COMP0005 Group Coursework
Academic Year 2023-24 Group Number: 26

Gesture Recognition via Convex Hull

1 Overview of Experimental Framework

1.1 Framework Design/Architecture

Test Data Generator

The TestDataGenerator class contains methods to generate two different types of data: random points and convex polygon points with fixed h points on the convex hull. The generate_random_points method takes the number of points to generate as an argument and returns a list containing that many unique points. The generate_convex_polygons method has parameters for the number of points and an optional h value, if h isn't specified then it will be equal to the number of points. We then use Valtr's algorithm to randomly generate a convex polygon with h vertices where all generated points are on the convex hull. If h is specified, then the remaining points (number of points - h) are placed randomly within the polygon. This is accomplished by generating random points and adding them only if they are within the hull which we check using a ray casting method. All points generated are integers in the range [0, 32767].

Experimental Framework

The ExperimentalFramework class is designed to provide an abstraction to comprehensively test each algorithm under various circumstances. To achieve this there are three types of experiments that can be run on the algorithms, each using different types of data. There is a random experiment which tests the runtime as n varies, a convex experiment that does the same but for data where all points lie on the convex hull, and an experiment where h is varied under a constant n to test how the output size impacts the performance of each algorithm. To minimise repeating code, each experiment method calls the run_experiment method with different arguments which handles the running of the experiment and stores the results in a corresponding dictionary with algorithm names as its keys. Additionally, every experiment method has an optional parameter 'repeats' which if given, will repeat the experiment with newly generated points on every algorithm. The mean time of these repeats is then used to reduce uncertainty. For the results presented in this report every experiment has been run using 10 repeats. Moreover, each algorithm is compared on the same data by storing the points used on the first algorithm which allows for a fairer comparison than generating new points. To visualise the results from the experiments, the framework also contains two methods that plot either h or n against the mean runtime of the algorithms. One of these methods is used to compare different data sets against each other on a single algorithm whereas the other compares how the three algorithms perform on a specific data set.

1.2 Hardware/Software Setup for Experimentation

All results in this report were obtained using Python 3.11.8 running in a Jupyter Notebook environment in Visual Studio Code. The system used to run the experiments included, Windows 11 Home Edition OS, Intel i7 CPU @2.30 GHz, and 16.0GB RAM.

2 Performance Results

2.1 Jarvis march algorithm

Average Case

Jarvis march has a time-complexity of O(nh), where n is the number of points in the input set and h is the number of the points on the convex hull. To obtain results for the average case scenario we randomly generate n unique 2D integer points in the range [0, 32767].

With random points we theorise that Jarvis march will tend towards a linear time complexity as n grows with some variation when h differs. This is because with random data, h will be significantly smaller than n when n is large.

Worst Case

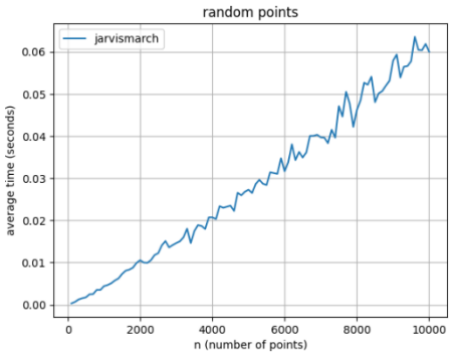
The worst-case scenario occurs when all points lie on the convex hull. This would mean that h = n so the time complexity in the worst-case scenario for Jarvis march is predicted to be $O(n^2)$. To trigger the worst-case scenario, we generate a random convex polygon with n vertices such that all vertices are part of the convex hull.

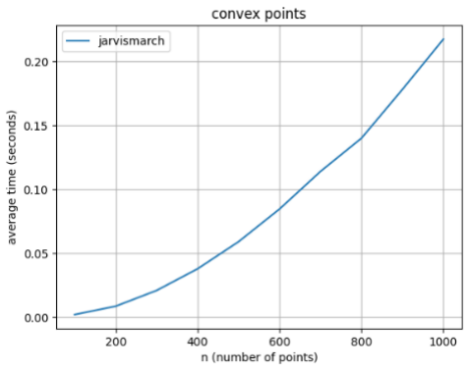
Analysis

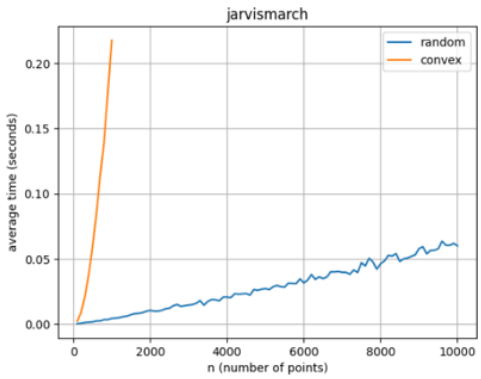
Jarvis march – Random points	
N (number of points)	Average time taken (seconds)
100	0.00030483
500	0.00173238
1000	0.00440057
5000	0.02726475
10000	0.05998934

Jarvis march – Convex points	
N (number of points)	Average time taken (seconds)
100	0.00228699
500	0.05917632
1000	0.21737064

*Experimented up to 1000 points for better comparison with random points







From our results we can see that Jarvis march shows a linear behaviour for random data and quadratic for convex points which matches our predictions.

Comparing the different data sets, it is evident that random data significantly outperforms convex data. Due to Jarvis march's quadratic growth with convex data, the algorithm is only suitable for use when the size of the input is small, or the input is guaranteed to have a small h relative to n.

2.2 Graham scan algorithm

Average Case

Graham scan has a time complexity of O(nlogn) where n is the number of points in the input set. This comes from the sort that is performed on the polar coordinates of the input points – our implementation uses merge sort which has an average and worst-case time complexity of O(nlogn). Similar to Jarvis march, we generate random points to test the average time of Graham scan as n varies.

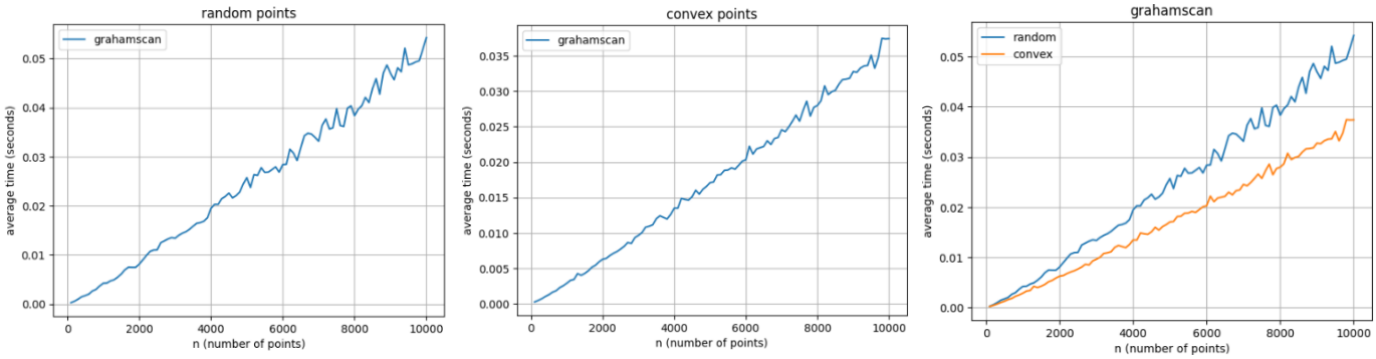
Worst Case

Unlike Jarvis march and Chan's, Graham scan is not an output-sensitive algorithm as its time complexity is O(nlogn) in all cases and is not affected by h. Although certain data sets may lead to the sorting algorithm performing more comparisons, there is no case that performs significantly worse for Graham scan. Therefore, the time taken for convex points where every point is on the convex hull should be very similar to the time taken for random points.

Analysis

Graham scan – Random points	
N (number of points)	Average time taken (seconds)
100	0.00028712
500	0.00174775
1000	0.00424273
5000	0.02574489
10000	0.05422192

Graham scan – Convex points	
N (number of points)	Average time taken (seconds)
100	0.00025038
500	0.00128801
1000	0.00289863
5000	0.01708513
10000	0.03740845



As expected, the behaviour of Graham scan is linearithmic for both random and convex points. Comparing the performance of the two data sets we observe that convex points slightly outperform random points. A possible explanation for this behaviour is that the number of pop operations on the stack is reduced as convex points form a continuous left turn. Additionally, with convex data we control the values of both n and h whereas with random data h can vary which is likely the reason why the times for convex points are more consistent.

2.3 Chan's Algorithm

Average Case

Chan's algorithm has a time complexity of O(nlogh) where n is the number of points in the input set and h is the number of points on the convex hull. An estimate m is made for h, then the Graham scan part takes O(nlogm) as an O(mlogm) Graham scan is performed on each of the n/m subsets of the input set.

Then the Jarvis tangent step also is O(mlogm) as the tangent algorithm uses binary search which is logarithmic and for each m point in the hull the tangent is found for all n/m subsets. The implementation uses a logarithmic complexity guessing approach to find the value of m, until it is >= h. The overall complexity ends up as O(nlogh) from this. We generate random points in the average case scenario the same way we do for Jarvis march and Graham scan.

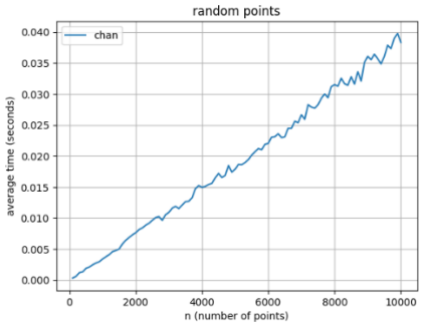
Worst Case

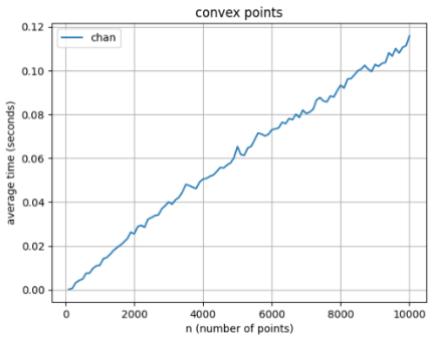
The worst-case scenario occurs when h = n and so all points lie on the convex hull. We generate convex data to verify that this case has an O(nlogn) time complexity.

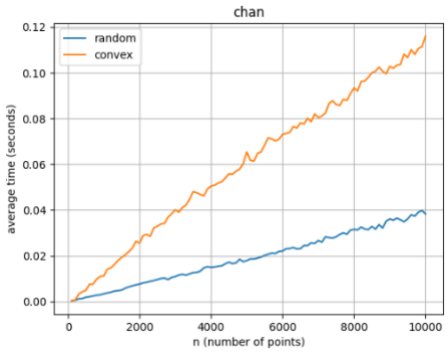
Analysis

Chan's – Random points	
N (number of points)	Average time taken (seconds)
100	0.00032821
500	0.00187998
1000	0.00338614
5000	0.01787886
10000	0.03833112

Chan's – Convex points	
N (number of points)	Average time taken (seconds)
100	0.00029498
500	0.00492319
1000	0.01120346
5000	0.06531511
10000	0.11576328

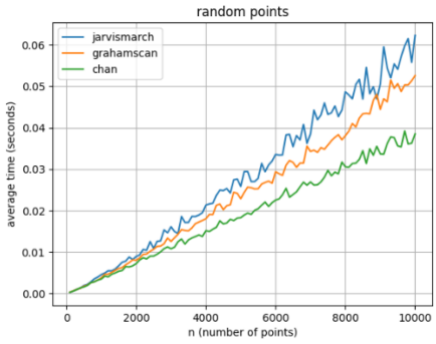


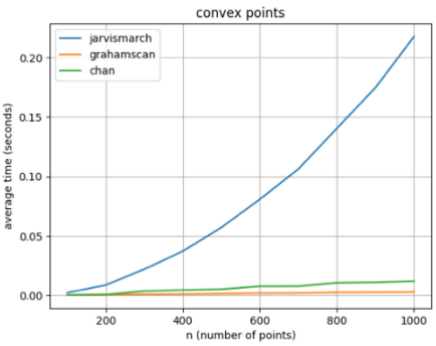


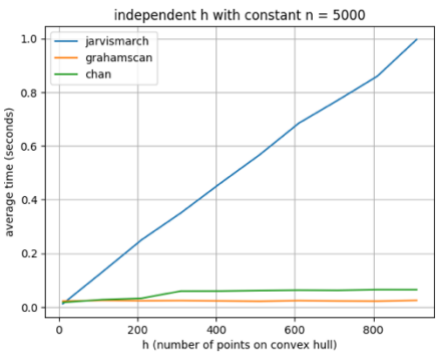


We can observe that Chan's worst case is in fact h = n, making it perform somewhat worse than random data. Still, the worst case only has O(nlogn) time complexity making it suitable for large values of n.

3 Comparative Assessment







As shown by the data, the average running time of the following algorithms goes from (highest to lowest) Jarvis march, Graham scan and Chan's. Considering the theoretical computational time complexities of these algorithms (O(nh), O(nlogn), O(nlogh) respectively), we can conclude that the algorithms in our results do in fact

follow this order of growth. Although the run times for random points are still relatively close, they will likely grow more distant as n increases. Due to this, Chan's is the recommended algorithm to maximise the efficiency of calculating the convex hull of random points.

An interesting observation to be made is that Graham scan outperforms Chan's in an environment of convex points. Graham scan tends to do better in an environment of convex points as it is theoretically the "best-case" for the Graham scan, while as Chan's incorporates the Jarvis march algorithm, it tends to do worse in an environment of convex points. However, even though Graham scan does marginally beat Chan's with convex points, the difference in performance is insignificant at best, especially compared to the Jarvis march algorithm which has the worst-case time complexity in an environment of convex points. This is because the time-complexity of Graham scan and Chan's in the worst case is O(nlogn) vs Jarvis march which is quadratic $O(n^2)$. Thereby it makes sense that there is a marginal difference between the Graham Scan and Chans algorithms while an apparent discrepancy between the Jarvis march algorithm.

Another observation is to be made when n is set at a constant value and the h-value is increased. As Graham scan is not output sensitive, it is seen that increasing the value of h has close to no impact on the average run time of the algorithm. And although Chan's is output sensitive, the effect is insignificant due to the logarithmic complexity (logh). Contrarily, increasing the h value increases the average run-time of Jarvis march significantly as the h-value affects the Jarvis march algorithm linearly.

Conclusion

According to our experimental analysis, in general, Chan's algorithm is the most efficient due to its lowest average-time complexity. Even though in an environment of convex points the Graham scan algorithm tends to have marginally lower average running time, the difference in average run time between the two algorithms in that specific environment is too marginal to discredit the superiority of Chan's time complexity in the average-case. While both Graham scan and Chan's are reliable algorithms for computing the convex hull, Jarvis march is only comparable to them when h is very small relative to n. Otherwise, it should not be used as not only is it the slowest in the average case, but it also becomes very inefficient when the convex hull contains most of the input points.

Based on experimental testing and data, Group 26 has concluded that the Chan's algorithm is most efficient (in terms of time-complexity) for convex-hull recognition and thereby is the most efficient algorithm to identify gestures.

4 Team Contributions

Student Name	Student Portico ID	Key Contributions	Share of Work
Nicholas Blandos	23109687	Experimental Framework/Test Data Generator, part of Chan's algorithm, helped with report	40%
Rivan Chanian	23002028	Graham scan algorithm, part of Chan's algorithm, helped with report.	30%
Junho Daniel Chey	23025478	Jarvis march algorithim, initial Experimental Framework/Test Data Generator, helped with report	30%
Sadman Chowdhury	22167989	N/A	0%