Convex Hulls in Practice

Dhruv Mehta
Department of Computer Science
University of Maryland
College Park, MD 20904
dhruvnm2@gmail.com

December 3, 2020

Abstract

This paper presents an empirical analysis of several planar convex hull algorithms on various point distributions. The results of the experiments show that reducing the number of floating point operations is very important to runtime. Among the algorithms tested, SymmetricHull was the fastest on the distributions tested.

1 Introduction

The convex hull of a set of points in two dimensions is the smallest convex polygon that contains all of the points in the set. This problem has been studied for a long time and algorithms for computing the convex hull have been published reaching back to 1970. From a theoretical standpoint, there is not much more to learn about convex hulls. It has already been proven that the lower bound of the asymptotic runtime of any convex hull is $\Omega(n \log n)$ [Yao81]. If an output sensitive algorithm is used, and the number of points on the hull h is less than the total number of points, then there is a known lower bound of $\Omega(n \log h)$. Thus, it is well known that no asymptotically faster convex hull algorithms exist.

However, it is still useful for us to know which algorithms are fastest in practice. Convex hulls are applied in many different scenarios in the real world, most notably in computer vision and computer graphics. In these scenarios, it is not enough to just use an asymptotically fast algorithm. We ideally want to use the algorithm that runs the fastest in real time. This leads into the motivation for this paper: which convex hull algorithm is fastest in practice? To answer this question, I implemented several convex hull algorithms and timed them against point sets generated by different distributions.

2 Algorithms

I implemented seven different algorithms and one heuristic for calculating the convex hull. In this section, I will briefly describe each algorithm and how it works. A common primitive used for convex hulls and in other problems in computational geometry is the *orientation test* [BCKO08]. Intuitively, given 3 points, an orientation test determines if the three points form a left turn or right turn in constant time. I will make references to this primitive throughout this section.

2.1 Graham's Scan

Graham's Scan was published by R. Graham in 1972. The first step of the algorithm is to determine the point p with the lowest y-coordinate. Then all points should be sorted based on the the angle they form with the horizontal line from p. This allows us to iterate over the the points counter clockwise starting from p. We start with the first three points in this new order including p as our initial hull in a stack. Then we attempt to add each new point to the hull in counter clockwise order. If the a new point forms a left turn with the top two points in the stack, it is added to the hull. If it forms a right turn, we pop off the stack until it forms a left turn. Once we iterate through all of the points, the stack will contain the full hull. This algorithm is dominated by the intial sort and runs in $\mathcal{O}(n \log n)$ time [Gra72].

2.2 Andrew's Monotone Chain

Andrew's Monotone Chain was published by A. M. Andrew in 1979. This algorithm is really just a variant of Graham's Scan. Instead of sorting by angle, we sort the points by x-coordinate. Then starting from the leftmost point, we can calculate the upper-hull by looking for right turns and the lower-hull by looking for left turns using the same stack method described above. Finally we concatenate the upper and lower hulls to get the final answers. This algorithm is also dominated by the initial sort and runs in $\mathcal{O}(n \log n)$ time [And79].

2.3 Divide and Conquer

The Divide and Conquer algorithm was published by F. P. Preparata and S. J. Hong in 1977. This is a recursive algorithm that splits the points in half and calculates the hull on the left and right side. The upper and lower hulls are calculated separately similar to Andrew's Monotone Chain. The base case is when you see three or fewer points in which case you can return 2-3 of the points as the upper/lower hull depending on the orientation. To merge the two hulls we find a line that is tangent to both hulls and has all of the points of both hulls on one side of the line. Once again after recursively calculating the upper and lower hulls, we can concatenate them to get the final hull. This algorithm has a similar recurrence to the mergesort recurrence, and thus runs in $\mathcal{O}(n \log n)$ time [PH77].

2.4 Jarvis March

The Jarvis March, otherwise known as gift-wrapping, was created independently by Chand Kapur in 1970 and R. A. Jarvis in 1973. The idea behind this algorithm is to start at the bottom most point and then search through all of the points until the next point on the hull is found. We find the next point on the hull by looking for the point that causes the least amount of "turning" as we move counter-clockwise. This can be computed with two orientation tests. One to make sure the candidate point is turning left (i.e. moving counter-clockwise), and another to make sure the candidate point turns less than the current best candidate. This is an output sensitive algorithm that run in $\mathcal{O}(nh)$ time where h is the number of points on the hull [Jar73].

2.5 Chan's Algorithm

Chan's Algorithm was published by T. M. Chan in 1996. This algorithm combines using an $\mathcal{O}(n \log n)$ algorithm with the Jarvis March. This algorithm guess a value for the number of points on the hull h and then computes n/h mini-hulls by using Graham's Scan or some other algorithm with the same time complexity. These mini-hulls are then combined into a single hull by using a modified version of the Jarvis March where each mini-hull is treated as a "fat" point. If the we have found h points and the hull is still not complete, we stop and restart the algorithm with a greater value of h. It turns out that repeatedly squaring the guess for h leads to $\mathcal{O}(n \log h)$ time where h is the number of points on the final hull. I also tested a variation of Chan's Algorithm where the interior points of each mini-hull are discarded with every iteration since those points can never be on the final hull anyway [Cha96].

2.6 Quickhull

Quickhull was created independently by W. Eddy in 1977 and A. Bykat in 1978. The first step of the algorithm is to find the leftmost (P) and rightmost (Q) points. The pointset is then divided into points above and below $\overline{PQ}$. The upper and lower hulls are then computed separately by calling the recursive portion of Quickhull with the $\overline{PQ}$ and the correct set of points. Within a recursive call of Quickhull, the algorithm searches for the point C which is furthest the line. This point must be on the hull and is added in. The algorithm then divides the points into points above $\overline{PC}$ and $\overline{CQ}$ and then makes two recursive calls of Quickhull for each partition. After the upper and lower hulls are computed they are concatenated together. Much like Quicksort, this algorithm runs in $\mathcal{O}(n \log n)$ time in the average case but degenerates to quadratic time in the worst case [Edd77].

2.7 Symmetric Hull

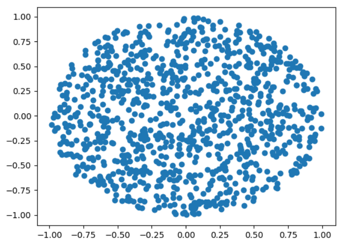
Symmetric Hull was published by A. Beltrán and S. Mendoza in 2018. This algorithm takes advantage of the geometric properties of the Akl-Toussaint heuristic to compute the hull in 4 separate quadrants without using any orientation tests. Points are sorted lexicographically and then 4 most extreme points in each cardinal direction is found. When solving for one of the quadrant hulls (say between the leftmost and topmost point), one can determine the next point by comparing its slope with the last point on the hull with the slope of the last two points on the hull. A stack is used to push and pop off values similar to other algorithms described above. Since calculating a slope is fewer floating point operations than an orientation test, the authors claim this algorithm runs faster than traditional convex hull algorithms. The asymptotic runtime is still $\mathcal{O}(n \log n)$ due to the initial sort [BM18].

2.8 Akl-Toussaint Heuristic

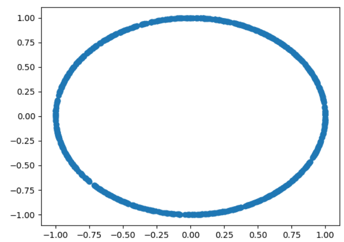
This is a heuristic published in 1981 by L. Devroye and G. T. Toussaint which was based on the algorithm devloped by S. Akl and G. T. Toussaint in 1978. The heuristic says to calculate the four most extreme points in each cardinal direction and then discard all points that lie within the quadrilateral since they clearly cannot be on the final hull. This can clearly be done in linear time, so running this heuristic before running any of the above algorithms will not increase their asymptotic complexity. For my experiment, I tested every algorithm both with and without this heuristic [DT81].

3 Tested Distributions

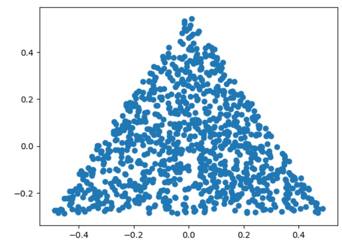
I ran every algorithm on 11 different distributions to test if certain algorithms performed better than others depending on the distribution. Refer to Figure 1 for an image of each distribution.



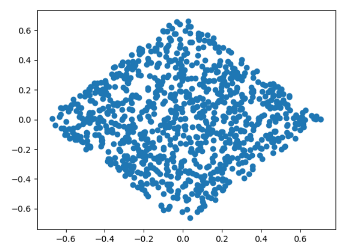
(a) Uniform Disk



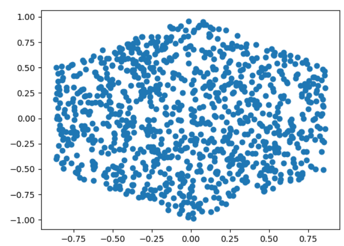
(b) Uniform Circle



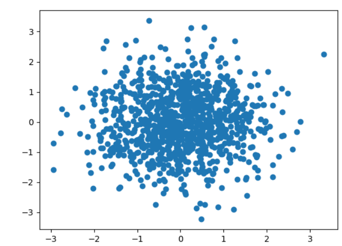
(c) Uniform Triangle



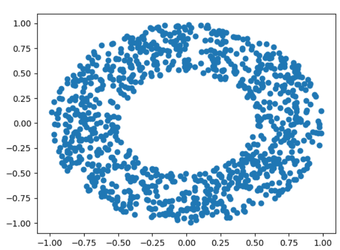
(d) Uniform Square



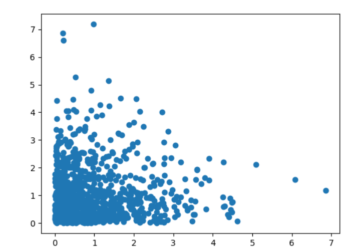
(e) Uniform Hexagon



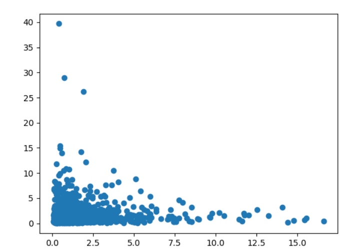
(f) Gaussian



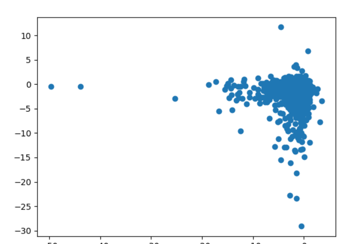
(g) Uniform Annulus



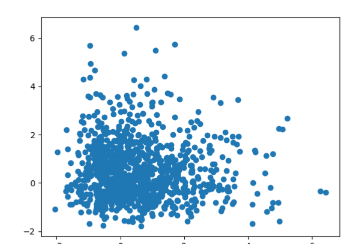
(h) Exponential



(i) Lognormal



(j) Johnson SU



(k) Extreme Value

Figure 1: Distributions used for the experiment

4 Results

I implemented every algorithm described in Section 2 in Python 3.7.4 [1]. The algorithms were timed on an Intel i5-6300HQ CPU @ 2.30 GHz. In Tables 1, 2, and 3, I present the average runtime of for each algorithm with and without the Akl-Toussaint heuristic over five runs on each distribution. Within each table, the fastest three times for each distribution are **bolded**. The largest improvement due to the Akl-Toussaint heuristic is *italicized*. Each successive table increases the number of points by an order of magnitude. "N/A" in certain entries signifies that I did not test the algorithm on that distribution with that number of points because the previous test was already slow.

Firstly, the obvious result here is that SymmetricHull is significantly faster than all of the other algorithms. This algorithm averages well under half a second on even 10^6 points for every distribution. For any practical case of computing a convex hull, SymmetricHull is the clear choice compared to any of these other algorithms.

The next thing to notice is how the Akl-Toussaint heuristic affected the runtimes of various algorithms. Notably, it did not improve the runtime of most of the algorithms. For SymmetricHull this makes sense considering the entire idea behind the algorithm was to eliminate the orientation test to minimize the amount of floating point computation. The Akl-Toussaint heuristic reintroduces the orientation test to the algorithm and thus slows it down. For the other algorithms it was more surprising since they all use the orientation test. In general, it seems that the Jarvis March, Chan's Algorithm, and Quickhull benifited from the heuristic while it affected the others negatively. Jarvis seems to benefit the most in terms of pure speedup, but this is also due to the fact that Jarvis has the worst initial running time of all of the algorithms on all of the tested distributions.

Chan's algorithm, which is asymptotically the fastest of all of the algorithms presented, did not come close to SymmetricHull. However, when combined with the Akl-Toussaint heuristic, it did seem to show performance comparable to the all of the other algorithms. For the original runs, I used Graham's Scan to compute the mini-hulls in Chan's algorithm. Since Chan's algorithm can make use of any $\mathcal{O}(n \log n)$ algorithm, I wanted to see if using SymmetricHull instead would speed it up.

Tables 4, 5, and 6 show the result of this follow up experiment. I also added two rows for SymmetricHull itself to see the comparison against the best algorithm. Once again, I have bolded the top 3 times for each distribution tested. In general, using SymmetricHull instead of Graham's Scan seemed to trend towards better performance. For example, on a uniform disk, using SymmetricHull seems to significantly improve performance. For other distributions the change is less pronounced. Using SymmetricHull was worse on uniform circle and using both SymmetricHull and the Akl-Toussaint heuristic with the modified version of Chan's algorithm seemed to be worse than using Graham's Scan version. Regardless, none of the variations of Chan's was able to come close to SymmetricHull.

5 Conclusion and Future Work

After evaluating the algorithms presented in this paper, it is clear that SymmetricHull is the best choice for computing convex hulls in practice for $n \leq 10^6$. SymmetricHull will definitely outperform all of the $\mathcal{O}(n \log n)$ algorithms for larger n as well since the experimental data suggests that SymmetricHull has a smaller constant. It would be interesting to see if some variation of Chan's algorithm could achieve similar or even better performance than SymmetricHull given a large

[1] Implentation hosted here: https://github.com/dhruvnm/convex

enough input size. Given more time and resources, this would be an interesting experiment to run, but considering SymmetricHull was running at least 5 times faster than any variation of Chan's algorithm even at 10^6 points, I hypothesize SymmetricHull will be faster for quite a few orders of magnitude beyond 10^6.

	Unif Disk	Unif Circle	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Graham	**0.009649**	**0.00576**	**0.00867**	**0.008723**	**0.009111**	**0.005599**
Graham w/A-T	0.01533	0.014174	0.012496	0.011891	0.01343	0.007094
Andrew	0.013083	**0.009456**	**0.012134**	0.012202	0.012364	0.007327
Andrew w/A-T	0.01545	0.017431	0.012319	0.012284	0.014793	0.007707
Div+Conq	0.015418	0.010452	0.013851	0.01402	0.013954	0.008755
Div+Conq w/A-T	0.016372	0.01819	0.012637	0.01234	0.015331	0.007283
Jarvis	0.184523	6.293434	0.104806	0.111553	0.152664	0.038498
Jarvis w/A-T	0.089413	6.306596	0.027964	0.023137	0.074271	0.011996
Chan	0.15736	0.480714	0.113563	0.11832	0.121227	0.040911
Chan w/A-T	*0.053571*	0.459062	*0.017637*	*0.02105*	*0.045643*	*0.009738*
Chan Mod	0.084854	0.421134	0.086167	0.08289	0.081373	0.043045
Chan Mod w/A-T	0.050682	0.482962	0.024873	0.024117	0.051158	0.011641
Quickhull	0.041641	0.229173	0.027751	0.032947	0.040817	0.027132
Quickhull w/A-T	0.027375	*0.163608*	0.014341	0.015182	0.024845	0.009512
SymmetricHull	**0.002024**	**0.003042**	**0.00229**	**0.002212**	**0.002116**	**0.001573**
SymmetricHull w/A-T	**0.012322**	0.010353	0.012143	**0.011348**	**0.011017**	**0.006737**

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Graham	**0.008561**	**0.005174**	**0.005476**	**0.009092**	**0.009737**
Graham w/A-T	0.014425	0.008505	0.007702	**0.01178**	0.012528
Andrew	0.012085	0.00726	0.00759	0.012494	**0.012089**
Andrew w/A-T	0.015894	0.009453	0.008112	0.011804	0.012738
Div+Conq	0.013938	0.009639	0.009085	0.013646	0.014098
Div+Conq w/A-T	0.016299	0.009924	0.008287	0.011951	0.012712
Jarvis	0.22276	0.050296	0.036193	0.043386	0.092115
Jarvis w/A-T	0.132671	0.033587	0.015619	0.013717	0.023627
Chan	0.10225	0.046474	0.047804	0.057429	0.060484
Chan w/A-T	0.060462	0.025129	0.013706	0.012744	0.017469
Chan Mod	0.11415	0.049532	0.047878	0.07258	0.08417
Chan Mod w/A-T	*0.063254*	0.023339	*0.013088*	*0.012644*	*0.021619*
Quickhull	0.059286	0.039045	0.035817	0.037227	0.027104
Quickhull w/A-T	0.033884	*0.016916*	0.011433	0.012249	0.015096
SymmetricHull	**0.002233**	**0.001491**	**0.001612**	**0.001835**	**0.001998**
SymmetricHull w/A-T	**0.010801**	**0.006922**	**0.007068**	0.011853	0.01211

Table 1: Average time in seconds on 10^4 points

	Unif Disk	Unif Circle	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Graham	**0.098999**	**0.077456**	0.119417	**0.108702**	**0.098574**	**0.064943**
Graham w/A-T	0.157455	0.138312	**0.11537**	0.112109	0.133554	0.077445
Andrew	0.127166	**0.100595**	0.135386	0.12632	0.124114	0.082593
Andrew w/A-T	0.179345	0.175926	**0.114428**	**0.111687**	0.145935	0.082283
Div+Conq	0.165363	0.108508	0.153522	0.14741	0.142036	0.105916
Div+Conq w/A-T	0.149335	0.178044	0.122377	0.135677	0.157259	0.068674
Jarvis	3.985723	N/A	1.291311	1.33936	1.979021	0.492064
Jarvis w/A-T	*1.636059*	N/A	*0.145752*	*0.155862*	*0.707876*	*0.078485*
Chan	1.090195	4.629648	1.001763	0.960337	0.98759	0.440343
Chan w/A-T	0.518169	*4.400622*	0.14062	0.136956	0.44824	0.081611
Chan Mod	0.862825	4.348852	0.788255	0.834802	0.843527	0.408744
Chan Mod w/A-T	0.463924	4.372306	0.13882	0.132573	0.412936	0.081353
Quickhull	0.300867	2.092958	0.21892	0.222211	0.282914	0.17751
Quickhull w/A-T	0.278187	2.135659	0.130551	0.12465	0.240774	0.077808
SymmetricHull	**0.028441**	**0.036393**	**0.025652**	**0.025731**	**0.025411**	**0.019999**
SymmetricHull w/A-T	**0.116143**	0.111238	0.121886	0.118083	**0.114799**	**0.072971**

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Graham	**0.127858**	**0.059887**	**0.058978**	**0.094959**	**0.09287**
Graham w/A-T	0.148259	0.075153	0.07558	0.122577	0.126316
Andrew	0.171235	0.10337	0.081407	0.140029	0.127023
Andrew w/A-T	0.199004	0.09682	0.087964	0.122894	0.121586
Div+Conq	0.151169	0.092176	0.090239	0.141724	0.135995
Div+Conq w/A-T	0.173387	0.077663	0.079082	**0.110436**	**0.115137**
Jarvis	4.878911	0.66702	0.50799	0.345135	0.804169
Jarvis w/A-T	*2.665971*	*0.149403*	*0.133289*	0.116237	*0.156387*
Chan	1.099122	0.509636	0.417495	0.41203	0.679262
Chan w/A-T	0.652603	0.140989	0.128494	0.120974	0.13974
Chan Mod	0.907298	0.451488	0.398651	0.407192	0.645533
Chan Mod w/A-T	0.584919	0.132766	0.125527	*0.117719*	0.13848
Quickhull	0.330158	0.195804	0.201121	0.209223	0.220111
Quickhull w/A-T	0.325875	0.101113	0.110116	0.117144	0.131249
SymmetricHull	**0.024934**	**0.018065**	**0.017008**	**0.025228**	**0.023004**
SymmetricHull w/A-T	**0.109815**	**0.071885**	**0.072308**	0.116279	0.118256

Table 2: Average time in seconds on 10^5 points

	Unif Disk	Unif Circle	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Graham	**1.095789**	**0.783676**	**1.096937**	**1.099022**	**1.086555**	0.718688
Graham w/A-T	1.443012	1.545065	**1.187467**	**1.160641**	1.41593	**0.716805**
Andrew	1.333447	**1.048083**	1.476658	1.38179	1.340574	0.853012
Andrew w/A-T	1.515785	1.852827	1.249764	1.169353	1.532108	**0.716035**
Div+Conq	1.502333	1.132524	1.934084	1.492417	1.50803	0.961925
Div+Conq w/A-T	1.584914	1.904875	1.206434	1.174092	1.591325	0.721565
Jarvis	91.2689	N/A	15.14653	15.50658	24.28271	5.363527
Jarvis w/A-T	*35.92466*	N/A	*1.319085*	*1.299456*	*9.335436*	*0.820929*
Chan	12.85858	81.02195	8.820718	9.612595	10.12924	4.770879
Chan w/A-T	6.515399	82.03259	1.265462	1.24395	4.739971	0.79344
Chan Mod	9.587833	81.55107	7.701913	8.70033	9.379771	4.662617
Chan Mod w/A-T	5.543736	82.73623	1.242253	1.23921	4.235208	0.784204
Quickhull	2.990062	27.21007	2.427607	2.141164	2.718453	1.773051
Quickhull w/A-T	2.901514	28.46199	1.21426	1.192682	2.355108	0.764131
SymmetricHull	**0.346876**	**0.489063**	**0.342821**	**0.39136**	**0.328705**	**0.234634**
SymmetricHull w/A-T	**1.217069**	1.276877	1.205508	1.199868	**1.403314**	0.730052

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Graham	**1.093555**	**0.727376**	**0.727267**	**1.077704**	**1.147844**
Graham w/A-T	1.480208	0.809693	0.908942	1.14573	**1.2102**
Andrew	1.406805	0.855042	0.863112	1.349753	1.390386
Andrew w/A-T	1.631363	0.838189	0.988371	1.158485	1.219887
Div+Conq	1.502106	0.976608	0.971131	1.478084	1.483789
Div+Conq w/A-T	1.691314	0.864024	1.033708	1.152973	1.235454
Jarvis	99.15347	7.549983	4.965297	2.708198	8.926097
Jarvis w/A-T	*49.70838*	*2.092856*	*2.668891*	1.181139	*1.532366*
Chan	13.73066	5.813109	4.605394	3.606551	7.853224
Chan w/A-T	7.762465	2.146377	2.528007	1.17274	1.468046
Chan Mod	11.31171	5.574911	4.617684	3.753927	7.704738
Chan Mod w/A-T	6.697954	1.697647	2.423917	*1.167141*	1.454568
Quickhull	3.294849	2.259551	2.167258	2.054201	2.314305
Quickhull w/A-T	3.560901	1.273497	1.69163	**1.13192**	1.365554
SymmetricHull	**0.323596**	**0.276506**	**0.418172**	**0.432527**	**0.340725**
SymmetricHull w/A-T	**1.17218**	**0.731178**	**0.757835**	1.187266	1.318459

Table 3: Average time in seconds on 10^6 points

	Unif Disk	Unif Circle	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Chan w/GS	0.17522	0.405504	0.074074	0.087852	0.093917	0.040704
Chan w/GS+A-T	0.096073	0.423661	0.017034	0.020116	0.04971	0.010604
Chan w/SH	0.08036	0.458335	0.07751	0.072847	0.077616	0.036927
Chan w/SH+A-T	**0.046744**	0.422112	0.017257	**0.018905**	0.043724	**0.010603**
Chan Mod w/GS	0.086213	**0.372278**	0.068449	0.077772	0.082286	0.03814
Chan Mod w/GS+A-T	0.049301	0.383399	**0.016908**	0.019438	0.042763	0.010287
Chan Mod w/SH	0.075325	0.415031	0.070253	0.071445	0.07226	0.036792
Chan Mod w/SH+A-T	0.046868	0.425942	0.018185	0.019252	**0.041266**	0.010778
SymmetricHull	**0.002198**	**0.003974**	**0.002687**	**0.002428**	**0.002023**	**0.001636**
SymmetricHull w/A-T	**0.013036**	**0.010567**	**0.012279**	**0.011604**	**0.01083**	**0.007127**

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Chan w/GS	0.09992	0.04647	0.038208	0.075088	0.107717
Chan w/GS+A-T	0.060454	0.02135	0.015826	0.012056	0.01782
Chan w/SH	0.079531	0.044587	0.036939	0.0425	0.054461
Chan w/SH+A-T	0.054919	0.020316	0.016186	**0.011922**	**0.017596**
Chan Mod w/GS	0.08699	0.043292	0.037075	0.042115	0.058085
Chan Mod w/GS+A-T	0.057645	**0.019859**	**0.015537**	**0.011957**	0.018121
Chan Mod w/SH	0.075792	0.043479	0.038157	0.041364	0.054063
Chan Mod w/SH+A-T	**0.05377**	0.020357	0.016518	0.012592	0.017963
SymmetricHull	**0.002142**	**0.001302**	**0.001615**	**0.00203**	**0.002013**
SymmetricHull w/A-T	**0.01076**	**0.007167**	**0.007189**	0.012316	**0.011347**

Table 4: Average time in seconds on 10^4 points

	Unif Disk	Unif Circle	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Chan w/GS	1.022499	4.329926	0.844371	0.993385	0.95675	0.42269
Chan w/GS+A-T	0.510449	4.585392	0.138197	0.138857	0.444088	0.083544
Chan w/SH	0.833755	4.862556	0.789763	0.772015	0.773288	0.384185
Chan w/SH+A-T	0.4753	4.926181	0.1381	0.135982	0.405872	0.082725
Chan Mod w/GS	0.851632	**4.277343**	0.72674	0.806218	0.829407	0.39099
Chan Mod w/GS+A-T	0.457973	4.349176	**0.136712**	0.137456	0.414057	**0.079596**
Chan Mod w/SH	0.732645	5.194565	0.776968	0.738274	0.733671	0.392188
Chan Mod w/SH+A-T	**0.449589**	4.880574	0.137291	**0.13509**	**0.388236**	0.081222
SymmetricHull	**0.026624**	**0.035431**	**0.02258**	**0.023832**	**0.025316**	**0.017899**
SymmetricHull w/A-T	**0.111361**	**0.107022**	**0.117942**	**0.115213**	**0.112515**	**0.071009**

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Chan w/GS	1.129867	0.508219	0.469818	0.797634	0.7288
Chan w/GS+A-T	0.667624	0.207604	0.173259	0.117726	0.190808
Chan w/SH	0.916285	0.461546	0.410953	0.448781	0.596435
Chan w/SH+A-T	0.618948	0.200546	0.174827	**0.116116**	0.192757
Chan Mod w/GS	0.893029	0.466509	0.410686	0.472679	0.645776
Chan Mod w/GS+A-T	**0.568446**	**0.191534**	**0.163156**	**0.117239**	**0.1922**
Chan Mod w/SH	0.834474	0.455272	0.409317	0.454475	0.615736
Chan Mod w/SH+A-T	0.584934	0.204094	0.174645	0.117585	0.205978
SymmetricHull	**0.0241**	**0.019322**	**0.01765**	**0.024251**	**0.024505**
SymmetricHull w/A-T	**0.108226**	**0.070352**	**0.070382**	0.117615	**0.123968**

Table 5: Average time in seconds on 10^5 points

	Unif Disk	Unif 3-gon	Unif 4-gon	Unif 6-gon	Gaussian
Chan w/GS	1.07409	0.878722	0.998325	1.045219	0.421924
Chan w/GS+A-T	0.527773	0.145791	0.142237	0.464863	0.082642
Chan w/SH	0.873434	0.809012	0.809064	0.788248	0.398947
Chan w/SH+A-T	0.489393	**0.139632**	**0.140602**	**0.408987**	**0.081656**
Chan Mod w/GS	0.891718	0.7802	0.840499	0.860629	0.401907
Chan Mod w/GS+A-T	**0.478857**	0.141513	0.14415	0.426564	0.082749
Chan Mod w/SH	0.751675	0.803085	0.755989	0.751915	0.404161
Chan Mod w/SH+A-T	0.544502	0.147172	0.14839	0.426637	0.08894
SymmetricHull	**0.024359**	**0.023643**	**0.024229**	**0.024575**	**0.016745**
SymmetricHull w/A-T	**0.113339**	**0.121618**	**0.118151**	**0.115967**	**0.069771**

	Unif Annulus	Exponential	Lognormal	Johnson's SU	Extreme Value
Chan w/GS	1.117635	0.521302	0.407456	0.456726	0.675934
Chan w/GS+A-T	0.667244	0.194103	0.147158	0.121281	0.155597
Chan w/SH	0.945644	0.470657	0.414348	0.432159	0.620597
Chan w/SH+A-T	0.644071	0.185991	0.147394	**0.118649**	**0.153699**
Chan Mod w/GS	0.949496	0.461746	0.414272	0.441388	0.649226
Chan Mod w/GS+A-T	**0.592008**	**0.175235**	**0.141453**	0.119157	0.154651
Chan Mod w/SH	0.845597	0.462796	0.47683	0.556289	0.723994
Chan Mod w/SH+A-T	0.836784	0.186095	0.149111	0.122047	0.155477
SymmetricHull	**0.024532**	**0.018154**	**0.018588**	**0.025565**	**0.023965**
SymmetricHull w/A-T	**0.110179**	**0.070209**	**0.070735**	**0.118637**	**0.127264**

Table 6: Average time in seconds on 10^6 points

References

[And79] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Inf. Process. Lett.*, 9:216–219, 1979.

[BCKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[BM18] A. Beltrán and Sonia Mendoza. Symmetrichull: A convex hull algorithm based on 2d geometry and symmetry. *IEEE Latin America Transactions*, 16:2289–2295, 2018.

[Cha96] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Computational Geometry*, 16:361–368, 1996.

[DT81] L. Devroye and G. Toussaint. A note on linear expected time algorithms for finding convex hulls. *Computing*, 26:361–366, 1981.

[Edd77] W. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3:398–403, 1977.

[Gra72] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1:132–133, 1972.

[Jar73] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inf. Process. Lett.*, 2:18–21, 1973.

[PH77] F. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[Yao81] A. Yao. A lower bound to finding convex hulls. *J. ACM*, 28:780–787, 1981.