Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

SwiftSina: Proposing a Fast and Efficient Exact Algorithm for Convex Hull

Sina Saadati

sina.mountain95@gmail.com

Amirkabir University of Technology Department of Computer Engineering

Research Article

Keywords:

Posted Date: November 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11886718/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

SwiftSina: Proposing a Fast and Efficient Exact Algorithm for Convex Hull

Sina Saadati[1]*

Abstract

The Convex Hull is a well-known problem in computational geometry, in which the smallest convex set containing all given points in a plane is determined. It has important applications in various computational fields, including artificial intelligence, computer vision, and astronomical computation. However, existing algorithms can suffer from high computational costs and reduced efficiency as the size of the input dataset increases. In this research, a novel method is proposed for solving the Convex Hull problem more efficiently. First, an agent-driven algorithm is employed to obtain an estimation of the final solution, through which points that are unlikely to belong to the final Convex Hull are identified and eliminated. Subsequently, an optimal algorithm is applied to the reduced set of points to determine the exact Convex Hull. By removing irrelevant points prior to the execution of the exact algorithm, the overall computational cost is reduced while the exactness of the final solution is maintained. It can be mathematically proven that the proposed method provides an efficient solution to the Convex Hull problem, with a time complexity of O(n + h log h), where n denotes the total number of input points, and h denotes the number of points belonging to or located in significant proximity to the final Convex Hull.

Key Words: Natural Computing, Computational Geometry, Convex Hull, High Performance Computing, Parallel Computing, Algorithm and Data Analysis.

1.Introduction

Computational geometry is a subfield of computer science concerned with the design and analysis of algorithms for solving geometric problems efficiently. A primary objective in this field is to develop algorithms that minimize computational resources, particularly execution time and space requirements [1-2]. One of the fundamental problems in computational geometry is the convex hull problem, which seeks to determine the smallest convex set containing a given set of points in the plane [1-4]. Geometrically, the convex hull can be illustrated using the analogy of an elastic band stretched around a collection of fixed points. When the band is released, it contracts around the outermost points while enclosing all the remaining points. The resulting boundary delineates the convex hull of the given point set [2,5].

The convex hull algorithm has been applied in various domains, including artificial intelligence [6], computer vision [7], medical image processing [8-9], and even astronomical studies [10], where it can support tasks such as object detection, image segmentation, feature extraction, and noise detection and correction. Despite the availability of several computationally efficient convex hull algorithms, their execution can remain costly when processing large and densely distributed point sets, particularly in applications such as computer vision and astronomy. A significant portion of this computational effort is often devoted to points that do not ultimately contribute to the final convex hull. Therefore, identifying and removing non-contributing points before performing the main convex hull computation can reduce the effective input size, decrease the number of required operations, and improve the overall computational efficiency and execution time of the algorithm.

[1] Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran.
sina.saadati@aut.ac.ir
* Corresponding Author

In this research, an algorithm is proposed for computing the exact solution to the convex hull problem. First, a scalable agent-driven algorithm is employed to rapidly identify points that do not contribute to the final convex hull polygon. These non-contributing points are subsequently removed from the input set to reduce the computational workload. The remaining points are then processed using the Sina algorithm, an optimal algorithm for convex hull computation, to determine the final exact convex hull. By eliminating non-contributing points during the first stage, the proposed approach reduces the number of points processed by the Sina algorithm, ideally approaching the number of points that constitute the final convex hull. Experimental results demonstrate that the proposed approach provides improved computational efficiency, particularly for densely distributed point configurations commonly encountered in applications such as computer vision and astronomical data processing. Furthermore, the proposed approach is amenable to parallelization through distributed computing and high-performance computing paradigms, offering further potential for reducing execution time and improving scalability.

The remainder of this paper is organized as follows: Section 2 presents a review of the relevant literature. Section 3 describes the proposed methodology and algorithm in detail, while Section 4 presents its experimental evaluation and results.

2. Related Works

This section presents a review of the existing literature on convex hull computation and its applications. The convex hull is a well-established concept in computational geometry and has been widely employed across various computational domains, including artificial intelligence [6], computer vision [7], medical image processing [8], medical detection [9], astronomy [10], and even electrical power systems [11-12]. In image processing, for instance, convex hull techniques have been utilized for tasks such as noise detection and the correction of inaccuracies arising from computational models [5-6]. In astronomy, convex hull computation has been applied to determine the boundaries of astronomical structures, including star clusters and galaxies [10]. Despite its broad range of applications, computing the convex hull remains computationally demanding in many scenarios, particularly as the number of input points increases substantially. This challenge has motivated the development of more efficient algorithms and computational approaches for reducing the processing time and resource requirements associated with convex hull computation.

Several algorithms have been proposed to solve the convex hull problem, each offering different computational characteristics. The Graham scan algorithm is a classical approach with a worst-case time complexity of $O(n\ log\ n)$, where n denotes the number of input points [13]. Another well-known method is the Jarvis March algorithm, whose computational efficiency can be advantageous when the number of points on the final convex hull is substantially smaller than the total number of input points [14]. Other established approaches include Andrew's monotone chain algorithm, which also achieves $O(n\ log\ n)$ time complexity [15], as well as estimation-based methods such as DeepHull [16], which employ alternative strategies to accelerate convex hull computation. These approaches demonstrate the continuing effort to improve the efficiency and scalability of convex hull computation for large and complex point sets.

In previous research, a natural computing-based algorithm was proposed for convex hull computation [5]. The approach employs autonomous agents to simulate the behavior of an elastic band surrounding a set of fixed points. The results demonstrated that the computational performance of the agent-driven algorithm is strongly influenced by the geometric configuration of the output points and the shape of the resulting convex hull polygon. Subsequently, the Sina algorithm was introduced as an approach inspired by the agent-driven method, with a worst-case time complexity of $O(n\ log\ n)$. In addition to its computational efficiency, the Sina

algorithm can be parallelized to further reduce execution time and improve its applicability to large-scale convex hull problems [17].

In this research, a novel algorithm is proposed that integrates the agent-driven algorithm with the Sina algorithm to efficiently solve the convex hull problem. The proposed approach has a worst-case time complexity of $O(n + h\ log\ h)$, where n denotes the total number of input points and h represents the number of points that lie on or are significantly close to the final convex hull polygon. By reducing the number of points requiring full convex hull computation, the proposed method can provide substantial computational advantages, particularly for large and densely distributed point sets. This characteristic makes the proposed algorithm especially suitable for applications involving large-scale geometric data, such as computer vision and astronomical data processing.

3. Proposed Methodology

This section presents the proposed methodology in detail. First, the concept of bad points is introduced. Bad points are defined as input points that lie inside the final convex hull and are sufficiently far from its boundary edges, such that they cannot contribute to the construction of the final convex hull polygon. Since processing these non-contributing points can significantly increase the computational cost of convex hull computation, an agent-driven algorithm is initially employed to identify and eliminate them from the input set. Following this preprocessing stage, the remaining points are expected to be equal or sufficiently close in number to the points constituting the final convex hull. The Sina algorithm, an exact convex hull algorithm, is then applied to the reduced set of points to compute the final convex hull. By removing a substantial number of non-contributing points before the exact computation, the proposed methodology aims to reduce the computational workload and improve the overall efficiency of convex hull computation, particularly for large and densely distributed point sets.

In many cases, bad points represent a major source of computational inefficiency in convex hull computation. Although these points do not belong to the final convex hull and may be located considerably far from its boundary, they can impose substantial computational overhead because conventional algorithms generally process the input points without prior knowledge of their geometric relevance. Computers do not inherently possess an understanding of the spatial relationships among points; consequently, conventional approaches must examine and process a large portion of the input set to determine which points contribute to the final solution. If bad points can be accurately identified and eliminated using an efficient preprocessing algorithm, the effective size of the input can be substantially reduced. As a result, fewer points need to be processed by the subsequent convex hull algorithm, potentially leading to improved computational efficiency and reduced execution time.

The early detection and elimination of bad points can be performed using the agent-driven algorithm for convex hull computation. The scalability of the agent-driven approach enables it to efficiently process point sets of varying sizes and geometric configurations, thereby reducing the computational demands of the overall convex hull computation. In particular, the computational workload of the agent-driven algorithm is influenced by the geometric structure and shape of the resulting convex hull rather than solely by the total number of input points. This characteristic enables the algorithm to identify bad points rapidly and eliminate them during the preprocessing stage. Moreover, the agent-driven algorithm preserves all points that belong to the final convex hull, ensuring that no valid hull point is incorrectly removed during the elimination process. This property is essential for maintaining the correctness and accuracy of the subsequent exact convex hull computation.

Although the agent-driven algorithm can be employed to compute the exact solution of the convex hull problem, it can also be used to scale the input point set and obtain an estimation-based solution with improved computational efficiency. In this context, scaling can be interpreted as virtually zooming out from the initial representation of the point set to obtain a summarized view of the overall spatial distribution. As the environment is scaled, points located sufficiently close to one another may appear as a single representative point, thereby reducing the effective number of points that must be processed. This reduction can substantially improve computational efficiency, as the computational workload and overall execution time of the estimation-based approach are primarily influenced by the geometric structure of the resulting solution. Consequently, points located well inside the estimated convex hull have a limited impact on computational complexity and do not significantly increase the processing requirements. This property makes the scaling mechanism particularly useful for efficiently analyzing large and densely distributed point sets.

After scaling the input points using the agent-driven algorithm, an estimation-based solution can be obtained. This estimated solution can then be used to accurately identify and eliminate bad points from the original point set. This approach is particularly beneficial in applications such as computer vision and astronomy, where points often exhibit densely distributed configurations and the majority of the input points lie within the convex hull. In such cases, conventional convex hull algorithms may require substantial computational resources and execution time because they process a large number of points that ultimately do not contribute to the final convex hull. In contrast, the agent-driven approach can efficiently identify and eliminate these non-contributing points before the exact convex hull computation, thereby reducing the effective problem size and improving computational efficiency. This advantage is particularly pronounced for point sets following uniform or Gaussian distribution patterns, where a large proportion of the points are typically located within the interior region of the convex hull.

Following the detection and elimination of bad points, the number of remaining points tends to approach the number of points constituting the final convex hull. However, because the solution obtained during the agent-driven preprocessing stage is estimation-based, an exact solution may still be required for applications where computational accuracy is essential. Therefore, the Sina algorithm can be applied to the reduced set of remaining points to determine the exact convex hull. Since the majority of non-contributing points have already been eliminated, Sina operates on a substantially smaller input set, thereby reducing the computational workload while preserving the exactness of the final result. Consequently, the proposed methodology combines the computational efficiency of the agent-driven estimation stage with the accuracy of an exact convex hull algorithm, enabling efficient computation of the optimal convex hull solution.

The proposed method is presented in Pseudocode 1. As shown in the pseudocode, the algorithm initially identifies and eliminates the bad points from the input set in order to reduce the computational workload and improve overall efficiency (Line 1). Subsequently, the Sina algorithm is applied to the reduced point set to compute the exact solution to the convex hull problem.

Pseudocode 1, The proposed algorithm for solving convex hull problem in a swift approach.

```
    Algorithm SolvingConvexHull()
    Input: A set of points in the plane.
    Output: Exact convex hull polygon for the given points.
1   Call eliminateBadPoints() to identify and remove bad points that are
    irrelevant to the problem.
2   Apply the Sina algorithm to the remaining points to compute the final
    exact solution.
```

The scaling and bad-point detection procedure is detailed in Pseudocode 2. As shown, a scale value is first specified. A larger scale value can improve the accuracy of the estimation process; however, it may also increase the computational workload. In the second line, a two-dimensional binary array is generated to simulate the environment in which the elastic band and the input points are modeled. In the third line, the minimum and maximum coordinate values of the input points are calculated. These values are subsequently used in Line 4 to normalize the point coordinates and map them to the environment data structure.

The environment data structure represents the occupancy status of the sub-spaces within the simulation framework. Specifically, it indicates whether each sub-space contains a molecular particle or is free, thereby determining whether other particles can move through that sub-space. In Line 5, an elastic band model is generated and positioned around the input points. The band is initialized such that it covers the relevant sub-spaces of the environment data structure. In Line 6, the elastic band is released, allowing it to contract and form an approximate convex hull polygon around the given points. The resulting convex hull is an estimation rather than an exact solution. This is primarily because the normalization and scaling operation in Line 4 maps the original points to a relatively small discrete space. Consequently, multiple input points may be mapped to the same sub-space in the environment data structure. Despite this limitation, the estimated convex hull provides an effective mechanism for identifying and eliminating irrelevant points, thereby reducing the size of the problem before the exact convex hull computation.

In Line 7, the final configuration of the elastic band is examined to identify bad points. A point is considered a bad point if it is mapped to a sub-space that does not lie on the final convex hull obtained in Line 6. These points can therefore be safely removed from the subsequent computation. Eliminating such points can significantly improve computational efficiency because the number of points passed to the next stage is reduced to a value equal to or close to the number of points forming the final convex hull.

Pseudocode 2, The proposed algorithm for detection and elimination of bad points using agent-based simulation methodology.

```
    Algorithm eliminateBadPoints()
    Input: A set of points in the plane, points, and a recursion depth value, depth.
    Output: A reduced set of points after eliminating the identified bad points.
1   scale ← Consider a predefined value as the scale factor;
2   Environment ← Create a two-dimensional Boolean array of size scale × scale, initialized to 0. Each
    element represents a sub-space of the environment and indicates whether that sub-space is empty or
    occupied;
3   MinX, MinY, MaxX, MaxY ← Find the minimum and maximum x- and y-coordinates among the input
    points;
4   For each point ∈ points do
4.1     (x, y) ← Normalize point.x and point.y using MinX, MinY, MaxX, and MaxY;
4.2     Environment[x, y] ← 1 ;
5   ElasticBand ← Create an agent-based elastic-band model and initialize it around the Environment so
    that it covers the relevant sub-spaces;
6   Release ElasticBand so that it contracts around the occupied sub-spaces. The particles forming the
    elastic band cannot move through a sub-space whose corresponding Environment value is 1. The final
    configuration of the ElasticBand forms an estimated convex hull of the input points;
7   Using the final configuration of ElasticBand, identify and remove from points all points mapped to sub-
    spaces that do not lie on the estimated convex hull;
8   If depth > 0 then
8.1     For each subspace ∈ Environment do
8.2         If number of points in the subspace is significantly high do
8.3             points ← eliminateBadPoints(points, depth − 1);
9   Return points;
```

In cases where the input points are densely distributed in certain regions, a single execution of the procedure may not identify all irrelevant points. Therefore, the proposed procedure can be applied recursively to the remaining points to detect and eliminate additional bad points. However, in sparse or other less favorable cases, excessive recursive calls may introduce additional computational overhead and consequently reduce the overall efficiency of the algorithm. To address this issue, a depth-based threshold is introduced in Line 8 to limit the recursion depth and maintain the computational efficiency of the proposed method. Finally, the identified bad points are removed from the input set, and the resulting reduced point set is passed to the subsequent stage of the algorithm for exact convex hull computation. The values, such as scale or depth threshold are configurable to enhance the usability of the algorithm.

After the bad points have been removed, the number of points remaining in the problem is significantly reduced. In many cases, the number of remaining points is equal to, or at least very close to, the number of points that constitute the convex hull polygon. Consequently, applying an optimal algorithm, such as the SINA algorithm, to the reduced point set can produce the final exact convex hull solution while substantially reducing the computational cost. This combination of the bad-point elimination procedure and the SINA algorithm therefore provides an effective and computationally efficient approach for solving the convex hull problem.

4. Discussion and Evaluation

In this section, the proposed method is evaluated from three perspectives: correctness, computational complexity, and parallelizability. The algorithm initially employs an agent-driven modeling and simulation framework to represent the input points of the convex hull problem as fixed points (nails) in the plane. An agent-driven simulation of an elastic band is then performed to obtain an estimated convex hull and identify the bad points defined in Section 3. These points are subsequently removed from the problem to reduce the computational workload. Finally, the Sina algorithm is applied to the remaining points to compute the exact optimal solution of the convex hull problem.

First, the mathematical correctness of the proposed method is discussed to establish whether it is guaranteed to produce the exact optimal solution. Second, its execution time is analyzed under different input configurations. Finally, the potential for parallelizing the proposed method is investigated, particularly in distributed-computing and high-performance-computing environments, including GPU-based architectures.

4.1. Evaluation of the Correctness of the Algorithm

It can be shown that the proposed method preserves the exactness of the final convex hull solution. The key issue is whether the bad-point elimination procedure can incorrectly remove a point that belongs to the final convex hull. The proposed scaling and elastic-band procedure provides an estimated representation of the spatial distribution of the input points. After the points are mapped to the scaled environment, the elastic band is released and contracts around the occupied sub-spaces. The resulting configuration represents an estimation of the convex hull.

The points considered for elimination can be divided into two categories: points that may contribute to the final convex hull and bad points. As defined in Section 3, bad points are points that are located strictly inside the convex hull and therefore cannot influence the boundary of the final convex hull polygon. According to the

proposed elimination procedure, points mapped to environment sub-spaces that have no intersection with the final configuration of the elastic band are identified as bad points.

The elimination of these points does not change the final convex hull, provided that the elimination procedure removes only points that are guaranteed not to lie on the boundary of the convex hull. In particular, removing points located strictly inside the convex hull cannot change the position or shape of the convex hull polygon. Therefore, the subsequent application of the exact Sina algorithm to the reduced point set produces the same convex hull as would be obtained by applying Sina directly to the original point set.

The scaling operation is introduced primarily to obtain an abstract representation of the spatial distribution of the input points and to facilitate the initial estimation of the convex hull. Although this representation may map multiple original points to the same environment sub-space, the estimated hull is not used as the final solution. Instead, it is used only to identify points that can be safely eliminated. The final result is always obtained by applying the exact Sina algorithm to the remaining points. Consequently, the approximation introduced during the agent-driven simulation phase does not compromise the exactness of the final result.

4.2. Evaluation of the Worst-Case Complexity of the Algorithm

In this section, the computational efficiency and time complexity of the proposed algorithm is evaluated.The primary advantage of the proposed approach is the reduction in the number of points provided to the exact convex hull algorithm. For many point distributions, including relatively uniform and Gaussian-like spatial distributions, a large proportion of the input points may lie inside the final convex hull. These interior points do not contribute to the final boundary but can substantially increase the computational workload of an exact convex hull algorithm. By identifying and removing such points during the agent-driven simulation phase, the proposed method can significantly reduce the size of the subsequent computational problem. After the elimination phase, the number of remaining points can, in favorable cases, become equal to, or close to, the number of points belonging to the final convex hull. Consequently, the computational cost of the subsequent Sina algorithm execution is considerably reduced.

This reduction becomes particularly important for large-scale problems in which the ratio between the total number of input points and the number of convex-hull points is high. The worst-case behavior occurs for specific and relatively uncommon spatial configurations in which the scaling operation does not expose all of the bad points during a single execution of $eliminateBadPoints()$. An example of such a configuration is illustrated in Figure 1.

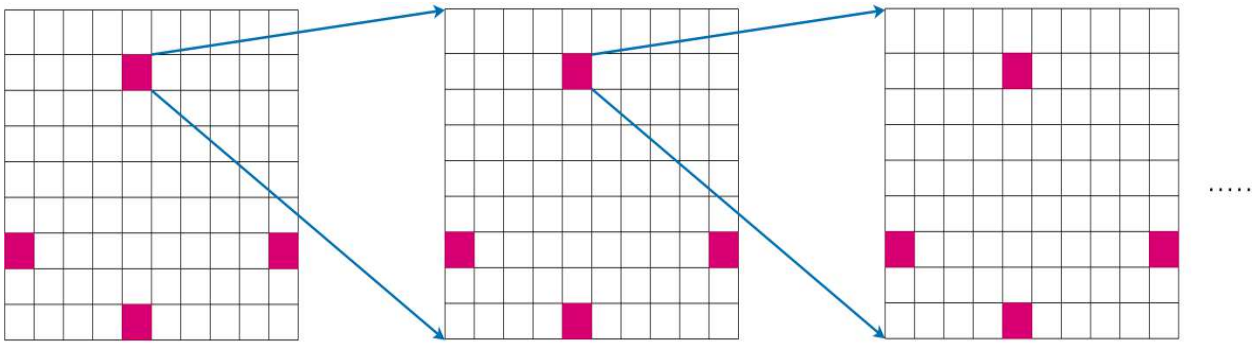


Figure 1, An example of worst-case scenario where bad points have a structured form.

In this situation, the remaining points may be distributed across the environment in a structured manner such that additional recursive executions are required to identify and remove further bad points. Without a limit on the recursion depth, repeated execution of the elimination procedure could introduce substantial computational overhead. In an extreme case, the recursion depth may increase with the number of input points, resulting in repeated processing of the same or overlapping sets of points and potentially increasing the computational complexity toward $O(n^2)$. To prevent this degradation, the proposed method introduces a depth-based threshold, as described in Pseudocode 2.

The threshold limits the number of recursive executions and prevents the recursive elimination process from continuing indefinitely. Therefore, even for unfavorable point configurations, the computational overhead introduced by recursive processing remains bounded. Under the proposed depth-limiting strategy, the worst-case complexity of the elimination phase can be bounded by $O(n \log n)$, depending on the implementation and the number of levels permitted by the depth threshold. It should be emphasized that these unfavorable configurations represent cases in which the spatial distribution of the points prevents the scaling and elastic-band procedure from eliminating a large number of bad points in each iteration. In contrast, for favorable distributions, the number of points can decrease substantially after the first execution, making the subsequent exact convex hull computation considerably more efficient.

The dimensions of the simulation environment are determined adaptively according to the number of input points. For relatively small datasets, containing fewer than 10,000 points, the simulation environment can be represented by a fixed grid, such as 100×100 or 200×200 cells. As the number of input points increases substantially, the dimensions of the simulation environment can be adjusted so that the width and height of the grid do not exceed $\sqrt{n}$, or a specified proportion of this value, where n denotes the number of input points. Consequently, the total number of cells in the simulation environment is bounded by O(n). Since the number of arithmetic operations performed by the proposed algorithm is proportional to the number of cells in this environment, its computational complexity with respect to these operations remains O(n).

Furthermore, for extremely large datasets, the dimensions of the simulation environment can be maintained at a fixed size, regardless of the number of input points. For instance, grids of 400×400 or 1000×1000 cells can be employed to process datasets containing billions of points or even more. In this configuration, the computational workload associated with grid-based operations remains bounded by the fixed number of cells rather than increasing directly with the dataset size. This adaptive spatial discretization strategy enables the proposed algorithm to maintain computational efficiency and scalability, making it potentially suitable for processing exceptionally large, including galaxy-scale, datasets. Importantly, the choice of grid resolution should be governed by the required spatial precision and the characteristics of the input data, ensuring that computational efficiency is achieved without compromising the accuracy and reliability of the results.

4.3. Evaluation of the Parallelization of the Algorithm

The proposed method is also amenable to parallel implementation. In this section, the parallelization of the proposed method is evaluated. The agent-driven modeling framework consists of multiple agents whose operations can potentially be distributed across multiple processing cores or computing nodes. In particular, operations associated with mapping points to the environment, updating the environment data structure, and processing independent spatial regions can be executed concurrently when the underlying implementation permits such parallelization. The exact Sina stage may likewise benefit from parallel-computing techniques. The processing of spatial regions or agent interactions can be distributed among multiple processing units.

Consequently, the complete proposed framework can be implemented on high-performance computing architectures, including multicore CPUs, GPUs, or distributed computing systems.

Assuming that the workload is effectively distributed among m computing nodes, the computational cost of the parallel implementation can be expressed in terms of the number of input points n, the number of points h on the final convex hull or very close to it, and the number of computing nodes m. In an average case, the computational workload associated with processing the input points can be reduced approximately in proportion to the number of available computing nodes. However, the actual speedup depends on factors such as workload balance, memory access, and the parallel scalability of both the agent-driven simulation and the Sina algorithm. Therefore, the proposed method has two complementary sources of computational improvement. First, the bad-point elimination phase reduces the number of points that must be processed by the exact convex hull algorithm. Second, the remaining computations can potentially be distributed across multiple processing units. The combination of these two strategies can make the proposed approach suitable for large-scale convex hull problems and high-performance computing environments.

The findings of this study demonstrate that the proposed algorithm can significantly improve computational efficiency in a wide range of cases. In particular, computer vision applications involving dense configurations of points can benefit from the proposed bad-point elimination procedure, which reduces the number of points that must subsequently be processed by the exact convex hull algorithm and consequently improves execution time. The proposed method can also be beneficial in astronomical applications, where spatial objects such as stars may exhibit clustered or dense spatial distributions. In such cases, a considerable number of points may be located inside the convex hull and therefore do not contribute to its boundary. Eliminating these points before applying the exact Sina algorithm can substantially reduce the computational workload and improve the overall execution time. The experimental results indicate that, for the evaluated cases, large point sets can be processed within a few seconds. These results show that the proposed algorithm in this study is reliable and can be used in various applications, such as computer vision and astronomical processing.

References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer, 2000. https://doi.org/10.1007/978-3-540-77974-2

[2] M. de Berg and B. Speckmann, "Computational geometry: Fundamental structures," in *Handbook of Data Structures and Applications*. Boca Raton, FL, USA: Chapman and Hall/CRC, 2018, pp. 1013–1026. https://doi.org/10.1201/9781315119335

[3] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York, NY, USA: Springer, 2012. https://doi.org/10.1007/978-1-4612-1098-6

[4] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[5] S. Saadati and M. A. Razzazi, "A natural way of solving a convex hull problem," *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences*, 2025. https://doi.org/10.1007/s40010-025-00932-z

[6] S. Saadati, M. Hashemi, and C. Nezhat, "Unsupervised deep learning for semantic segmentation using laparoscopic videos: A self-detection and self-learning approach," *Machine Learning with Applications*, Art. no. 100767, 2025. https://doi.org/10.1016/j.mlwa.2025.100767

[7] S. Saadati and A. Sepahvand, "An interpretable and efficient KNN-based approach for object segmentation," in *Proc. 11th Int. Conf. Signal Process. Intell. Syst. (ICSPIS)*, Babol, Iran, 2025, pp. 144–147. https://doi.org/10.1109/ICSPIS68676.2025.11551697

[8] J. Zheng, L. Wang, J. Gui, *et al*., "Study on lung CT image segmentation algorithm based on threshold-gradient combination and improved convex hull method," *Scientific Reports*, vol. 14, Art. no. 17731, 2024. https://doi.org/10.1038/s41598-024-68409-4

[9] [Initials]. Liu, Y. Lu, J. Xu, *et al*., "3D reconstruction of bone CT scan images based on deformable convex hull," *Medical and Biological Engineering and Computing*, vol. 62, pp. 551–561, 2024. https://doi.org/10.1007/s11517-023-02951-7

[10] T. Chand, S. Sharma, K. Singh, J. Pandey, A. Verma, H. Kaur, Mamta, M. Chakraborty, D. K. Ojha, and A. K. Singh, "Long-term investigation of an open cluster Berkeley 65," *Frontiers in Astronomy and Space Sciences*, vol. 12, Art. no. 1608196, 2025. https://doi.org/10.3389/fspas.2025.1608196

[11] D. Pozo, "Convex hull formulations for linear modeling of energy storage systems," *IEEE Transactions on Power Systems*, vol. 38, no. 6, pp. 5934–5936, Nov. 2023. https://doi.org/10.1109/TPWRS.2023.3304131

[12] M. Qu, T. Ding, Y. Sun, C. Mu, K. Pan, and M. Shahidehpour, "Convex hull model for a single-unit commitment problem with pumped hydro storage unit," *IEEE Transactions on Power Systems*, vol. 38, no. 5, pp. 4867–4880, Sep. 2023. https://doi.org/10.1109/TPWRS.2022.3215463

[13] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, 1972. https://doi.org/10.1016/0020-0190(72)90045-2

[14] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters*, vol. 2, no. 1, pp. 18–21, 1973. https://doi.org/10.1016/0020-0190(73)90020-3

[15] A. M. Andrew, "Another efficient algorithm for convex hulls in two dimensions," *Information Processing Letters*, vol. 9, no. 5, pp. 216–219, 1979. https://doi.org/10.1016/0020-0190(79)90072-3

[16] R. Balestriero, Z. Wang, and R. G. Baraniuk, "DeepHull: Fast convex hull approximation in high dimensions," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Singapore, 2022, pp. 3888–3892. https://doi.org/10.1109/ICASSP43922.2022.9746031

[17] S. Saadati and M. Razzazi, "Sina algorithm: Proposing a distributed agent-driven algorithm for solving convex hull problem," *Research Square*, 2025. https://doi.org/10.21203/rs.3.rs-7160843/v1