CS23820: C & C++ Main Assignment
Fractal Colouring Book Generator

Zeyad S M Moustafa

Zes3@aber.ac.uk

Table of Contents

Introduction 3
C Program Overview 3
  Introduction to the C program 3
    Key functionalities of the C program include: 3
    Purpose 3
  Design choices 4
    Design Choices in main.c 4
    Design Choices in fractal.h 4
    Design Choices in svg_generator.h 4
    Design Choices in parse_NFSF.h 4
  Implementation Details 5
  Issues Faced 5
  Testing and Robustness 6
  Conclusion for C Program 6
My C++ Approach to the program 7
  Advantages: 7
  Exploring Different Viewpoints: 8
  Conclusion: 8

Introduction

The purpose of this report is to provide an overview of the C program developed, highlighting its functionality, design choices, implementation details, and testing procedures. The C program is designed to parse and interpret NFSF (Nested Fractal Scene Format) files, capturing information about transformations, graphics, and branching structures to generate visual scenes in SVG format. The report will explore into the details of the program, discussing challenges faced during development, the robustness of the implemented solution, and considerations for potential improvements or changes. Additionally, the report explores the theoretical transition to a C++ implementation, discussing how the program's design and features might differ in an object-oriented scenario. Through this report, I aim to give understandings of the development process, decision-making, and potential directions for future improvements.

C Program Overview

Introduction to the C program

The C program is designed to interpret and process a custom text-based file format known as NFSF (Nested Fractal Scene Format). This program serves as a graphics generator that interprets the NFSF file, which describes a hierarchical scene of graphical elements, transformations, and fractal structures. The primary objective is to convert the NFSF input into Scalable Vector Graphics (SVG), a widely used XML-based vector image format.

Key functionalities of the C program include:

1. **Parsing NFSF Files:** The program parses NFSF files containing instructions for transformations, graphics, and fractal structures. It identifies keywords and extracts relevant information to construct a hierarchical representation of the scene.
2. **Transformation Handling:** The program interprets transformation instructions, including rotation, translation, and scaling, to manipulate graphic elements in the scene. Each transformation is represented as a distinct entity, enhancing the flexibility of scene composition.
3. **Graphics Representation:** Graphic elements, defined by coordinates, are processed to create a visual representation in the SVG output. The program supports various graphic shapes and configurations, providing a versatile tool for scene generation.
4. **Fractal Structures:** The program interprets branching structures, allowing the creation of recursive and self-replicating patterns. Fractal elements contribute to the richness and complexity of the generated scenes.
5. **SVG Output Generation:** Upon parsing the NFSF file, the program generates an SVG output file that faithfully represents the specified scene. The SVG format ensures scalability and compatibility with various graphic editing tools.

Purpose

The primary purpose of the C program is to offer a flexible and extendable platform for creating complex graphical pictures through the NFSF file format. The NFSF format enables users to specify not only individual graphic elements and transformations but also complex hierarchical structures, promoting the creation of visually appealing and geometrically complex designs.

By converting the NFSF descriptions into SVG format, the program facilitates the visualization and potential further editing of the generated scenes. The modularity of the codebase, with distinct modules for parsing, transformation handling, and SVG generation, contributes to ease of maintenance and potential future enhancements.

In summary, the C program serves as a powerful tool for artists, designers, and enthusiasts seeking a versatile and customizable solution for creating nested fractal scenes with a focus on simplicity, expressiveness, and extensibility.

Design choices

Design Choices in main.c

1. **File Parsing and Structure:** The main responsibility of main.c is to initiate the parsing of NFSF files. The decision to structure the code in a way that separates parsing logic from the main program allows for modularity and easier maintenance.
2. **Memory Management:** Dynamic memory allocation is used for data structures like transforms, graphics, and branches. This choice allows for flexibility in handling varying numbers of elements and avoids fixed-size limitations.

Design Choices in fractal.h

1. **Struct Definitions:** The definition of Transform, Graphic, and Branch structures encapsulates related information, promoting clean and organized code. Each structure aligns with the respective components of an NFSF file.
2. **Abstraction of Fractal Elements:** The Transform struct abstracts rotational, translational, and scaling transformations, providing a comprehensive representation. Similarly, Graphic and Branch structures encapsulate graphic elements and branching information, respectively.

Design Choices in svg_generator.h

1. **Separation of Concerns:** The svg_generator module focuses specifically on generating Scalable Vector Graphics (SVG) output. This separation of concerns enhances code readability, maintainability, and the potential for reuse in other projects.
2. **Clear Function Interfaces:** Functions like generateSVGForTransform and generateSVGForGraphic have clear interfaces, taking specific data structures as input. This choice enhances code readability and ease of understanding.

Design Choices in parse_NFSF.h

1. **Responsibility for File Parsing:** The parse_NFSF module is dedicated to parsing NFSF files. The decision to create a separate module for parsing encapsulates this functionality, making the codebase modular and facilitating easier modifications or extensions to the parsing logic.
2. **Error Handling:** The code includes error-checking mechanisms, such as checking for file opening failures and skipping lines without keywords. This design choice enhances the program's robustness and user-friendliness.

Implementation Details

1. **Modular Structure:**
    - The code is structured into multiple files: main.c, fractal.h, svg_generator.c, and parse_NFSF.h.
    - main.c contains the main function, user input handling, and file I/O.
    - fractal.h and parse_NFSF.h define structures and function prototypes related to fractals and NFSF file parsing.
    - svg_generator.c contains the functions for generating SVG files.
2. **User Input:**
    - User input for the input NFSF file name is obtained using scanf.
    - This allows the user to provide the input filename when running the program.
3. **NFSF File Parsing (parseNFSFFile function):**
    - The code successfully opens the NFSF file, skips comments, and tokenizes lines to identify keywords.
    - Transformation, graphic, and branch data are correctly parsed from the NFSF file.
    - Error checking is implemented to handle cases where the file cannot be opened.
4. **SVG File Generation (generateSVGFile function):**
    - The SVG file is successfully opened for writing.
    - The SVG header and footer are written to the file.
    - Transformations are applied to graphics coordinates, and SVG polyline elements are generated.

Issues Faced

1. **Error Handling in NFSF Parsing:**
    - Initially faced challenges in error handling during NFSF file parsing.
    - Implemented a workaround by allowing the user to input the filename after the program starts running, bypassing the initial error caused by the absence of a filename.
2. **Memory Allocation Issues:**
    - The program uses fixed-size arrays for transforms, graphics, and branches (MAX_TRANSFORMS, MAX_GRAPHICS, MAX_BRANCHES).
    - This can lead to buffer overflow issues if the number of elements exceeds these predefined limits.
    - Consider using dynamic memory allocation to handle a dynamic number of transformations, graphics, and branches.
3. **Coordinate Transformation Order:**
    - The order of applying transformations to coordinates might affect the final result.

  - Ensure that the order of applying scale, rotation, and translation is appropriate for the intended transformation.

4. **Input Validation:**
   - The code checks for errors in opening the NFSF file but could benefit from additional input validation within the NFSF parsing loop.
   - Validate the input data to handle unexpected cases more gracefully.
5. **Warning: 'sscanf' and 'strtod':**
   - Received warnings about using sscanf for floating-point conversions.
   - Consider replacing sscanf with strtod for better error reporting.

Testing and Robustness

During the testing phase, I conducted a comprehensive evaluation of the fractal generation program, scrutinizing both functionality and robustness. I systematically tested the program with diverse NFSF files, manipulating the complexity of transformations, graphics, and branches. These tests included deliberate introduction of errors, exploration of malformed files, and examination of edge cases. While the program demonstrated robustness by providing insightful error messages, adeptly handling unexpected input, and accurately parsing NFSF files, an issue surfaced during the visualization of generated images. Despite successful SVG file generation, the images were not displayed as anticipated when examined using an SVG viewer. This unexpected behaviour prompted a closer examination of the coordinate transformations, scaling factors, and SVG formatting. The user input mechanism, which allowed filename input during runtime, effectively circumvented initial errors related to file absence. Addressing warnings tied to memory allocation and transformation order was crucial for supporting the program's robustness. In summary, while the testing experience instilled confidence in the program's overall reliability.

Conclusion for C Program

In conclusion, the journey of developing the fractal generation program has been both rewarding and challenging. The program successfully handles diverse scenarios, demonstrating robustness during testing. The user-friendly input mechanism addressed initial challenges, and the program maintained efficiency with larger NFSF files. However, an issue arose where the SVG files do not visually represent the intended fractal images, necessitating further investigation. Overcoming challenges related to memory allocation and transformation order enhanced the program's resilience. This experience underscores the importance of thorough testing and iterative refinement in software development. Addressing the SVG display issue will be a key focus for future improvements.

My C++ Approach to the program

Advantages:

1. **Object-Oriented Design:**
   - C++ allows for a more structured, object-oriented design with classes and encapsulation. This can lead to clearer organization and maintainability of the code.
2. **Dynamic Memory Management:**
   - C++ provides advanced memory management features, such as dynamic memory allocation with new and delete operators. This flexibility is advantageous when handling variable-sized data structures like arrays of transforms, graphics, and branches.
3. **File Handling with Streams:**
   - The use of stream classes (ifstream and ofstream) in C++ simplifies file handling, making the code more concise and expressive. This could enhance the readability and maintainability of the file I/O operations.
4. **Exception Handling:**
   - C++ supports robust exception handling through try, catch, and throw. Introducing exception handling in the program could improve its reliability by providing a systematic approach to handle exceptional situations.
5. **STL Algorithms and Data Structures:**
   - The C++ Standard Template Library (STL) offers a wealth of algorithms and data structures. Utilizing these could lead to more efficient and optimized code for certain operations, such as coordinate transformations.

Disadvantages:

1. **Learning Curve:**
   - *Disadvantage:* Transitioning from C to C++ might require developers to familiarize themselves with new language features and concepts, potentially causing a learning curve.
2. **Potential Overhead:**
   - *Disadvantage:* The use of object-oriented features and dynamic memory management, while providing flexibility, might introduce some runtime overhead compared to the simpler, more manual approach in C.
3. **Complexity:**
   - *Disadvantage:* The adoption of advanced C++ features could introduce additional complexity to the program. While object-oriented design can enhance modularity, it might also introduce more layers of abstraction.

Exploring Different Viewpoints:

Developer Perspective:

From a developer's viewpoint, transitioning to C++ can be advantageous for code organization and readability. The availability of modern language features simplifies certain aspects of the implementation.

Maintainability and Scalability:

In the long run, a C++ implementation may offer better maintainability and scalability. Object-oriented principles and dynamic memory management facilitate adapting to evolving requirements and handling larger datasets efficiently.

Performance Considerations:

There could be concerns about potential runtime overhead associated with C++ features. The efficiency gained through more advanced memory management and STL algorithms may, however, offset this concern.

Conclusion:

While adopting C++ in the fractal generation program introduces some complexities and a learning curve, the benefits in terms of code organization, maintainability, and potential performance improvements make a compelling case. The decision should be guided by the specific requirements of the project, the developer familiarity with C++, and the required balance between simplicity and modern language features. Overall, a C++ implementation could contribute to a more modular, maintainable, and expressive codebase, aligning with contemporary software development practices.