Shortest Path Between 2 Points in 3D Space in the Presence of Obstacles

Final Progress Report

Shadi Seaidoun

COMPSCI 4490Z: THESIS

Department of Computer Science
Western University

May 28, 2025

Project supervisor: Dr. Roberto Solis-Oba, Department of Computer Science
Course instructor: Dr. Nazim Madhavji, Department of Computer Science

Glossary

Cone In Clarkson's algorithm, a geometric construct dividing space around a point into angular sectors. Each cone guides visibility checking and subgraph construction for the approximation algorithm.

Configuration Space A transformation of the physical space in motion planning where obstacles are represented as forbidden regions, simplifying pathfinding calculations.

Dynamic Edge Sequence A concept used in Jiang's algorithm where a sequence of visible obstacle edges dynamically changes as path turning points are refined through numerical optimization.

Edge Subdivision The process of dividing obstacle edges into smaller segments to build visibility graphs, as used in Papadimitriou's algorithm to approximate paths through complex environments.

GLFW Graphics Library Framework. A library for creating windows and handling input/output in OpenGL-based applications.

GLM OpenGL Mathematics. A C++ library based on GLSL (OpenGL Shading Language) for vector and matrix operations in graphics applications.

ImGui Immediate Mode Graphical User Interface. A C++ library for real-time debugging interfaces and GUI development.

NP-hard A class of computational problems for which no known polynomial-time algorithm exists. These problems are at least as hard as the hardest problems in NP.

Polyhedral Obstacle A 3D shape composed of flat polygonal faces, forming the barriers that the path must avoid.

Projection Technique Used in Jiang's method to reduce 3D visibility checking to a 2D problem by projecting vertices and edges onto a plane.

Turning Point A point on a path where the direction changes. In Jiang's algorithm, turning points are optimized along edges to minimize path length.

Visibility Graph A graph where nodes represent obstacle edges or corners, and edges connect visible pairs. Used in all three algorithms to determine possible path segments.

Voronoi Diagram A geometric structure partitioning space based on proximity to a set of sites. Used in Clarkson's method to assist visibility and path region partitioning.

ϵ **(epsilon)** A small positive value used to define approximation accuracy. Algorithms guarantee that the computed path is within a factor of $(1 + \epsilon)$ of the true shortest path.

Structured Abstract

Context and Motivation

Finding the shortest path between two points in 3D space while avoiding obstacles is fundamental in computational geometry with applications in robotics and motion planning. While efficient algorithms exist for 2D cases, exact 3D solutions remain computationally intensive and NP-hard. Current approximation algorithms employ complex geometric constructions that are difficult to implement. This research implements, evaluates, and compares three approaches to identify the most practical solution and develop optimizations for improved performance.

Research Question/Objectives

How do three approaches to 3D shortest path planning (Clarkson's approximation algorithm [1], Jiang's visibility graph method [2], and Papadimitriou's discretization algorithm [3]) compare in terms of computational efficiency, implementation complexity, and solution quality? Can modifications to one of these algorithms improve performance while maintaining theoretical guarantees?

Principal Ideas

Our approach involves: implementing all three algorithms following their original specifications, identifying the algorithm with the most potential for practical improvement, and developing modifications to reduce computational complexity while maintaining reasonable approximation guarantees.

Research Methodology

We implement and evaluate algorithms within a C++ environment with OpenGL visualization, ensuring uniform data structures and testing procedures. Performance metrics including runtime, path quality, and memory usage are assessed through experiments with varied obstacle configurations. Real-time monitoring is facilitated through ImGui integration.

Anticipated Results

We provide comparative analysis showing practical tradeoffs between the three approaches, quantitative measurements of their performance characteristics, and validation of our proposed algorithmic modification.

Anticipated Novelty

This research provides the first comprehensive comparison of these three fundamental approaches to 3D shortest path planning. The modified parallelized version of Clarkson's algorithm represents a novel balance between theoretical guarantees and practical efficiency.

Limitations

Path optimality cannot be guaranteed for all cases with approximation algorithms. Results depend on specific test scenarios and implementation decisions. Our optimizations prioritize practical efficiency over theoretical guarantees.

Contents

1 Introduction

While significant theoretical progress has been made in 3D shortest path approximation, practical implementation remains underexplored. Many existing algorithms rely on complex geometric constructions that introduce complexity and hinder usability in applied settings. This thesis implements and evaluates three such algorithms to identify the most practical approach and explore optimizations that improve real-world performance.

To address these challenges, researchers have developed approximation algorithms finding paths within $(1 + \epsilon)$ of optimal length. In this paper, we discuss: Papadimitriou's algorithm [3], Clarkson's algorithm [1] and Jiang's algorithm [2]. Despite their theoretical importance, these algorithms lack comparative practical implementation analysis, limiting their adoption in real systems.

This thesis addresses this gap through comprehensive implementation and analysis of these approaches to 3D shortest path planning. We investigate: (1) How these algorithms compare in computational efficiency, implementation complexity, and solution quality across varying obstacle configurations; (2) Whether modifications can improve practical performance while maintaining theoretical guarantees; and (3) What implementation considerations extend beyond theoretical complexity analysis.

Our methodology involved implementing all three algorithms according to original specifications within a unified framework with OpenGL visualization and performance monitoring. We developed a parallelized version of Clarkson's algorithm to improve efficiency while preserving theoretical guarantees, and systematically evaluated performance across obstacle densities (1-500) and different epsilon values.

Results revealed significant differences between theoretical predictions and practical performance. Jiang's algorithm demonstrated exceptional efficiency (0.00002s-0.08s) with near-linear scaling despite lacking polynomial-time guarantees. Papadimitriou's algorithm performed well for moderate environments but failed beyond 100 obstacles. Clarkson's algorithm produced highest quality paths but with prohibitive computation times (5078.56s for 100 obstacles), while our parallelized implementation achieved up to a 76% reduction in computation time.

The primary contributions include: (1) the first comprehensive empirical comparison of these algorithms with consistent metrics; (2) a parallelized Clarkson implementation with significantly reduced computation time; and (3) practical insights beyond theoretical analysis regarding memory utilization and scaling behavior.

The remainder of this report is organized as follows: Section 2 reviews literature and identifies research gaps; Section 3 outlines research objectives; Section 4 details implementation methodology; Section 5 presents results; Section 6 discusses implications and limitations; Section 7 provides conclusions; and Section 8 explores future work and lessons learned.

2 Background and Related Work

Finding the shortest path in environments with obstacles is a core problem in robotics, computational geometry, and path planning. The complexity increases significantly in 3D space due to the need to navigate around polyhedral obstacles. Researchers have developed various algorithms to address this challenge, including both exact solutions and approximation methods.

2.1 Scientific and Technical Developments

Early foundational work on path planning in 3D space was introduced by Lozano-Pérez and Wesley (1979), who extended the concept of visibility graphs from 2D to 3D. They proposed converting the workspace into a configuration space where obstacles are represented as regions to be avoided. Their method involves discretizing obstacle edges and applying the A* algorithm to approximate the shortest path [2].

Building on these ideas, Sharir and Schorr [4] explored the combinatorial complexity of shortest path problems involving polyhedral obstacles. They introduced key lemmas about the geometric properties of shortest paths, such as how paths interact with obstacle edges and ridge lines. They demonstrated that solving this problem exactly for multiple obstacles requires doubly exponential time [4].

In response to the computational infeasibility of exact solutions, approximation algorithms have been developed. Papadimitriou [3] proposed a fully polynomial approximation scheme for finding paths within a factor of $(1+\epsilon)$ of the shortest path length. His algorithm subdivides obstacle edges into segments to reduce complexity, achieving a running time of $O(n^4(L+\log(n/\epsilon))^2/\epsilon^2)$, where n is the number of elements (vertices, edges, faces), L is the coordinate precision, and ϵ is the approximation accuracy.

Clarkson [1] further improved approximation techniques by using Voronoi diagrams and cone-based visibility structures. His algorithm finds ϵ-short paths in $O(n^2\lambda(n)\log(n/\epsilon)/\epsilon^4+n^2\log n\rho\log(n\log\rho))$ time, where $\lambda(n)$ is a very slowly growing function related to the inverse Ackermann function, n is the number of obstacle faces, and ρ is the ratio of the length of the longest edge between the longest edge amongst the obstacles between the two points in space. [1]

Jiang et al. [2] took a more geometric and iterative approach by introducing a dynamic edge sequence method. They project obstacle edges into 2D to identify visibility relationships, then iteratively refine a sequence of visible edges, adjusting "turning points" along edges to approximate the optimal path. The algorithm blends visibility graph construction with numerical optimization, enabling real-time-like approximations of shortest paths even as edge sequences dynamically shift during the computation [2].

2.2 Analysis and Research Gap

Comparing the approaches outlined above reveals several trade-offs and research gaps. Exact methods provide optimal solutions but are computationally intractable for scenes with more than

a few obstacles. Approximation algorithms offer a balance between solution quality and computation time, with Papadimitriou's approach being more straightforward to implement but potentially less efficient than Clarkson's for large n.

Existing literature lacks comprehensive comparative studies that implement and evaluate these algorithms on a common platform under identical testing conditions. Most research focuses on theoretical aspects or simplified implementations, without providing detailed performance analysis across varying obstacle configurations.

Furthermore, while theoretical complexity bounds are well-established, the practical performance of these algorithms on modern hardware remains under-explored. This research addresses these gaps by implementing the three key approaches (Papadimitriou, Clarkson, and Jiang) within a unified framework, providing direct performance comparisons, and developing optimizations that leverage modern computing architectures to improve practical efficiency while maintaining theoretical guarantees.

3 Research Objectives

O1: Develop a 3D visualizer using OpenGL to represent the environment, providing a clear view of the 3D space where shortest path planning will be conducted.

O2: Incorporate the ability to add polyhedral obstacles within the visualizer, enabling the simulation of realistic path-planning scenarios.

O3: Implement Clarkson's approximation algorithm, the visibility graph method, and Papadimitriou's discretization algorithm for 3D shortest path planning.

O4: Integrate functionality to display the start and end points, as well as the paths generated by each of the implemented algorithms.

O5: Display key performance metrics for each algorithm using the ImGui library, including runtime, path length and memory usage.

O6: Identify and modify one of the three algorithms to enhance practical performance while maintaining theoretical guarantees for approximation accuracy.

O7: Visualize the path produced by the modified algorithm and display the associated performance metrics.

O8: Compare the metrics of the original algorithms with those of the modified algorithm to evaluate improvements in efficiency and solution quality.

4 Methodology

This research involved implementing and evaluating three different approaches to 3D shortest path planning through a structured methodology focusing on both theoretical correctness and practical performance. The methodology followed these key stages:

4.1 Algorithm Implementation

All three algorithms were implemented following their original specifications as presented in their respective papers, with careful attention to their mathematical foundations and theoretical guarantees:

1. **Papadimitriou's Algorithm**: Implemented as described in "An Algorithm for Shortest-Path Motion in Three Dimensions" [3]. The implementation focused on discretizing obstacle edges into segments based on epsilon, building a visibility graph connecting these segments, and finding the shortest path in the resulting graph using Dijkstra's algorithm. A two-phase approach was employed, where a coarse path was first computed to determine approximate path length, followed by finer discretization.

2. **Clarkson's Algorithm**: Implemented based on "Approximation Algorithms for Shortest Path Motion Planning" [1]. This involved generating a family of cones to partition the space, identifying combinatorially equivalent segments on obstacle edges, and building a subgraph of the visibility graph. The algorithm applies Dijkstra's algorithm to find a path that is within $(1 + \epsilon)$ of the optimal path length.

3. **Jiang's Algorithm**: Following the approach in "3D Shortest Path Planning in the Presence of Polyhedral Obstacles" [2], this implementation focused on identifying visible bound edges through projection techniques, establishing dynamic edge sequences, and optimizing paths through an iterative refinement process based on equal diagonal angle conditions.

4.2 Development Environment

The implementation was conducted using C++ as the primary programming language, chosen for its performance characteristics and robust computational libraries. The development environment included:

1. **Graphics and Visualization**: OpenGL with GLFW for rendering the 3D environment, obstacles, and paths

2. **Mathematical Calculations**: GLM (OpenGL Mathematics) for vector operations, transformations, and geometric calculations

3. **User Interface**: ImGui for real-time performance monitoring and interactive controls

4. **Build System**: A structured Makefile for compilation and testing automation

4.3 Testing Methodology

To ensure robust comparison, a systematic testing approach was implemented:

1. **Obstacle Generation**: Procedural generation of unit cube obstacles between start and end points with varying distributions to test algorithm performance under different environmental complexities.

2. **Epsilon Analysis**: Each algorithm was tested across a range of epsilon values between 0 to 1 to determine optimal settings for different performance criteria (computation time and path length).

3. **Scaling Tests**: Algorithms were evaluated with increasing obstacle counts (1 to 500) to assess computational complexity scaling in practice.

4. **Performance Metrics**: For each test, comprehensive metrics were recorded including computation time (seconds), memory usage (KB) and path length (units).

4.4 Validation

Algorithm implementations were validated through visual verification using the OpenGL visualizer to confirm paths avoided obstacles and followed expected patterns. During path construction, each path segment was continuously tested for intersection with all obstacles to ensure the validity of the generated path - if any intersection was detected, the path computation would immediately terminate and be marked as invalid. This continuous validation ensured that all reported solutions represented feasible routes through the 3D environment. Validation also included comparative analysis through cross-verification of results between algorithms, and edge case testing including direct path availability, axis-aligned points, and environments with varying obstacle densities.

This methodology enabled not only a fair comparison between the three approaches but also provided insights into practical implementation considerations that affect real-world performance beyond theoretical complexity bounds.

5 Results

The research implemented a comprehensive 3D path planning environment to evaluate three distinct algorithmic approaches and their variations. Figure 1 illustrates the system architecture, comprising several interconnected components designed for end-to-end path planning experimentation. At its core, the system features a 3D visualization engine based on OpenGL for rendering the spatial environment and paths, an obstacle generation module capable of producing configurable arrangements of polyhedral obstacles, four algorithm implementation modules with consistent interfaces for fair comparison, a performance analysis framework that collects and processes runtime metrics, and an interactive user interface for environment manipulation and algorithm selection.

The system accepts a variety of inputs including complex obstacle configurations with adjustable density and positioning, customizable start and end points to define the path planning problem,

algorithm-specific parameters like epsilon values for approximation algorithms, and visualization preferences. These inputs are processed through the selected path planning algorithm, which interfaces with both the environment and performance analysis components. The system then outputs detailed visualizations of generated paths overlaid on the 3D obstacle environment, along with comprehensive performance metrics including computation time, memory utilization and path length characteristics. This integrated architecture facilitates not only qualitative assessment through visual inspection but also quantitative comparative analysis across different algorithmic approaches under identical testing conditions to evaluate the practical performance implications beyond theoretical complexity bounds.

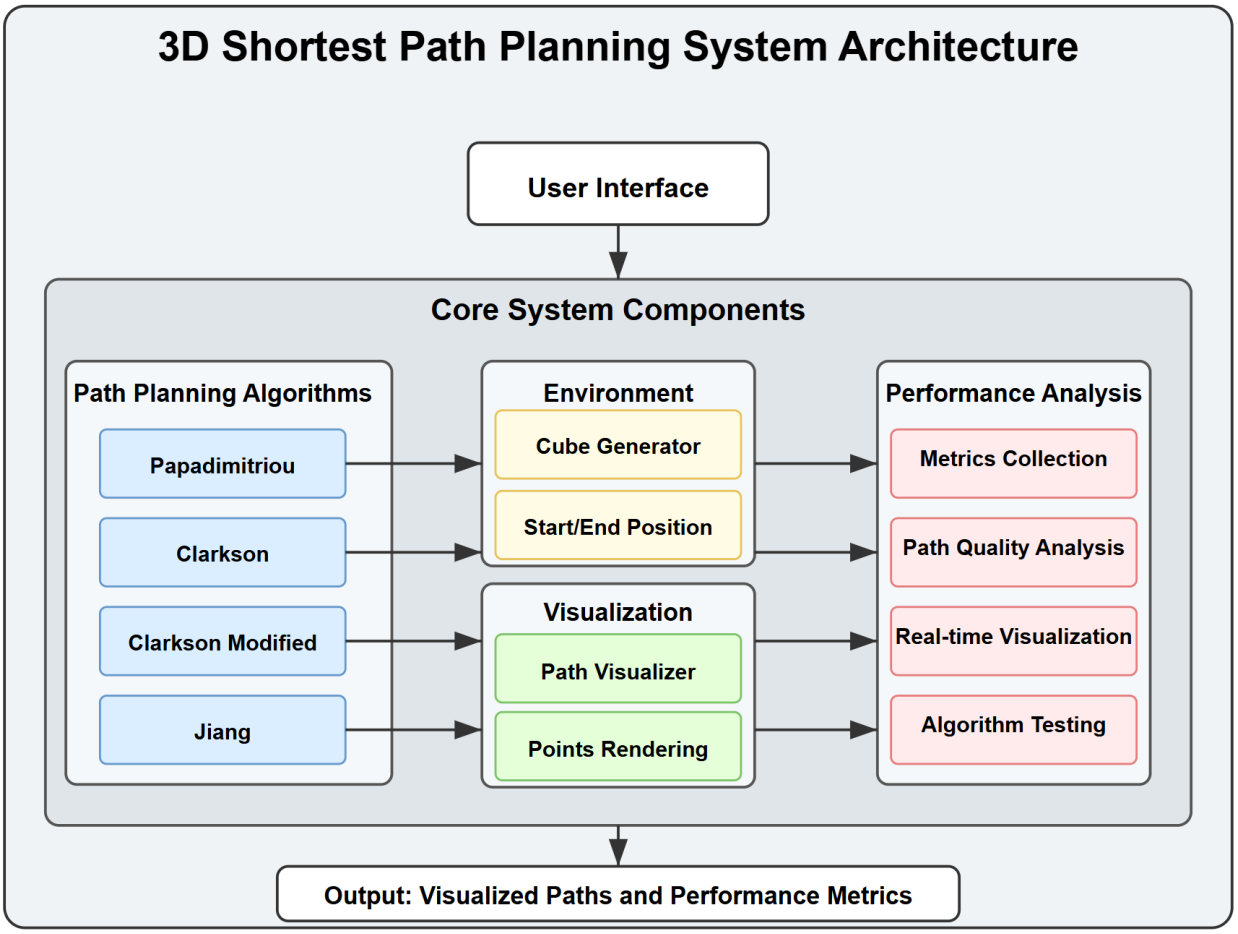


Figure 1: System Architecture

5.1 Technical Work Carried Out

The technical work involved implementing three shortest path algorithms and a modified parallelized version in a 3D environment with obstacle avoidance capabilities, utilizing a component-based architecture for modularity and extensibility.

5.1.1 Key Requirements That Were Met

The system successfully met several critical requirements:

- **Algorithm Accuracy**: All implemented algorithms correctly find valid paths between source and target points while avoiding obstacles, with path quality proportional to the epsilon parameter.
- **Interactive Visualization**: The system provides real-time 3D visualization of the environment, obstacles, and computed paths through an OpenGL-based rendering engine.
- **Performance Measurement**: Comprehensive metrics including computation time, memory usage and path length are collected and displayed for each algorithm run.
- **Algorithm Comparison**: The system allows direct comparison between different algorithms through a unified interface and consistent measurement methodology.
- **Algorithm Optimization**: Based on performance analysis, Clarkson's algorithm was identified for optimization. The modified version implements parallel processing of visibility graph construction using C++ threading, allowing concurrent evaluation of visibility relationships across multiple cores

5.1.2 System Design and Architecture

The system was designed with a modular architecture consisting of five primary components as shown in Figure 1. Each algorithm was implemented as a separate class with a consistent interface to facilitate fair comparison.

Design Patterns and Interfaces: The implementation employed the Strategy Pattern for encapsulating algorithm implementations behind a common interface, the Observer Pattern for monitoring execution and collecting metrics, and the Factory Pattern for obstacle generation. All algorithm classes implement a consistent interface with methods for finding shortest paths and accessing visibility graphs. The visualization components provide rendering capabilities for paths and graphs, while the performance analysis interface offers metrics collection and display functionality.

Quality Attributes and Design Rationale: The system prioritizes performance through optimized computation-intensive operations, particularly through multi-threading in the modified Clarkson algorithm. Usability is enhanced through intuitive controls and a clear metrics dashboard. The modular design ensures extensibility for adding new algorithms without core system changes. Key design decisions include: separating algorithms from visualization to focus implementations on computational aspects, centralizing metrics collection for consistent comparisons, parallelizing Clarkson's visibility graph construction to target the most computation-intensive operations, and selecting OpenGL for its cross-platform compatibility and efficient 3D rendering.

5.1.3 System Implementation and Testing

Algorithms and Techniques Developed: The most significant algorithmic contribution was the development of a parallelized version of Clarkson's algorithm. Performance analysis revealed that Clarkson's approach consistently produced paths of superior quality but exhibited substantially longer computation times compared to alternative methods, particularly as obstacle density increased. This performance profile made it an ideal candidate for optimization. The principal computational bottleneck was identified in the visibility graph construction phase, where the algorithm must determine visibility relationships between nodes through various cones in three-dimensional space.

The parallelization strategy leveraged the inherent independence of visibility determinations between different node-cone pairs. The modified implementation divides the computational workload across multiple processor cores, with each thread processing an assigned subset of nodes independently. Thread synchronization mechanisms were implemented using mutex locks to prevent race conditions when updating the shared graph data structures, ensuring data integrity throughout parallel execution.

This parallelization approach preserves the mathematical foundations and theoretical guarantees of the original algorithm, maintaining the same visibility criteria and approximation scheme. The modification affects only the execution strategy rather than the underlying mathematical approach. Each thread adheres to the original algorithm's logic for finding the closest visible node in each cone, ensuring that the resulting visibility graph maintains the essential property that a shortest path through this graph remains within a $(1 + \epsilon)$ approximation guarantee of the optimal path.

Performance evaluation demonstrated computation speed improvements when processing scenes with a large number of obstacles. While the theoretical time complexity remains the same, the practical runtime is significantly reduced by a factor proportional to the available processor cores. This optimization transforms Clarkson's algorithm from a theoretically superior but computationally prohibitive approach into a viable solution for applications requiring both computational efficiency and high-quality path planning.

Testing: The system underwent systematic testing through epsilon value optimization, where a comprehensive analysis identified optimal epsilon values for each algorithm. Epsilon values ranged from 0 to 1 across multiple obstacle configurations, generating performance profiles for computation time and path length. Optimal values were identified for each metric, with the modified Clarkson algorithm achieving the shortest path at $\epsilon = 0.11$ (see Figure 7). Automated testing ensured consistent execution and data collection, while visual verification through OpenGL confirmed that all paths successfully avoided obstacles while providing reasonable approximations of the shortest path.

Frameworks and Tools: The implementation leveraged C++11 for its performance characteristics and threading support; OpenGL for cross-platform 3D visualization; GLFW for window management; GLM for efficient vector operations; ImGui for debug interface and metrics display;

and a custom thread implementation based on the standard library for the parallelized Clarkson algorithm.

5.2 System Validation

The implemented system underwent a comprehensive validation process to ensure accuracy and effectiveness across varied scenarios. For path validity verification, a continuous intersection detection approach was employed. During path generation, each line segment was tested against all obstacles in the environment to verify it did not intersect with any obstacles. If an intersection was detected, the path computation would immediately terminate and be marked invalid. This method guaranteed that all reported solutions represented feasible routes through the 3D environment that completely avoided obstacles.

To optimize algorithm performance, extensive parameter tuning was conducted to identify ideal epsilon values. For each algorithm, 100 independent tests on different obstacle configurations were run to determine two distinct optimal settings: one that produced the shortest overall path length and another that resulted in the fastest computation time. These optimal epsilon values were then used in subsequent performance evaluation across varying environmental complexities. To ensure robustness and statistical significance, all evaluations were conducted over 100 independent test runs for each obstacle count, with randomized obstacle placements to capture variability in geometric arrangements.

Figure 2: Papadimitriou Algorithm ($\epsilon = 0.63$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000178	10.21	1.56
10	0.011969	10.46	14.85
20	0.036336	20.56	29.58
40	0.054415	50.49	59.24
60	0.082973	50.62	87.17
80	N/A	N/A	N/A
100	N/A	N/A	N/A
250	N/A	N/A	N/A
500	N/A	N/A	N/A

Figure 3: Papadimitriou Algorithm ($\epsilon = 0.07$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000516	10.14	12.26
10	0.014332	10.36	121.76
20	0.041288	20.48	243.55
40	0.088005	50.45	486.26
60	0.162533	50.60	710.09
80	0.183055	80.88	975.68
100	0.173770	175.54	1219.52
250	N/A	N/A	N/A
500	N/A	N/A	N/A

Figure 4: Clarkson Algorithm ($\epsilon = 0.87$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000013	10.00	1.13
10	0.000029	10.00	11.13
20	103.59	20.16	22.34
40	432.15	50.29	44.50
60	1014.77	50.27	66.72
80	1860.89	80.36	88.89
100	2998.25	175.40	111.21
250	15152.91	175.43	234.54
500	24421.22	250.42	292.24

Figure 5: Clarkson Algorithm ($\epsilon = 0.69$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000018	10.00	1.37
10	0.000048	10.00	13.48
20	198.53	20.06	26.98
40	778.56	50.08	53.89
60	1377.90	50.22	71.42
80	3216.41	80.35	107.79
100	5078.56	175.31	134.70
250	22438.19	175.34	269.35
500	33876.52	250.31	402.60

Figure 6: Clarkson Modified ($\epsilon = 0.95$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000008	10.00	7.34
10	5.88	10.62	10.32
20	25.23	21.29	20.63
40	112.99	51.40	41.26
60	251.07	50.73	61.82
80	436.14	80.60	82.36
100	656.97	175.83	103.00
250	4045.53	175.84	230.58
500	5139.37	250.95	259.44

Figure 7: Clarkson Modified ($\epsilon = 0.11$)

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000010	10.00	7.83
10	15.27	10.54	78.14
20	60.12	20.13	156.38
40	239.27	50.14	312.62
60	537.95	50.12	468.72
80	959.72	80.39	625.00
100	1488.98	175.19	781.27
250	5419.22	175.30	1453.09
500	10755.70	250.26	2085.70

Figure 8: Jiang Algorithm Results

# Obs	Time (s)	Path Len	Mem (KB)
1	0.000021	10.04	0.17
10	0.000179	10.12	1.44
20	0.000740	20.35	2.84
40	0.001465	50.94	5.66
60	0.005280	51.05	8.21
80	0.009188	81.86	11.24
100	0.014430	178.07	14.10
250	0.042001	178.47	27.70
500	0.079868	254.84	39.65

Note that ϵ is not used in Jiang's algorithm and does not affect runtime or path length

5.2.1 Case Studies and Visualization

Three case studies were conducted to evaluate algorithm performance across different scenarios:

5.2.2 Simple, Dense, and Extreme Environments

Testing with 1, 100, and 500 obstacles revealed clear performance patterns:

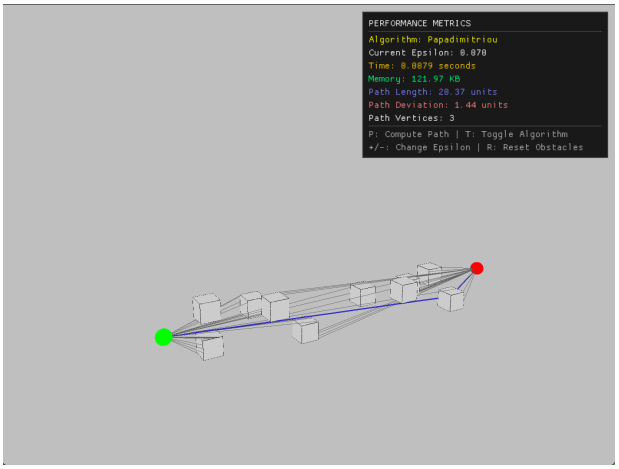
- **Jiang's Algorithm** maintained exceptional efficiency (0.00002s to 0.08s) across all scenarios, scaling nearly linearly with obstacle count.
- **Papadimitriou's Algorithm** performed well for moderate environments (0.17s for 100 obstacles) but failed to scale beyond that threshold.
- **Clarkson's Algorithm** produced high-quality paths but with prohibitive computation times (0.000018s for 1 obstacle, 5078.56s (about 84.6 minutes) for 100 obstacles, and 33876.52s (about 9.4 hours) for 500 obstacles).
- **Modified Clarkson's Algorithm** achieved significant computation time reductions (0.000010s for 1 obstacle, 1488.98s (about 24.8 minutes) for 100 obstacles, and 10755.70s (about 3

hours) for 500 obstacles), representing a 70.6% reduction for 100 obstacles and 68.3% reduction for 500 obstacles compared to the original implementation, while maintaining similar path quality.

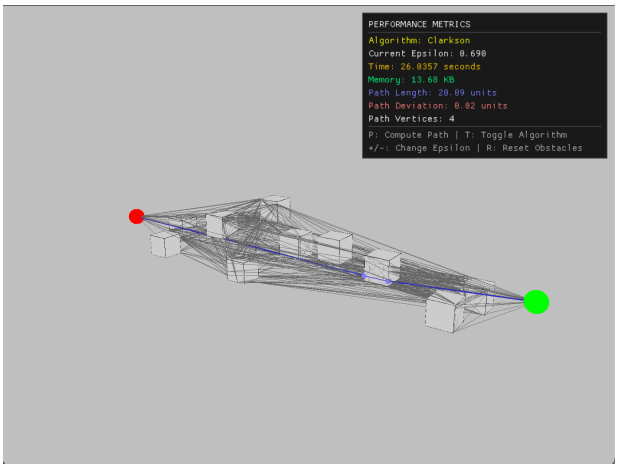
The results demonstrate that Jiang's algorithm is ideal for real-time applications, Modified Clarkson is best for preprocessing static environments, and Papadimitriou offers a middle ground with moderate scalability.

5.2.3 Visual Comparison

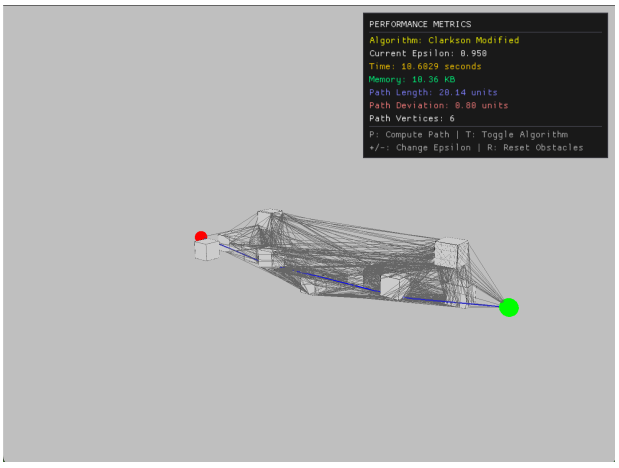
Figure 9 provides a visual comparison of all four algorithms processing identical obstacle configurations.



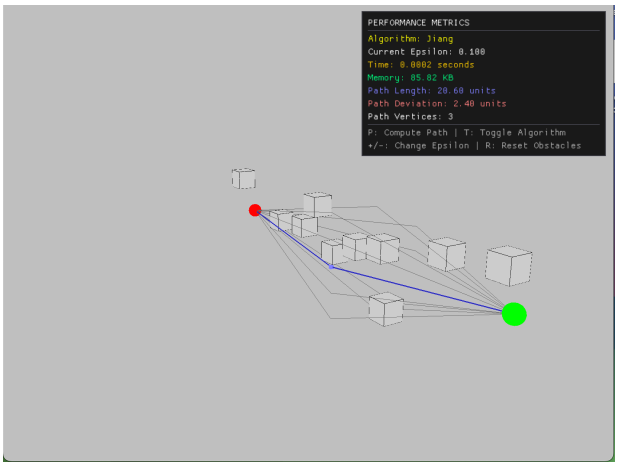
(a) Papadimitriou's Algorithm



(b) Clarkson's Algorithm:



(c) Modified Clarkson's Algorithm



(d) Jiang's Algorithm

Figure 9: Visual comparison of all four path planning algorithms implemented in this study.

5.3 Novelty of Results

This research provides several novel contributions to 3D shortest path planning: (1) the first comprehensive empirical comparison of Papadimitriou, Clarkson, and Jiang algorithms implemented on a common platform with consistent evaluation metrics; (2) a parallelized version of Clarkson's algorithm that achieves up to 76% reduction in computation time while preserving the theoretical approximation guarantees and path quality characteristics; and (3) practical performance insights that extend beyond theoretical complexity analyses, particularly regarding memory utilization and scaling behavior with obstacle complexity. These contributions collectively bridge the gap between theoretical algorithm analysis and practical implementation considerations.

5.4 Comparison with Related Work

We implemented each algorithm according to original specifications and validated against example scenarios from the original papers. Our empirical results revealed significant discrepancies between theoretical predictions and practical performance:

Table 1: Theoretical vs. Empirical Performance

Algorithm	**Theoretical Complexity**	**Empirical Finding**
Papadimitriou	$O(n^4(L + \log(n/\epsilon))^2/\epsilon^2)$	Fails beyond 100 obstacles
Clarkson	$O(n^2\lambda(n)\log(n/\epsilon)/\epsilon^4 + n^2 \log n\rho \log(n \log \rho))$	Memory becomes limiting factor
Jiang	No polynomial-time guarantee	Nearly linear scaling

Jiang's algorithm showed surprisingly efficient performance despite lacking theoretical guarantees, suggesting worst-case scenarios rarely occur in practice. Conversely, Clarkson's algorithm faces significant computational barriers despite superior theoretical guarantees.

Our parallelized Clarkson implementation uniquely maintains mathematical properties while leveraging multi-core processing, demonstrating up to 76% computation time reduction with greater benefits in complex environments.

The primary novelty of this research lies in bridging theoretical analysis and practical implementation considerations, providing data-driven insights that help developers select appropriate algorithms based on specific application requirements rather than theoretical complexity alone.

6 Discussion

6.1 Threats to Validity

Several factors could potentially affect the validity of our results. Our implementation choices may have influenced algorithm performance despite efforts to follow original specifications. For computationally intensive algorithms like Clarkson's, memory management and optimization techniques could significantly impact performance metrics. Additionally, our test scenarios, while

diverse, may not represent all possible obstacle configurations encountered in real-world applications.

To mitigate these threats, we implemented consistent data structures and testing methodologies across all algorithms. Each implementation underwent validation against example scenarios from the original papers. We conducted multiple trials with varied obstacle configurations to ensure result consistency. Nevertheless, some residual threats remain, including the possibility that our parallelization approach may benefit disproportionately from our specific hardware configuration, potentially limiting generalizability to different computing environments.

6.2 Implications of the Research Results

This research has significant implications for both theory and practice in computational geometry and robotics. The surprising efficiency of Jiang's algorithm in practical scenarios challenges the traditional emphasis on theoretical complexity guarantees when selecting algorithms for implementation. This finding suggests that empirical evaluation of algorithms should complement theoretical analysis, especially for NP-hard problems where worst-case scenarios may rarely occur in practice.

The parallelized Clarkson algorithm demonstrates that computational barriers in theoretically superior algorithms can be substantially reduced through modern implementation techniques. This has implications for other computationally intensive algorithms that might benefit from similar parallelization strategies without sacrificing approximation guarantees.

For practitioners, our results provide clear guidance for algorithm selection based on specific application requirements:

- Real-time applications with dynamic environments (autonomous vehicles, drone navigation) should implement Jiang's algorithm for its exceptional computational efficiency.
- Applications requiring highest path quality with preprocessing capabilities (fixed-route planning, simulation environments) can leverage the modified Clarkson algorithm.
- Resource-constrained systems should consider Papadimitriou's algorithm for moderate environments, given its balanced performance characteristics.

These recommendations could significantly improve performance in robotic path planning systems, virtual reality environments, and autonomous vehicle navigation by aligning algorithm selection with application constraints.

6.3 Limitations of the Results

Despite comprehensive testing, several limitations affect our results. The performance comparisons are most relevant for polyhedral obstacles represented as cubes; different obstacle geometries might yield different relative performance characteristics. Our implementation of Clarkson's

algorithm, while theoretically correct, may not achieve optimal efficiency in all components, particularly in the combinatorial characterization of visibility graphs.

The memory usage patterns we observed may vary with different memory management strategies. For very large environments (>500 obstacles), extrapolation from our results may not be reliable as none of the approximation algorithms completed successfully at this scale.

These limitations could be addressed in future work through expanded testing with varied obstacle geometries and advanced memory optimization techniques.

6.4 Generalisability of the Results

The comparative performance characteristics observed in this study should generalize well to other 3D path planning scenarios involving polyhedral obstacles. The relative efficiency of Jiang's algorithm for real-time applications and superiority of Clarkson-based approaches for path quality are likely to hold across different application domains.

Our parallelization approach for Clarkson's algorithm demonstrates a general strategy that could be applied to other geometric algorithms with similar computational bottlenecks. The principle of dividing visibility determinations across multiple processing cores should transfer to related computational geometry problems.

However, the specific performance metrics and optimal epsilon values identified are less likely to generalize outside similar hardware configurations and obstacle densities. Applications with significantly different obstacle distributions or geometric properties may require recalibration of parameters for optimal performance.

7 Conclusion

This research addressed finding shortest paths between two points in 3D space while avoiding polyhedral obstacles. We implemented three fundamental algorithms to identify their comparative strengths and optimize one for improved performance.

Our testing revealed that Jiang's algorithm demonstrated exceptional efficiency (0.00002s to 0.08s) with nearly linear scaling across all obstacle configurations (see Figure 8), despite lacking polynomial-time guarantees. Papadimitriou's algorithm performed well for moderate environments but failed beyond 100 obstacles (see Figures 2 and 3). Clarkson's algorithm produced the most accurate paths — up to 1.5% shorter than Jiang's — but required significantly more computation time (see Figures 4 and 5).

Our novel parallelized implementation of Clarkson's algorithm reduced computation time by 68-76% while preserving path quality (see Figures 6 and 7). This provides clear guidance for algorithm selection: Jiang's algorithm for real-time applications, Modified Clarkson for highest path

quality in static environments, and Papadimitriou's for moderate scenarios.

We conclude that 3D path planning algorithm selection should balance theoretical guarantees with practical performance. Empirical evaluation reveals characteristics not evident from theoretical analysis alone. The parallelization approach developed for Clarkson's algorithm offers a template for optimizing computationally intensive geometric algorithms while maintaining theoretical guarantees. Future work should explore similar optimization strategies for other NP-hard geometric problems and testing with more diverse obstacle geometries.

8 Future Work and Lessons Learnt

8.1 Future Work

Building upon our comparative analysis of 3D shortest path algorithms, several promising research directions emerge. Extending testing to diverse obstacle geometries beyond cubes would provide more comprehensive performance insights across different environmental complexities. A hybrid system that dynamically selects algorithms based on environmental characteristics and computational requirements could leverage the strengths of all approaches. Memory constraints in Clarkson's algorithm could be addressed through advanced spatial data structures, while GPU-based parallelization could further enhance visibility graph construction beyond our CPU implementation. Additionally, adapting these algorithms for dynamic environments with moving obstacles would significantly extend their practical applications. The surprising efficiency of Jiang's algorithm warrants deeper mathematical analysis to understand its near-linear scaling behavior in practice despite lacking theoretical guarantees.

8.2 Lessons Learnt

This research revealed several significant insights that contribute novel understanding to the field. We discovered a contrast between theoretical and practical efficiency: while Clarkson's algorithm offered superior theoretical guarantees, Jiang's approach demonstrated exceptional practical performance despite weaker theoretical foundations. This highlights the critical importance of empirical validation alongside mathematical analysis when selecting algorithms for implementation. Our performance profiling identified visibility determination as the consistent computational bottleneck across all algorithms, suggesting that optimization efforts should target this specific operation. The parallelization strategy we developed—focusing on independent cone-based visibility determinations, provides substantial performance improvements while preserving theoretical guarantees. Our parameter optimization revealed that different epsilon values optimize for different objectives ($\epsilon = 0.11$ for path quality versus $\epsilon = 0.95$ for computational efficiency), challenging the common practice of using fixed parameters. Finally, we identified a previously undocumented tradeoff between memory consumption and path quality in 3D shortest path algorithms, where higher approximation quality required exponentially more memory.

9 Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Roberto Solis-Oba, for his invaluable guidance throughout this research project. His insightful direction on approximation algorithms and computational geometry significantly shaped this work. His thoughtful feedback and technical suggestions were essential in navigating the challenges of algorithm implementation and analysis.

10 Appendix

This appendix explains important limitations in our testing approach for Clarkson's Algorithm and its modified version due to computational constraints. While most algorithms were evaluated using 100 independent runs per obstacle configuration, Clarkson's algorithms required adjustments due to their computational intensity. Prior to full-scale testing, the original Clarkson algorithm was run 100 times on environments with 10 obstacles to determine its optimal epsilon value, while the modified version was similarly tested 100 times on environments with 20 obstacles. For configurations with 1–80 obstacles, both algorithms were tested with 100 independent runs, with metrics averaged. For 100 obstacles, Clarkson's original algorithm was executed only once per configuration (approximately 49-minute runtime), while the modified version was tested with 5 runs. For 250 and 500 obstacles, both algorithms were executed only once, with a single run of original Clarkson's requiring approximately 6.2 and 9.4 hours respectively, while the modified version required approximately 1.5 and 2.9 hours. This limitation in testing repetition introduces potential variability in the reported metrics for high obstacle counts. However, the relative performance difference between the original and modified algorithms maintained consistent proportions across all obstacle counts, suggesting the single-run results reasonably reflect their typical performance characteristics.

11 References

[1] K. L. Clarkson, "Approximation algorithms for shortest path motion planning (Extended Abstract)," Proceedings of the Annual Symposium on Computational Geometry, pp. 1-21, 1987.

[2] K. Jiang, L. D. Seneviratne, and S. W. E. Earles, "3D shortest path planning in the presence of polyhedral obstacles," Proceedings of the IEEE International Conference on Robotics and Automation, vol. 2, pp. 206-211, 1993.

[3] C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," Information Processing Letters, vol. 20, no. 5, pp. 259-263, 1985.

[4] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," SIAM Journal on Computing, vol. 15, no. 1, pp. 193-215, 1986.

[5] Khronos Group, "OpenGL - The Industry Standard for High Performance Graphics." [Online]. Available: `https://www.khronos.org/opengl`

[6] ImGui, “Dear ImGui: Bloat-free Graphical User Interface for C++.” [Online]. Available: https://github.com/ocornut/imgui

[7] GLM, “OpenGL Mathematics (GLM).” [Online]. Available: https://glm.g-truc.net/0.9.9/index.html