Software Requirements Specification (SRS) for Rubik's Cube Solver

1. Introduction

1.1 Purpose

This SRS document describes the requirements for a Rubik's Cube solver application using **Kociemba's Two-Phase Algorithm** for solving the cube and **OpenGL** for 3D visualization and user interaction.

1.2 Scope

The application will:

- Provide an interactive 3D visualization of a Rubik's Cube.
- Allow users to manually scramble and rotate the cube.
- Solve the cube using Kociemba's algorithm and display the solution visually.

1.3 Definitions, Acronyms, and Abbreviations

- **Rubik's Cube**: A 3D combination puzzle.
- **Kociemba's Algorithm**: A two-phase algorithm for solving Rubik's Cube efficiently.
- **OpenGL**: Open Graphics Library for rendering 2D and 3D vector graphics.
- **UI**: User Interface.
- **SRS**: Software Requirements Specification.

1.4 References

- Kociemba, H. (1992). The two-phase algorithm for solving Rubik's Cube.
- Rokicki, T. et al. (2010). *God's Number is 20*.

1.5 Overview

This document outlines the functional and non-functional requirements, system design, and technical specifications of the Rubik's Cube solver application.

2. Overall Description

2.1 Product Perspective

The Rubik’s Cube solver is a standalone desktop application that renders a Rubik’s Cube in 3D and allows users to interact with and solve it.

2.2 Product Features

- 3D rendering of a Rubik's Cube using OpenGL.
- Cube manipulation through user inputs (mouse and keyboard).
- Solver engine using Kociemba's Two-Phase Algorithm.
- Visual display of the solution steps.

2.3 User Classes and Characteristics

- **Beginners**: Users with basic knowledge of the Rubik’s Cube and no programming expertise.
- **Enthusiasts**: Rubik’s Cube solvers who want to explore efficient algorithms.
- **Researchers**: Developers and researchers interested in cube-solving algorithms.

2.4 Operating Environment

- Operating Systems: Windows, macOS, Linux.
- Development Framework: OpenGL for rendering, C++ for backend and algorithm implementation.

2.5 Design and Implementation Constraints

- Real-time rendering must not compromise system performance.
- Kociemba's algorithm should efficiently compute solutions for all cube states.

2.6 Assumptions and Dependencies

- The system assumes the user has a mouse and keyboard for input.
- The application relies on OpenGL for rendering and requires a system with graphics hardware support.

3. Functional Requirements

3.1 Rendering of Rubik’s Cube

- The system shall render a 3D Rubik’s Cube using OpenGL.
- The cube must support different camera views and rotations.

3.2 User Input Handling

- The system shall allow users to manipulate the cube through mouse input.
- The user shall be able to rotate and scramble the cube manually.

3.3 Cube Solver

- The system shall implement Kociemba’s Two-Phase Algorithm for solving the cube.
- The system must provide a step-by-step visual representation of the solution.

3.4 Solution Display

- The system shall display the number of moves required to solve the cube.
- The solution steps must be animated in real-time for user clarity.

4. Non-Functional Requirements

4.1 Performance

- The application should provide real-time response to user inputs.
- The algorithm must solve the cube within a maximum of 2 seconds for any valid input state.

4.2 Usability

- The user interface must be intuitive, allowing easy interaction with the cube.
- The system must display clear instructions for users.

4.3 Compatibility

- The system should be compatible across Windows, macOS, and Linux platforms.
- The application should run efficiently on both high-end and standard hardware configurations.

4.4 Maintainability

- The code must be modular to allow for future updates or algorithm improvements.
- OpenGL rendering and Kociemba's algorithm should be separated into different modules.

5. System Design

5.1 System Architecture

- **UI Layer**: Responsible for displaying the 3D cube and handling user inputs.
- **Solver Layer**: Contains Kociemba's algorithm for solving the cube.
- **Rendering Layer**: Manages the cube's 3D visualization using OpenGL.

5.2 Data Flow

1. **User Interaction**: The user scrambles or rotates the cube via mouse and keyboard input.
2. **Solver**: The solver is triggered to compute the optimal solution.
3. **Visualization**: The solution is applied to the 3D model and animated for the user.

6. Other Non-functional Attributes

6.1 Reliability

- The system must handle edge cases such as invalid cube states (e.g., improperly scrambled cubes).

6.2 Security

- Since this is a standalone application, no security measures related to data handling are required.

6.3 Portability

- The system should be easily portable across different platforms by compiling the source code in respective environments.

7. Appendices

- **OpenGL Setup Instructions**: Guide to setting up the OpenGL environment.
- **Algorithm Explanation**: Detailed documentation of Kociemba's Two-Phase Algorithm.
- **User Manual**: Instructions on how to interact with the cube and solve it using the application.