Convex Hull Algorithms Simulator

Lab Report - Final Project

Muhommod Toufik Islam
Reg. No: 2021331538
Software Development II Course
2nd year, 2nd Semester

June 23, 2025

Contents

1 Introduction

Convex hull algorithms are fundamental components in computational geometry with extensive applications across various domains including computer graphics, pattern recognition, image processing, and geographic information systems (GIS). A convex hull is defined as the smallest convex polygon that encloses all points in a given set, conceptually similar to stretching a rubber band around the outermost points.

Despite their critical importance in computational geometry, understanding and visualizing these algorithms can be challenging for students and professionals due to their abstract mathematical nature and complex geometric operations. Traditional learning methods through textbooks and static diagrams often fail to provide the intuitive understanding necessary to grasp the step-by-step execution of these algorithms.

This project addresses this educational gap by developing an interactive Convex Hull Algorithm Visualizer that provides a comprehensive platform for understanding, visualizing, and comparing multiple convex hull algorithms. The visualizer serves as both an educational tool for students learning computational geometry and a practical utility for professionals working with geometric algorithms.

The significance of this project lies in its ability to transform abstract algorithmic concepts into tangible, interactive experiences. By providing real-time visualization of algorithm execution, step-by-step animation, and performance comparison capabilities, this tool bridges the gap between theoretical knowledge and practical understanding.

2 Objectives

The primary objectives of this project are structured around creating a comprehensive educational and analytical tool for convex hull algorithms:

2.1 Primary Objectives

- **Algorithm Implementation**: Implement and optimize multiple convex hull algorithms including Graham's Scan, Jarvis March (Gift Wrapping), Monotone Chain, QuickHull, and Chan's Algorithm
- **Interactive Visualization**: Develop a dynamic, user-friendly interface that demonstrates step-by-step algorithm execution with animated graphics
- **Performance Analysis**: Provide real-time comparison of algorithm performance, including time complexity analysis and comparison counting
- **Educational Enhancement**: Create an accessible learning platform that aids in understanding computational geometry concepts

2.2 Secondary Objectives

- Enable custom data point input and random point generation capabilities
- Implement zoom and pan functionality for detailed visualization
- Provide algorithm selection and configuration options
- Create comprehensive documentation and user guides
- Ensure cross-platform compatibility through Java implementation

2.3 Learning Outcomes

- Deep understanding of convex hull algorithm mechanics and complexity analysis
- Practical experience in Java Swing GUI development and event-driven programming
- Knowledge of computational geometry principles and geometric algorithm optimization
- Experience in software architecture design and modular programming practices

3 Functionality & Features

The Convex Hull Algorithm Simulator provides a comprehensive set of features designed to enhance learning and analysis of convex hull algorithms:

3.1 Core Algorithms Implemented

1. **Graham's Scan**: O(n log n) time complexity algorithm using polar angle sorting
2. **Jarvis March (Gift Wrapping)**: O(nh) time complexity where h is the number of hull points
3. **Monotone Chain**: O(n log n) algorithm processing points in lexicographic order
4. **QuickHull**: Divide-and-conquer approach with average O(n log n) complexity
5. **Chan's Algorithm**: Optimal O(n log h) algorithm combining multiple techniques

3.2 User Interface Features

- **Interactive Point Input**: Click-to-add point functionality with mouse interaction
- **Random Point Generation**: Automated generation of random point sets for testing
- **Algorithm Selection**: Dropdown menu for choosing different convex hull algorithms
- **Animation Controls**: Play, pause, reset, and step-through animation capabilities
- **Speed Control**: Adjustable animation speed for detailed observation
- **Zoom and Pan**: Enhanced viewing capabilities for detailed analysis
- **Theme Support**: Multiple UI themes using FlatLaf library (Dark, Light, Dracula, etc.)

3.3 Visualization Capabilities

- **Real-time Animation**: Step-by-step visualization of algorithm execution
- **Color-coded Elements**: Different colors for input points, hull points, and construction lines
- **Progress Indicators**: Visual feedback on algorithm completion status
- **Geometric Highlighting**: Emphasis on current algorithm operations and decisions
- **Anti-aliased Graphics**: High-quality rendering for smooth visual experience

3.4 Analysis Features

- **Performance Metrics**: Real-time display of algorithm execution statistics
- **Comparison Counter**: Track and display the number of geometric comparisons
- **Time Complexity Analysis**: Theoretical and practical complexity demonstration
- **Algorithm Comparison**: Side-by-side performance analysis capabilities

4 System Architecture

The Convex Hull Algorithm Visualizer follows a modular, object-oriented architecture designed for maintainability, extensibility, and clear separation of concerns.

4.1 Architecture Overview

The system is organized into three primary modules:

```
Convex Hull Algorithms Visualizer/
 algorithms/          # Algorithm implementations
 setup/              # Core utilities and data structures
 userinterface/      # GUI components and visualization
```

Figure 1: Project Structure Overview

4.2 Module Descriptions

4.2.1 Algorithms Module

This module contains the implementation of all convex hull algorithms:

- **ConvexHullAlgorithm.java**: Abstract base class defining the common interface
- **GrahamScan.java**: Implementation of Graham's Scan algorithm
- **JarvisMarch.java**: Gift wrapping algorithm implementation
- **MonotoneChain.java**: Andrew's monotone chain algorithm
- **QuickHull.java**: Divide-and-conquer QuickHull implementation
- **ChanAlgorithm.java**: Optimal Chan's algorithm implementation

4.2.2 Setup Module

Core utilities and data structures supporting the algorithms:

- **Point.java**: Fundamental point class with geometric operations
- **PointStack.java**: Specialized stack for point operations
- **HeapSort.java**: Efficient sorting implementation for point ordering
- **Line.java**: Line segment representation and operations
- **Stack.java**: Generic stack implementation
- **Median.java**: Median finding utilities for optimization
- **PointCircular.java**: Circular point list for specific algorithms

4.2.3 User Interface Module

GUI components and visualization logic:

- **Visualizer.java**: Main application window and control logic
- **AnimationArea.java**: Canvas component for algorithm visualization

4.3 Design Patterns Implemented

- **Strategy Pattern**: Algorithm selection through abstract base class
- **Observer Pattern**: Event-driven GUI updates and animation control
- **Template Method**: Standardized algorithm execution framework
- **Factory Pattern**: Dynamic algorithm instantiation based on user selection

4.4 Technology Stack

- **Java SE**: Core programming language for cross-platform compatibility
- **Java Swing**: GUI framework for desktop application development
- **FlatLaf**: Modern look-and-feel library for enhanced UI aesthetics
- **Java AWT**: Low-level graphics and event handling
- **BufferedImage**: Double-buffered graphics for smooth animation

5 Project Approach

The development of the Convex Hull Algorithm Simulator followed a systematic, phase-based approach designed to ensure thorough implementation and testing of all components.

5.1 Development Methodology

The project employed an iterative development approach with clear phases:

5.1.1 Phase 1: Requirements Analysis and Research (Weeks 1-2)

- Comprehensive study of convex hull algorithms and their mathematical foundations
- Analysis of existing visualization tools to identify improvement opportunities
- Definition of functional and non-functional requirements
- Technology stack selection and architecture planning

5.1.2 Phase 2: Core Algorithm Implementation (Weeks 3-5)

- Implementation of fundamental data structures (Point, Stack, etc.)
- Development of sorting and utility functions
- Sequential implementation of convex hull algorithms
- Unit testing and algorithm verification with known datasets

5.1.3 Phase 3: User Interface Development (Weeks 6-7)

- Design and implementation of the main application window
- Development of the animation canvas with double-buffered graphics
- Integration of algorithm implementations with the visualization system
- Implementation of user interaction features (mouse input, controls)

5.1.4 Phase 4: Testing and Optimization (Weeks 8-9)

- Comprehensive testing with various datasets and edge cases
- Performance optimization and animation smoothness improvements
- Bug fixing and stability enhancements
- User experience testing and interface refinements

5.1.5 Phase 5: Documentation and Finalization (Week 10)

- Code documentation and commenting
- User manual creation
- Final testing and quality assurance
- Project packaging and deployment preparation

5.2 Implementation Challenges and Solutions

5.2.1 Algorithm Complexity Management

Challenge: Implementing multiple complex algorithms while maintaining code clarity and performance. **Solution**: Adopted abstract base class pattern with standardized interfaces, enabling consistent implementation across all algorithms.

5.2.2 Animation Synchronization

Challenge: Coordinating real-time visualization with algorithm execution without blocking the user interface. **Solution**: Implemented step-based algorithm execution with state management, allowing controlled progression through algorithm steps.

5.2.3 Performance Optimization

Challenge: Ensuring smooth animation performance while handling large datasets. **Solution**: Utilized double-buffered graphics and optimized rendering techniques to maintain responsive user interface.

5.3 Quality Assurance Approach

- **Algorithm Verification**: Tested each algorithm against known convex hull results
- **Performance Testing**: Evaluated algorithms with datasets of varying sizes
- **User Interface Testing**: Conducted usability testing with potential users
- **Cross-platform Compatibility**: Tested on multiple operating systems

6 Advantages

The Convex Hull Algorithm Simulator offers significant advantages over traditional learning methods and existing tools:

6.1 Educational Advantages

- **Visual Learning Enhancement**: Transforms abstract mathematical concepts into intuitive visual representations, significantly improving comprehension
- **Step-by-Step Understanding**: Allows learners to observe each algorithmic decision and geometric operation in detail
- **Interactive Exploration**: Enables hands-on experimentation with different point configurations and algorithm behaviors
- **Comparative Analysis**: Provides direct comparison between algorithms, highlighting their relative strengths and weaknesses
- **Self-Paced Learning**: Users can control animation speed and progression according to their learning needs

6.2 Technical Advantages

- **Cross-Platform Compatibility**: Java-based implementation ensures operation across Windows, macOS, and Linux systems
- **Modular Architecture**: Clean separation of concerns enables easy maintenance and extension
- **Performance Optimization**: Efficient algorithm implementations with optimal time and space complexity
- **Modern User Interface**: Contemporary design using FlatLaf themes for enhanced user experience
- **Scalable Design**: Architecture supports easy addition of new algorithms and features

6.3 Practical Advantages

- **Real-World Application**: Directly applicable to fields requiring geometric computation
- **Research Tool**: Useful for algorithm comparison and performance analysis
- **Teaching Aid**: Valuable resource for instructors teaching computational geometry
- **Professional Development**: Enhances understanding of algorithm design and implementation
- **Open Source Potential**: Codebase can be extended and customized for specific needs

6.4 Competitive Advantages

- **Comprehensive Algorithm Coverage**: Implements multiple algorithms in a single, unified platform
- **Advanced Visualization**: Superior graphics quality with anti-aliasing and smooth animations
- **User-Friendly Design**: Intuitive interface requiring minimal learning curve
- **Performance Metrics**: Built-in analysis tools for algorithm comparison and evaluation
- **Customizable Experience**: Multiple themes and adjustable settings for personalized usage

7 Future Updates

The Convex Hull Algorithm Simulator provides a solid foundation for numerous enhancements and extensions:

7.1 Algorithm Enhancements

- **3D Convex Hull Algorithms**: Extension to three-dimensional space with algorithms like QuickHull 3D and Incremental Construction
- **Approximate Algorithms**: Implementation of approximation algorithms for very large datasets
- **Parallel Algorithms**: Multi-threaded implementations for improved performance on large datasets
- **Online Algorithms**: Dynamic convex hull maintenance as points are added or removed
- **Specialized Variants**: Algorithms optimized for specific point distributions or constraints

7.2 User Interface Improvements

- **Advanced Controls**: More sophisticated animation controls including backward stepping and bookmarking
- **Multiple Viewport Support**: Side-by-side algorithm comparison with synchronized execution
- **Customizable Visualization**: User-defined colors, point sizes, and animation styles
- **Touch Interface Support**: Adaptation for tablet and touch-screen devices
- **Accessibility Features**: Support for screen readers and keyboard navigation

7.3 Analysis and Reporting Features

- **Performance Profiling**: Detailed timing analysis and memory usage tracking
- **Statistical Analysis**: Comprehensive algorithm performance statistics over multiple runs
- **Export Capabilities**: Generation of reports, images, and animation sequences
- **Benchmark Suite**: Standardized test cases for algorithm evaluation
- **Complexity Visualization**: Graphical representation of time and space complexity

7.4 Educational Enhancements

- **Tutorial Integration**: Built-in guided tutorials for each algorithm
- **Exercise Generation**: Automatic problem generation for practice
- **Learning Path Tracking**: Progress monitoring and adaptive learning features
- **Collaborative Features**: Multi-user support for classroom environments
- **Assessment Tools**: Quiz and evaluation capabilities for educational use

7.5 Platform Extensions

- **Web Application**: Browser-based version using WebGL or Canvas API
- **Mobile Applications**: Native iOS and Android implementations
- **Virtual Reality**: Immersive 3D algorithm visualization in VR environments
- **Cloud Integration**: Online collaboration and cloud-based computation
- **API Development**: RESTful API for integration with other educational platforms

8 Conclusion

The development of the Convex Hull Algorithm Simulator represents a successful integration of computational geometry theory with practical educational technology. This project successfully bridges the critical gap between abstract mathematical concepts and their practical understanding through interactive visualization.

The completed application stands as a testament to the effectiveness of visual learning approaches in complex algorithmic domains. By transforming static textbook concepts into dynamic, interactive experiences, the visualizer demonstrates how modern software development can directly enhance educational outcomes in computer science education.

From a personal development perspective, this project provided invaluable hands-on experience with the complete software development lifecycle - from initial concept through final implementation. The challenges encountered during development, particularly in animation synchronization and performance optimization, resulted in deeper understanding of both algorithmic complexity and user interface design principles.

The project's modular architecture and comprehensive implementation create a solid foundation for future educational tools in computational geometry. The successful completion within the planned timeline demonstrates effective project management and iterative development practices that are directly applicable to professional software development environments.

Most importantly, this visualizer addresses a real educational need in the computer science curriculum. The gap between theoretical algorithm study and practical

understanding is a persistent challenge in computational geometry education. By providing an interactive platform where students can experiment with and observe algorithm behavior in real-time, this tool contributes meaningfully to modern pedagogical approaches that emphasize active learning and visual comprehension.

The Convex Hull Algorithm Simulator ultimately serves as both a practical educational resource and a demonstration of how thoughtful software design can transform the learning experience in technical disciplines.