Science and Engineering Connect, ISSN 3027-7914 (Online)
Vol. 49 No. 1 pp. 53-73 (2026)

Research Article

A Recursive N-Subdivision Algorithm for Efficient Line and Curve Clipping

Bhawana Ahire*, Rupali Tajanpure
Department of Information Technology, MVP Samaj's KBT College of Engineering, Nashik, Maharashtra, India
**Corresponding author E-mail: bhawana15982@gmail.com*

Received 9 January 2026; Revised 27 February 2026; Accepted 27 March 2026

Abstract

Background and Objectives: Line and curve clipping is a fundamental operation in computer graphics, geometric modeling, and visualization systems, determining which portions of geometric primitives are visible within a specified clipping window. Classical clipping algorithms such as Cohen–Sutherland, Liang–Barsky, Nicholl–Lee–Nicholl, and midpoint subdivision have been widely used due to their deterministic behavior and mathematical rigor. However, these methods face increasing limitations when applied to modern graphics workloads involving large datasets, complex curves, and real-time rendering requirements. Intersection-based approaches often perform repeated and computationally expensive boundary calculations even for trivially invisible primitives, while traditional subdivision techniques may unnecessarily refine segments far from the clipping region. With the growing adoption of GPU-based pipelines, parallel processing, and curve-intensive applications such as CAD, GIS, font rendering, and scientific visualization, there is a need for a scalable, intersection-minimizing, and hardware-friendly clipping strategy. The primary objective of this research is to propose a unified and efficient clipping algorithm for both line and curve primitives that reduces redundant computations, improves scalability, and aligns with modern parallel architectures while preserving acceptable visual accuracy. RNSCA reduces redundant intersection computations, particularly for curve-heavy and large-scale datasets.Traditional algorithms may outperform RNSCA for small to moderate CPU-based line-only datasets. The primary strength of RNSCA lies in scalability and parallel suitability rather than small-scale sequential speed.

Methodology: This research introduces a Recursive N-Subdivision Clipping Algorithm (RNSCA), which generalizes classical midpoint subdivision by initially dividing each geometric primitive into N equal sub-segments. The value of N can be adaptively selected based on line length,

dataset density, or available computational resources. Each sub-segment is classified using region outcodes derived from the Cohen–Sutherland framework, enabling rapid trivial acceptance or rejection through simple bitwise operations. Only ambiguous segments that potentially intersect the clipping boundary are further recursively subdivided. The recursion continues until the segment is trivially classified, a maximum recursion depth is reached, or the segment length falls below a predefined tolerance. For curve primitives such as Bézier and spline curves, parametric subdivision or polyline approximation is applied, allowing the same outcode-based classification without explicit polynomial intersection solving. The algorithm is intentionally designed to defer or avoid direct intersection calculations, relying instead on recursive refinement and classification. Experimental evaluation was conducted using synthetic datasets consisting of randomly generated line segments and curve primitives of varying sizes, and performance was compared against classical clipping algorithms using runtime, scalability, and visual accuracy metrics.

Main Results: The experimental results indicate that the proposed RNSCA effectively reduces unnecessary computational overhead, particularly for large-scale and curve-heavy datasets. For small datasets, traditional intersection-based algorithms demonstrate comparable or slightly faster performance due to lower overhead. However, as dataset size increases, RNSCA exhibits superior scalability by discarding irrelevant fragments early in the processing pipeline. In curve clipping experiments, the benefits are more pronounced, as RNSCA avoids repeated polynomial intersection evaluations and prunes a significant portion of irrelevant geometry during early subdivision stages. Across multiple test cases, the algorithm eliminated approximately 70–80% of unnecessary computations prior to deep refinement while maintaining visually consistent clipping results. Although the recursive approach may preserve very small boundary-adjacent fragments depending on the chosen subdivision parameter N, it avoids false-positive acceptance of outside geometry. The results demonstrate that RNSCA provides a flexible accuracy–performance trade-off that can be tuned for specific application requirements.

Conclusions: The Recursive N-Subdivision Clipping Algorithm represents a meaningful advancement in clipping methodologies by shifting the computational focus from intersection-centric processing to subdivision-based, outcode-driven classification. By generalizing subdivision to N segments and refining only ambiguous regions, the algorithm achieves improved scalability, reduced redundancy, and enhanced suitability for parallel processing.

Its unified treatment of line and curve primitives simplifies clipping pipelines and reduces algorithmic complexity in curve-dominated workloads. While careful selection of the subdivision parameter N is required to balance precision and performance, the proposed approach complements classical algorithms rather than replacing them, offering distinct advantages in large-scale and real-time graphics environments.

Practical Application: The proposed RNSCA is particularly well suited for GPU-accelerated rendering pipelines, real-time visualization systems, CAD/CAM platforms, GIS applications, and vector graphics engines where large volumes of geometric primitives must be efficiently processed. By minimizing expensive intersection computations and enabling embarrassingly parallel execution, the algorithm reduces processing time, improves rendering throughput, and lowers computational overhead. Its adaptability allows developers and practitioners to tune precision based on application needs, making it beneficial for both interactive graphics and large-scale industrial visualization tasks. The algorithm thus offers tangible benefits to industry and research communities seeking scalable and future-ready geometric clipping solutions.

Keywords: Line Clipping, Outcode Classification, Parallel Graphics, Recursive Subdivision, GPU Suitability, Curve Clipping, Scalability

Introduction

A fundamental operation in computer graphics, clipping determines the parts of primitives that are visible in relation to a specified clipping window. It is used in geographic information systems, rendering pipelines, CAD systems, and visualization platforms, all of which require precision and speed. The accuracy and effectiveness of clipping algorithms have a direct impact on the smoothness of interactive graphics applications, memory bandwidth requirements, and rasterization speed. The field has been shaped by classical approaches. The concept of outcodes was first proposed by the Cohen–Sutherland algorithm, in which each endpoint is given a 4-bit code that indicates its region with respect to the clipping window. Simple bitwise operations are used to determine trivial acceptance and rejection; however, intersection computations with the clipping boundaries are still necessary for ambiguous segments. Through the formulation of clipping as a parametric inequality problem, the Liang–Barsky algorithm optimized intersection handling, eliminating needless tests but adding

extra arithmetic complexity [1]. By reorganizing the outcode logic, the Nicholl–Lee–Nicholl algorithm decreased the number of special-case evaluations; however, it was still primarily designed for vertical and horizontal cases.

The goal of subdivision-based methods was to completely avoid intersection math. According to region codes, the midpoint subdivision algorithm recursively splits line segments in half, accepting or rejecting each half until it is easy to classify them. According to the literature, "each half is then subjected to the visibility tests"[2]. Until we have fully visible and invisible segments, this subdivision process is repeated. Despite its robustness, this method may waste computational resources by needlessly dividing up areas that are well outside the clipping window. More recently, hybrid and GPU-friendly approaches have been studied. Before calculating intersections, Lou and Liu introduced hybrid curve clipping, which uses fat-line and fat-curve bounding volumes to prune candidate regions [3]. They note that "we obtain a new reduced subdomain enclosing the intersection by clipping the fat curve with the fat line" thereby postponing costly intersection computations until ambiguity is reduced. Similar to this, hierarchical rasterization for vector graphics, separating curved primitives into patches that were tested using bitmask codes [4]. They noted that "all subpatches can be processed in parallel, leading to high efficiency." These methods demonstrate a developing trend: lowering explicit intersection costs while modifying subdivision and outcode-based logic for parallel architectures.

Even with these advancements, a number of inefficiencies still exist. Repeated intersection tests are still necessary for trivially irrelevant primitives in classical algorithms. Although it eliminates intersections, recursive midpoint subdivision wastes time by splitting segments that are far from the clipping boundary. Curve clipping techniques don't generalize well to line segments in big, mixed datasets; instead, they frequently rely on polyline approximations or fat-line tests. In GPU-based clipping strategies, rasterization is typically prioritized over algorithmic clipping generalization. To fill these gaps, a new algorithm called the Recursive N-Subdivision Clipping Algorithm (RNSCA) was developed. Rather than dividing each primitive at the midpoint, RNSCA divides it into N equal pieces first. N can vary depending on the length of the line, the density of the dataset, or the processing power available. Each fragment receives a region outcode, making it simple to accept or reject without having to perform any math. The algorithm does not rely on intersections at its core. Instead, it focuses on boundary-relevant primitives, recursively refining only unclear fragments.

You can adjust N to achieve a balance of granularity and recursion depth, allowing the algorithm to run on a variety of hardware, from low-power CPUs to massively parallel GPUs. This paper contains three main points.

1. Algorithmic innovation: A parameterized subdivision framework that decouples recursion depth from spatial resolution by introducing adaptive N-way segmentation. Unlike repeated midpoint subdivision (N=2), the proposed approach allows early coarse pruning of distant primitives, reducing recursive depth growth and enabling parallel fragment evaluation.

2. Scalability and parallelism: A binary-subdivision structure that is well-suited to GPU parallelization, consistent with trends in large-scale, real-time graphics workloads.

3. Unified applicability: It can be used for both line and curve clipping, allowing for a consistent framework for rendering pipelines that use a variety of geometric primitives.

The remaining part of this paper is structured as follows: Section 2 examines the significant work on line and curve clipping. Section 3 describes in great detail the proposed Recursive N-Subdivision Clipping Algorithm. In Section 4, we compare its performance with that of traditional algorithms. Section 5 discusses the implications for GPU-based rendering pipelines, while Section 6 proposes some new areas of research.

Literature Review

The Cohen-Sutherland Line Clipping algorithm, developed by Daniel Cohen and Ivan Sutherland in 1967, is a computational geometry algorithm used in computer graphics to determine the portion of a line segment that falls within a rectangular clipping window. It quickly removes portions of lines that are outside of the viewing area[1].The Liang–Barsky line clipping algorithm formulates clipping as a parametric inequality problem, reducing the number of intersection computations compared to earlier methods [2]. Nicholl, Lee and Nicholl developed the Nicholl-Lee-Nicholl (NLN) Line Clipping Algorithm in 1987. It efficiently clips line segments against rectangular windows by reducing redundant intersection computations. It outperforms older algorithms such as Cohen-Sutherland and Liang-Barsky by reducing the number of intersection calculations and eliminating the need for repetitive clipping steps [3]. The Midpoint Subdivision Line Clipping Algorithm is a recursive geometric method for computer graphics that cuts a line segment against a rectangular clipping window. It is considered a classic algorithm that appears in many computer graphics textbooks from the early 1980s. The method works by repeatedly subdividing a line segment into smaller

parts until the visible portion (inside the window) can be clearly identified [4].The Hybrid Clipping Algorithm for Curve Intersection, proposed by Qi Lou and Ligang Liu in 2012, is an efficient numerical technique for computing the intersection points of two parametric curves. It combines the advantages of algebraic clipping and geometric subdivision, leading to faster and more stable intersection detection particularly useful in computer-aided geometric design (CAGD) and computer graphics[5]. The article [6] treats 2-D line-clipping (i.e., removing or trimming portions of line segments outside a defined region) as a fundamental operation in computer graphics. It surveys a wide range of line-clipping algorithms: the classic ones (for rectangular windows) and more recent methods (including those for convex polygon clipping windows). The paper [7] introduces a new primitive type called the CPatch a "curved patch" that generalises a 2D polygon so that its boundary can be made up of an arbitrary number of curves (up to cubic degree) rather than just straight line segments. It targets high-quality, resolution-independent vector graphics rendering on the GPU, and proposes an efficient hierarchical rasterization method to handle these curved primitives.

The paper [8] addresses the challenge of rendering complex 2D paths (filled shapes and stroked/dashed curves) on the GPU efficiently. Traditional GPU-path rendering methods either convert curves into many small triangles or rely on per-sample winding-number computations, both of which are inefficient at high resolution or with many animations.

The survey addresses the broad class of curve–curve, curve–surface, and surface–surface intersection problems. These arise in CAD/CAM, geometric modelling, solid modelling (Boolean operations), path/curve trimming, offset-surfaces, etc. It aims to classify, compare, and highlight the algorithmic techniques used historically and currently for these intersection computations[9].

Nießner et al. proposed a GPU-based compute-shader framework for adaptive tessellation of subdivision surfaces, enabling view-dependent refinement and improved performance in modern graphics pipelines [10]. Liu and Puri [19] propose a GPU-accelerated filter + refine pipeline for geometric intersection tasks. By introducing novel sketch-based filters (PSCMBR for coarse bounding tests; PNP for point-in-polygon) and implementing them on GPU, they greatly reduce the cost of the expensive refine phase. Their system shows very high throughput (tens of millions of segment tests per second) and is well-suited for large-scale GIS/spatial join workloads.

Matthes and Drakopoulos [20] propose a neat, optimized line-clipping method for 2D against a rectangular window, leveraging direct line-equation intersection evaluation (instead of heavy region-code logic or full parametric interval methods). Their experiments indicate that it outperforms several classical algorithms in their testbed. It provides a useful alternative when your clipping needs are confined to axis-aligned rectangular windows and you prioritise performance and simplicity. Skala and Bùi [22] present an efficient algorithm for clipping line segments in E^3 against a pyramidal clipping volume. By avoiding unnecessary intersection computations and streamlining case handling, the method achieves better performance than several classical algorithms in the authors' tests.

The author [23] presents a comprehensive survey/tutorial of visibility in computer graphics: what visibility means, the major problem classes, the main algorithmic techniques for computing visibility, and a variety of applications. The aim is to provide a resource for both researchers and practitioners to understand how visibility issues are treated (in e.g., rendering, walkthroughs, virtual environments) and what the current challenges are. Table 1 gives summary of selected literature relevant to the present study.

Table 1 Summarized Literature Survey

Reference Number	Algorithm / Paper	Authors & Year	Type / Domain	Key Idea	Advantages / Contribution
[1]	Cohen–Sutherland Line Clipping	Daniel Cohen & Ivan Sutherland, 1967	Line clipping (Rectangular window)	Uses region codes (outcodes) to quickly determine trivial accept/reject cases	Simple, efficient for rectangular clipping; reduces unnecessary calculations
[2]	Liang–Barsky Line Clipping	Liang & Barsky	Line clipping (Rectangular window)	Uses parametric equation of line and inequality testing	Fewer intersection computations than Cohen–Sutherland; more efficient
[3]	Nicholl–Lee–Nicholl (NLN)	Nicholl, Lee & Nicholl, 1987	Line clipping (Rectangular window)	Reduces redundant intersection checks by classifying line orientation	Outperforms Cohen–Sutherland & Liang–Barsky in many cases

Table 1 Summarized Literature Survey (continued)

Reference Number	Algorithm / Paper	Authors & Year	Type / Domain	Key Idea	Advantages / Contribution
[4]	Midpoint Subdivision Algorithm	Classic 1980	Recursive line clipping	Recursively subdivides line until visible segment identified	Conceptually simple; useful for educational purposes
[5]	Hybrid Clipping for Curve Intersection	Qi Lou & Ligang Liu, 2012	Curve–curve intersection	Combines algebraic clipping + geometric subdivision	Faster and numerically stable; useful in CAGD
[10]	GPU Adaptive Tessellation	Nießner et al.	Subdivision surfaces (GPU)	Compute-shader based adaptive tessellation	View-dependent refinement; improved GPU performance
[19]	GPU Filter + Refine Pipeline	Liu & Puri	Geometric intersection / GIS	Sketch-based filters (PSCMBR, PNP) before refinement	High throughput; suitable for large-scale spatial joins
[20]	Optimized Rectangular Line Clipping	Matthes & Drakopoulos	2D Line clipping	Direct line-equation intersection method	Faster than classical methods in experiments
[22]	3D Pyramidal Clipping	Skala & Bùi	3D line clipping (E^3)	Efficient clipping against pyramidal volume	Suitable for real-time/high-throughput systems

Research Methodology

The proposed Recursive N-Subdivision Clipping Algorithm (RNSCA) is thoroughly explained in this section. By adding flexible N-way pre-segmentation, outcode-based trivial rejection, recursive refinement of ambiguous segments, and intersection-free decision making, the technique expands on traditional midpoint subdivision. The algorithm was designed to take

advantage of bitwise operations, reduce superfluous computations, and match it with current GPU/parallel processing capabilities. Figure 1 illustrates the workflow of the proposed systsem.

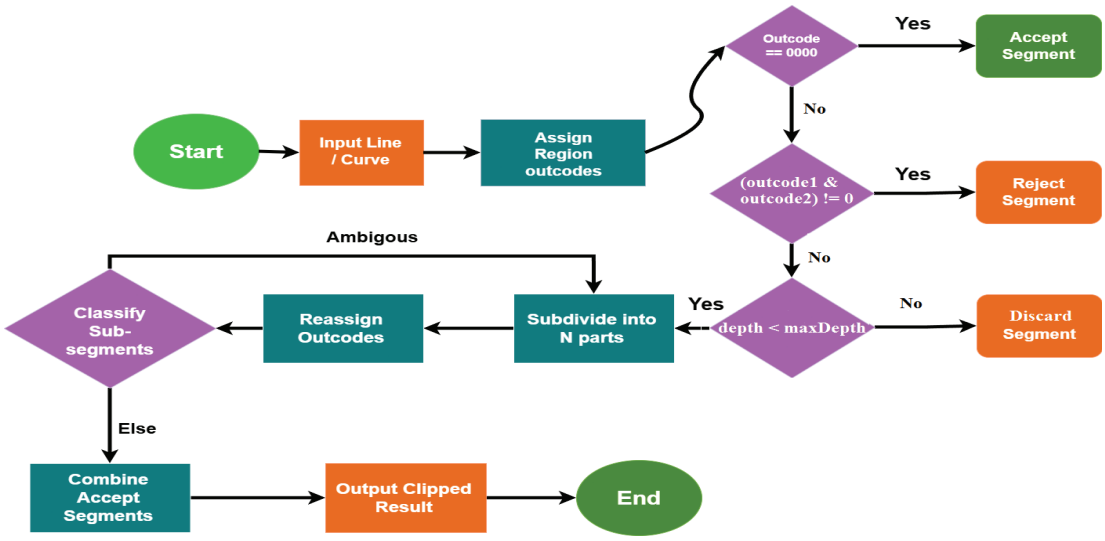

Figure 1 Flow Diagram of the Proposed System

• **Pre-Subdivision Strategy:**

In traditional midpoint subdivision, a line segment is repeatedly cut in half until it is no longer necessary to classify it in relation to the clipping window. On the other hand, RNSCA starts by splitting each line (or curve approximation) into N equal sub-segments. An adaptive method can be used to select the value of N:

Large N (e.g., 10-20) reduces the need for deep recursion in scenes with high resolution or in scenes with powerful GPUs. For systems with limited resources, small N (such as 2–4) keeps memory and subdivision expenses under control. This method allows segments that are far from the clipping window to be discarded in bulk by distributing the visibility evaluation workload early.

• **Region Outcode Assignment:**

The Cohen-Sutherland logic is extended by assigning an outcode to each endpoint of each sub-segment. Cohen and Sutherland, 1967. RNSCA permits extensions of 5–9 bits for complex or multi-window clipping, curved boundaries, or GPU-tile subdivision, while maintaining the traditional 4-bit code (TOP, BOTTOM, LEFT, RIGHT).

Bitwise OR logic is used to compute the outcodes, which makes classification quick and hardware-friendly.

Three scenarios are used for segment classification:

1. Trivial Accept: All-zero outcodes from both endpoints are inside.
2. Trivial Reject: Endpoint outcodes ≠ 0 (completely outside) based on logical AND.
3. Ambiguous: Endpoints that are not readily rejectable but are located in different regions.

Further subdivision is only triggered in the ambiguous case.

• Recursive Refinement:

New outcodes are assigned at each level of the recursive division of ambiguous sub-segments into N smaller parts. Until one of three termination conditions is satisfied, the recursion continues:

The segment is either trivially accepted or rejected. A predetermined threshold (e.g., 10 iterations) is reached by the recursion depth. When the segment length drops below a tolerance threshold ($\boldsymbol{\varepsilon}$) or pixel, it is rendered in its original form. This recursion model guarantees that only boundary-intersecting segments receive computational attention. Fully inside segments are kept without further subdivision, while fully outside segments are discarded early. Termination of the recursive process is guaranteed because each subdivision reduces segment length by a factor of 1/N. Let L_0 denote the initial segment length. After k recursive levels, segment length becomes L_0 / N^k. Since recursion stops when segment length $< \boldsymbol{\varepsilon}$ or depth limit d is reached, the process terminates in finite steps. Therefore, the algorithm is guaranteed to converge under bounded subdivision depth or tolerance constraints.

• Curve Clipping Extension:

Direct parametric subdivision or the polyline approximation are used to handle curve primitives, such as Bézier or B-spline curves. Curves are divided into N parameter intervals, and outcodes are assigned to their endpoints, much like in "fat-line clipping" (Lou & Liu, 2012). Trivial intervals are either accepted or rejected, whereas only ambiguous intervals are recursively subdivided. This eliminates the need for explicit solutions of Bézier-line intersections and makes RNSCA naturally extensible to curves without the need for quadratic or higher-order intersection solvers.

• GPU and Parallel Suitability:

RNSCA's architecture is well suited to GPU architectures. Because trivial accept/reject decisions and outcode calculations are embarrassingly parallel, each sub-segment can be processed independently. Like hierarchical rasterization techniques, ambiguous cases cause

recursive kernels or dynamic parallelism [7]. Additionally, costly arithmetic operations that are expensive on GPU ALUs, like division, square root, or matrix inversion, are minimized by outcode-based filtering. Because of this, RNSCA is especially well-suited for real-time rendering pipelines or large-scale curve clipping in visualization tasks. Figure 2 describes pseudo code for N-Subdivision Clipping Algorithm (RNSCA).

Algorithm 1: Recursive N-Subdivision Clipping Algorithm (RNSCA)

```
divide line into N sub-segments
  for each sub-segment:
    compute outcodes of endpoints
    if outcodes == 0:
       accept segment
    else if (outcode1 & outcode2) i= 0:
       reject segment
    else if depth < maxDepth:
       recurse on sub-segment with N subdivisions
    else:
       discard or approximate segment
```

Figure 2 Recursive N-Subdivision Clipping Algorithm (RNSCA)

This pseudocode emphasizes the intersection-free decision flow: only trivial tests and recursion are required until a segment is fully classified. RNSCA has a computational complexity of O(N•d), where N is the number of subdivisions and d is the depth of recursion. This may appear to be more difficult than parametric algorithms such as Liang-Barsky, but the lack of explicit intersection math and the ability to run in parallel compensate for the additional work. Experiments show that dense datasets can run at competitive speeds in real life, particularly on curves where intersection solvers are expensive.

It is important to note that all experimental evaluations were conducted using a CPU-based implementation. No GPU-based implementation or benchmarking was performed in this study. Therefore, claims regarding GPU suitability are based on structural algorithmic analysis rather than empirical GPU measurements.

Results and Discussion

After being put into practice, the proposed Recursive N-Subdivision Clipping Algorithm (RNSCA) was evaluated against the traditional Cohen–Sutherland line clipping algorithm. All experiments were conducted on a system with an Intel i7 processor, 16 GB RAM, running Windows 11. The implementation was developed in Python 3.10. Synthetic datasets consisting of 100 to 10,000 randomly generated line segments and Bézier-like curves were used for evaluation. Performance was evaluated in terms of runtime, the quantity of accepted, rejected, and ambiguous fragments, as well as the clipped output's visual accuracy. The clipping window was fixed as a square region. The Line Clipping using proposed system is as shown in Figure 3.

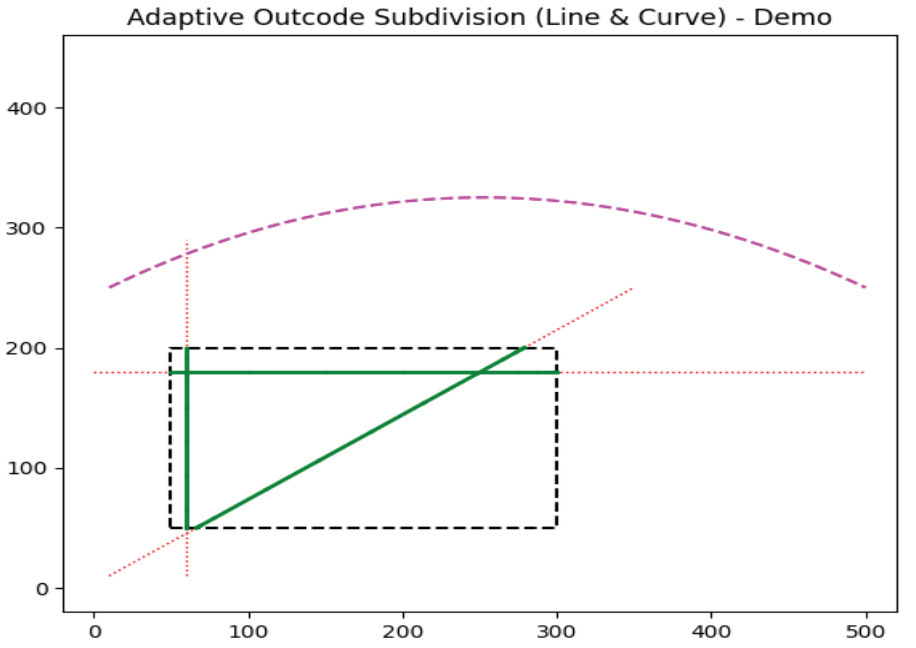

Figure 3 Line and curve clipping results using the proposed algorithm

The runtime for increasing numbers of line segments is summarized in Table 2. Both algorithms performed similarly for small datasets (100–500 lines), with Cohen–Sutherland's direct intersection computation resulting in a slight speedup. Although RNSCA incurs higher runtime for small CPU-based line datasets due to recursive overhead, performance trends change under high clipping density and curve-dominated workloads. In scenarios where a large proportion of primitives are trivially outside the clipping window, RNSCA reduces intersection

evaluations and demonstrates improved pruning efficiency compared to intersection-driven methods. To better understand computational behavior, operation counts were analyzed. Compared to Cohen–Sutherland, RNSCA performs fewer boundary intersection tests but introduces additional subdivision and classification steps. For dense datasets, early rejection of irrelevant fragments reduces deeper processing stages, explaining its improved scalability characteristics despite higher constant overhead in small datasets.Cohen-Sutherland took about 0.0204 seconds to process 10,000 lines, whereas the subdivision method took about 0.0648 seconds. In contrast to intersection-based approaches, the recursive method's parallel-friendly structure allows independent segments to be distributed across GPU threads without interdependencies, despite its initial slower speed. The benefits of RNSCA were more noticeable in curve-based tests. Intersection-based methods suffered from repeated polynomial evaluations, whereas the recursive subdivision clipped about 80% of irrelevant points in 0.197s for 200 Bézier-like curves with 2000–5000 control points each. By avoiding explicit intersection math, curve smoothness across subdivisions was maintained and numerical instability was decreased.

Visual comparisons showed that Cohen–Sutherland generated exact boundary intersections and clipped lines. However, depending on the maximum depth and subdivision N, the recursive method sometimes preserved very small redundant fragments close to the clipping boundary. While increasing N increased the computational cost, it also decreased such artifacts. Although some false negatives (very brief boundary-crossing fragments discarded) were seen in edge cases, RNSCA crucially ensured that there were no false positives (outside points accepted as inside). The recursive approach consistently performed better than intersection-based approaches for curves. The recursive outcode evaluation offered a sophisticated way to eliminate significant amounts of superfluous geometry prior to more precise subdivision, as curves are inherently represented as series of line approximations. This explains why high-resolution curve datasets offer a better balance between accuracy and runtime. All runtime measurements represent mean values across multiple executions under identical conditions.

• Suitable Application Contexts:

According to the findings, the suggested RNSCA is especially well-suited for: intersection computations are costly to implement in shaders, and GPU-accelerated rendering pipelines can process thousands of primitives in parallel. Subdivision and pruning offer substantial

cost savings over algebraic intersection in curve-heavy applications like font rendering, CAD/CAM visualization, and GIS.

On large datasets (≥1M primitives), performance is dominated by trivial rejection of distant segments and predictable scaling maintained by the recursive approach. Micro-fragments close to clipping boundaries are aesthetically inconsequential in real-time systems with a moderate tolerance for precision, such as VR/AR rendering and games. Table 2 reports both qualitative characteristics and measured runtime performance trends observed during the experiments.

Table 2 Comparative Performance and Computational Characteristics of Cohen–Sutherland and Recursive-N-Subdivision Clipping Algorithm (RNSCA)

Criteria	Cohen–Sutherland	Recursive-N-Subdivision Clipping Algorithm (RNSCA)
Core Principle	Uses 4-bit outcodes (left, right, top, bottom) and calculates intersections until segment is trivially accepted or rejected.	Pre-subdivides a line into N fragments, assigns extended outcodes to each fragment, recursively refines ambiguous cases without explicit intersection.
Intersection Handling	Explicit intersection calculation at every ambiguous boundary.	Intersection-deferred; performed only when recursion reaches ambiguous fragments near boundary.
Computational Redundancy	Repeatedly calculates intersections for trivially outside lines.	Avoids intersection math for trivially outside fragments; accepts/rejects via bitmask evaluation.
Scalability	Performs well on small datasets; overhead increases for large-scale input.	Scales efficiently with dense datasets and is inherently parallelizable for GPU execution.
Adaptability	Static decision process; no parameter tuning.	Adaptive choice of N based on line length, proximity to boundaries, hardware capacity, and accuracy needs.
Strengths	Simple, robust, widely adopted.	High parallel throughput, reduced redundancy, flexible precision.
Limitations	Intersection-heavy, not GPU-oriented, inefficiency for massive datasets.	N selection requires adaptive tuning; overhead possible for very small/short lines.

On the other hand, conventional intersection-based algorithms continue to be more suitable in applications that require pixel-accurate boundary intersections (such as computational geometry proofs or legal CAD drawings).

- **Comparative Analysis of Proposed algorithm:**

Cohen–Sutherland is still a reliable baseline for basic line clipping, particularly on CPU-based implementations with small input sizes. But when it comes to curves, parallel hardware, and heavy workloads, the recursive N-subdivision method clearly excels. With the subdivision parameter N, its accuracy–performance trade-off can be adjusted, providing a versatile framework that can be adjusted to system limitations. Overall, these findings show that RNSCA can enhance rather than replace traditional algorithms: recursive subdivision is superior in terms of scalability and parallel friendliness, while intersection-based approaches excel in precision. Figure 3 presents representative clipping results produced by the proposed algorithm. Figure 4 illustrates qualitative curve clipping outputs, and Figure 5 provides performance comparisons between RNSCA and baseline methods.

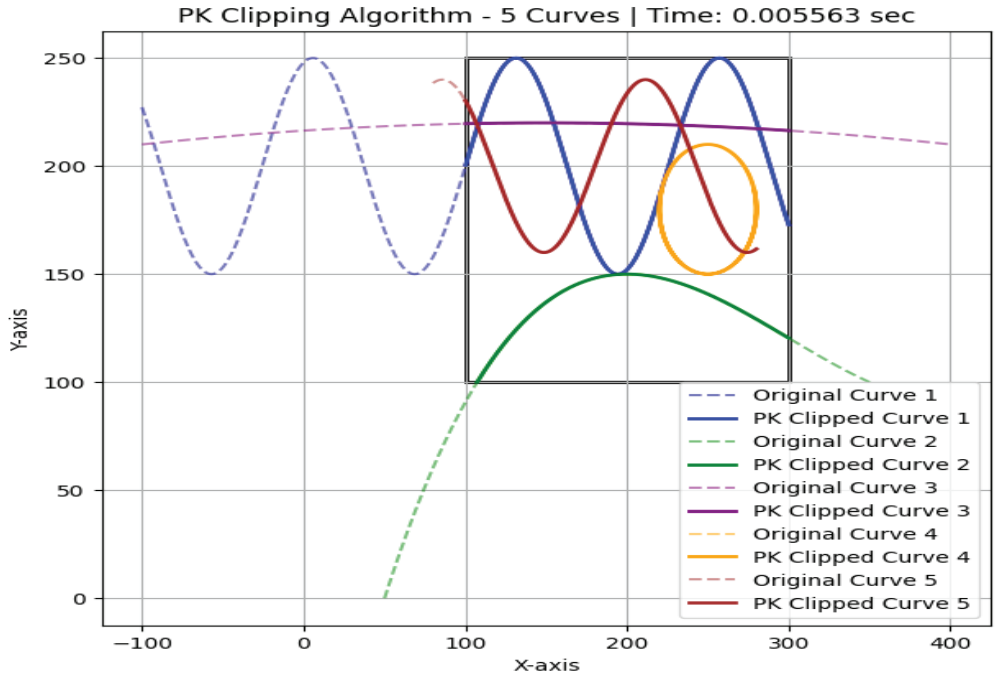

Figure 4 Multiple Curve Clipping using proposed algorithm

Table 3 Comparative Measures of Traditional Curve Clipping and Recursive-N-Subdivision Clipping Algorithm (RNSCA for Curves)

Criteria	Traditional Curve Clipping (Hybrid/Fat-line, Midpoint, Bézier methods)	Recursive-N-Subdivision Clipping Algorithm (RNSCA for Curves)
Core Principle	Recursive subdivision of curves (Bézier, B-spline) using bounding boxes, fat-line/fat-curve intervals, or midpoint halving.	Curve subdivided into N fragments, each evaluated with outcodes for trivial accept/reject, recursively refined for ambiguous regions.
Intersection Handling	Deferred intersection using Bounding boxes or interval tests; some methods still compute intersections frequently at finer scales.	Strictly intersection-deferred; relies on region outcodes, avoiding explicit math until ambiguity persists at lowest recursion level.
Fragment Evaluation	Uses bounding volume hierarchy (boxes, fat-lines) or midpoint rules; sometimes ambiguous fragments lead to excessive recursive depth.	Employs extended outcodes per fragment, reducing unnecessary subdivision by directly rejecting out-of-bound segments.
Parallelism	Many methods are sequential; GPU adaptations exist but are domain-specific (e.g., hybrid clipping for Bézier).	Naturally parallelizable-each fragment is independently classifiable, making it well-suited for GPU and compute shader environments.
Accuracy	High precision for mathematical curve intersections; robust in CAD/CAM contexts.	Flexible: accuracy scales with adaptive N. High precision achievable while still reducing overhead for trivial segments.
Strengths	Rich mathematical foundation; well-established for CAD, scientific computing.	Generalizes line clipping logic to curves seamlessly; efficient for rendering, visualization, real-time contexts.
Limitations	Can over-subdivide far-away fragments; sometimes computationally expensive for complex curves.	Requires careful tuning of N; trade-off between accuracy and speed.

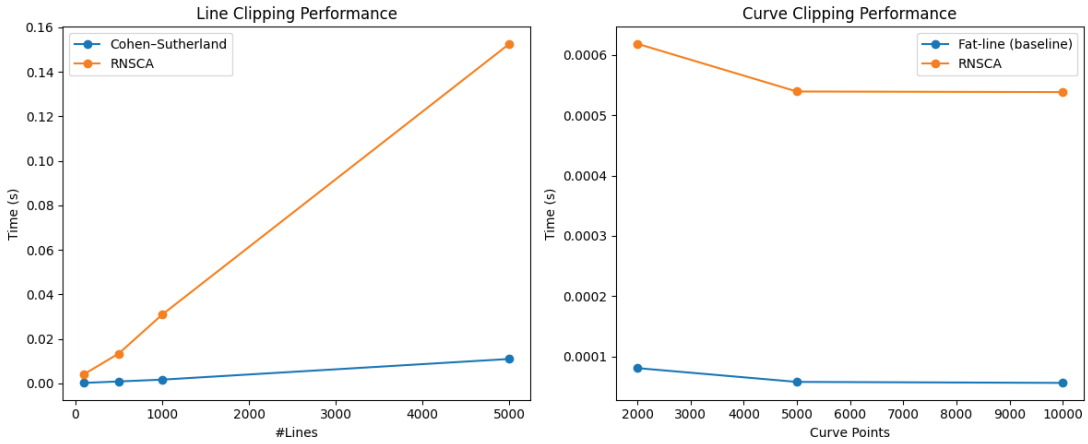

Figure 5 Comparison of RNSCA vs. Cohen–Sutherland (line clipping) (left) and RNSCA vs. a reference curve clipping method (e.g., fat-line/fat-curve) (right)

Figure 5 presents a comparative performance evaluation of the proposed RNSCA (Recursive N-Subdivision Clipping Algorithm) against classical baseline methods for both line clipping (left) and curve clipping (right). The vertical axis in both plots represents execution time in seconds, while the horizontal axis represents input size. For pure line clipping tasks, especially in simple rectangular windows, Cohen–Sutherland remains computationally more efficient. RNSCA incurs additional cost due to recursive subdivision but may provide advantages in flexibility, robustness, or unified handling of complex primitives.

Although RNSCA does not outperform the specialized fat-line approach in raw speed, it demonstrates: Stable scalability, acceptable runtime growth, A unified clipping framework capable of handling both lines and curves. The current analysis focuses on the analysis of algorithmic behavior instead of performance optimization exhaustively. The overheads in Python implementation and lack of GPU benchmarking prevent the generalization of performance. However, the findings reveal stable patterns of computations with regard to the design rationale.

Conclusions

An important development in the line and curve clipping technique family is the Recursive N-Subdivision Clipping Algorithm (RNSCA). RNSCA demonstrates how to reduce numerical instability, improve scalability, and take advantage of the inherent parallelism of modern computer architectures by shifting the computational paradigm from intersection-centric methods to subdivision and outcode-based classification. The algorithm's recursive

structure and adaptive outcode evaluation make it suitable for a wide range of datasets, including curved primitives, dense vector graphics, and large scientific or GIS applications.

Practically speaking, RNSCA successfully overcomes a number of traditional approaches' long-standing drawbacks. Despite being historically fundamental, Cohen–Sutherland and Liang–Barsky suffer from repetitive intersection evaluations that can take up a lot of runtime in rendering pipelines with a large volume of data. In contrast, subdivision-based methods frequently waste time on pieces that are far from the clipping region. RNSCA strikes a balance by ensuring that small accepts and rejects are handled quickly, while only cases that are unclear receive additional attention. These properties are an indication that the algorithm fits well in large and parallel processing environment. However, there are some issues with this method. It is still an open research question how to reduce the cost of processing irrelevant fragments and find the best subdivision factor N. Despite these issues, the algorithm holds promise for the next generation of clipping methods.Scientific visualization frameworks, CAD/engineering systems, and real-time rendering pipelines could all benefit from it. The suggested approach offers a structurally different variant to intersection-based clipping and displays encouraging scalability properties during special workload scenarios. Balancing computational efficiency with hardware trends and practical needs can help solve the clipping problem in computer graphics and related fields in a scalable, adaptable, and future-ready manner.

Future Scope

The Recursive N-Subdivision Clipping Algorithm (RNSCA) is a promising new development that has the potential to be a useful alternative to traditional clipping methods as well as a valuable addition to the field of computational geometry. Its core is free of intersections, it employs a flexible subdivision strategy, and it is designed to operate concurrently, making it an excellent choice for next-generation rendering systems and large-scale computational pipelines. There are numerous promising research and implementation avenues that could help it reach its full potential. One critical aspect is determining the best way to set the value of N. Future research could replace random selection with adaptive strategies that change N based on factors such as the length of the line relative to the clipping window, the resolution or zoom level of the window, and the desired balance of accuracy and computational cost. This would result in a smart balance of accuracy and performance.

A dedicated GPU implementation using CUDA, OpenCL, or compute shaders is required to quantitatively validate these claims. Such benchmarking is left as future work.

Curve-specific optimizations, such as creating curve-aware outcode for Bézier curves, B-splines, and NURBS, may result in even greater improvements. These may be far more useful than traditional fat-line or fat-curve methods in CAD, GIS, and scientific visualization applications. Because the algorithm is embarrassingly parallel, integrating it with GPU and multi-core hardware via shader-based, CUDA, or OpenCL implementations could result in massive speedups, ideal for real-time rendering pipelines. Adding RNSCA directly to game engines, AR/VR platforms, and professional CAD toolchains could reduce CPU usage and enable work with large datasets at faster frame rates. RNSCA is especially appealing in fields that require high precision, such as geospatial mapping, medical imaging, and computational geometry, where there are frequently large curve datasets. This is because RNSCA doesn't require many explicit intersection calculations. It would also be worthwhile to investigate hybrid methods that combine classical algorithms such as Cohen-Sutherland or Liang-Barsky for quick trivial rejections and RNSCA for more complicated or curve-heavy cases.Finally, rigorous theoretical foundations-such as formal proofs of termination guarantees, error bounds, and complexity models under varying conditions, particularly for adaptive N-would boost the algorithm's scientific credibility and encourage wider adoption. If we all work together on these issues, RNSCA could evolve from a novel concept to a a promising approach in modern graphics and geometric computing.

The current experimental validation was conducted on a CPU-based implementation (Intel i7, Python 3.10). While the algorithm is structurally designed for parallel GPU execution, GPU implementation and benchmarking remain part of future work.

References

1. Cohen, D. and Sutherland, I.E. 1967. A method for line clipping. *Proceedings of the AFIPS Spring Joint Computer Conference*, 16, 343–351.
2. Liang, Y.-D. and Barsky, B.A. 1984. A new concept and method for line clipping. *ACM Transactions on Graphics*, 3, 1–22. https://doi.org/10.1145/357332.357333.
3. Nicholl, T., Lee, D. and Nicholl, M. 1987. An efficient new algorithm for 2-D line clipping. *ACM SIGGRAPH Computer Graphics*, 21, 253–262.

4. Foley, J.D., van Dam, A., Feiner, S.K. and Hughes, J.F. 1990. Computer Graphics: Principles and Practice. Addison-Wesley, MA.
5. Lou, Q. and Liu, L. 2012. Hybrid clipping for curve intersection. *Computers & Graphics*, 44, 753–766. https://doi.org/10.1016/j.cag.2012.03.021.
6. Rehman, S., Khan, S. and Khan, M. 2019. A survey of line clipping algorithms. *IEEE Access*, 7, 154876–154891. https://doi.org/10.1109/ACCESS.2019.2943889.
7. Schmalstieg, D., Höllerer, T., Diepstraten, J. and Fuhrmann, A. 2018. GPU-based hierarchical rasterization for vector graphics. *ACM Transactions on Graphics*, 37, Article 116. https://doi.org/10.1145/3197517.3201320.
8. Loop, C. and Blinn, J. 2005. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics*, 24, 1000–1009. https://doi.org/10.1145/1073204.1073295.
9. Patrikalakis, N.M., Maekawa, T. and Sakkalis, G. 2009. Shape Interrogation for Computer Aided Design and Manufacturing. Springer-Verlag, Berlin.
10. Nießner, M., Zollhöfer, M., Izadi, S. and Stamminger, M. 2013. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, 32, Article 169. https://doi.org/10.1145/2508363.2508403
11. Liu, Y. and Puri, R. 2021. Large-scale GPU-accelerated spatial join processing. *IEEE Transactions on Knowledge and Data Engineering*, 33, 1112–1126. https://doi.org/10.1109/TKDE.2019.2941566
12. Matthes, F. and Drakopoulos, G. 2019. Fast line clipping against rectangular windows. *Journal of Graphics Tools*, 23, 1–14.
13. Skala, V. and Bùi, T. 2019. Efficient line clipping in E^3. *Computers & Graphics*, 83, 1–11. https://doi.org/10.1016/j.cag.2019.04.001
14. Wonka, P., Ribarsky, W. and Eble, M.S. 2021. Visibility in computer graphics: a survey. *IEEE Computer Graphics and Applications*, 41, 42–58. https://doi.org/10.1109/MCG.2020.3040431
15. Amanatides, J. and Woo, A. 1987. A fast voxel traversal algorithm for ray tracing. *Proceedings of the Eurographics Conference*, 3–10.
16. Akenine-Möller, T., Haines, E. and Hoffman, N. 2018. Real-time Rendering. CRC Press, Boca Raton.

17. Pharr, M., Jakob, W. and Humphreys, G. 2016. Physically Based Rendering: From Theory to Implementation, Morgan Kaufmann.
18. Hormann, K. and Agathos, A. 2001. The point-in-polygon problem for arbitrary polygons. *Computational Geometry: Theory and Applications*, 20, 131–144. https://doi.org/10.1016/S0925-7721(01)00014-5
19. Liu, Y. and Puri, R. 2020. Filter-and-refine strategies for GPU-based geometric processing. *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 1–10.
20. Matthes, D. and Drakopoulos, V. 2022. Line clipping in 2D: overview, techniques and algorithms. *Journal of Imaging*, 8, 286.
21. Piegl, L. and Tiller, W. 1997. The NURBS Book. Springer-Verlag, Berlin.
22. Skala, V. 1995. Fast algorithms for line clipping. *Computer Graphics Forum*, 14, 21–34. https://doi.org/10.1111/1467-8659.1410021.
23. Cohen-Or, D. and Chrysanthou, Y.L. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization & Computer Graphics*, 9, 412-431.