Convex Hulls: Comparing Theoretical and Practical Runtimes

Zach Havens, Danika Passler Bates

December 10, 2021

1 Abstract

In this project, we have implemented the Gift Wrapping/Jarvis March (GW), Graham's Scan, divide and conquer (D&C), and Chan's algorithms for computing the convex hull of a point set. After implementing the algorithms we tested them with randomly generated point sets of varied input sizes and distributions. Next, we analyzed the data and compared our experimental run times with the expected runtimes given the algorithms' theoretical bounds. We found that the divide and conquer algorithm performed better than expected, while Chan's algorithm performed much worse than expected. Gift wrapping and Graham's scan performed as expected for each input. Finally, we present a profile on our implementation of Chan's algorithm and a discussion of why the algorithm didn't perform well.

2 Introduction

The convex hull of a set of n points P is defined as the smallest possible convex polygon such that all points in P are either contained within the polygon or on its boundary [1]. After briefly touching on the topic in class, we were motivated to learn more about convex hulls and felt that implementing algorithms to solve the problem would be the best way to do so. We also anticipated that learning different ways to determine convex hulls would expose us to concepts that might help us with future algorithms and problems. In addition, we were curious to explore how the theoretical run times of these algorithms compared to their practical run times. To meet these goals, we implemented convex hull algorithms and experimentally measured their performance. We implemented four convex hull algorithms: Gift Wrapping/Jarvis March, Graham's Scan, divide and conquer, and Chan's Algorithm. Following implementation, we conducted an experiment where we ran the algorithms with varied input sizes and distributions of the input points. Finally, we analyzed the results to compare the theoretical and practical run times of the algorithms.

3 Related Work

Chadnov and Skvortsov [2] also compared the experimental runtimes of various algorithms with diverse input sets. In their work they compared the Jarvis march, Graham's scan, and divide and conquer algorithms just as we did. In contrast, they did not examine Chan's algorithm, but they did look at Andrew's algorithm and the Quickhull algorithm. They concluded that the best algorithm depends on the distribution of points used.

4 Algorithms

4.1 Gift Wrapping Algorithm

The gift wrapping algorithm, also known as the Jarvis march algorithm, was discovered independently by Chand and Kapur in 1970 [3] and R.A. Jarvis in 1973 [4]. The algorithm starts by finding the leftmost point and then iteratively choosing the next point such that all other points are to the right side of the line between the current and next point. The algorithm is called the gift wrapping algorithm because the hull is formed by "wrapping" it around the point set. The algorithm has $O(nh)$ complexity, where h is the size of the hull.

4.2 Graham's Scan

Graham's scan algorithm was introduced by Ronald Graham in 1972 [5]. The algorithm represents the working hull as a stack. It starts with an extreme point and sorts all the other points according to the angle they make with that initial extreme point. It then iterates through all of the sorted points, popping points off of the stack until the angle formed between the current and top points form a consistent winding (clockwise or counter-clockwise), then adding the current point to the stack once the condition is satisfied. The resulting stack contains the hull and is constructed in $O(n \log n)$ time.

4.3 Divide and Conquer

In 1977, Preparata and Hong [6] applied a standard divide and conquer method to the convex hull problem. First, the algorithm sorts the point set by either the x or y coordinates. Then the sorted point set gets divided into two halves and the convex hull of each half is found. This is done recursively until a point set contains 3 or fewer points and hull construction is $O(1)$. Merging is done by first finding either the upper and lower tangents between the two hulls (if the point set is sorted by x coordinates) or the left and right tangents between the two hulls(if the point set is sorted by y coordinates). The merged hull is the two original hulls, and the tangent lines, minus the points on the original hulls on the interior of the tangents. The merge can be done in linear time, and so the overall hull construction takes $O(n \log n)$.

4.4 Chan's Algorithm

Chan's algorithm was outlined by Timothy Chan in 1996 [7]. Chan's algorithm combines the use of an $O(n \log n)$ algorithm such as divide and conquer or Graham's scan with the gift wrapping algorithm. Chan's algorithm first divides the input into m subsets of size n/m where m approximates h. For each subset, hulls are calculated using the $O(n \log n)$ algorithm. It then combines the subset hulls using a modified version of the gift wrapping algorithm in $O(\log n)$ time by looking for m contiguous extreme points from the subset hulls. If a complete hull is formed then it is returned. Otherwise, m is increased and a new construction attempt begins. The success of the algorithm depends on the accuracy of m, so the algorithm uses a squaring scheme of $m = 2^{2^t}$ where t ranges from to $\log_2 \log_2 n$. With this squaring scheme, the algorithm takes $O(n \log h)$.

5 Method

5.1 Implementation Details

The algorithms were implemented in Python 3.8 and leveraged the NumPy library for calculations. Our code can be found on GitHub: https://github.com/zhavens/convex_hull. Most of our algorithms did not differ significantly from their descriptions in their original papers. For the gift wrapping algorithm, the approach was inspired by Jarvis's paper [4], pseudocode on Wikipedia [8], and an implementation of the algorithm in Java by Tushar Roy [9]. The implementation of Chan's algorithm was aided by a tutorial by Pankaj Sharma [10]. Our Chan's implementation enabled easy configuration of the choice of sub-hull construction between Graham's scan and divide and conquer. In the end, experimental results were generated primarily with divide and conquer as it performed better practically. Additionally, the implementation of Chan's does include the suggested optimization that points that are not part of a sub-hull can be discarded when generating the next set of sub-hulls as they are guaranteed to not be extreme.

5.2 Divide and Conquer Merge

For the divide and conquer algorithm, we used a different merging algorithm than the one presented by Preparata and Hong [6] as the merge algorithm in the original paper is "sketchy" according to Chan [11] and does not describe all of the cases required for a full implementation. We instead used a technique outlined in MIT's OpenCourseWare notes [12] and in O'Rourke's *Computational Geometry in C* [13] to merge the hulls.

To find the upper tangent, the merge algorithm first finds the vertical line centered between the rightmost point of the left hull L and the leftmost point of the right hull R. It then sets the current tangent endpoints to be the rightmost point of the left hull at index i and the leftmost point of the right hull at index j. It then iteratively "walks up" the hull until the current endpoints define the

upper tangent. This is done by finding the next point in the left hull in counter clockwise order L_{i-1} and the next point on the right hull in clockwise order R_{j+1}. A comparison is done between the intersections of the dividing vertical line and 3 lines: $(L_i,\ R_j)$, $(L_i,\ R_{j+1})$, and $(L_{i-1},\ R_j)$. If the intersection between the vertical line and $(L_i,\ R_{j+1})$ is higher than the intersection between the vertical line and $(L_i,\ R_j)$, the j is incremented so R_{j+1} becomes the next current endpoint of R. If the intersection between the vertical line and the line $(L_{i-1},\ R_j)$ is higher than the intersection between the vertical line and the current endpoints, i is decremented and L_{i-1} becomes the next current endpoint in L. In either case, the loop is continued until the intersection of the vertical line and the line $(L_i,\ R_j)$ is maximized and the tangent is found. A similar procedure is used to find the lower tangent, but the endpoints "walk down" the hulls instead.

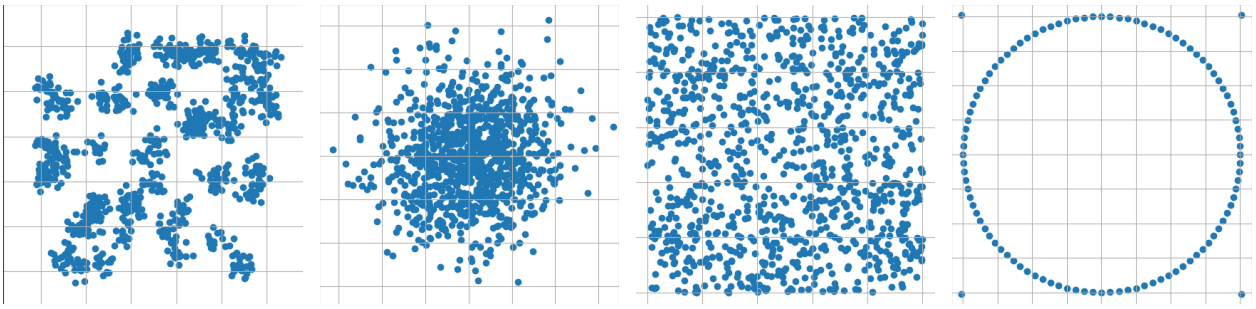
5.3 Input Data Sets

We used four different distributions in order to give a varied set of inputs to the algorithms: normal, uniform, clustered, and circular. Normal and uniform distributions were generated using standard randomization algorithms. For the clustered input sets we defined 100 centers uniformly within a unit square, and placed an $n/100$ points around each center with uniform angle and distance from it. The circular inputs have points evenly distributed around the circumference of a unit circle, with very small random perturbations to keep points in general position while still ensuring that every point was extreme.

Input sets of 50K, 100K, 250K, and 500K points were generated for each of the normal, uniform, and clustered distributions. The size of the constructed hulls for these inputs varied from 13 to 45 points. Notably, for normally distributed inputs h was always less than or equal to $\log n$. Input sets were limited to 500K points to allow for reasonable runtimes on available hardware. For the circular distribution, only sets of 25K points were generated to allow output-sensitive algorithms to construct hulls within a reasonable amount of time. "Boxed" inputs were also generated at the highest point count for distributions where a bounding box of four points was added to analyze the performance of the algorithms when the size of the hull is minimized. Examples of generated input sets can be found in Figure 1.

5.4 Environment

Code was run using the standard Python 3.8 interpreter on a Ubuntu 20.04 virtual machine (VM) managed by a machine running the XCP-Nng hypervisor in its bare-metal configuration. The VM was given exclusive access to 4 logical cores of an Intel Core i3-10110U processor and 16GB of memory.



(a) Clustered (b) Normal (c) Uniform (d) Circle (w/ Box)

Figure 1: Input Types

6 Results

6.1 Expected Results

Based on theoretical upper bounds, we expected that Graham's scan and divide and conquer would perform similarly since they are both $O(n \log n)$. We also expected that for our input sets with small hulls, gift wrapping would perform better than these two algorithms, but that for input sets with large resulting hulls, gift wrapping would perform poorly compared to these two algorithms, since gift wrapping is $O(nh)$. Since Chan's algorithm is $O(n \log h)$, we expected it to have the fastest runtime for all inputs, although with only marginal improvement over gift wrapping when hull sizes were small (as in the boxed inputs).

6.2 Measured Results

Results were measured by a difference in system clock time immediately prior to calling the appropriate hull construction algorithm to the time immediately after completion. Any additional steps such as validating the hull or writing any output were performed after the time measurement was completed. Results are shown in tabular format in Figure 2 and graphical format in Figure 3. The system clock has nanosecond precision, although results are reported to hundredths of a second as that was the most significant digit. All results for Chan's algorithm were measured using divide and conquer to calculate the sub-hulls during execution, except where noted otherwise.

Additionally, each algorithm was profiled when run against the 500K point clustered input without bounding box. Data was collected using the built-in cProfile module in order to get per-method runtimes and invocation counts during hull construction. Profile charts with annotations can be found in Figure 4, and results are discussed below.

$n(h)$	Gift-Wrap	Graham's	D&C	Chan's
50K (31)	1.20	0.66	0.23	0.72
100K (37)	2.83	1.45	0.54	1.79
250K (45)	9.06	3.56	1.60	4.52
500K (38)	14.90	7.51	3.53	9.29
500K (4)	1.73	7.12	3.61	11.08

(a) Clustered

$n(h)$	Gift-Wrap	Graham's	D&C	Chan's
50K (15)	0.57	0.66	0.23	0.72
100K (13)	0.99	1.44	0.52	1.53
250K (18)	3.62	3.49	1.67	4.64
500K (18)	7.18	7.02	3.45	9.28
500K (4)	1.72	7.31	3.59	7.82

(b) Normal

$n(h)$	Gift-Wrap	Graham's	D&C	Chan's
50K (31)	1.24	0.69	0.23	0.92
100K (28)	2.28	1.37	0.54	1.90
250K (33)	6.53	3.55	1.65	4.67
500K (37)	14.47	7.05	3.64	9.63
500K (4)	1.71	7.17	3.52	6.90

(c) Uniform

$n = 25$K	Gift-Wrap	Graham's	D&C	Chan's
Ordered	515.69	0.32	0.16	12.27*
Random	500.35	0.33	0.16	13.76*
Boxed (4)	0.08	0.34	0.15	0.95

(d) Circle

* Runtimes for Chan's using Graham Scan instead of D&C for sub-hull construction.

Figure 2: Runtimes (s)

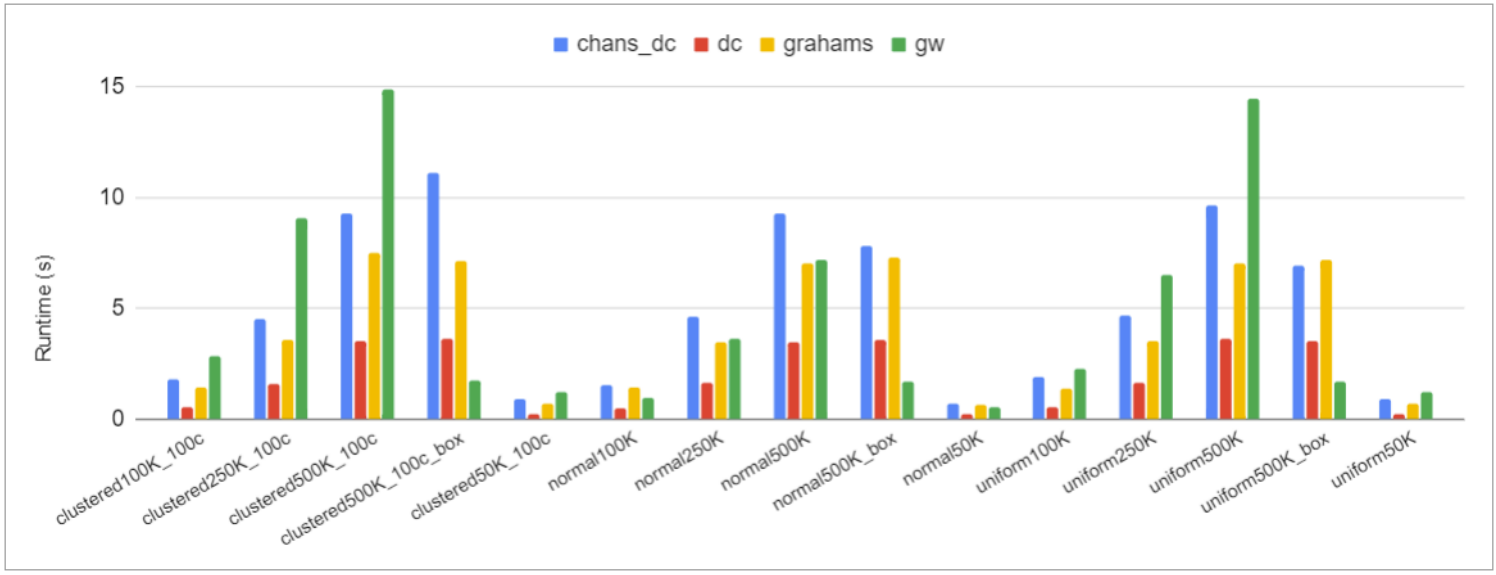


Figure 3: Runtime Chart

6.3 Graham's and Gift Wrapping Results

Graham's algorithm generally performed better than the gift wrapping algorithm, with a few exceptions. With normally distributed points the two algorithms performed similarly due to the fact that $h \approx \log n$ for those inputs, while for input sets with a bounding box where $h < \log n$ gift wrapping performed better. These results are expected since the gift wrapping is $O(nh)$ and Graham scan is $O(n \log n)$.

6.4 Divide and Conquer Performance

Despite having an upper bound that was theoretically equal to Graham's scan and worse than Chan's, the divide and conquer algorithm consistently performed best. This is due in part to the fact that the sorting done by our implementation of Graham's sorts points by angle relative to an arbitrary extreme point while the divide and conquer implementation sorts by x-coordinate. This eliminates the more expensive angle calculations, as shown by the blue boxes in Figures 4b and 4c. As well, the merge takes $O(n)$ in the worst case, but we hypothesize that it is taking less time than that practically because the algorithm would only need to consider the "inner" points as it walks up/down the hulls.

6.5 Chan's Disparity

Chan's algorithm performed significantly worse than expected based on its theoretical complexity. This is less surprising for circular input sets where $h = n$, but Chan's performed poorly even for inputs with bounding boxes added (where $h = 4$). This was true regardless of whether Graham's scan or divide and conquer was used to calculate sub hulls, with runtime always being longer than the given algorithm would take to calculate the hull of the entire input set. The version using divide and conquer were much faster practically than using Graham's as would be expected based on their relative performance.

To investigate why Chan's performed so poorly, we profiled the number of calls and amount of time it spent in each method it called in Figure 4d. Our profiling shows that the majority of the time was spent calculating sub-hulls (red outline) and finding the rightmost point in the hull using a binary search (orange outline). Interestingly enough, each of the calculation of sub-hulls alone and finding the rightmost points of said hulls took longer than the chosen sub-hull algorithm would take to calculate the full hull (Figure 4c, red outline). This means that completely eliminating the cost of one of those operations would not reduce the runtime below that of the given $O(n \log n)$ algorithm. The profiling data shows that the correct number of invocations and the right sized inputs were used to call Graham's or divide and conquer in each case, so we believe the poor performance to be due to constant-time operations introduced by the implementation of those algorithms which is causing significant performance impact given large number of invocations ($O(n)$).

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	24.23	24.23	46.77	46.77	convex_hull.py:200(gift_wrapping)
56999998	22.14	3.884e-07	22.14	3.884e-07	<string>:2(__eq__)
1	0.26	0.26	0.4033	0.4033	~:0(<built-in method builtins.sorted>)
500000	0.1434	2.867e-07	0.1434	2.867e-07	convex_hull.py:206(<lambda>)

(a) Gift Wrapping

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	1.605	1.605	20.58	20.58	convex_hull.py:322(grahams_scan)
1	0.4519	0.4519	14.55	14.55	~:0(<built-in method builtins.sorted>)
500000	1.612	3.225e-06	14.1	2.82e-05	convex_hull.py:334(<lambda>)
1499997/999998	3.224	3.224e-06	9.555	9.555e-06	~:0(<built-in method numpy.core._multiarray_umath.implement_array_function>)
499999	0.5309	1.062e-06	6.626	1.325e-05	<__array_function__ internals>:2(norm)
999998	1.041	1.041e-06	5.457	5.457e-06	<__array_function__ internals>:2(dot)
499999	1.827	3.654e-06	5.336	1.067e-05	linalg.py:2363(norm)

(b) Grahams

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
475711/1	2.458	2.458	19.52	19.52	convex_hull.py:276(divide_and_conquer)
262143	3.513	1.34e-05	6.795	2.592e-05	~:0(<built-in method builtins.sorted>)
237855	1.832	7.7e-06	4.179	1.757e-05	convex_hull.py:237(find_upper_tangent)
8927136	3.26	3.652e-07	3.26	3.652e-07	convex_hull.py:287(<lambda>)
237855	1.572	6.609e-06	2.982	1.254e-05	convex_hull.py:259(find_lower_tangent)
4354146	1.823	4.186e-07	1.823	4.186e-07	convex_hull.py:194(y_intesection)
5681068	1.516	2.669e-07	1.516	2.669e-07	~:0(<built-in method builtins.len>)
487917	0.3432	7.034e-07	0.9014	1.847e-06	convex_hull.py:49(vplot_is_on)
237855	0.474	1.993e-06	0.7527	3.165e-06	~:0(<built-in method builtins.max>)
475710	0.4159	8.742e-07	0.7441	1.564e-06	~:0(<method 'index' of 'list' objects>)
237855	0.4558	1.916e-06	0.7327	3.081e-06	~:0(<built-in method builtins.min>)

(c) D&C

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	1.351	1.351	49.3	49.3	convex_hull.py:349(chans_algorithm)
993290	9.819	9.885e-06	23.61	2.377e-05	convex_hull.py:88(find_rightmost_in_hull)
1019620/154688	3.95	2.554e-05	21.79	0.0001409	convex_hull.py:276(divide_and_conquer)
3597957	2.467	6.857e-07	6.342	1.763e-06	convex_hull.py:49(vplot_is_on)
432466	2.797	6.468e-06	6.246	1.444e-05	convex_hull.py:237(find_upper_tangent)
10646477	5.577	5.238e-07	5.577	5.238e-07	convex_hull.py:53(find_orientation)
18660775	4.997	2.678e-07	4.997	2.678e-07	~:0(<built-in method builtins.len>)
432466	2.428	5.613e-06	4.575	1.058e-05	convex_hull.py:259(find_lower_tangent)

(d) Chan's

Figure 4: Profiling Data - 500K Clustered

7 Future Work

7.1 Additional Algorithms

There are a large number of convex hull construction algorithms that we have not included in this study. Appropriate candidates include other output-sensitive algorithms such as the Kirkpatrick-Seidel algorithm [14] or another $O(n \log n)$ algorithm with a degenerate case such as Quickhull [15].

7.2 Practical Improvements to Chan's

As discussed in Section 6.5, Chan's performed significantly worse than expected. We consider 3 possible practical optimizations to improve runtimes:

- One optimization proposed by Chan [7] that was not implemented is to reuse the existing sub-hulls and merge them instead of calculating new larger sub-hulls when t is incremented. Using a linear-time merge algorithm would reduce the complexity of the sub-hull step from $O(n \log m)$ to $O(n \log(m/m'))$ time.
- The number of sub-hulls that need to be calculated can theoretically be reduced with a heuristic-based selection of initial t values to better approximate m. These could potentially be based loosely on the size of the input set, or on known distribution characteristics.
- As large portion of the runtime for Chan's algorithm is spent calculating the rightmost point with a $O(\log n)$ binary search. There could be small optimizations in the implementation of this algorithm efficient that could significantly improve the runtime over many calls.

7.3 Parallelization

While given access to multiple cores, all of our implementations were single-threaded and didn't take advantage of paralellization. Day and Tracey [16] applied paralellization to divide and conquer and saw significant improvements. They also suggested that even more performance gains could be realized with improved data passing methods. We also believe that paralellization could be applied to Chan's algorithm as constructions of each of the sub-hulls are independent of one another. This would be true whether constructing from scratch or merging existing hulls as discussed above. Given that the construction of the sub-hulls is a significant portion of the runtime for Chan's, we believe this could be a significant practical improvement.

7.4 Preprocessing P

Golin and Sedgewick [17] proposed a method of preprocessing P in order to reduce the number of points given to the hull construction algorithms. Their

algorithm works by removing points within a bounding box defined by 4 points in P, as points within the box are guaranteed to not be extreme. This can be done in $O(n)$ time and reduces the input to $O(\sqrt{n})$ points. This reduced set P' can then be passed to any of the convex hull algorithms discussed in this paper and will theoretically benefit the runtime as they have super-linear complexity. We would like to practically test this with our chosen input distributions and construction implementations.

8 Conclusion

In this paper, we have implemented and compared the practical and theoretical runtimes of four convex hull algorithms: Gift Wrapping/Jarvis March, Graham's Scan, divide and conquer, and Chan's Algorithm. Our results show the importance of examining practical runtimes in addition to theoretical bounds, as we noticed a significant disparity in the expected and actual runtimes of the Divide and Conquer algorithm and Chan's algorithm, which performed better and worse than their theoretical analysis would suggest, respectively.

References

[1] T. H. Cormen, C. Stein, C. E. Leiserson, and R. Rivest, *Introduction to algorithms*. MIT Press, 2009.

[2] R. Chadnov and A. Skvortsov, "Convex hull algorithms review," in *Proceedings. The 8th Russian-Korean International Symposium on Science and Technology, 2004. KORUS 2004.*, IEEE, vol. 2, 2004, pp. 112–115.

[3] D. R. Chand and S. S. Kapur, "An algorithm for convex polytopes," *Journal of the ACM*, vol. 17, no. 1, pp. 78–86, 1970. DOI: 10.1145/321556.321564.

[4] R. 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, ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(73)90020-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020019073900203.

[5] R. 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, ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(72)90045-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020019072900452.

[6] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM*, vol. 20, no. 2, pp. 87–93, Feb. 1977, ISSN: 0001-0782. DOI: 10.1145/359423.359430. [Online]. Available: https://doi-org.uml.idm.oclc.org/10.1145/359423.359430.

[7] T. M. Chan, "Optimal output-sensitive convex hull algorithms in two and three dimensions," *Discrete & Computational Geometry*, vol. 16, no. 4, pp. 361–368, 1996.

[8] Wikipedia, *Gift wrapping algorithm*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Gift_wrapping_algorithm.

[9] T. Roy, *Interview/jarvismarchconvexhull.java at master mission-peace/interview*, 2021. [Online]. Available: https://github.com/mission-peace/interview/blob/master/src/com/interview/geometry/JarvisMarchConvexHull.java.

[10] P. Sharma, *Chan's algorithm to find convex hull*, 2021. [Online]. Available: https://iq.opengenus.org/chans-algorithm-convex-hull/.

[11] T. M. Chan, "A minimalist's implementation of the 3-d divide-and-conquer convex hull algorithm," Tech. Rep., 2003.

[12] MIT OpenCourseWare, *Lecture 2: Divide and conquer*, 2015. [Online]. Available: https://ocw.aprende.org/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/MIT6_046JS15_lec02.pdf.

[13] J. O'Rourke, *Computational Geometry in C*, 2nd ed. Cambridge University Press, 1998, pp. 91–95.

[14] D. G. Kirkpatrick and R. Seidel, "The ultimate planar convex hull algorithm?" *SIAM Journal on Computing*, vol. 15, no. 1, pp. 287–299, 1986. DOI: 10.1137/0215021. eprint: https://doi.org/10.1137/0215021. [Online]. Available: https://doi.org/10.1137/0215021.

[15] W. F. Eddy, "A new convex hull algorithm for planar sets," *ACM Trans. Math. Softw.*, vol. 3, no. 4, pp. 398–403, Dec. 1977, ISSN: 0098-3500. DOI: 10.1145/355759.355766. [Online]. Available: https://doi-org.uml.idm.oclc.org/10.1145/355759.355766.

[16] A. Day and D. Tracey, "Parallel implementations for determining the 2d convex hull," *Concurrency: Practice and Experience*, vol. 10, no. 6, pp. 449–466, 1998.

[17] M. Golin and R. Sedgewick, "Analysis of a simple yet efficient convex hull algorithm," in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, ser. SCG '88, Urbana-Champaign, Illinois, USA: Association for Computing Machinery, 1988, pp. 153–163, ISBN: 0897912705. DOI: 10.1145/73393.73409. [Online]. Available: https://doi-org.uml.idm.oclc.org/10.1145/73393.73409.