Project P2 : Set 1 (Pre-MidTerm Break)

[1]Arjun Verma : IMT2017008
CS/DS 715: Computational Geometry
Course Instructor : Prof. Pradeesha Ashok
Project Report

Abstract. *This technical report contains a brief overview of the methodology involved in implementing the Graham Scan algorithm and the Divide and Conquer algorithm to compute the convex hull of n points in the plane. It then attempts to compare the performance of the two algorithms for different values of n. The code has also been modified so that different stages of the algorithm can be visualised.*

1. Tools and Methodology

1.1. Tools

The following subsections throw a light on the exhaustive set of libraries used for coding the algorithms and generating the visualizations.

1.1.1. For the Interfaced Solution

- math (atan2, degrees)
- functools
- numpy
- PyQt5.QtCore

1.1.2. For Non-Interfaced Solution

- PIL
- pylab (ginput)
- numpy
- matplotlib
- seaborn

Only the additional libraries used in the Non-Interfaced solution have been mentioned. Some of the common libraries from the Interfaced Solution have been omitted to remove redundancy.

1.2. Methodology

In this section, we provide the distinct characteristics involved in implementing each technique.

1.2.1. Graham Scan Algorithm

Distinct characteristics of the implemented Graham Scan algorithm are as follows:

- Given a set of points, we first find the anchor point. This is the point with the lowest y-coordinate value. If there are more than one such points, we find the one with the lowest x-coordinate.
- We now find the polar angles that all other points make with this anchor point. The points are then sorted according to these computed polar angles. If more than one point makes the same polar angle, we sort the points by the distance from the anchor point.
- We now initialize the final convex hull array with the anchor point and the first element in the sorted array.
- We now iterate over each point in the sorted array and see if traversing to a point from the previous two points makes a clockwise or a counter-clockwise direction. If it makes a clockwise turn, then reject the point and move on to the next point. Continue this till the end of the sorted array.

1.2.2. Divide and Conquer Algorithm

Distinct characteristics of the implemented Divide and Conquer algorithm are as follows:

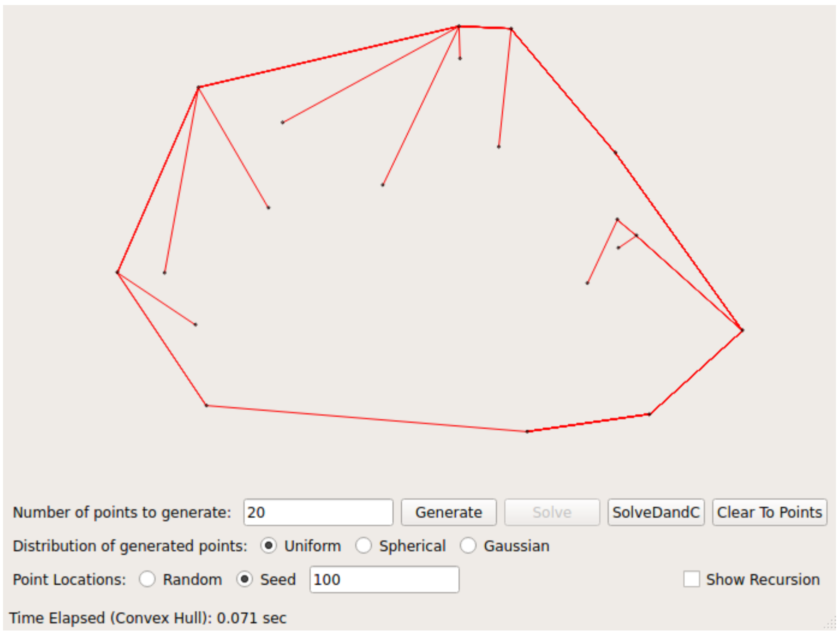
- Given a set of points P, we split the set into two more or less equal halves P1 and P2 and compute the convex hulls for these set of points recursively and call them C1 and C2. This is the divide step. The merge step is to simply combine the recursively computed convex hulls C1 and C2 by joining them along their upper and lower tangents. Note, the divide step only continues as long as there are six or more points in the step For five or less points, brute force solution is used to compute the hull.
- We begin by finding the upper tangent. We take the rightmost point of the left convex hull C1 and the leftmost point of the right convex hull C2 and join them by a line. If this line now crosses through the polygon C2, we move the line up by changing the end point on C2 to the next point upward. This step is repeated till the line does not cross through C2. Once it does not, we now check whether this line passes through C1 and if it does, shift the end point on C1 one point upward. These steps are repeated till we find the upper tangent.
- The lower tangent is computed in the exact same manner, only we now move inversely. That is, if we find that the line is passing through the polygon, we move one point downward.

2. Sample Visualizations

2.1. Interfaced Implementation

Distinct characteristics of the interfaced implementation are as follows:

- You are given a text box to input as many number of points you want
- You are given the option to choose the distribution by which you wish to generate these points. The options for choosing a distribution are either Uniform, Gaussian or Spherical. You may or may not choose a seed. Simply click on *Generate* after choosing appropriate settings.
- After generating the points, you can now choose the algorithm by which you wish to compute the convex hull. To implement the Graham Scan algorithm, click on *Solve* meanwhile to choose the Divide and Conquer algorithm, click on *SolveDandC*.
- Maybe after you have seen the solution to the problem, you now wish to see the recursive steps on the same set of points. To do this, simply click on *Clear to Points* and check the *Show Recursion* checkbox. Now, click on the implementation of the algorithm whose recursive steps you wish to visualize as previously mentioned in point 2 and you are good to go.
- The time taken to execute the algorithm is displayed in the lower left half of the interface.



Number of points to generate: 20

Distribution of generated points: [✓ Uniform] Spherical Gaussian

Point Locations: Random [✓ Seed] 100

Show Recursion

Time Elapsed (Convex Hull): 0.071 sec

Figure 1. n : 20, Distribution : Uniform, Algorithm : GrahamScan

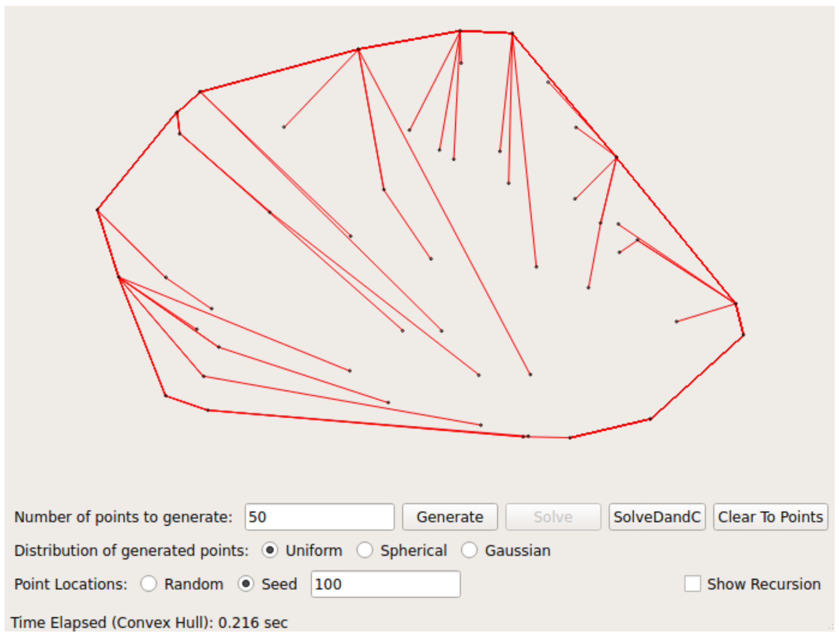


Figure 2. n : 50, Distribution : Uniform, Algorithm : GrahamScan

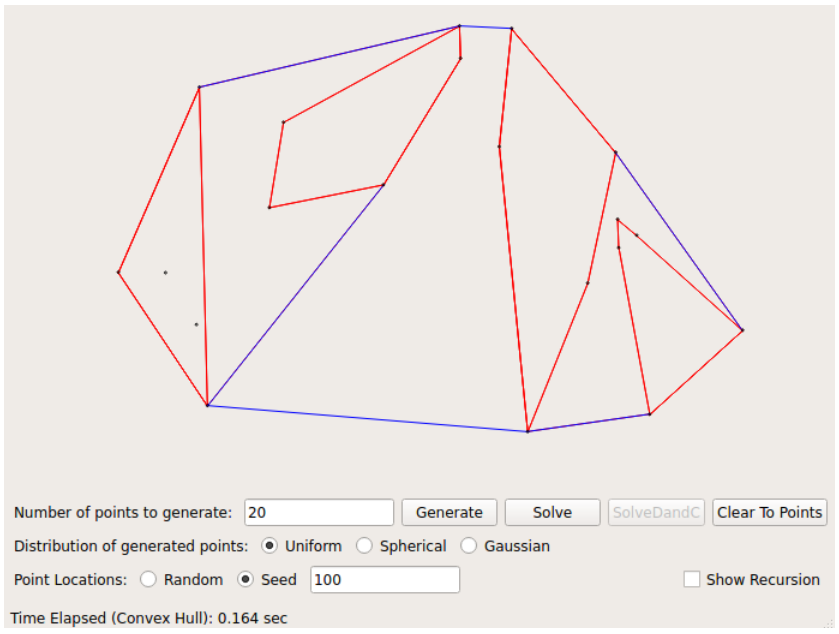
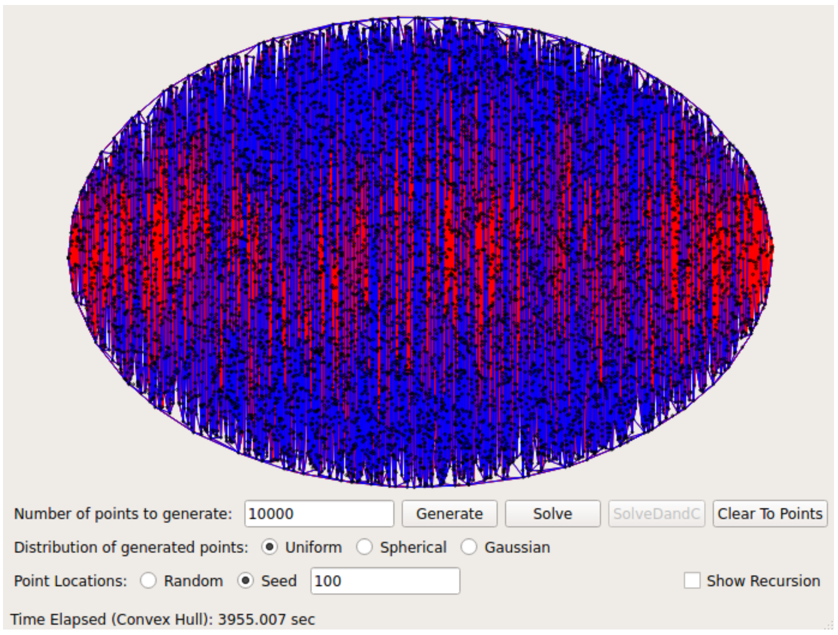


Figure 3. n : 20, Distribution : Uniform, Algorithm : Divide and Conquer


Number of points to generate: 10000

Distribution of generated points: [✓ Uniform] Spherical Gaussian

Point Locations: Random [✓ Seed] 100

Show Recursion

Time Elapsed (Convex Hull): 3955.007 sec

Figure 4. n : 10000, Distribution : Uniform, Algorithm : Divide and Conquer

2.2. Non-Interfaced Implementation

Distinct characteristics of the non-interfaced implementation are as follows:

- These are executable programs which generate gifs as final outputs. The gif animation displays the entire process of the computation of the convex hull. These come in two files, one files takes in an input *n* and generates random numbers and visualizes the algorithm on them. Another file takes in the input *n* and then opens up a plot to let you choose the n points with your cursor and then visualizes the algorithm.

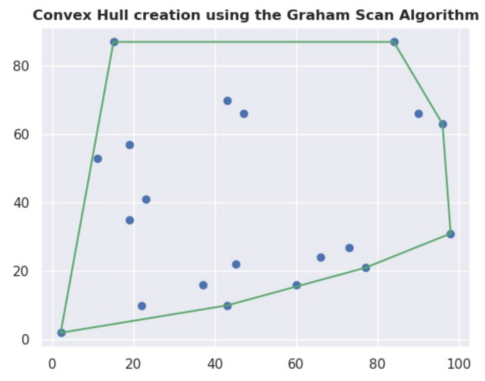


Figure 5. n : 20, Input Mode : Random, Algorithm : GrahamScan

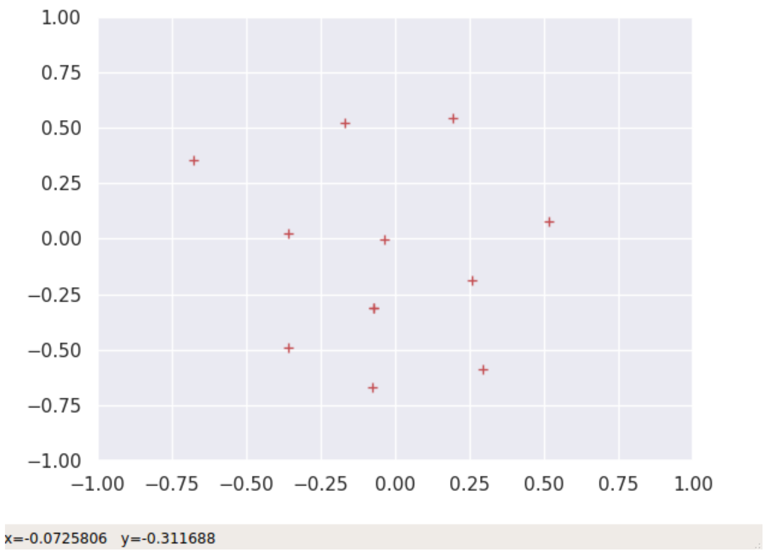


Figure 6. Input Step (n : 20, Input Mode : Cursor Click on Graph)

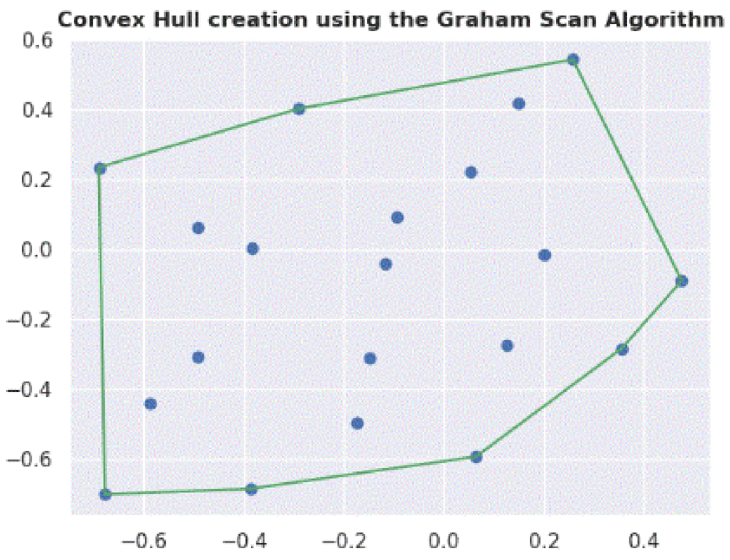


Figure 7. n : 20, Input Mode : Cursor Click on Graph, Algorithm : GrahamScan

3. Comparison of Algorithms

3.1. Without Recursive Visualizations

Without loss of generality, the distribution chosen is Uniform.

Number of DataPoints	Graham-Scan	Divide and Conquer
10	0.020 sec	0.041 sec
20	0.071 sec	0.156 sec
40	0.183 sec	0.297 sec
80	0.263 sec	0.563 sec
160	0.941 sec	1.516 sec
320	3.611 sec	4.583 sec
640	13.651 sec	17.765 sec
1280	56.937 sec	68.073 sec

Table 1. Comparison of Graham-Scan against Divide and Conquer

3.2. With Recursive Visualizations

Without loss of generality, the distribution chosen is Gaussian.

Number of DataPoints	Graham-Scan	Divide and Conquer
10	2.271 sec	2.529 sec
20	4.805 sec	7.871 sec
40	9.911 sec	18.393 sec
80	20.317 sec	41.558 sec
160	41.892 sec	88.455sec
320	88.806 sec	182.223 sec
640	187.560 sec	389.879 sec
1280	393.602 sec	814.989 sec

Table 2. Comparison of Graham-Scan against Divide and Conquer

3.3. Key Points and Observations

Few observations and points to be noted with respect to the running of the comparative experiments are as follows :

1. Although the time complexity for both the algorithms is same, i.e **O(nlog(n))**, the Graham-Scan algorithm slightly outperforms the Divide and Conquer approach which may point out to the fact that the constant added to the time complexity for the Graham Scan approach is lesser than the Divide and Conquer approach.

2. The time differences between the two algorithms is much lesser in the 'without visualizations' table than those seen in the 'with visualizations' table. This is solely because of greater number of calls being made to the visualization libraries in the Divide and Conquer recursive steps when compared to the Graham Scan approach.

3. The comparisons have been made on the same set of points for a particular fixed *n*.

4. Another thing to note is that the divide and conquer approach does not work for all inputs (generally in the range of *n* between *30-600*. It works almost perfectly for *n* greater than *1000*). The reason for failing may be due to the non-generalizability of the implementation across all four quadrants and missing out on some corner case due to a certain permutation of point distribution in quadrants. Since the points generated by the interface are random according to either of the three provided distributions, we cannot cap them to follow a particular quadrant combination. Hopefully, these can be ignored and for comparison purposes only those which generate the correct output have been taken into consideration.

4. Conclusion

In conclusion, it is worth to mention that the report has successfully elaborated on all the deliverables :-

1. Both the approaches : **Graham-Scan** and **Divide and Conquer** have been implemented.

2. Intermediate visualizations for both approaches has been implemented.

3. The Non-Interfaced solution also allows a user to manually plot points on a graph and then see a visualization of the underlying algorithm that generates the convex hull.

4. The Interfaced Solution provides a user-friendly approach to the visualizations.

5. A comparison of the time taken by the algorithms for a certain set of *n* has also been made.