Mega Exercise 1: convex hulls

Vitor Mazal Krauss

Abstract

In this project, I implement four algorithms for finding convex hulls on the plane. Namely, I start by implementing the following four methods: Incremental; Gift Wrapping; Graham-Scan; Divide and Conquer. For each of these methods I comment on a few details regarding their practical implementation, and more specifically, by doing so with the C++ Standard Template Library. Finally, I analyze the obtained results, with a special concern for the performance and proneness to floating-point error of each algorithm.

1. Introduction

The purpose of this project is to implement the four methods for finding convex hulls on the plane. A detailed description of the algorithms are given in [1]. In this report, I merely intend to discuss a few of the details regarding my own implementation of these methods, which were made in the C++ language, using the Standard Template Library for general-purpose algorithms and data structures, such as sorting and linked lists. The GLM library is used for the mathematical parts. I also present some of the results obtained, focusing on the run time of each method. Finally, for a visual validation of the results, a very unsophisticated rendering is made with OpenGL.

1.1. Incremental

At each step of the Incremental Algorithm, besides adding a new point to the hull, there is also the possibility of removing a subset of the current hull. This necessity of removal suggest the use of linked list to store the hull points. In fact, with a linked list implementation, once the positions of insertion and removals are found, the insertion of the new point can be made in constant time and the removal is linear on the number of points to be removed, because the STL implementation ,in fact, clears the removed values in memory. In contrast, an implementation using a STL vector would require not only the removal of points, but a resizing of the structure, i.e, copying and moving values in memory and, thus, yielding a greater number of operations. Hence the linked list implemention should be more efficient.

1.2. Gift Wrapping

In the case of the Gift Wrapping, note that this algorithm only adds points that are in the hull. In fact, once a point has been added, it shall not be removed. For this reason we do not have to worry about removing operations. With this, an implementation either with a linked list or a vector would require the same number of operations. In this case, for the convinience of having all the data stored contiguously, I chose the vector. This essentially leads to simpler and more readable code. It is also more practical for rendering purposes, since we can directly pass that area of memory to a OpenGL vertex buffer object.

1.3. Graham Scan

For the Graham Scan, like in the Incremental Algorithm, we must perform removal of points. However, the key difference between these two algorithm is that the Graham Scan will remove at most one points at each step, while the Incremental might remove several elements at a single step. For both a linked list or a vector, the cost of removing a single member is only one operation. This means a both the linked list and vector implementation would present similar performance. Again, for convinience and readability, I chose the vector implementation.

1.4. Divide and Conquer

With the Divide and Conquer algorithm, at each step, we again have the possibility of removing multiple points at a single step. Besides, differently from the previous three algorithms, multiple points will be added at each step. Again, this suggests the use of a linked list in order to represent the set of points in the hull. Like for the previous algorithms, I chose to represent the convex hull by a linked list of 2d points, each one being a point in the current hull. It should be noted, however, that a more efficient implementation of the Divide and Conquer algorithm could be achieved by storing a linked list of indices instead. The idea is to assign each vertex an index and, when needed, access the vertex coordinates by its index. By doing this, we can avoid splitting the list by half and copying the two halves at for each recursive call. Instead, we would pass a minimum and maximum index as parameters, such that in for each recursive call we would simply have to divide an integer by two. Like I said, this could lead a faster implementation for the Divide and Conquer method. Here, I decided to stick with the other option merely to stay in accordance with the previous three methods. For these methods the gain in efficiency would be very marginal, since they do not copy elements of the list.

2. Results

In this section I present the results obtained for each of the implemented algorithms. Our main concern here is the run time. However, it should also be noted that all these four method are, at some level, prone to floating-point errors. Both the Incremental and Divide and Conquer algorithms rely on deciding wether an edge is visible to a point or not. This is done by evaluating a determinant and checking its sign. The common practice is to test this value against an arbitrary, sufficiently small ε. However, of course, this does not exclude the possibility of error. On the other hand, the Gift Wrapping and Graham Scan rely on comparing angles. Firstly, it should be remarked that the angle itself does not need to be computed. A wiser alternative is to simply compute the cosine by performing dot products. Since cos is a monotonic function in the $[0,\pi]$ range, this is sufficient. This not only reduces floating-points errors that arise from evaluating arccos, but also provides a speed up. However, likewise, these two method are also prone to error arising from this angle comparisons.

2.1. Verifying the results

In order to test the correctness of the results, I used the following two ad-hoc processes to generate the points sets which were used for testing: For a set in which all points are also hull points, we can simply sample random, positive values x_i and add the points (x_i, x_i^2) to the set. Since the function x^2 is convex, the generated points will be the vertices of a convex polygon. This results in a set with n points, all of which are hull points. For a set with interior(i.e non hull) points, we start in a similar way by sampling positive values x_i and adding (x_i, x_i^2) to the set. This will generate a parabola-like hull. Once this is done, for every three points on this parabola, we can add their center of mass to the set. Again, because x^2 is convex, this center of mass will be an interior point. So if we added k points to the parabola, the total number of points n of the set will be $n = 2k-2$. These kind of sets are convenient because we know beforehand the exact number of interior and hull points.

The running time of the different method for these two classes of sets with varying n is shown in the tables below.

Execution Time				
Algorithm	n=10	n=100	n=1000	n = 2000
Incremental	186μs	4194μs	0.389s	1.581s
Gift Wrapping	167μs	6585μs	0.456s	1.652s
Graham Scan	66μs	437μs	0.007s	0.018s
Divide and Conquer	249μs	1080μs	0.017s	0.028s

Execution Time				
Algorithm	n=18	n=198	n=1998	n = 3998
Incremental	253μs	8628μs	0.802s	3.339s
Gift Wrapping	208μs	8605μs	0.834s	3.320s
Graham Scan	103μs	1616μs	0.241s	0.575s
Divide and Conquer	225μs	2509μs	0.281s	0.569s

We can see that in terms of runtime the Graham Scan presented the overall best performance, as could be expected, since its complexity is the lowest. However, it should be noted that performance is also set dependent. Some point sets might slightly favor other algorithms. However, due to their lower complexity, the Graham Scan and Divide and Conquer generally perform better.

As for the floating-points errors, an incorrect number of hull points was only found in the set with 3998 points for the Incremental, Graham Scan and Divide and Conquer algorithm. By the way this set is generated, it indeed leads to points which are close to colinear, which is problematic for measuring angles as well as testing visibility, which is made by checking if three points are in counter-clockwise order or not.

Finally, we generate a sample of 100 points in the $[-1,1]^2$ square and render them and their hull obtained by the Graham Scan algorithm. The rendering is very simple and was made with OpenGL.

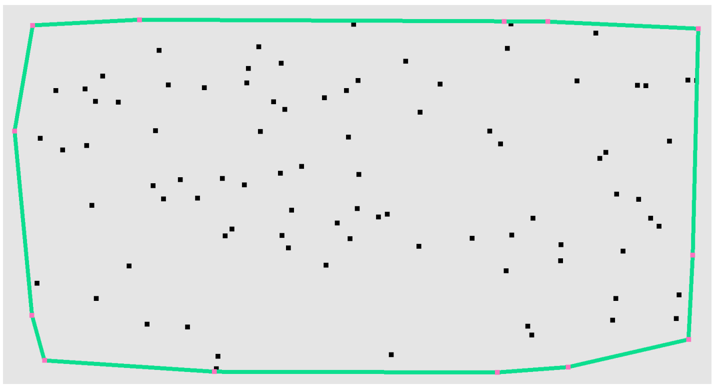

Fig. 1: 100 *points and their convex hull, found using the Graham Scan algorithm.*

References

[1] S. T. Devadoss, J. O'Rouke. *Discrete and Computational Geometry*, Princeton University Press, 2011.