Unified User Interface

Development for

Cross- Disciplinary Scientific Computing Applications

by

Prerak Shah
B.E., A.D.Patel Institute of Technology, 2019

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Unified User Interface

Development for

Cross- Disciplinary Scientific Computing Applications

by

Prerak Shah
B.E., A.D.Patel Institute of Technology, 2019

Supervisory Committee

Dr. Teseo Schneider, Supervisor
(Department of Computer Science)

Dr. Jens Weber, Departmental Member
(Department of Computer Science)

ABSTRACT

In this study, we create a cross-platform unified user interface for general-purpose scientific computing. Our project work is guided by the intention to face the growing importance of scientific computing, especially in the field of geometric computing. Making user visual interfaces has remained a non-trivial task, even with the swift advancement of computational tools and their incorporation into other fields. The ability to combine developer integration features with user accessibility is something that many current systems struggle with. Finding out whether it is possible to create a single user interface that encompasses all of the primary paradigms of scientific computing and allows developers to easily integrate their code bases while still being understandable to users in unrelated professions is the main purpose of this project.

In order to achieve this, the suggested user interface solution makes use of the cross-platform compatibility and stability offered by web-based applications. By leveraging a robust ecosystem of TypeScript and JavaScript libraries, the user interface (UI) operates within mainstream browsers and benefits from the stability of browsers such as Chrome. The project also presents a novel strategy utilising an on-site Representational State Transfer (REST) server and tackles the challenging problem of cross-platform functionality. This server is in charge of binary execution, shape conversion (particularly for geometric objects), file storage, and query handling. The ultimate goal is to create a general interface specification protocol and a user interface that can be customised to meet the specific needs of different scientific computing projects, even those that support dynamic components.

Contents

List of Figures

ACKNOWLEDGMENTS

To start, I would like to extend my heartfelt thanks to Dr. Teseo Schneider for his consistent support and valuable guidance during the duration of this project. His dedication and mentorship throughout my studies at the University of Victoria have been truly praiseworthy.

The University of Victoria and the Department of Computer Science have greatly contributed to fostering an exceptional atmosphere where I could enhance and develop my technical and employability skills.

I am immensely grateful to my close friends Sai Chandra Madhuri, Priyadharsini Srinivasan, and Sachin Chaudhary for their steadfast backing and motivation. They have truly been my rock during the toughest moments I've faced.

My teammate, Yuelong Li's dedication and valuable input in this project deserve recognition. I feel privileged to have collaborated with such a committed and skilled colleague.

DEDICATION

I'd like to express my dedication for this project to my cherished family. To my mother, Binaben Anilkumar Shah, whose unwavering belief in me, support, and inspiration have accompanied me throughout my academic journey. To my father, Anilkumar Shah, whose counsel and wisdom have played a significant role in shaping the person I have become today. I also want to thank my brother, Nilay Shah, who has been my mentor and a dependable friend when I needed one. Your unwavering presence and belief in me have been instrumental in my journey.

Chapter One

Introduction

1.1 Background and Motivation

In the present exploration, scientific computing presents a significant part in driving new discoveries and innovations in various scientific domains. Researcher progress in their fields has been extraordinarily supported by the multiplication of computational strategies and tools. The Deep Mind AlphaFold project's success in protein folding and biology exemplifies the transformative potential of scientific computing in cross-disciplinary collaborations [1].

However, there are a number of issues with the accessibility and usability of these state-of-the-art computing systems. Creating intuitive user interfaces has often been neglected as the scientific computing community thrives at finding innovative solutions to complex technical problems. This hinders the democratisation of scientific computing by erecting obstacles in the way of researchers from diverse areas using computational tools to their full potential.

The motivation behind this project is to address the critical gap in the development of unified user interfaces for cross-disciplinary scientific computing applications like polyFEM. By creating a cross-platform unified user interface that combines the best of both developer friendly integration and user accessibility, this project aims to democratize scientific computing. The central question this endeavor seeks to answer is whether it is possible to

design and implement a user interface that is versatile enough to accommodate the diverse needs of scientific computing while being straightforward for developers to integrate into their existing codebase and intuitive for users from entirely separate fields.

1.2 Contribution and Significance of the Study

Given the challenges mentioned above, we at the University of Victoria have proposed and in making a cross-platform user interface and predetermined toolset for graphical visualizations and interactions. Making an easy-to-understand, versatile interface that can consolidate significant designs in scientific computing is the aim of this project. Bridging the gap between researchers from different fields and complex computational technologies is a problem that this directly addresses. Wide-ranging effects could result from enabling scientists to use sophisticated computing power without having to be computer science specialists, which would speed up research and foster interdisciplinary cooperation.

In addition, we have added an inventive approach to managing local file access, cross-platform functionality, and binary task execution via a REST server. The visualization of 3D data is a crucial aspect of scientific computing, and it is made feasible by utilizing the Javascript library for graphical interactions and visualizations.

Perhaps most importantly, our ambition to design a universal toolset and user interface specification protocol for scientific computing is groundbreaking. By creating a unified user interface that is versatile, accessible, and customizable, In the project we can foster cross-disciplinary collaborations, and set a new standard for user interface design in the scientific computing domain.

1.3 Organization of the Report

The rest of the project report is organized as follows:

- Chapter Two offers a literature review and an overview of existing methods and approaches related to 3D Visualization, geometry modeling, and libraries for feature extraction.

- Chapter Three introduces the terminology, definitions, and technical background required for understanding the proposed user interface and its implementation.

- Chapter Four details the user interface design, components, and system architecture that supports to develop a platform for Cross-Disciplinary Scientific Computing Applications.

- Chapter Five discusses the use case which examines how a user interacts with the PolyFEM UI by using a configuration file named cube.json in the developed user interface.

- Chapter Six presents the testing, validation, optimization, and analysis of the UI's performance and the results obtained from different JSON Files.

- Chapter Seven summarizes the contributions and findings of the study and its limitations and outlines potential future work and research directions in the field of Scientific Computing.

Chapter Two

Background Work

2.1 Literature Review

2.1.1 Geometric Modeling

Geometry modeling is extremely important, in the creation of user interfaces for computing applications that span multiple disciplines. When it comes to visualizing data geometric representations are often utilized. Techniques like 3D rendering, surface modeling, and volume rendering are vital, for representing data from a range of domains[2]. Tasks like mesh generation, surface reconstruction, and transformation are all part of geometry modeling. Completing these tasks is essential to producing educational visualizations. Scientific computing has incorporated geometry modeling in a number of research projects. As an illustration, consider the development of aspect-focused visualization tools for applications in biology and material science [3].

2.1.2 3D Visualisation and Interaction

In scientific computing, managing complicated datasets like multidimensional data, models, and simulations is a typical task. Researchers can now more easily represent and understand such data thanks to the powerful tools that 3D visualization provides. Meaningful 3D data

representations have been made possible by the widespread use of techniques like surface visualization, volume rendering, and 3D scatter plots [4]. A key component of developing a cohesive user interface for applications involving scientific computing across disciplines is three-dimensional(3D) visualization. It enables researchers to engage in intuitive and enlightening interactions with complex data from multiple scientific domains. Researchers can learn new things and make discoveries that might be difficult to find with conventional two-dimensional interfaces when given access to a three-dimensional view.

Similar to the above, a key component of developing a cohesive user interface that supports applications in scientific computing across disciplines is effective interaction design. Working with data, running simulations, and assessing results are all significantly impacted by how users interact with the interface. It also improves data exploration's tactile quality and intuitiveness. This level of interaction breaks down barriers between domains by involving users with different scientific backgrounds in the analysis process [5].

Fostering cross-disciplinary collaboration in the field of scientific computing is greatly impacted by the incorporation of 3D visualization and interaction into unified user interfaces. These features can be utilized by researchers across various disciplines to efficiently convey intricate discoveries. In medical research, for example, 3D visualization makes complex anatomical structures or disease models easier to view and collaborate on during diagnosis and treatment planning [6]. Also, interactive components empower researchers to investigate informational data sets together, modify simulations continuously, and direct joint analyses, separating barriers between scientific disciplines. Such cooperative interfaces contribute to the collaboration of information, cross-fertilization of thoughts, and the development of innovative solutions for complex scientific difficulties.

2.1.3 Mesh Generation

Mesh generation is a vital component of computational science and designing that makes complex geometries discretizable for exact simulations and analysis [7]. Also, complex geometries like unpredictable shapes and multi-material interfaces ought to be dealt by mesh generation algorithms since they are most generic in various scientific fields. Accurate outcomes also depend on the capacity to produce excellent, unstructured meshes that respect boundary conditions and change on the fly during simulations.

2.1.4 Scientific Simulations

Scientific simulations unified user interface development has a number of obstacles. An interface that is flexible enough to meet the needs and preferences of a broad spectrum of users including researchers from a variety of scientific fields are necessary [8]. In interface design, striking the ideal balance between usability and versatility is a difficult task. To tailor simulations to their particular domains, researchers need to have the freedom to do so while maintaining an interface that is understandable to users of different computational skill levels.

To solve these problems, techniques like integration standards, visualization techniques, and modular design are applied. Researchers can select and configure individual simulation components to personalize the interface through a modular approach. The interface is more user-friendly thanks to visualization tools and interactive dashboards that make it easier to interpret the results of simulations.

2.2 Related Tools and Libraries

2.2.1 Mesh Processing Tools

1. MeshLab: A variety of tools are available for processing and editing three-dimensional meshes in MeshLab, an open-source mesh processing program. Scientific visualization, computational geometry, and computer graphics are among the domains where it is frequently employed. Strong mesh manipulation capabilities, which are necessary for many scientific simulations and visualizations, can be made available to researchers by integrating MeshLab into the unified user interface [9].

2. Gmsh: Gmsh is an open-source finite element mesh generator and post-processor. The advanced meshing capabilities it provides for complex geometries are used extensively in engineering and scientific simulations. Integration of Gmsh can empower researchers to generate high-quality meshes for their simulations within the unified interface, streamlining the computational workflow [10].

2.2.2 Visualisation Toolkit (VTK)

It is a flexible library for image processing, visualization, and 3D computer graphics. Scientific computing applications can benefit from its ability to handle and visualize both structured and unstructured meshes through the provision of tools. Researchers can improve information analysis and perception by utilizing VTK to make intuitive 3D representations of their computational information inside the unified user interface [11].

2.2.3 ParaView

ParaView is an open-source data analysis and visualization application built on top of VTK. Large datasets are its area of expertise, and it supports a wide range of data formats that are frequently used in scientific computing. Cross-disciplinary collaboration is facilitated by the

integration of ParaView, which allows researchers to carry out sophisticated data analysis and visualization tasks directly within the unified user interface [12].

2.2.4 Challenges and Opportunities in Geometry Processing

Even though the field of geometry has a lot of untapped potential and unmet research needs, this review of the literature provides a solid foundation for the project.

Several data types, such as mesh data, experimental results, and numerical simulations, are frequently integrated in scientific computing. It is challenging to design an interface that keeps usability while integrating and visualizing these data sources in a seamless manner. It might take training for researchers who aren't familiar with certain scientific computing tools to use the unified interface efficiently. It is essential to create intuitive features and offer instructional materials in order to overcome this obstacle.

On the other hand, a modular user interface lets users personalize their experience by choosing and allocating components according to their requirements. Researchers can find it more straightforward to comprehend and check complex information visualization when they utilize refined visualization methods like interactive plots, 3D rendering, and information-driven visualizations.

2.2.5 Project Overview and Connection to Background Work

Aiming to meet the expanding relevance of scientific computing across research disciplines, the present project is focused on creating a uniform and cross-platform user interface for general-purpose scientific computing. The harmony between user accessibility and developer integration is absent from current systems. The goal of this research is to ascertain whether it is possible to create a user interface that satisfies the stringent standards of scientific computing while yet being adaptable and intuitive, easy for developers to maintain, and used by a variety of users and professions.

The underlying work for the project highlights the significance of mesh creation, scientific simulations, 3D visualization and interaction, and geometry modeling within the framework of scientific computing. It emphasizes how important it is to have user-friendly interfaces that can manage complicated geometry, multidimensional data, and simulations while being accessible to academics from other fields. Through the simplification of data exploration and analysis, the integration of 3D visualization and interactivity into unified user interfaces promotes cooperation across disciplines. The project's objectives of developing a flexible and user-friendly interface for scientific computing are further supported by the use of visualization techniques, modular design, and integration of programs such as MeshLab, Gmsh, VTK, and ParaView. This background research gives the project's developers and implementers a crucial understanding of the needs and difficulties it faces.

Chapter Three

Preliminaries

The following sections include background information and important terminology that help to further comprehend the project. The dictionary of important terminology and their definitions used in this report is listed in Section 3.1. The technical background, including libraries and computational ideas pertinent to this project, is covered in Section 3.2.

3.1 Definitions and Terminology

- Geometry [13]: The spatial representation of a topic item or framework is referred to as geometry with regard to computational simulations. For extra computational assessments, such as meshing and simulations, to be done, it offers the underlying structure by indicating the shape, size, and limits of the thing.

- Finite Element Method (FEM) [14]: FEM is a numerical method for estimating solutions to differential equation boundary value problems. It tracks the arrangement by dismantling a more complex framework into more modest, simpler to-gather parts (finite elements) and assembling them all. In engineering and physics, FEM is often used to solve structural, thermal, and fluid dynamics questions.

- Surface Selection [15]: The demonstration of choosing explicit surfaces or segments of geometry for the use of determined models or characteristics is referred to as surface

selection. In computational simulations, this lets users to define boundary constraints, loads, or other unique qualities to various parts of the model.

- Solver [16]: In computer simulations, a solver is an algorithmic instrument that works out the answer for a bunch of equations. Different solutions are used depending on the kind of issue (linear, nonlinear, static, or dynamic). In order to compute results, solvers use the given data regarding geometry, boundary conditions, and material properties.

- Simulation [17]: Simulation implies impersonating a certifiable interaction or framework over time within a computational environment. It contains utilizing numerical models and algorithms to imagine how a UI will be acknowledged in specific situations.

- Computational Libraries [18]: Computational libraries are collections of different functions and works that are utilized in software applications to perform specific tasks, particularly those types that are connected with math, science, and Engineering design. Because these libraries are designed with accuracy and speed in mind, developers can rely on them when creating complex applications.

- Spec: Specs are detailed descriptions of the objectives and requirements of a software component or piece of software; in software development, the term "spec" usually refers to specifications. As a sort of diagram or guide for designers, particulars guarantee that the application meets the expected requirements of usefulness and execution.

- SpecEngine: An engine or system intended to read and enforce standards (specs) is referred to in this project as SpecEngine. Such an engine would ensure that the program or any of its parts meet the specified requirements.

- Configuration/JSON file [19]: Software program setup data is stored in a file called a configuration file. JSON (JavaScript Object Notation) format is most often used for it. JSON is a small, easily interpreted, and readable data format for transfers. Users can modify software behavior with these files without having to change the code directly.

- UfileSystem [20]: A "file system" in software parlance is the process and structure that an operating system employs to manage files on storage devices; the term "UfileSystem" is not standard. A specific or user-defined file management system in our project referred as "UfileSystem".

- Services [21]: An independent functional unit that is accessible to other components is referred to as a service in software architecture. There is adaptability and versatility in framework design when services are approximately coupled, and that implies they cooperate without serious areas of strength. Microservices architectures or service-oriented architectures can include them.

- Service Engine [22]: A runtime environment that operates and oversees services in a service-oriented architecture is called a service engine. In addition to handling message processing and the lifecycle of service, it also includes more features like event management, security, and other things.

- Visualization: In order to make complicated data, numerical findings, or simulations accessible and interpretable for people, a procedure known as "visualization" must be carried out. In order to support scientific information analysis, comprehension, and communication, graphical representations of data and outcomes are created.

- Mesh [23]: In computational simulations, a mesh is a discrete representation of a geometry, composed of elements (like cubes, triangles, cylinders, or tetrahedra). The mesh is utilized to inexact the arrangement space, considering the arrangement of incomplete differential conditions over the geometry.

- 2D-3D Views [24]: Two and three-dimensional representations of data or models are referred to as 2D and 3D perspectives in computer graphics and visualization. While 2D perspectives show data on a level plane, 3D perspectives offer depth, taking into

consideration a more vivid and point-by-point visualization of complicated geometries or datasets.

3.2 Technical Background

Software engineering, computational science, and user experience design methods must be combined in an interdisciplinary manner to create a Unified User Interface (UI) for Cross-Disciplinary Scientific Computing Applications. The details needed to comprehend this project better, including the technical aspects like Javascript libraries, Computational Geometries, and Python libraries [25] like meshio and libigl.

3.2.1 Python Libraries

1. Meshio [26]: For several mesh formats used in computer simulations, this Python library provides I/O methods. In this project, by facilitating the reading and writing of different mesh formats, it enables seamless integration and data transfer between different simulation tools and visualization platforms.

2. libigl [27]: It is a library dedicated to geometry processing. It provides techniques and algorithms for meshing, smoothing, and optimization. IGL can be the groundwork of geometry-related operations in UI, supporting many scientific fields that need complex geometric computations.

3.2.2 Javascript Libraries

1. React Typescript [28]: React TypeScript is a development framework and programming language combination used to build versatile and cross-platform user interfaces. It benefits from the rich ecosystem of libraries and tools available in JavaScript and TypeScript, facilitating rapid development. Its flexibility enables the design of dynamic and

customizable user interfaces, making it suitable for accommodating the diverse needs of scientific computing projects, from 3D visualizations to handling various computing options.

2. threejs: It is a cross-browser JavaScript library/API used to create and display animated 3D computer graphics in web browsers [29]. Improved user experience and deeper insight into the results of the calculations are the two main purposes for it. Improving user engagement makes 3D geometry, data, and simulations easier to visualize for scientific purposes [30].

3. Prism.js [31]: It is a lightweight, extensible syntax highlighter. For a cross-disciplinary UI, it can enhance code readability and presentation, especially when users need to input scripts, equations, or view code snippets related to their simulations or computational tasks.

3.2.3 Computational Geometry

computational geometry, with its rich collection of algorithms and principles, forms the backbone of many functionalities required in a unified user interface for cross-disciplinary scientific computing applications.

1. Fundamental Representations [32]: The fundamental tools and algorithms for representing geometrical structures and shapes are provided by computational geometry. These representations are crucial for manipulating and visualizing geometric data when designing a user interface for scientific computing applications.

2. Mesh Generation and Refinement: Computational geometry plays a major role in mesh generation, which is a fundamental aspect of FEM simulations. Computational geometry provides the means and techniques to enable unified user interfaces (UIs) with effective mesh generation, visualization, and refinement capabilities. [33].

Chapter Four

User Interface Design, Architecture, and Components

4.1 Introduction

This chapter aims to provide a comprehensive overview of the Design, Features, components, and Architecture used to develop Unified User Interface for Cross- Disciplinary Scientific Computing Applications. This chapter will focus on the problem statement, design, architecture, and proposed solution, while the previous chapter provides the necessary background knowledge. This chapter provides the detailed implementation of the general-purpose software for computing applications. Section 4.2 covers User Interface Design & features, Section 4.3 explains the process and architecture of the system including code snippets and pseudocode, and Section 4.4 covers a detailed explanation of all UI components.

4.2 User Interface Design

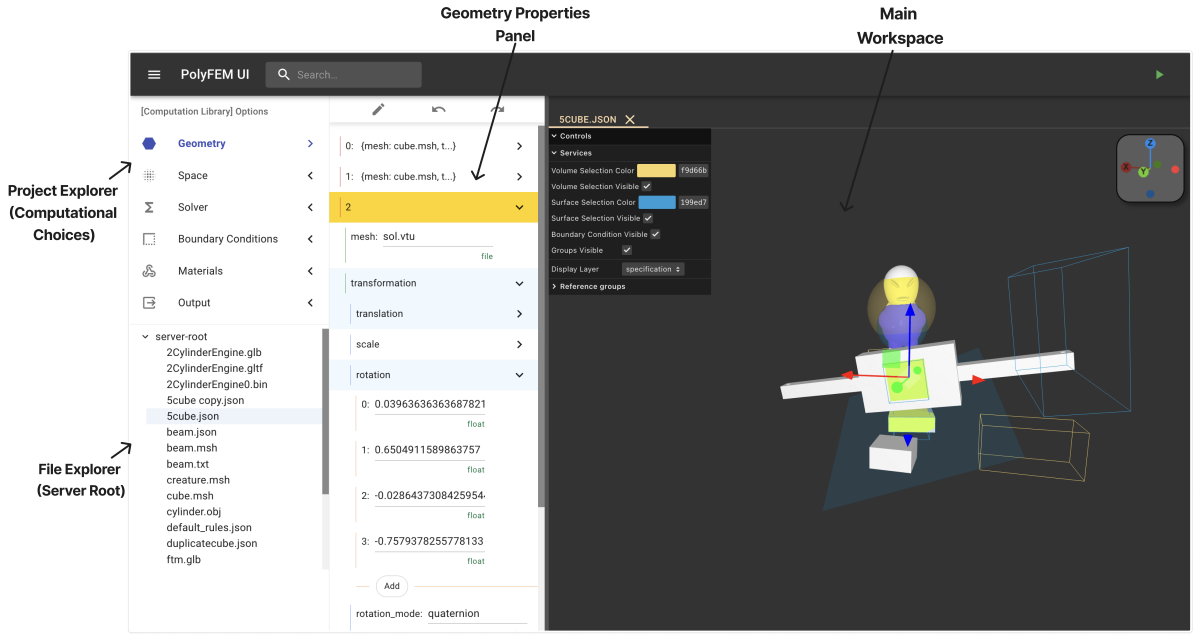

Figure 4.1 User Interface

Various Specs and Various procedures give a thorough technique for mathematically settling and testing actual problems. A very much organized User Interface(UI) for Basic Specs fills in as an extension between complex computational calculations and the user's design goals. By arranging functionalities sensibly and focusing on easy-to-use configuration, such UIs enable users to tackle the maximum capacity of Limited Component Strategies. Engineers, researchers, and students can collaborate with these intense computational apparatuses through the comparing UI. Users can investigate and utilize geometry computation abilities easily on account of a powerful and clear User Interface Design. As per fig. 4.1, user interface layout and Design includes different sections like Main workspace, geometry Properties panel, Project Explorer, and File Explorer.

1. Main Workspace (Fig. 4.2)

   - Graphics or Visualization Canvas:
     - Collaborative visualization of models and results.
     - Covering most of the central part of the user interface, this area provides an interactive view of the geometry and simulation results.
     - Users can rotate, pan, zoom in-out, examine closely, focus on specific areas, and interact with the model by seeing deformations, stress distributions, and other examinations in addition to the geometry.
     - Color-coded visualization based on result values.

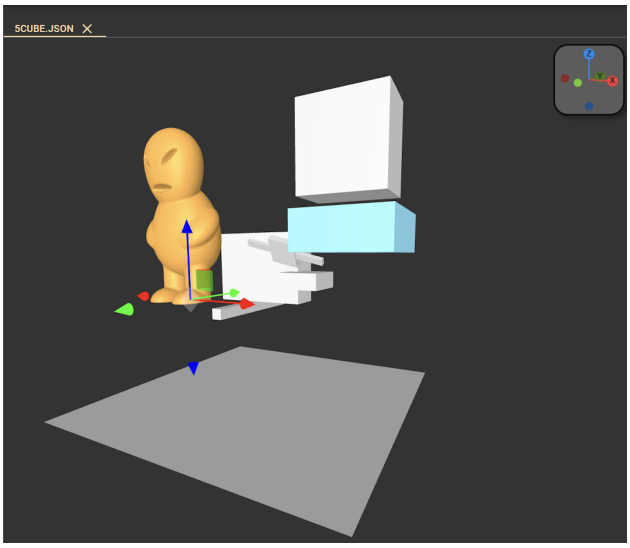

Figure 4.2 Visualization/Graphics Canvas

   - Grid Overlay and Axes:
     - Assist users in understanding the orientation and scale of the model.
     - Different Arrows to move any sides in straight Directions.
     - In Fig. 4.2, arrows representing directions, often used for displacements or fluid flows.

– An optional reference grid, which helps in orienting the model and understanding its scale.

- Services: Gives a few settings or determinations connected with the 3D model or calculation, references to volume and surface choice, varieties, and understandable choices. There are likewise settings connected with the showcase of limit conditions, groupings, and layer specifications. For example in Fig. 4.3. different services with color pickers are displayed.

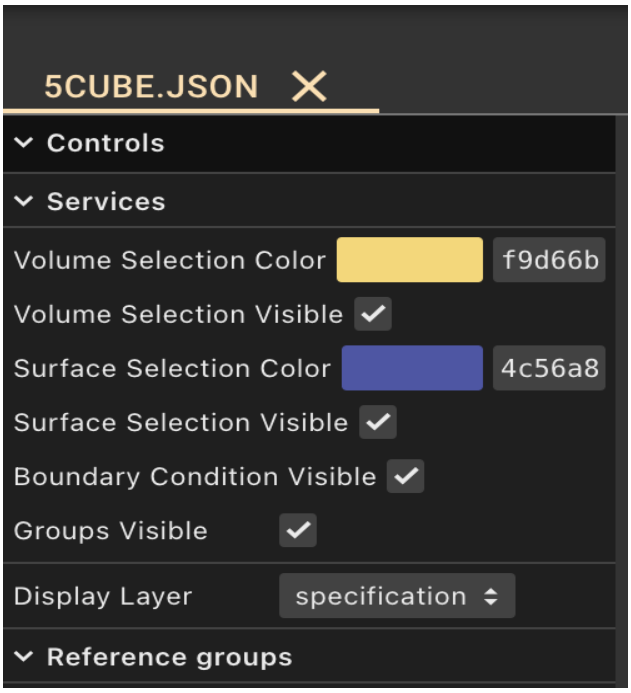

Figure 4.3 Services

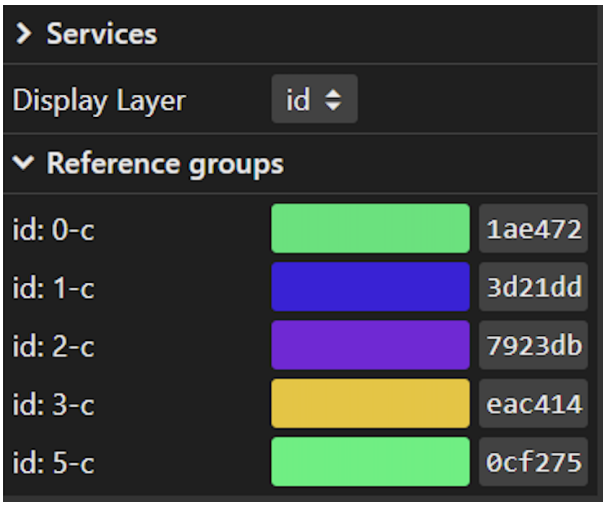

Figure 4.4 Reference Groups

- Reference Groups: This segment, as its name implies, recognizes the various groups or layers that make up a model or reproduction. Each group has an exceptional number and a value that accompanies it. They are representations for different areas of the 3D model, boundary conditions, or more isolated parts of the simulations. Users may effectively make modification, alter, or reference to these groupings on account of labels and values. For example in Fig. 4.4. groups are defined by different IDs.

2. Left Sidebar:

   - Project Explorer:
     - In Fig. 4.1, left side is having a panel containing different pre-characterized computational numerical choices to change the representation.
     - Hierarchical tree view of design components including different geometries, boundary conditions, mesh properties, and result sets.
     - When you right-click an item, you can access actions relevant to that item, such as editing or deleting.
     - Users can easily select or customize any calculation option to use in the rendering.
     - Options to initiate mesh generation, refine, or check mesh quality.
   - Server-Root File Explorer:
     - In Fig. 4.1, bottom left side is having multiple different types of files which is a Overview of Materials or JSON files with their properties, such as different shapes, creatures, cubes, cylinders, etc.
     - Users can characterize and save their material by determining properties like volume, transformation scale, thickness, and more.
     - Immediately look for materials by their names, sets, or properties.

3. Properties or Details Panel:

   - It acts as a contextual panel, powerfully changing its items in light of the client's ongoing choice or action. It provides detailed settings, properties, and tools for fine-tuned control over the simulation. Fig. 4.5 and Fig. 4.6 displayed the example detailed modification panel for visualization.

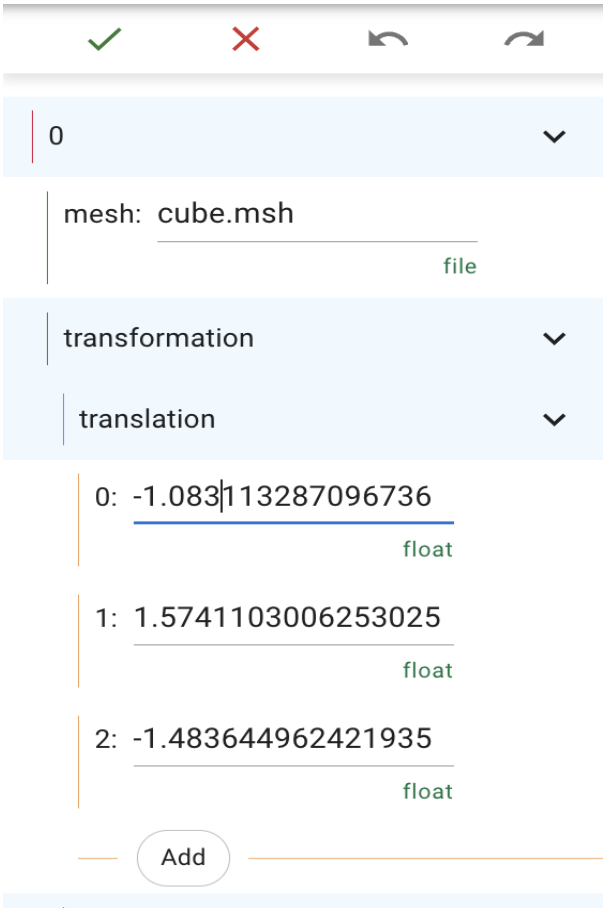

Figure 4.5 Transformation Parameters Modification

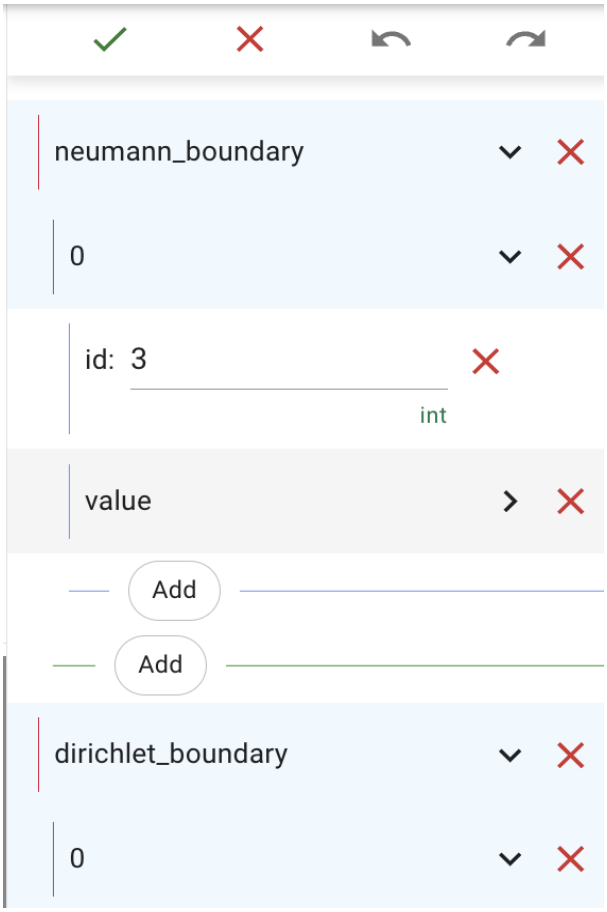

Figure 4.6 Boundary Conditions Modifications

- Dynamic Content Display:
  - Depending on the selected item in the Project Explorer or visualization window, this panel displays relevant properties.
  - For instance, if a user selects a boundary condition, they might see its type, magnitude, and direction here.
- Editable Fields:
  - Users can directly modify multiple values from this panel.
  - users can add new properties with add button functionality.
  - users can easily undo or redo the last modification made to visualization.

  - users can remove or confirm the property before applying it to visualization.
  - Input types can vary from text boxes, dropdown lists, and more.
- Sections:
  - For items with numerous properties, the panel has collapsible sections to organize information.
  - For example, a boundary condition might have sections for adding multiple geometric properties, loading conditions, and advanced settings.
- Validation and Feedback:
  - The panel often provides immediate feedback on entered values.
  - Invalid entries might be highlighted, and tooltips can offer guidance on acceptable value ranges or formats.
- Contextual Help and Descriptions:
  - Next to each property, there might be icons or links that provide additional information or definitions.
  - This aids in understanding, especially for novice users or for complex properties.

4.3 System Architecture

As shown in Fig. 4.7, The presented architecture provides a system that seamlessly integrates different parts to create dependable and effective visualization. The Arrows displayed in architecture are the flow of the system and how it works. The administration, visualization, and processing of geometric and simulation data are all handled holistically.

The Service Engine, at the heart of this system, is in charge of everything and serves as the main interface between user input and computational operations. Users can sculpt and polish their models using the Geometry Controller before sending them via a precise Query

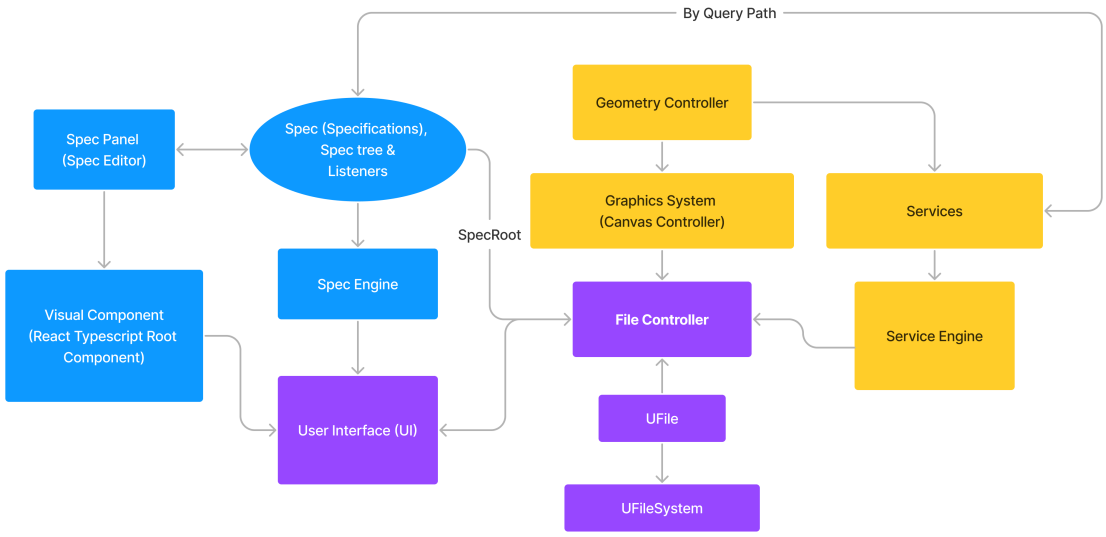

Figure 4.7 System Architecture: React Typescript in Blue, General User interface system in purple, and Graphics & Services in Yellow

Path to make sure every geometrical detail is recorded and handled. Users can set specific simulation conditions and parameters with the help of the Spec Panels and Spec Engine, which guarantees that every study is properly matched to the needs.

In addition, the system makes use of contemporary web technologies like as React Typescript, guaranteeing a dynamic and responsive user experience that uses Listeners to actively communicate with the back-end. Geometries, outcomes, and project files are all systematically arranged and easily retrieved thanks to the File Controllers and UFile System, which also protect user data integrity and accessibility. Every element in this ecosystem from the Spec Root to the different services works together harmoniously as users traverse it, guaranteeing that the intricate mathematical foundations of FEM are converted into concrete, useful insights.

4.3.1 Components of Architecture and Their Interconnections

1. File System: It is the bedrock of data management in the architecture. It ensures organized storage, retrieval, and manipulation of all General-purpose geometry simulation results. Manages file operations like saving, loading, importing, or exporting. Store the outcomes of simulations, including stress, strain, displacement values, etc. It is getting used to Save user-defined settings or simulation parameters. Interacts with the UFileSystem to access the underlying file storage.

    - Server Obstruction: The complexity of direct file operations is included in the layer known as server obstruction. It gives support, guaranteeing secure and optimized data handling. It additionally works with productive ordering, search, and recovery functionalities, guaranteeing quick admittance to needed files.
    - UFile: Manages individual file operations such as saving, reading, writing, updating, or deleting. It ensures that data integrity is maintained, especially during concurrent accesses.
    - UFileSystem: Governs the overarching organization and hierarchy of stored data. It might also maintain metadata about files, like creation dates, last accessed dates, file sizes, etc.

    The following is an example of using the UFile Class implementation to access a file: The below snippet is for files having vtu format.

```
class UFile{
    url: string;
    name: string;
    isDir: boolean = false;
    children: UFile[] = [];
    extension: string;
    constructor(url: string, name: string, isDir: boolean){
```

```
8           this.url = url;
9           this.extension = this.url.split('.').pop();
10          this.name = name;
11          this.isDir = isDir;
12      }
13      /* Creates a URL from which contents of this file can be accessed
            through, deals with file type conversions here depending on
           the geometric file format */
14      accessURL(){
15          switch(this.extension){
16              case 'glb':
17              case 'gltf':
18              case 'obj':
19                  return 'http://localhost:8081/queryFile/?address=./'+
                        this.url;
20              case 'msh':
21              case 'vtu':
22                  let nameComponents = this.name.split('.');
23                  nameComponents.pop();
24                  nameComponents.push('obj');
25                  let name = encodeURIComponent(nameComponents.join('.'
                        ));
26                  return `http://localhost:8081/mesh-convert/${
                        encodeURIComponent(this.url)}/${name}`;
27          }
28      }
29  }
```

Listing 4.1 Accesing a new file

2. Geometry Controller: This is likely the interface through which users interact with geometric entities in the system.

- Functionality: To define, modify, or import/export geometries.
- Interaction: Communicates directly with the Service Engine to request geometric operations. Receives data from File Controllers when importing geometries. It Updates the UI through React Typescript components when changes are made.

3. Visual Component: The root or main component of the React-based UI. All visual components and UI elements stem from this root. It represents the main UI layer, built with React and Typescript, that interacts with the user and displays data. It also communicates with the system mainly through a request path, which can be an API or method call path to request and receive data.

4. Spec: The "Spec" system, sometimes shortened to specification, is a crucial component of this software since it establishes the parameters, characteristics, and circumstances in which the simulations are conducted.

    - The specification is needed to manage interactions, state transitions, and event trees.
    - The Spec system allows users to create or alter any kind of property required for simulations, including material properties, load conditions, boundary conditions, and more.
    - These characteristics may apply to all geometric entities in the model or just some of them.
        - SpecNode: Individual elements or nodes inside the specification hierarchy are represented by SpecNodes.
            * The many qualities, conditions, or parameters that are necessary for the geometry computation process can be found in each SpecNode.
            * For instance, density, Young's modulus, and Poisson's ratio are examples of properties of a SpecNode, which represents a material.

- Listeners:
  - Listeners continuously scan the system for particular events or triggers. User interactions, system events, or data changes could all be examples of this.
  - The Spec system is strongly connected to listeners. Listeners notice when specifications are changed, added, or removed.
  - They make sure that the system's other parts are updated accordingly. For instance, a listener might cause a visualization update to reflect a change in a material property in the Spec.

5. Spec Root: This presents a centralized location or module where all specifications (specs) for a simulation are stored and managed.

   - Functionality: Act as the primary source of truth for all parameters, conditions, and settings of a user interface simulation.
   - Interaction:
     - Spec Panels likely feed data into the Spec Root as users define or modify simulation parameters.
     - Services and the Service Engine frequently reference the Spec Root to access the current specifications for ongoing tasks.
     - The Spec Engine manages the contents of the Spec Root, validating and organizing data.

6. Spec Editor: It is a portion of the user interface where particular geometry simulation parameters and specifications can be entered, shown, and changed. By capturing the information required to run a simulation, they serve as a link between the user's intent and the underlying system.

- It is possible for users to create completely new specifications or parameters that define entities, descriptions, and other metadata in addition to values.
- For use in the primary project workflow, the Spec Editor can feed predefined or custom specifications into the Spec Root.
- The panel can carry out constant validation as users enter values, ensuring that the information is placed inside allowed ranges or formats.

7. Spec Engine: This automation powerhouse ensures uniformity and streamlines monotonous processes.

   - manages the particulars of different materials, circumstances, or simulations.
   - creates populated trees from the JSON specification processing and runs queries.
   - This part handles queries and specs validation in addition to managing the Spec Tree.
   - It interacts with the Spec (tree) to fetch or update specifications and might use queries to search or filter specifications.
   - Receives updates from Spec Panels when users define or modify parameters.

   This is how to implement a specification from the location specified in the query.

   - Parameters:
     - query: a query matchable to a pointer, for example: /geometry/object1 -> /geometry/*
     - parent: An instance of the Spec class, which likely represents the parent spec of the one being queried.
     - include: it selects which subSpecs to include from the tree
     - typeOverride: A number that allows the user to override which instance of the matched raw specification is to be utilized.

- Functionality:
  - The method splits the query string to derive individual keys.
  - It searches for these keys in a structure called specTree.
  - If a match is found, a new spec is constructed and populated with properties from the matched raw specification.
  - The type of the spec determines if it's a leaf node.
  - Default values are assigned, and the spec is marked if optional.
  - Child specifications are recursively added based on the include and required parameters.

The following is the code for Building a Spec from the given query It Returns a Spec object which is constructed and populated based on the given query and other parameters:

```
1    query(query: string, parent: Spec, include:string[]=[],
        typeOverride:number=0): Spec{
2        let keys = query.split('/');
3        let loc = this.specTree.query(keys);
4        if(loc==undefined)//Terminate if invalid query
5            return undefined;
6        let raw = loc.rawSpec[typeOverride];
7        let spec = new Spec(keys.pop(), parent);
8        spec.query = query;
9        spec.pointer = raw.pointer;
10       spec.type = raw.type;
11       spec.typename = raw.type_name;
12       spec.typeIndex = typeOverride;
13       spec.doc = raw.doc;
14       spec.isLeaf = ['object', 'list'].indexOf(raw.type)<0;
15       spec.value = raw.default;
16       spec.optional = raw.optional;
```

```
        for(let included of include){
            spec.pushChild(this.query(`${query}/${included}`, spec));
        }
        if(raw.required){//If required is not undefined
            for(let required of raw.required){
                spec.pushChild(this.query(`${query}/${required}`,
                    spec));
            }
        }
        return spec;
```

8. Services

   - Functionality:
     - Each service might handle a specific task like mesh generation, material property assignment, boundary condition setup, or solver execution.
     - Services verify input data for accuracy and consistency prior to job execution.
   - Interactions:
     - Services receive tasks from the Service Engine, execute them, and return results, errors, or status updates.
     - Some services might require detailed specifications or parameters, which they fetch from the Spec Engine.
     - Services involved in data I/O would interact with File Controllers to read or write data.

9. Service Engine:

   - Functionality:
     - As a master organizer, the Service Engine distributes various services according to client preferences or system needs.

  - It coordinates and forwards requests to the relevant services via a variety of UI elements, including Listeners, Spec Panels, and Geometry Controller.

- Interactions:
  - The Service Engine sends tasks to specific services, waits for their completion, and then either sends the results to other components or triggers other services based on the outcomes.
  - With UI Components, it processes requests and provides feedback or updates based on the operations performed by the services.
  - With Spec Engine it fetches detailed specifications or parameters required to execute certain tasks.

10. Service Engine and Services - Combined Workflow:

- Initiation: A user action (like requesting a mesh operation) triggers the Service Engine through a UI component.
- Task Delegation: The Service Engine delegates this task to the appropriate service, say the Mesh Generation Service.
- Specification Fetch: If detailed parameters are needed, the Mesh Generation Service might request these from the Spec Engine.
- Execution: The service performs the task, potentially interacting with other system components like File Controllers.
- Completion: Once done, the service sends the results back to the Service Engine, which then updates the UI or proceeds to the next step in the workflow.

Essentially, the Service Engine and services are "brain" and "organs" of this user interface, which guarantee the correct, efficient, and seamless execution of

tasks linked to simulations. Scalability, maintainability, and future extensions and upgrades are made easier by the modularity of this design.

11. Graphics System: The architecture's visualization center, the graphics system, renders simulation results, mesh structures, and geometric data. It is intimately integrated with other components, especially the Visual Component and Service Engine, to provide dynamic visuals that are changed in real time based on user inputs and simulation results.

In order to guarantee the correctness, effectiveness, and utility of the simulations, a general-purpose specifications program's system architecture consists of a complex network of interconnected components. The paths especially the Query Path help to regulate data flow by ensuring that each component gets access to the data it needs at the right time. Since the Spec Root centralizes and regulates all important simulation parameters, the results are assured to be dependable and reproducible.

4.4 User Interface Components

An extensible framework has to be developed in order to give a unified user interface for Cross-Disciplinary Scientific Computing Applications, which may be used for everything from structural analysis to material dynamics fluids. Such an interface must enable a process that is easy to use and understand for both inexperienced and seasoned users. Understanding the computational potential and applying best practices in user interface and user experience design are equally essential for this. Research publications on this subject include a wide range of subjects, from tool integration to usability. Given below is a list of UI Components organized by functionality that can be included:

1. Geometry Module: The geometry module is an essential part of scientific computation applications like FEM. since it lets the user adjust individual pixels in a picture and

determine the image plane. The parts and purposes of the geometry module are listed below:

- 3D Viewer:
  - To view, rotate, and zoom in/out on the geometry and mesh.
  - Information can be immediately added to, recorded on, or labeled in geometry. It is applicable to documentation.
  - Slice through the geometry to inspect internal details.
- File Import/Export:
  - To import mesh or geometry files or export output.
  - Importing many geometries at once allows you to study various patterns side by side, which is a handy function.
- Geometry Editor:
  - Rotate, scale, or manipulate geometric features with handles or input data.
  - Builds geometry based on parameters for easy user interface modification.
  - Recognise fillets, holes, and patterning and modify them accordingly.
  - Allow users to group numerous entities, for example, sides or edges to make it simpler to choose or apply conditions later.
  - Important factors like the size, values, or characteristics of a particular mesh may be saved and changed.

2. Mesh Generation Module: The mesh generation module is critically important for the user interface. In order to approximate the governing equations on the smaller, simpler domains known as elements, the computational domain (geometry) must be divided into them. This process is known as meshing.

   - Mesh Operations:

- Smoothing: Modifies the structure of the mesh in general by changing node places for better component appearance.
- Decimation: A valuable method to decrease too complex meshes, it lowers the amount of components.
- Subdivision: Used frequently in local refinement, it divides components to provide a finer mesh.

- Mesh Inspection and Repair:
  - Inspection Tools: Bring particular focus on areas that need development using quality indicators.
  - Automatic Repair: Attempts to detect and fix problems such as inverted components, remaining spaces, and overlaps.
  - Manual Repair: Allows users to manually re-mesh certain areas, alter nodes, and adjust element edges.

3. Material Properties: Material properties have a large influence on determining how an object or system responds to an external geometry input. It is important to precisely address these elements to get reliable simulation results. Consequently, when handling materials, a simple user interface is crucial.

   - Material Library: Assigning known materials to UI components is easy and error-free for users using an organized material library. The components are defined below:
     - Listing or Dropdown: To Display and select available materials. It includes materials like Laplacian, Stokes, SaintVenant, Polymers, etc.
     - Material Information View/Update Panel: The user can update the values of different parameters like density, Viscosity, etc.

- Assigning IDs: Every material may have a unique identification number issued to it. An internal reference to this identifier is generated by the application when an object or model is a component of the user's application of a material. The process of standardization guarantees that other software modules, services, or even external tools verify the material order.

4. Boundary Conditions: A fundamental part of the geometry computations are the boundary conditions, which distinguish the result's stability and accuracy. They control the model's way of behaving at its boundaries and the way that the framework cooperates with its current environment. We should look all the more carefully at the different boundary conditions that are available in the finite element methods for our user interface.

   - Dirichlet Boundary Condition:
     - It is also known as a value-prescribed or essential boundary condition.
     - Specifies the values that the solution must take at the boundary of the domain.
     - Often used to determine displacements in structural mechanics problems or to determine temperature in heat transfer analyses.
   - Neumann(Normal) Boundary Condition:
     - It is also known as a flux-prescribed boundary condition.
     - provides the solution's boundary derivative, which frequently translates into a significant flow or motion over the boundary area.
     - In structural mechanics, this might be compared to defining forces or grips on the boundary.
   - Mixed (Robin) Boundary Conditions:
     - Dirichlet and Neumann boundary conditions together to create a mixed (Robin) boundary.

- the solution's value in addition to its derivative associated with the boundary.
- For Example, prescribing both a displacement and a force on a boundary.

It is necessary to comprehend and apply boundary conditions correctly for simulation results to appropriately represent the real-world scenario that is being modeled.

5. Solver Settings and Controls: For our user interface, it is a necessary element. In addition to having a major impact on computation time, accuracy, and outcome stability, it controls the problem's numerical solution. Components that can be included in the user interface, are described below in depth.

   - Solver Type Selection: Selection includes the dropdown of the different solvers.
     - Linear vs. Nonlinear
     - Direct vs. Iterative
     - Static vs. Dynamic
   - Solver Parameters: Numerical inputs in the formats of float or integer numbers.
     - Maximum Iterations: Before terminating, the solver must try this maximum integer value.
     - Tolerance: Indicates the standards for convergence. When this value is reached by the residual (error estimate), the simulation comes to an end.
   - Advanced Solver Settings: In the interface, it can be an expandable Panel or a separate dropdown.
     - Matrix Storage: Options like sparse vs. dense matrix storage or specific matrix formats.
     - Iteration Counter: For displaying the current iteration number.
     - Real-time Text Fields: A real-time input field showing current convergence values. Change in the 3D Model will change the values in text fields as well.

6. Results and Post-Processing: This part is when the user visualizes and analyses the simulation's results. An easy to comprehend, clear, and detailed summary of the outcomes should be provided through an attractive user interface (UI) for this phase of the process. The following are the User Interface elements that were created for this phase:

    - Visualization (Results) Viewer:
        - 3D Visualization Canvas: An interactive area containing result data displayed over the modeled geometry that allows users to examine, rotate, pan, and zoom in. A large central area that displays results, mesh, and geometry.
        - Animation Controls: play, pause, and rewind depending on the time or transient remarkable results.
        - Display Modes: Pressure views, vector plots, and contour plots are just a few of the visualization options to pick from.
    - Reference Groups and Color Mapping:
        - Color Bar: Shows the range of values in the outcomes collectively with the associated color.
        - Color Mapping: Users are allowed to select from a wide range of color schemes or variations based on the kind of data they are interacting with or their personal preferences.
        - Reference Groups: Use various reference groups to define your services. Different colors were assigned to each service's unique reference group.
    - Advanced Visualization Elements:
        - X-Y Axis Plots: Users can select the X and Y Scales in visualizations according to their personal preferences.
        - Snapshot Capture: Taking screenshots of live visualizations is an easy task

for the user.

- Markup Layers: To highlight or identify important regions, place drawings, symbols, or other markups over them.

7. Help and Documentation: Documentation is always required for any new user interface. Users often need guidance from scratch introduction to detailed troubleshooting of their visualizations.

   - Theory and Background:
     - Basic Documentation: Introduction and fundamental concepts about the different specs and How it is getting used with finite elements.
     - Live URLs: Links to the useful and necessary online platforms that can be used for the user's visualization and for the User to understand the User Interface. Links or embedded views of the library's documentation.
   - Quick Help and Tool-tips:
     - Context-Sensitive Help: When a user hovers over or clicks on a UI element, display a brief description or tip related to that element.
     - Shortcut keys: A quick overlay that shows keyboard shortcuts for common actions.
   - Downloadable Resources:
     - PDF/Word Manuals: Provide users with different Manuals to go through the different topics for finite element methods.
     - Templates: Offer some common simulation files or project files to help new users.

Chapter Five

Case Study: Interaction with PolyFEM

For dealing with complex actual issues in a scope of fields, including liquid elements and primary designing, the Finite Element Method (FEM) is a powerful instrument. The development, execution, and evaluation of simulations can be streamlined and made simpler even though the computing core of FEM solvers is sophisticated. The innovative FEM program PolyFEM blends flexibility and usability so that users of all skill levels may make use of it. PolyFEM is a simple C++ and Python finite element library [34].

This use study examines how a user interacts with the PolyFEM by using a configuration file named 5cube.json and our generic user interface. A multitude of parameters and settings that characterize the simulation's nature are contained in this configuration file. The file acts as a blueprint for the simulation, describing in detail each step of how the program should analyze and comprehend the given problem, including solver settings and geometry transformations.

With the help of this case study, we intend to shed light on the many stages of user interaction with the PolyFEM program by demonstrating how crucial the 5cube.json file is to the setup and functioning of the simulation. Whether you are an under-study who needs to become familiar with FEM or an accomplished specialist who needs to exploit PolyFEM's capacities, this application gives knowledge into the consistent joining of processing power and UI plan. Case Study has been divided into several parts:

1. Launching the User Interface: The user starts our generic user interface application, revealing the main UI with PolyFEM including sections like Geometry, Space, Solver, Boundary Conditions, Materials, and Output.

2. Geometry Selection:

   - From the file list, the user selects the 5cube.json file. The system loads multiple cube geometries with various transformations, such as translations, scales, and rotations.
   - Mesh Files: The simulation uses various mesh files, such as cube.msh, sol.vtu, and creature.msh.
   - Transformation: Each geometry has transformation parameters, including translation, scale, and rotation. Quaternion and Euler angles are used for rotations.
   - As per Example JSON 5.1, Fig. 5.1 shows the 3D View of cube with directions and transformation dimensions.

```
"geometry": [{
"mesh": "cube.msh",
"transformation": {
    "translation": [-0.22, 0.23, 5.00],
    "scale": [1.59, 0.41, 1.25],
    "rotation": [-1.68, 1.39, -2.97]
    },
}]
```

Listing 5.1 Transformation JSON

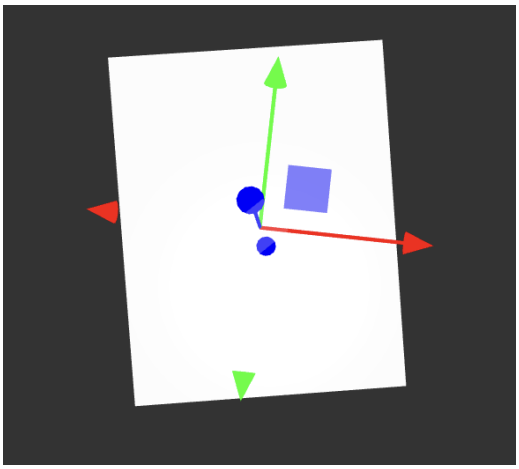

Figure 5.1 3D View of Geometry Transformation

3. Applying Volume/Surface Selection:

   - Specific volumes, surfaces, and points on the geometry can be selected for various purposes. This can be useful for applying boundary conditions or loads to specific regions.
   - PolyFEM identifies the volume selection linked to the cube.msh mesh within the JSON file.
   - The user sees options for volume and surface selection.
   - For Instance, According to 5.2 JSON Input, Fig. 5.2, shows the volume of the cube as yellow color and the surface as purple color.

```
"volume_selection": [
    {
        "id": 2,
        "box": [
            [-3,1,0.5],
            [2,1,1]
        ]
    }],
"surface_selection": [
    {
        "id": 2,
        "box": [
            [0,0,0],
            [1,1,1]
        ]
    }]
```

Listing 5.2 Volume/Surface Selection JSON

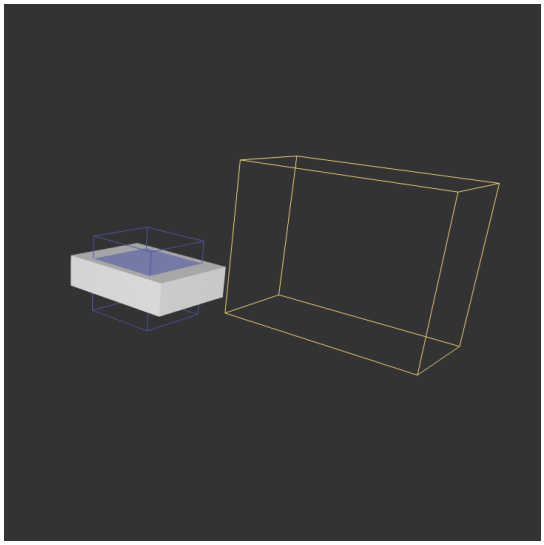

Figure 5.2 3D View of Volume/Surface selection

4. Configuring Boundary Conditions:

   - The user defines right-hand side (rhs) conditions with values [0, 9.81, 0, 21].
   - Pressure boundaries and Dirichlet boundaries are set with specific IDs and values/interpolations.
   - For this example, we want Neumann boundary condition of [100, 0] (a force of 100 in x) applied to the whole right side (pulling), so in the "neumann_boundary" array of "boundary_conditions", we add an entry with "id": 2 and "value": [100, 0].

```
"boundary_conditions": {
    "rhs": [0, 9.81, 0, 21],
    "neumann_boundary": [
    {
        "id": 1,
        "value": [ 100, 0]
    }]
},
```

Listing 5.3 Boundary Conditions JSON

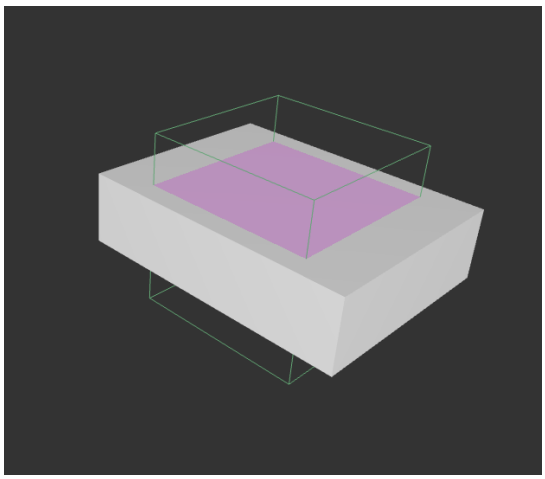

Figure 5.3 3D View of Boundary Conditions

5. Setting Material Properties:

   - The user assigns material properties like Young's Modulus (E), viscosity, and other material-specific parameters.
   - material properties can be possible to select from the dropdown or write manually in the input fields as well.
   - As per 5.4 JSON, materials can be added.

```
"materials": {
        "id": 1,
        "type": "LinearElasticity",
        "E": 210000,
        "nu": 0.3
},
```

Listing 5.4 Materials JSON

6. Output Configurations:

   - The user specifies the output format as 'paraview' and names the output file as 'cubes.pvd' (Exa.: 5.5).
   - They enable the options to output both velocity and acceleration results.

```
"output": {
    "paraview": {
    "file_name": "5-cubes.pvd",
    "options": {
        "velocity": true,
        "acceleration": true
        }
    }
},
```

Listing 5.5 Output Configuration JSON

7. Run Simulations and Visualize Results: Upon completion, the PolyFEM System displays results in the Visualisation Canvas/Viewer. User analyzes data plots and can export results if needed.

In Fig. 5.4, It showcases the PolyFEM user interface, highlighting various settings and parameters essential for finite element simulations. From the visual cues, users can define

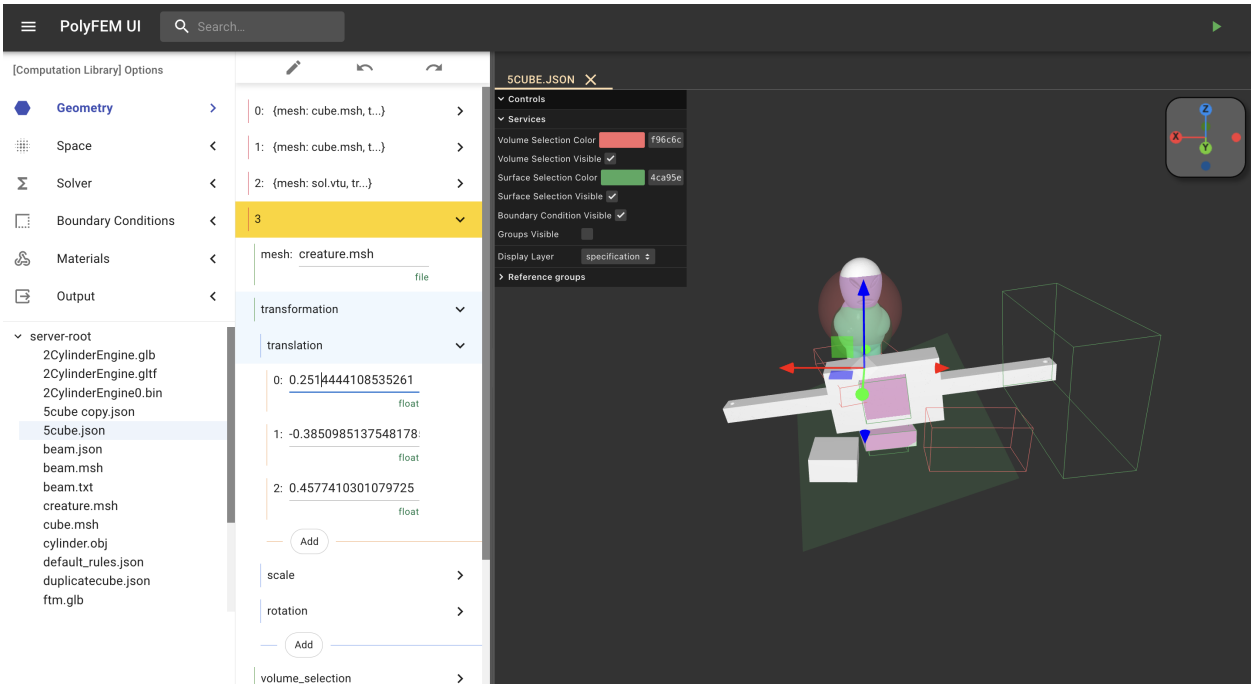

Figure 5.4 PolyFEM User Interface

geometries, set boundary conditions, choose solvers, and visualize results. Distinct geometries like cubes, creatures, and more abstract forms are visualized. Through visualizing these meshes within the UI, users can gain a deeper understanding of their simulation setup and make necessary adjustments for optimal outcomes.

Chapter Six

User Testing and Evaluation

This chapter is an insight into the testing methods utilized to develop a User Interface for Scientific Computing Applications There are four sections in this chapter. Section 6.1 mentions the processes used for the validation of the interface's accuracy and efficacy. Section 6.2 describes the techniques for optimizing the performance of the User interface. Section 6.3 discusses the challenges during process implementation, and Section 6.4 shows snapshots of 3D Visualisations developed using our user interface.

6.1 Testing and Validation

Any UI advancement that expects to further develop ease of use and accessibility must include testing. For a better understanding of testing, we have defined some methods and a use case diagram based on user interaction.

1. Visual inspection: Verifying that an image is correctly displayed and free of distortion or corruption is known as visual inspection.

    - Consistency Across Different Sections: Maintaining Uniformity Throughout Various Sections Check by hand that elements like text fields, menus, and buttons function and look the same across various program sections.

- Authenticate that the text size, style, and color are all applied uniformly across the program. Verify the text's readability and proper spacing.
- Examine how various UI elements respond and provide visual feedback (such as hover effects and clicked states).
- Responsive Design: Test the application on various devices and screen sizes, checking for any layout issues or usability concerns.
- Performance: Make that the user interface (UI) runs smoothly, even when heavy 3D simulations or intricate calculations are being run. When utilizing the application, monitor its performance and look for any slowness, unresponsive elements, or long load times.

2. Use Case - Geometry Input and Simulation Setup: The system is all-inclusive and made to make it easier for users to work with and view geometric data. A comprehensive grasp of its features and user interactions will be possible thanks to this use case graphic (Fig. 6.1) and description. The main Actor will be a User.

    - Load JSON: A JSON file can be manually loaded into the system by the user. The user can import geometric data that will be needed in later steps in this first part of the procedure.
    - 3D Visualization of JSON: When the JSON has been loaded, the user will see the geometrical data in 3D Visualization. By better comprehension of the geometry dimensions, parameters, and spatial associations, the user of this Interface will guarantee precision and exactness in their work.
    - Modify Geometry: The user can define or alter the geometry according to their needs thanks to the system's interactive user interface. Modifying dimensions, shapes, and other geometric properties can be part of this.
    - Define Material Properties: The geometry's material attributes may be defined

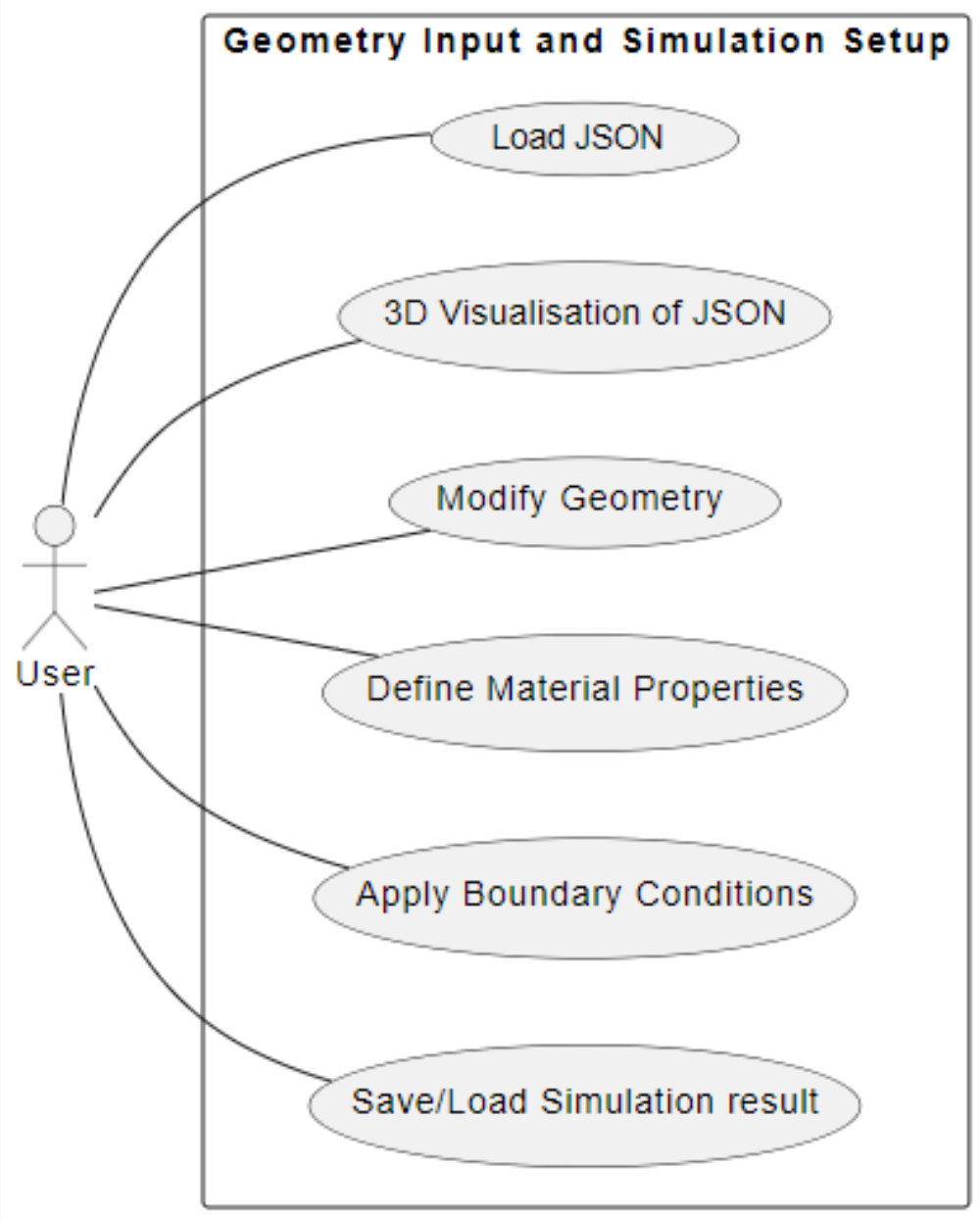

Figure 6.1 Diagram: Geometry Input and Simulation Setup

by the user using the system. correct simulation results depend on this because it guarantees the correct representation of material behavior.

- Apply Boundary Conditions: Setting up the geometry for the simulation requires the user to add boundary conditions. Assuring accurate and consistent simula-

tion results is part of this, as is specifying how the geometry interacts with its environment.

- Save/Load Simulation Result: In the future, the user has the option to preserve any modifications they have made to the setup, including any ongoing configurations, parameters, dimensions, and settings. Additionally, the user may import previous configurations, which facilitates and increases productivity in their process.

3. Performance Comparison: Terminal vs. Generic User Interface :

Aspect	**Terminal**	**Generic User Interface (UI)**
Ease of Use	Requires higher expertise and a steeper learning curve. Suitable for users with command-line experience.	User-friendly with an intuitive interface. Accessible to users with varying expertise levels.
Time Configuration	May take longer due to the need for command-line knowledge.	Significantly reduced configuration time.
Dependability	High when correctly configured.	High, with added robustness for novice users through error handling and validation.
Accessibility	Better suited for advanced configurations and users familiar with command-line interfaces.	Preferred for a streamlined and efficient approach, especially for users seeking ease of use.

Table 6.1 Terminal vs Generic User Interface

Overall, According to table 6.1, the UI Method works better than the Terminal method in terms of accessibility and usability. thus, the UI strategy is prompted for users

who are looking for a basic and effective way to deal with any scientific computing techniques. However, advanced users who needed more precise configuration control still found value in the Terminal method. Considering their particular necessities and level of involvement, users can pursue choice on the best decision with the help of these experimental results.

6.2 Optimization

we have used several optimization techniques in this project mainly focused on the design and implementation of a cross-platform, web-based user interface for scientific computing. The following lists some of the techniques we used :

1. Cross-Platform Web-Based UI: we decided to develop a cross-platform UI that runs within different web browsers. This choice leverages web technologies, offering several advantages, including:

    - Platform Independence: The UI is accessible from various operating systems through common web browsers, ensuring platform independence.
    - Stability and Reliability: Well-established browsers like Chrome are known for their stability and reliability, providing a consistent user experience.
    - Leveraging Web Technologies: Leveraging web technologies allows for rapid development, thanks to the abundance of libraries available in JavaScript and TypeScript.

2. REST Server for Local Operations:

    - A separate REST server is designed to handle file storage, data query, format conversion, and binary executions.

- This server takes care of handling local files and providing a secure way to perform operations that require local access.

3. Predefined Toolsets for Visualizations:

   - The project utilizes the ThreeJS library, a WebGL binding written in React TypeScript, for graphical visualizations.
   - This library enables the rendering of 3D objects, providing a solid foundation for visualizing scientific data.

4. Customizability and Universality:

   - Dynamic Components: The UI incorporates dynamical components that can be redefined for usage in different scenarios.
   - UI Specification Protocol: A universal UI specification protocol is developed to adapt to the specific requirements of scientific computing projects. While certain aspects of the protocol remain constant, such as material settings and geometry importing, it also allows for the selection, choice, and execution of various computing options specific to each task.

6.3 Challenges and Solutions

During the implementation process, we faced several challenges, some of which are listed below:

1. Programming-Heavy Interfaces: Although Blender and ParaView were designed mainly as tools for visualizing and analyzing geometry, using them to interact with simulations required a significant amount of code on the side of the user. Our objective is to minimize the required amount of code by providing a user interface that facilitates the identification of the scenarios and interactions inside the simulation.

2. Complexity of Scientific Computing: Scientific computing involves complex mathematical models and simulations, which can be challenging for users from non-computer science backgrounds. To address this, the solution focused on creating an intuitive and user-friendly interface that abstracts away the technical complexities.

3. Universal UI Specification Protocol: Complicated mathematical models and simulations are a feature of scientific computing that can be difficult for users with little or no experience with computers to understand. A user-friendly interface that abstracts away the technical complexities was the main focus of the solution developed to address this.

6.4 Sample Visualizations

In this section, we present different visualizations (Fig. 6.2 and Fig. 6.3) generated using our generic User interface. These visualizations showcase the versatility and responsiveness of the user interface when applied to various geometrical shapes and models that are generated from individual JSON files.

6.4.1 Visualizations Description

- Different samples display various 3D geometric shapes. These shapes include rectangles, cubes, creature (Human), lines, and some linear elements like beams.

- Overlaying the objects, there are colored arrows pointing in the X (red), Y (green), and Z (blue) directions. This tool is typically used to move objects along specific axes.

- Different colors represent boundary conditions, volume selection, and groupings by unique IDs.

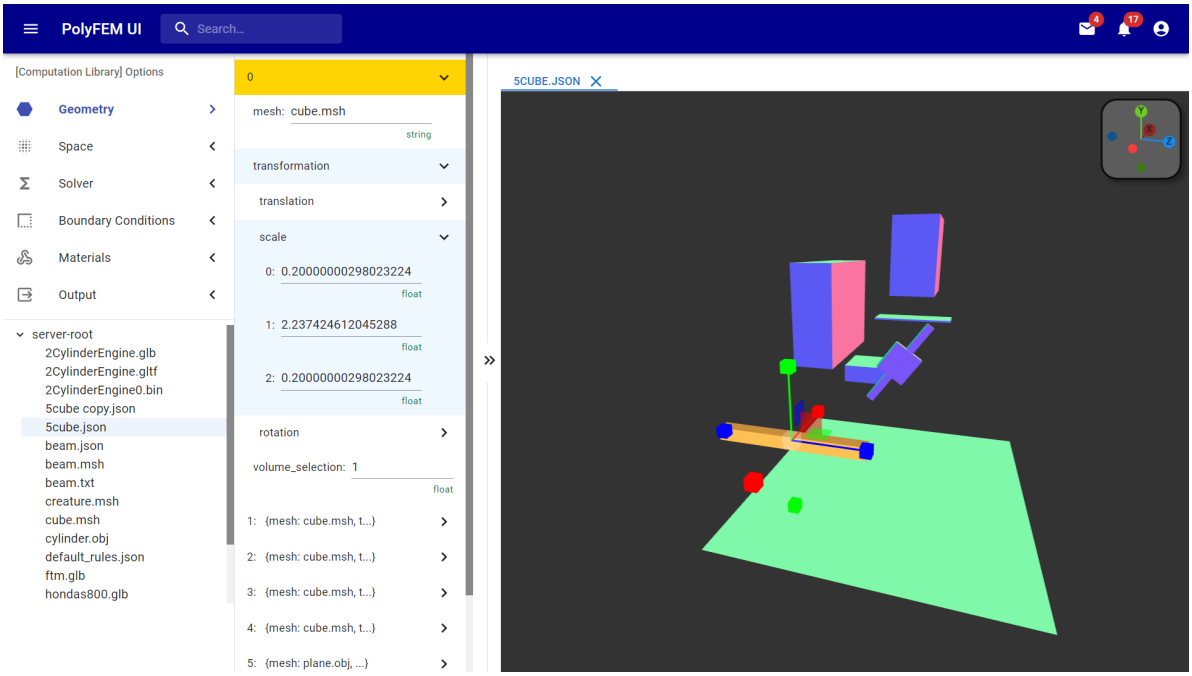

Figure 6.2 3D Visualization of cubes with respect to individual dimensions and geometric configurations

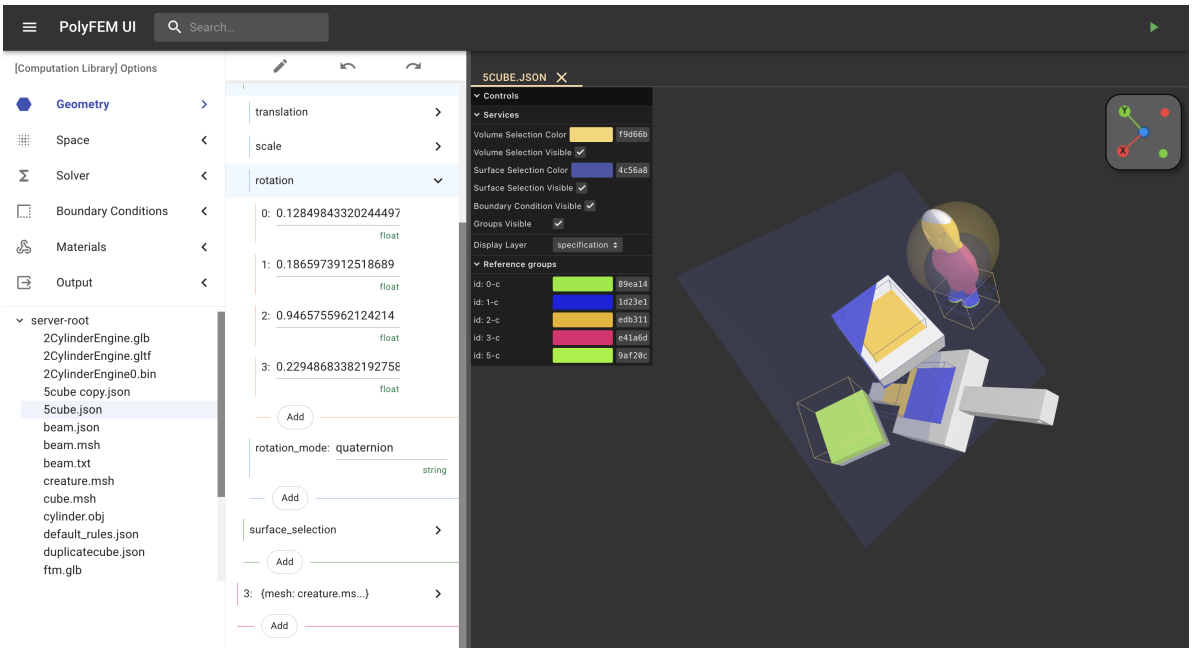

Figure 6.3 3D Visualization of different shapes with reference groups and their color

Chapter Seven

Conclusion and Future Work

7.1 Summary of Contributions and Findings

The goal of this project is to meet the urgent need for a unified, cross-platform user interface in the scientific computing domain. The research endeavor yielded huge discoveries and contributions, one of which is the capability making of a versatile and user-friendly interface that works flawlessly within web browsers, subsequently enlarging accessibility and usability for a wide range of users. In addition, the project takes a novel approach by handling file management and binary execution through the implementation of a local REST server, guaranteeing cross-platform functionality and customization. The project also looks into the creation of a standard protocol for UI specifications that can maintain core functionality while adapting to the particular needs of various scientific computing projects.

7.2 Limitations and Future Work

Ensuring the project's success, there are also certain limitations and areas that require further research work.

1. Usability Testing: In order to guarantee the efficiency and user-friendliness of the cross-platform user interface, usability testing is crucial within the framework of general-

purpose scientific computing. Here's a usability testing plan for this project:

- Specific tasks utilising the user interface will be assigned to participants. A variety of common user operations, such as importing data, setting up simulations, and visualizing outcomes, should be covered by tasks.
- It might be necessary for developers to integrate code, change parameters, and run simulations.
- Experts in unrelated domains may be required to perform tasks like data import, visualization customization, and result interpretation.
- Observers will record task completion time, errors, and user satisfaction.

2. Performance Improvement: Performance issues with the UI could arise as simulations get more complicated. Ongoing work should concentrate on enhancing resource management, optimizing algorithms, and utilizing hardware acceleration in order to maintain the user interface (UI) responsive and efficient. Especially in simulations that require a lot of preliminaries, the responsiveness of the user interface may be greatly enhanced by optimizing for parallel processing, multi-threading, and GPU performance enhancement.

3. Terminal Inside UI: Coordinating a terminal inside the UI could engage advanced users with direct control and scripting capabilities, upgrading the flexibility of the point of interaction for complex geometry processing tasks while keeping a consistent user experience for all expertise levels.

4. Engagement with the Open Source Community: Working together with the open-source community can result in beneficial additions and UI extensions that support more computation tasks and scientific domains. By putting the user interface through rigorous testing and quality assurance with community participation, the reliability and stability of the UI are guaranteed even in challenging scientific computing scenarios.

5. Enhanced Gadgets: Future developments could include extra editing gadgets and tools inside the UI to make complex simulations more sensible. This could include the Run Simulation button, Add animated scenes, Spec searching, Drag/Drop Functionality, etc.

6. Continuous User Feedback Integration: In order to meet changing user needs and make sure that the user interface (UI) is a useful tool for scientific computing in a variety of fields, it is imperative that user feedback be continuously gathered and integrated into the development process.

7.3 Conclusion and Final Remarks

To sum up, the goal of this project is to develop a flexible, cross-platform user interface for scientific computing. This project aims at democratizing the power of computational tools via putting an emphasis on simplicity of use, interoperability across several platforms, and integration of a wide range of scientific computing libraries. It attempts to offer accessibility and customization by utilizing web-based technology and putting in place a local REST server. The main difficulty is coming up with a universal UI specification protocol that can satisfy a variety of scientific computing requirements while maintaining user-friendliness. Researchers in a variety of professions may find it easier to use sophisticated computational tools and scientific discoveries might be expedited by this endeavour.

REFERENCES

[1] K. Tunyasuvunakool et al. "Highly accurate protein structure prediction for the human proteome". In: *Nature* 596.7873 (2022), pp. 590–596.

[2] Matthew Ward, Georges G. Grinstein, and Daniel Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. A K Peters, 2010.

[3] Author(s). *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*. 2014th. London: Springer London, 2014. DOI: 10.1007/978-1-4471-6497-5.

[4] J. Krüger, J. Schneider, and R. Westermann. "ClearView: An Interactive Context Preserving Hotspot Visualization Technique". In: *IEEE Transactions on Visualization and Computer Graphics* 9.4 (2003), pp. 454–462.

[5] T. A. DeFanti and M. D. Brown. "Visualization in Scientific Computing". In: *BYTE* (1988).

[6] E. Deines et al. "Simulation, Visualization, and Virtual Reality based Modeling of Room Acoustics". In: *The Journal Name* The Volume Number.The Issue Number (2007), The Page Range.

[7] B. Platzer. "Book Review: Rainald Löhner, Applied Computational Fluid Dynamics Techniques - An Introduction Based on Finite Element Methods". In: *ZAMM* 89.10 (2009), pp. 869–869. DOI: 10.1002/zamm.200990018.

[8] L. Boström and M. Sjöström. "MethodViz: designing and evaluating an interactive learning tool for scientific methods – visual learning support and visualization of research process structure". In: *Education and Information Technologies* 27.9 (2022), pp. 12793–12810. DOI: 10.1007/s10639-022-11139-9.

[9] Paolo Cignoni et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[10] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331.

[11] Will Schroeder, Ken Martin, and Bill Lorensen. "The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics". In: *IEEE Computer Graphics and Applications* 16.3 (1996), pp. 71–81. DOI: 10.1109/38.486688.

[12] James Ahrens, Berk Geveci, and Charles Law. "ParaView: An End-User Tool for Large Data Visualization". In: *Proceedings of the IEEE Visualization Conference*. 2005.

[13] J.F. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.

[14] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method for Solid and Structural Mechanics*. United Kingdom: Elsevier Science Technology, 2005.

[15] D. H. Norrie. "A first course in the finite element method". In: *Finite Elements in Analysis Design* 3.2 (), pp. 162–163. DOI: 10.1016/0168-874X(87)90008-4.

[16] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Philadelphia: SIAM, 2003.

[17] J. Haddock. "book reviews: Discrete-Event System Simulation by J. Banks and J. S. Carson II Prentice-Hall, Inc., Englewood Cliffs, New Jersey; 1984 $31.95 (hardcover); 500 pp". In: *SIMULATION* 46.2 (1986), pp. 67–68. DOI: 10.1177/003754978604600204.

[18] G.H. Golub and C.F. Van Loan. *Matrix Computations: Fourth Edition*. Johns Hopkins University Press, 2013.

[19] D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006.

[20] Andrew S. Tanenbaum. *Modern Operating Systems*. 3rd. Upper Saddle River, N.J: Pearson Prentice Hall, 2008.

[21] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[22] N. M. Josuttis. *SOA in Practice: The Art of Distributed System Design*. United States: O'Reilly Media, Incorporated, 2007.

[23] Daniel S.H. Lo. *Finite Element Mesh Generation*. Boca Raton: Taylor Francis, 2015. DOI: 10.1201/b17713.

[24] J.D. Foley et al. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1995.

[25] Sebastian Koch et al. "Geometric Computing with Python". In: *SIGGRAPH '19: ACM SIGGRAPH 2019 Courses*. 2019, pp. 1–45. DOI: 10.1145/3305366.3328067.

[26] Nico Schlömer. *meshio: Tools for mesh files*. https://orcid.org/0000-0001-5228-0946. 2018. DOI: 10.5281/zenodo.1173115. URL: https://github.com/nschloe/meshio.

[27] Alec Jacobson and Daniele Panozzo. "libigl: prototyping geometry processing research in C++". In: *Not Specified* 1 (2017). DOI: 10.1145/3134472.3134497.

[28] J. Qiu. *Test-Driven Development with React and TypeScript: Building Maintainable React Applications*. 2nd. Berkeley, CA: Apress L. P, 2023. DOI: 10.1007/978-1-4842-9648-6.

[29] R. Cabello et al. *three.js: JavaScript 3D Library*. GitHub Repository. 2010.

[30] Jos Dirksen. *Learning Three.js - the JavaScript 3D library for WebGL: create stunning 3D graphics in your browser using the Three.js JavaScript library*. 2nd ed. Birmingham, England: Packt Publishing, 2015.

[31] L. G. Rosenfeld. *Prism: Lightweight, robust, elegant syntax highlighting*. 2012.

[32] Mark de Berg et al. *Computational Geometry: Algorithms and Applications*. Berlin: Springer, 2008. DOI: 10.1007/978-3-540-77974-2.

[33] J. R. Shewchuk. "Delaunay Refinement Algorithms for Triangular Mesh Generation". In: *Computational Geometry: Theory and Applications* 22.1-3 (2002), pp. 21–74.

[34] Teseo Schneider et al. *Polyfem*. https://polyfem.github.io/. 2019.