get the Ownership for each line of a file at a specific commit with an example how the output is formatted. 60

Acronyms

AR Augmented Reality. 14

CAD Computer Aided Design. 65

CI Continuous Integration. 75

GLSL OpenGL Shading Language. 8

GPU Graphics Processing Unit. 8

GUI Graphical User Interface. 45–47, 64, 80

HLSL High-Level Shader Language. 8, 9, 11, 42

IDE Integrated Development Environment. 18, 80

MR Mixed Reality. 14

PC Personal Computer. 47

RCS Revision Control System. 16, 19

VCS Version Control System. 5, 6, 15, 29, 37, 67

VR Virtual Reality. 8, 14, 16, 18, 22

Bibliography

[1] C. L. Jeffery, „The city metaphor in software visualization“, vol. 2901, Vaclav Skala Union Agency, 2019, pp. 153–161, ISBN: 9788086943374. DOI: 10.24132/CSRN.2019.2901.1.18.

[2] A. M. Nababan and B. Hendradjaya, „Visualization of codes and code smells with 3d mushroom farm metaphor“, vol. 2019-July, IEEE, Jul. 2019, pp. 250–255, ISBN: 978-1-7281-2418-6. DOI: 10.1109/ICEEI47359.2019.8988796. [Online]. Available: https://ieeexplore.ieee.org/document/8988796/.

[3] H. Koike and H.-C. Chu, „Vrcs: Integrating version control and module management using interactive three-dimensional graphics“, IEEE, 1997, pp. 168–173, ISBN: 0818681446.

[4] A. Fiechter, R. Minelli, C. Nagy, and M. Lanza, „Visualizing github issues“, IEEE, Sep. 2021, pp. 155–159, ISBN: 978-1-6654-3144-6. DOI: 10.1109/VISSOFT52517.2021.00030. [Online]. Available: https://ieeexplore.ieee.org/document/9604892/.

[5] X. Liu and R. Holmes, „Exploring developer preferences for visualizing external information within source code editors“, IEEE, Sep. 2020, pp. 27–37, ISBN: 978-1-7281-9914-6. DOI: 10.1109/VISSOFT51673.2020.00008. [Online]. Available: https://ieeexplore.ieee.org/document/9240499/.

[6] M. Lungu, R.-H. Pfeiffer, M. D'Ambros, M. Lanza, and J. Findahl, „Can git repository visualization support educators in assessing group projects?“, IEEE, Oct. 2022, pp. 187–191, ISBN: 978-1-6654-8092-5. DOI: 10.1109/VISSOFT55257.2022.00030. [Online]. Available: https://ieeexplore.ieee.org/document/9978497/.

[7] J. Grabner, R. Decker, T. Artner, M. Bernhart, and T. Grechenig, „Combining and visualizing time-oriented data from the software engineering toolset“, 2018, pp. 76–86. DOI: 10.1109/VISSOFT.2018.00016.

[8] N. N. Zolkifli, A. Ngah, and A. Deraman, „Version control system: A review“, *Procedia Computer Science*, vol. 135, pp. 408–415, 2018, ISSN: 18770509. DOI: 10.1016/j.procs.2018.08.191. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050918314819.

[9] D. Spinellis, „Version control systems“, *IEEE Software*, vol. 22, pp. 108–109, 5 Sep. 2005, ISSN: 0740-7459. DOI: 10.1109/MS.2005.140. [Online]. Available: https://ieeexplore.ieee.org/document/1504674/.

[10] C. F. Malmsten and S. B. Sullivan, „Evolution of version control systems - comparing centralized against distributed version control models“, Sep. 2010, ISSN: 1651-4769. [Online]. Available: https://gupea.ub.gu.se/handle/2077/23474.

[12] N Deepa and K. Lb, „An analysis on version control systems“, *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020. DOI: 10.1109/ic-ETITE47903.2020.39. [Online]. Available: https://www.researchgate.net/publication/340973855.

[18] D. Spinellis, „Git“, *IEEE Software*, vol. 29, pp. 100–101, 3 May 2012, ISSN: 0740-7459. DOI: 10.1109/MS.2012.61. [Online]. Available: http://ieeexplore.ieee.org/document/6188603/.

[20] S. M. Ali, N. Gupta, G. K. Nayak, and R. K. Lenka, „Big data visualization: Tools and challenges“, IEEE, Dec. 2016, pp. 656–660, ISBN: 978-1-5090-5256-1. DOI: 10.1109/IC3I.2016.7918044. [Online]. Available: http://ieeexplore.ieee.org/document/7918044/.

[21] M. ORAL and A. A. ELMAS, „A brief history of 3d mesh compression“, 2017, pp. 136–140.

[23] R. Müller and D. Zeckzer, „Past, present, and future of 3d software visualization - a systematic literature analysis“, SCITEPRESS - Science, and Technology Publications, 2015, pp. 63–74, ISBN: 978-989-758-088-8. DOI: 10.5220/0005325700630074. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005325700630074.

[30] A. Teyseyre and M. Campo, „An overview of 3d software visualization“, *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 87–105, 1 Jan. 2009, ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.86. [Online]. Available: http://ieeexplore.ieee.org/document/4564449/.

[31] S. Ardigò, C. Nagy, R. Minelli, and M. Lanza, „Visualizing data in software cities“, pp. 145–149, 2021. DOI: 10.1109/VISSOFT52517.2021.00028.

[32] F. Pfahler, R. Minelli, C. Nagy, and M. Lanza, „Visualizing evolving software cities“, IEEE, Sep. 2020, pp. 22–26, ISBN: 978-1-7281-9914-6. DOI: 10.1109/VISSOFT51673.2020.00007. [Online]. Available: https://ieeexplore.ieee.org/document/9240484/.

[33] P. Khaloo, M. Maghoumi, E. Taranta, D. Bettner, and J. Laviola, „Code park: A new 3d code visualization tool“, IEEE, Sep. 2017, pp. 43–53, ISBN: 978-1-5386-1003-9. DOI: 10.1109/VISSOFT.2017.10. [Online]. Available: http://ieeexplore.ieee.org/document/8091185/.

[34] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, and L. Nafeie, „Islandviz: A tool for visualizing modular software systems in virtual reality“, IEEE, Sep. 2018, pp. 112–116, ISBN: 978-1-5386-8292-0. DOI: 10.1109/VISSOFT.2018.00020. [Online]. Available: https://ieeexplore.ieee.org/document/8530137/.

[35] A. Ciani, R. Minelli, A. Mocci, and M. Lanza, „Urbanit: Visualizing repositories everywhere“, IEEE, Sep. 2015, pp. 324–326, ISBN: 978-1-4673-7532-0. DOI: 10.1109/ICSM.2015.7332479. [Online]. Available: http://ieeexplore.ieee.org/document/7332479/.

[36] R. Brito, A. Brito, G. Brito, and M. T. Valente, „Gocity: Code city for go“, IEEE, Feb. 2019, pp. 649–653, ISBN: 978-1-7281-0591-8. DOI: 10.1109/SANER.2019.8668008. [Online]. Available: https://ieeexplore.ieee.org/document/8668008/.

[37] K. Dodia, „Softization-3d visualization of a version control repository“, 2019.

[38] R. Oberhauser, „Vr-gitcity: Immersively visualizing git repository evolution using a city metaphor in virtual reality“, 2024.

[39] A. Hori, M. Kawakami, and M. Ichii, „Codehouse: Vr code visualization tool“, IEEE, Sep. 2019, pp. 83–87, ISBN: 978-1-7281-4939-4. DOI: 10.1109/VISSOFT.2019.00018. [Online]. Available: https://ieeexplore.ieee.org/document/8900968/.

[40] R. Oberhauser, „Vr-git: Git repository visualization and immersion in virtual reality“, 2022.

[41] Y. Kim, J. Kim, H. Jeon, *et al.*, „Githru: Visual analytics for understanding software development history through git metadata analysis“, *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, pp. 656–666, 2 Feb. 2021, ISSN: 1077-2626. DOI: 10.1109/TVCG.2020.3030414. [Online]. Available: https://ieeexplore.ieee.org/document/9222261/.

[42] S. Elsen, „Visgi: Visualizing git branches“, IEEE, Sep. 2013, pp. 1–4, ISBN: 978-1-4799-1457-9. DOI: 10.1109/VISSOFT.2013.6650522. [Online]. Available: http://ieeexplore.ieee.org/document/6650522/.

[43] D. Baum, J. Schilbach, P. Kovacs, U. Eisenecker, and R. Muller, „Getaviz: Generating structural, behavioral, and evolutionary views of software systems for empirical evaluation“, IEEE, Sep. 2017, pp. 114–118, ISBN: 978-1-5386-1003-9. DOI: 10.1109/VISSOFT.2017.12. [Online]. Available: http://ieeexplore.ieee.org/document/8091194/.

[44] D. Baum, P. Kovacs, and R. Müller, „Fostering collaboration of academia and industry by open source software“, DOI: 10.18420/SE2020_52. [Online]. Available: http://www.gisa.de.

[45] V. S. S. Castillo, L. Merino, G. Hecht, and A. Bergel, „Vr-based user interactions to exploit infinite space in programming activities“, IEEE, Nov. 2021, pp. 1–5, ISBN: 978-1-6654-0956-8. DOI: 10.1109/SCCC54552.2021.9650396. [Online]. Available: https://ieeexplore.ieee.org/document/9650396/.

[53] A. Akhtar, „Role of apache software foundation in big data projects“, May 2020. [Online]. Available: http://arxiv.org/abs/2005.02829.

[67] B. Perens and M. Sroka, „The open source definition“, 2007. [Online]. Available: http://perens.com/OSD.html.

[68] A. Fuggetta, „Open source software—an evaluation“, *Journal of Systems and Software*, vol. 66, pp. 77–90, 1 Apr. 2003, ISSN: 01641212. DOI: 10.1016/S0164-1212(02)00065-1. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121202000651.

[71] A. H. Caudwell, „Gource“, ACM, Oct. 2010, pp. 73–74, ISBN: 9781450302401. DOI: 10.1145/1869542.1869554. [Online]. Available: https://dl.acm.org/doi/10.1145/1869542.1869554.

[72] M. Ogawa and K.-L. Ma, „Code_swarm: A design study in organic software visualization“, *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1097–1104, 6 Nov. 2009, ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.123. [Online]. Available: http://ieeexplore.ieee.org/document/5290717/.

[75] J. D. Watson and F. H. C. Crick, „The structure of dna“, *Cold Spring Harbor Symposia on Quantitative Biology*, vol. 18, pp. 123–131, 0 Jan. 1953, ISSN: 0091-7451. [Online]. Available: http://symposium.cshlp.org/cgi/doi/10.1101/SQB.1953.018.01.020.

[76] S. Sagiroglu and D. Sinanc, „Big data: A review“, IEEE, May 2013, pp. 42–47, ISBN: 978-1-4673-6404-1. DOI: 10.1109/CTS.2013.6567202. [Online]. Available: http://ieeexplore.ieee.org/document/6567202/.

[77] I. Yaqoob, I. A. T. Hashem, A. Gani, *et al.*, „Big data: From beginning to future“, *International Journal of Information Management*, vol. 36, pp. 1231–1247, 6 Dec. 2016, ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2016.07.009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0268401216304753.

[80] A. Quiña-Mera, P. Fernandez, J. M. García, and A. Ruiz-Cortés, „Graphql: A systematic mapping study“, *ACM Computing Surveys*, vol. 55, pp. 1–35, 10 Oct. 2023, ISSN: 0360-0300. DOI: 10.1145/3561818. [Online]. Available: https://dl.acm.org/doi/10.1145/3561818.

[95] M. Greiler, K. Herzig, and J. Czerwonka, „Code ownership and software quality: A replication study“, IEEE, May 2015, pp. 2–12, ISBN: 978-0-7695-5594-2. DOI: 10.1109/MSR.2015.8. [Online]. Available: http://ieeexplore.ieee.org/document/7180062/.

[96] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, „Don't touch my code!“, ACM, Sep. 2011, pp. 4–14, ISBN: 9781450304436. DOI: 10.1145/2025113.2025119. [Online]. Available: https://dl.acm.org/doi/10.1145/2025113.2025119.

Book References

[19] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012, ISBN: 1449345050.

[85] D. Ilett, *Building Quality Shaders for Unity®*. Apress, 2022, ISBN: 978-1-4842-8651-7. DOI: 10.1007/978-1-4842-8652-4.

[86] K. Halladay, „Writing shaders in unity", in K. Halladay, Ed. Apress, 2019, pp. 299–325, ISBN: 978-1-4842-4457-9. DOI: 10.1007/978-1-4842-4457-9_16. [Online]. Available: https://doi.org/10.1007/978-1-4842-4457-9_16.

Online References

[11] „Google trends cvcs vs dvcs“. (2024), [Online]. Available: https://trends.google.com/trends/explore?date=all&q=centralized\%20version\%20control,distributed\%20version\%20control (visited on 06/25/2024).

[13] „Revision control system“. (2022), [Online]. Available: https://www.gnu.org/software/rcs/ (visited on 06/07/2024).

[14] „Concurrent versions system“. (2019), [Online]. Available: https://cvs.nongnu.org (visited on 06/07/2024).

[15] „Apache subversion“. (2023), [Online]. Available: https://subversion.apache.org (visited on 06/07/2024).

[16] „Git“. (2024), [Online]. Available: https://git-scm.com (visited on 06/07/2024).

[17] „Stackoverflow git usage“. (2023), [Online]. Available: https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/ (visited on 06/07/2024).

[22] „Graphics compendium - chapter 2: The graphics pipeline“. (2024), [Online]. Available: https://graphicscompendium.com/intro/01-graphics-pipeline (visited on 07/01/2024).

[24] „Opengl“. (2024), [Online]. Available: https://www.opengl.org (visited on 05/25/2024).

[25] „High-level shader language (hlsl) - win32 apps | microsoft learn“. (2024), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl (visited on 07/01/2024).

[26] „Unity - manual: Writing shaders overview“. (2024), [Online]. Available: https://docs.unity3d.com/Manual/SL-ShadingLanguage.html (visited on 07/01/2024).

[27] „Unity shader reference examples“. (2024), [Online]. Available: https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html (visited on 07/27/2024).

[28] „About shader graph“. (2024), [Online]. Available: https://docs.unity3d.com/Packages/com.unity.shadergraph@17.0/manual/index.html (visited on 07/01/2024).

[29] „Unity learn - introduction to shadergraph“. (2024), [Online]. Available: https://learn.unity.com/tutorial/introduction-to-shader-graph (visited on 07/27/2024).

[46] „Github market share“. (2024), [Online]. Available: https://www.datanyze.com/market-share/source-code-management--315 (visited on 07/02/2024).

[47] „Github“. (2024), [Online]. Available: https://www.github.com (visited on 06/07/2024).

[48] „Github skyline“. (2024), [Online]. Available: https://skyline.github.com (visited on 06/07/2024).

[49] „Github network graph“. (2024), [Online]. Available: https://github.com/INSO-TUWien/Binocular/network (visited on 06/24/2024).

[50] „Github code frequency“. (2024), [Online]. Available: https://github.com/INSO-TUWien/Binocular/graphs/code-frequency (visited on 06/24/2024).

[51] „Apache kibble“. (2024), [Online]. Available: https://kibble.apache.org (visited on 06/07/2024).

[52] „Apache foundation“. (2024), [Online]. Available: https://apache.org (visited on 06/07/2024).

[54] „Apache kibble dashboard“. (2024), [Online]. Available: https://demo.kibble.apache.org/dashboard.html?page=repos (visited on 06/07/2024).

[55] „Autodesk hubble“. (2024), [Online]. Available: https://autodesk.github.io/hubble/ (visited on 06/07/2024).

[56] „Autodesk hubble demo page“. (2024), [Online]. Available: https://autodesk.github.io/hubble/housekeeping-git-traffic (visited on 06/07/2024).

[57] „Gitclear“. (2024), [Online]. Available: https://www.gitclear.com (visited on 06/07/2024).

[58] „Pluralsight flow“. (2024), [Online]. Available: https://www.pluralsight.com/product/flow (visited on 06/07/2024).

[59] „Code climate“. (2024), [Online]. Available: https://codeclimate.com (visited on 06/07/2024).

[60] „Waydev“. (2024), [Online]. Available: https://waydev.co (visited on 06/07/2024).

[61] „Linearb“. (2024), [Online]. Available: https://linearb.io (visited on 06/07/2024).

[62] „Gitclear demo“. (2024), [Online]. Available: https://www.gitclear.com/users/register (visited on 06/07/2024).

[63] „Chaoss“. (2024), [Online]. Available: https://chaoss.community (visited on 06/07/2024).

[64] „Grimoirelab“. (2024), [Online]. Available: https://chaoss.github.io/grimoirelab/ (visited on 07/27/2024).

[65] „8knot“. (2024), [Online]. Available: https://metrix.chaoss.io (visited on 07/27/2024).

[66] „Bitergia analytics“. (2024), [Online]. Available: https://chaoss.biterg.io/ (visited on 06/07/2024).

[69] „Gitinspector“. (2016), [Online]. Available: https://github.com/ejwa/gitinspector (visited on 08/02/2024).

[70] „Gource“. (2024), [Online]. Available: https://gource.io (visited on 06/07/2024).

[73] „Codeflower“. (2024), [Online]. Available: http://www.redotheweb.com/CodeFlower/ (visited on 06/07/2024).

[74] „Blender“. (2024), [Online]. Available: http://www.redotheweb.com/CodeFlower/ (visited on 06/07/2024).

[78] „Binocular“. (2024), [Online]. Available: https://github.com/INSO-TUWien/Binocular (visited on 06/07/2024).

[79] „Inso“. (2024), [Online]. Available: https://www.inso.tuwien.ac.at (visited on 07/27/2024).

[81] „Unity 3d“. (2024), [Online]. Available: https://unity.com (visited on 07/02/2024).

[82] „Unity virtual production“. (2024), [Online]. Available: https://web.archive.org/web/20240304204219/https://unity.com/roadmap/virtual-production (visited on 07/02/2024).

[83] „Unity industrial visualizations“. (2024), [Online]. Available: https://unity.com/solutions/design-visualization (visited on 07/02/2024).

[84] „Usage statistic itch.io“. (2024), [Online]. Available: https://itch.io/game-development/engines/most-projects (visited on 07/02/2024).

[87] „Arangodb“. (2024), [Online]. Available: https://arangodb.com (visited on 07/30/2024).

[88] „Arangoexport“. (2024), [Online]. Available: https://docs.arangodb.com/stable/components/tools/arangoexport/ (visited on 07/30/2024).

[89] „Microsoft learn serialization“. (2023), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/serialization/ (visited on 06/01/2024).

[90] „Runtime file browser“. (2024), [Online]. Available: https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006 (visited on 07/02/2024).

[91] „Unity ui“. (2024), [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/index.html (visited on 06/26/2024).

[92] „Unity gui reference“. (2024), [Online]. Available: https://docs.unity3d.com/ScriptReference/GUI.html (visited on 06/26/2024).

[93] „Unity gui window documentation“. (2024), [Online]. Available: https://docs.unity3d.com/ScriptReference/GUI.Window.html (visited on 06/26/2024).

[94] „Unity mesh documentation“. (2024), [Online]. Available: https://docs.unity3d.com/ScriptReference/Mesh.html (visited on 06/27/2024).

Appendices

Appendix A

Pre-Evaluation Holocular

Demographic & Experience

1. What is your Age?

   Mark only one oval.

   - -20
   - 20-25
   - 26-30
   - 31-40
   - 40-50
   - 51-

2. What is your experience level with version-controlled repositories?

   Mark only one oval.

	1	2	3	4	5	
Begi	◯	◯	◯	◯	◯	Professional

3. Do you use tools to analyze version-controlled repositories?

   Mark only one oval.

   - Yes
   - No

4. If the previous Question is answered with yes which tools do you use?

5. What is your experience level with analyzation tools of version-controlled repositories?

Mark only one oval.

	1	2	3	4	5	
Begi	◯	◯	◯	◯	◯	Professional

6. How regularly do you use those tools?

Mark only one oval.

- ◯ Never
- ◯ Rarely
- ◯ Often
- ◯ On a daily basis

3D Visualizations of Version-Controlled Repositories

7. Do you think a 3D visualization could help to visualize version-controlled Repositories over 2 visualizations?

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ I don't know.

8. Have you ever heard of any 3D visualization for version-controlled repositories?

   Mark only one oval.

   - ◯ Yes
   - ◯ No

9. If the previous question was answered with yes which one (name or type of visualization)?

   _____

10. Have you ever used a 3D visualization of a version-controlled repository?

    Mark only one oval.

    - ◯ Yes
    - ◯ No

11. If the previous question was answered with yes which one?

    _____

12. Why do you think 3D visualizations are rarely developed and used for version-controlled repositories?

    _____

    _____

    _____

    _____

    _____

13. "3D visualizations of version controlled repositories rarely use state of the art 3D graphics. Do you agree with this statement and why do you think this is the case?

14. What do you think about these proposed advantages of a 3D visualization over a 2D visualization and can you think of other advantages? (Sort from most 1 to least important

Mark only one oval per row.

	1	2	3	4
More Data can be displayed at once	◯	◯	◯	◯
You can get a new perspective on data	◯	◯	◯	◯
Looks good for presentations	◯	◯	◯	◯
Relations can be displayed more easily	◯	◯	◯	◯

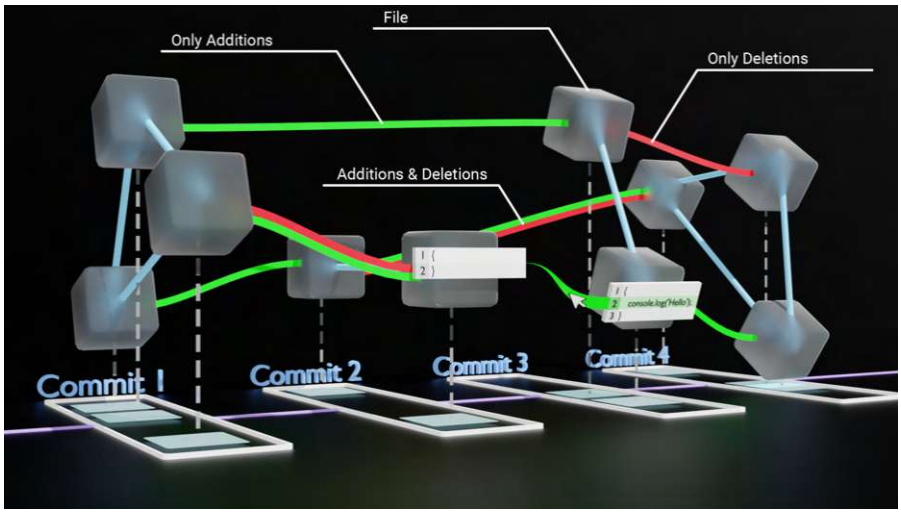
15. Can you think of other advantages of 3D visualizations over 2D visualizations?

My Vision

Holistic Helix like visualization of a whole version-controlled repository at once.

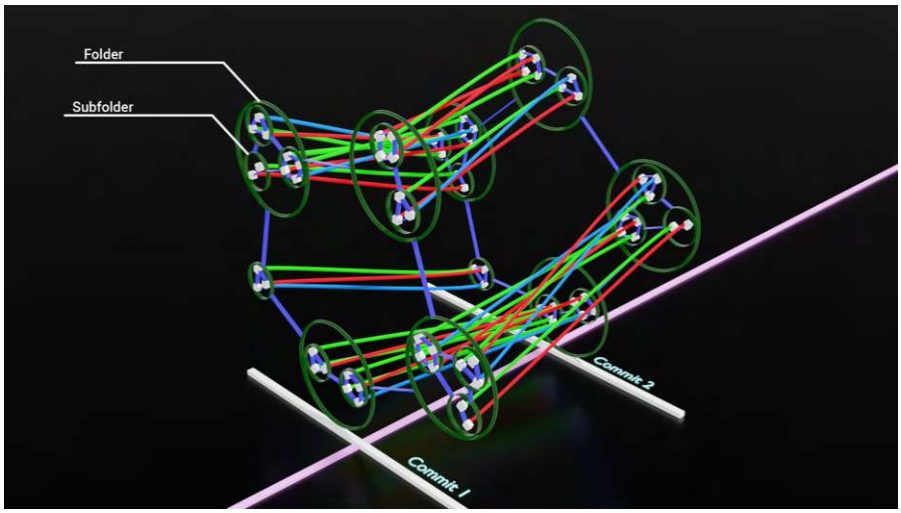
One Strand:

Here is an example of how one strand could look like that visualizes a branch with four commits. The gray cubes are stand-ins for files and the line between the files is there to help in t spatial understanding of which files belong to a commit. In Commit two and three only two files have changed in this example. The green and red lines between the commits show if there were additions or deletions or both between the commits. Additionally for the changes between comr two and three an exemplary interaction is shown to display detailed changes.



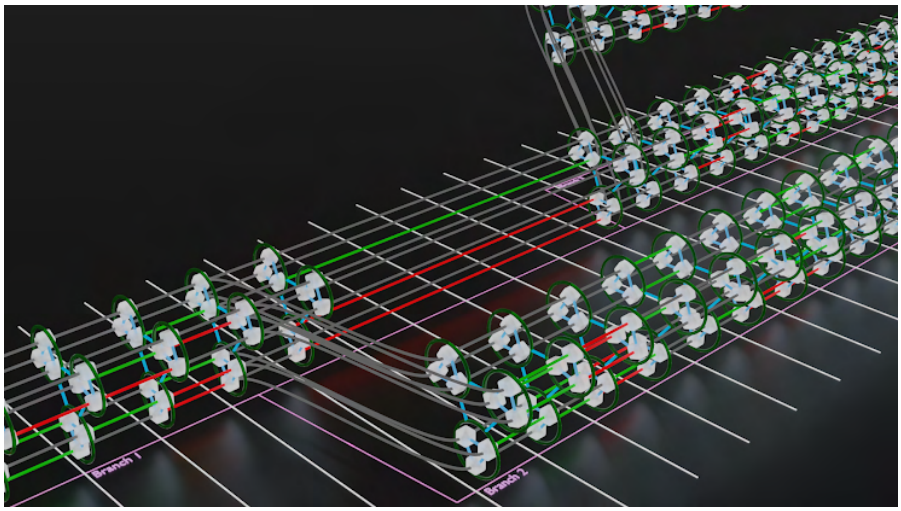
Nested File structure (only one commit):

Here you can see how a nested file structure could look like where each folder is visualized as a Strad as described above but the strands for each folder are nested into bigger strands.



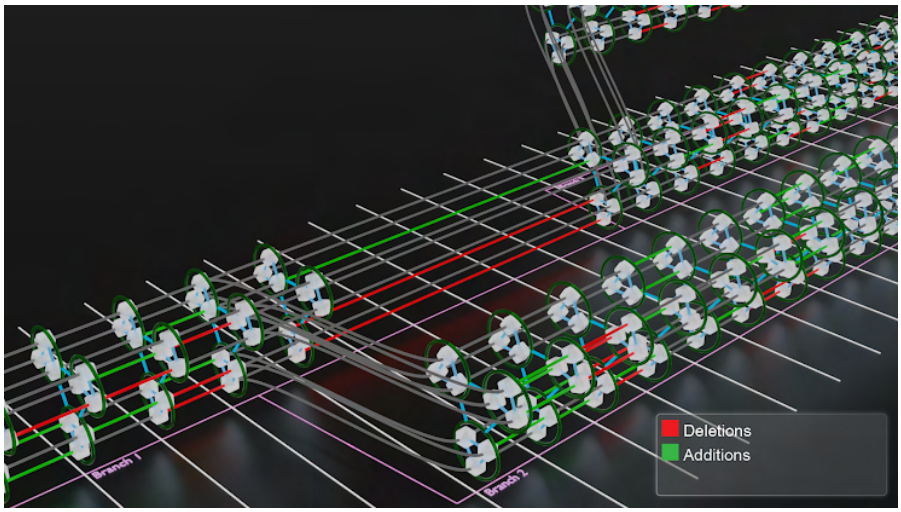
Multiple branches:

Here you can see a view of how multiple branches could look like. For simplicity, the example project only contains three folders with three files each.



Evaluation of benefits

Source Code Evolution



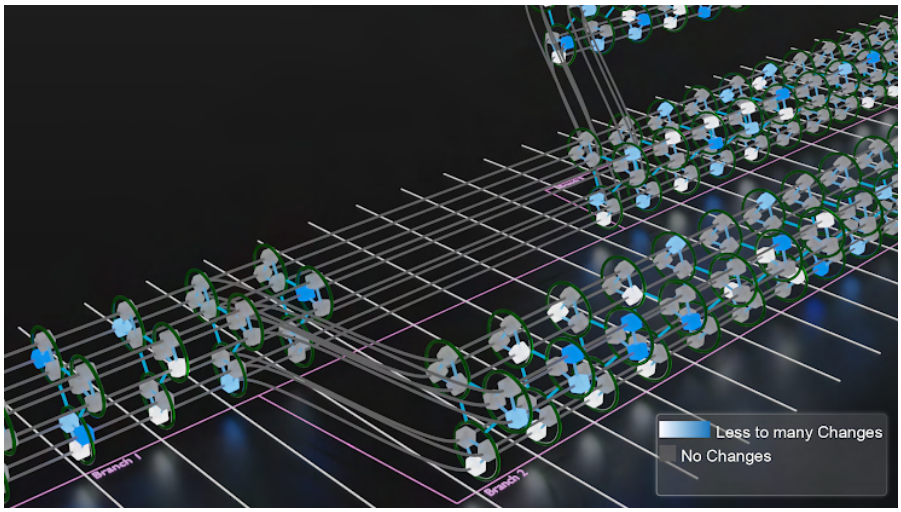
16. How do you rate the usefulness of this type of analysis for your work on a scale from 1 to

Mark only one oval.

	1	2	3	4	5	
reall	◯	◯	◯	◯	◯	not useful at all

17. What are your thoughts about analyzing the evolution of a repository this way and how do you think this could help developers in their daily work?

Code Hotspots



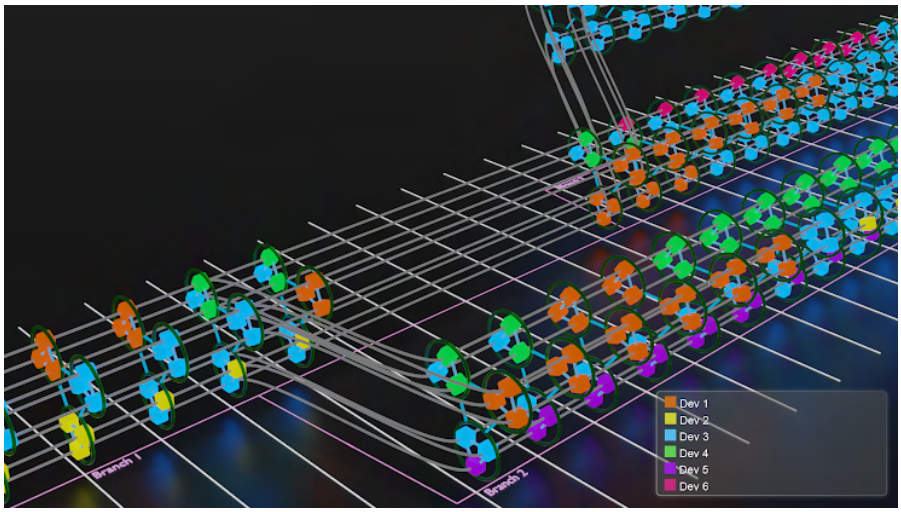
18. How do you rate the usefulness of this type of analysis for your work on a scale from 1 to

Mark only one oval.

	1	2	3	4	5	
reall	◯	◯	◯	◯	◯	not useful at all

19. What are your thoughts about analyzing hotspots in a repository this way and how do you think this could help developers in their daily work?

Code Ownership



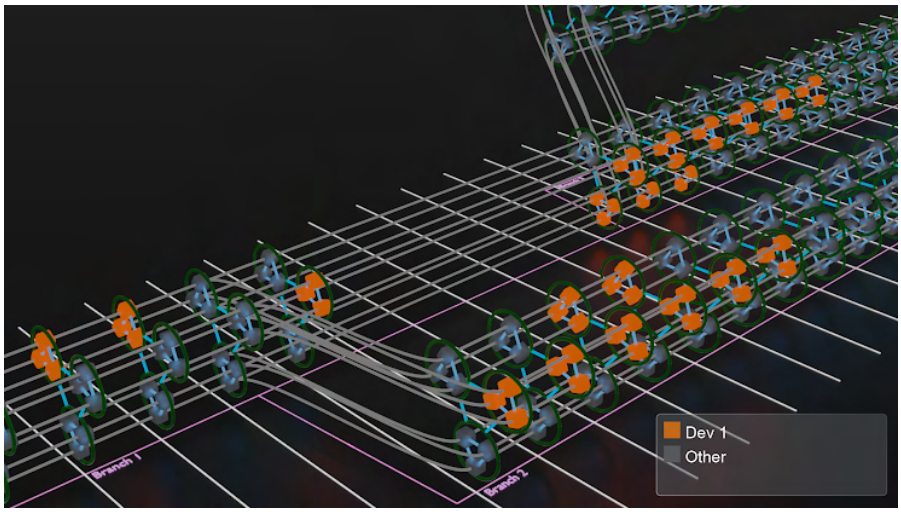
20. How do you rate the usefulness of this type of analysis for your work on a scale from 1 to

Mark only one oval.

	1	2	3	4	5	
reall	◯	◯	◯	◯	◯	not useful at all

21. What are your thoughts about analyzing the ownership of a repository this way and how d you think this could help developers in their daily work?

Code Expertise (Variation of Ownership with one Developer Extracted)



22. How do you rate the usefulness of this type of analysis for your work on a scale from 1 to

Mark only one oval.

	1	2	3	4	5	
reall	◯	◯	◯	◯	◯	not useful at all

23. What are your thoughts about analyzing who has expertise on what part of a repository lik shown abought and how do you think this could help developers in their daily work?

Data Filtering and Configuration

24. Rank the usefulness of the proposed filteringoptions below from 1 (most useful) to 5 (leas useful)

Mark only one oval per row.

	1	2	3	4	5
Date and Time	◯	◯	◯	◯	◯
Developer	◯	◯	◯	◯	◯
Branches	◯	◯	◯	◯	◯
Specific Files	◯	◯	◯	◯	◯
Developer Contribution in %	◯	◯	◯	◯	◯

25. What method do you think is better for distributing the individual Commits?

Mark only one oval.

◯ Equally Distribute Commits
◯ Distribute According to Commit Date

26. Can you think of different methods to filter/brush the data visualized?

Evaluation of Vision

27. What are your thoughts about the proposed visualization?

28. What additional insights into a repository do you think a visualization like this could give?

29. How do you think a holistic visualization like this could improve the workflow of developer that analyze version-controlled repositories on a regular basis?

Appendix B

Scenario-Based Expert Interview - Holocular

Demographic

1. What is your age?

   Mark only one oval.

   - 19
   - 20 - 25
   - 26 - 30
   - 31 - 40
   - 41- 50
   - 51 -

2. What is your experience level with version-controlled repositories?

   Mark only one oval.

	1	2	3	4	5	
Begi						Professional

3. Do you use tools to analyze version-controlled repositories?

   Mark only one oval.

   - Yes
   - No

4. What is your experience level with analysis tools of version-controlled repositories?

Mark only one oval.

	1	2	3	4	5	
Begi	◯	◯	◯	◯	◯	Professional

5. How regularly do you use those tools?

Mark only one oval.

- ◯ Never
- ◯ Rarely
- ◯ Often
- ◯ On a daly basis

3D Version Controls System Visualisations

6. Have you ever heard of or used a 3D VCS visualization? If yes which one?

7. What do you think are the main problems of 3D VCS visualizations and why are they rarely used?

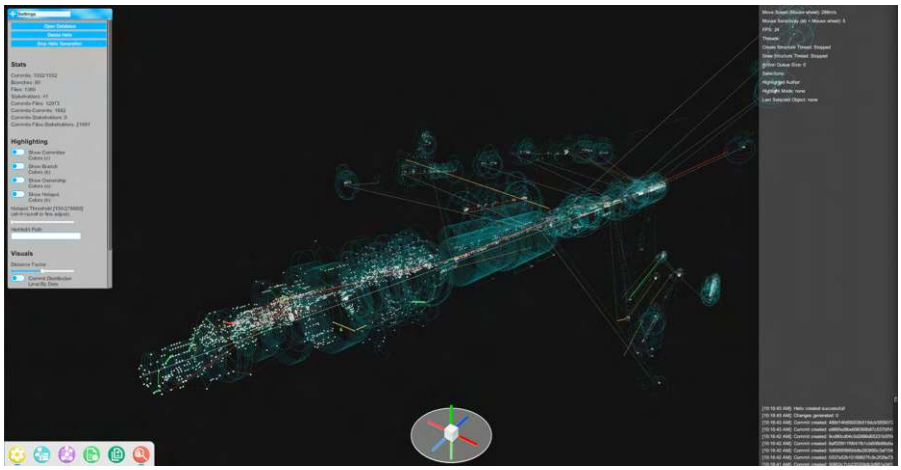
8. Do you think that in the future with more developments in 3D graphics and broader usage o technologies like VR, AR, and MR, 3D visualizations could become more common?

Mark only one oval.

	1	2	3	4	5	
Unlil	◯	◯	◯	◯	◯	Very Likely

Live Demo of Holocular

Holocular



Thoughts about the Prototype

9. What are your initial thoughts about the shown Prototype?

Scenarios

In the following sections, you will be walked through multiple scenarios to show use cases for this visualization. For each scenario you will be asked how suitable Holocular is for extracting the information and if you recognized any possible problems.

Scenario 1: Evolution

Open a database and analyse the connections between each file/commit. Find a connection with many changes (for example lib/git.js) and compare the source code of the two versions of the file.

10. How suitable do you think Holocular is to extract source code evolution data?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Really suitable

11. Did you recognize any problems with this scenario scenario and if yes which?

Scenario 2: Hotspots

Open a database and try to get an overview of where in the history of the project a lot of files got changed within one commit which could point at a refactoring or merge request and open the GitHub commit URL of one of those to check if it is really a refactoring.

12. How suitable do you think Holocular is to find hotspots?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Really suitable

13. Did you recognize any problems with this scenario scenario and if yes which?

Scenario 3: Ownership

Open a database and check who did the most commits and who has had the most ownership during the development of the provided repository. Did ownership change?

14. How suitable do you think Holocular is to get an overview of the ownership data of a proje

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Really suitable

15. Did you recognize any problems with this scenario scenario and if yes which?

Scenario 4: Expertise

Open a database and try to find the branch feature/137 and find out which developer did most of the commits on this feature.

16. How suitable do you think Holocular is to find expertise for a part of the repository?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Really suitable

17. Did you recognize any problems with this scenario scenario and if yes which?

Final Thoughts

18. After seeing the use cases of Holocular in action did your perception change?

19. Rank the four different coloring modes from most (1) to least (4) useful.

Mark only one oval per row.

	1	2	3	4
Committer	◯	◯	◯	◯
Branch	◯	◯	◯	◯
Ownership	◯	◯	◯	◯
Hotspots	◯	◯	◯	◯

20. Do you prefer the linear or by-date distribution of commits?

Mark only one oval.

- ◯ linear
- ◯ by-date

21. Do you prefer the helix with the folder rings or without?

Mark only one oval.

- ◯ With
- ◯ Without

22. What additional data do you think could be shown as part of the visualization?

23. What additional filters or brushing options do you think could be added to improve the visualization?

Thank you for your participation!