Quickhull algorithm

700038669

Abstract

Quickhull algorithm is a divide-and-conquer algorithm used to solve the convex hull problem. It has a very efficient approach to solving the problem by ignoring points contained within the shapes, making it faster to compute than other alternative algorithms. Thus, it has seen many uses in the real world with many groundbreaking applications that are currently changing the world such as image processing. This report will explore how exactly the algorithm can achieve this along with providing a simplified explanation of the functionality of the program, its limitations and uses.

Introduction

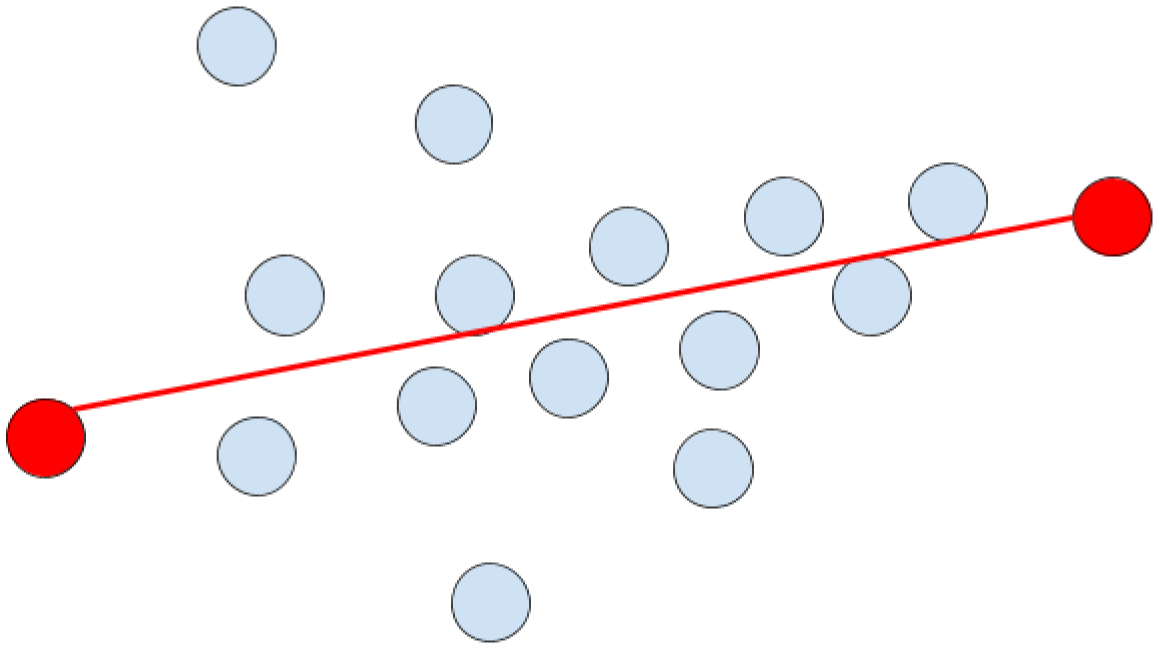
The quick hull algorithm is mainly used to compute the convex hull for a set of points. The convex hull for a set of points is calculated by finding the smallest convex polygonal shape for the points. It was first proposed in 1990 by Jonathan Greenfield but only for two-dimensional problems. It was further developed in 1996 by C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa to work with N-dimensions. It is primarily used in sports analysis, collision meshes and data sciences.

Principles and Aims

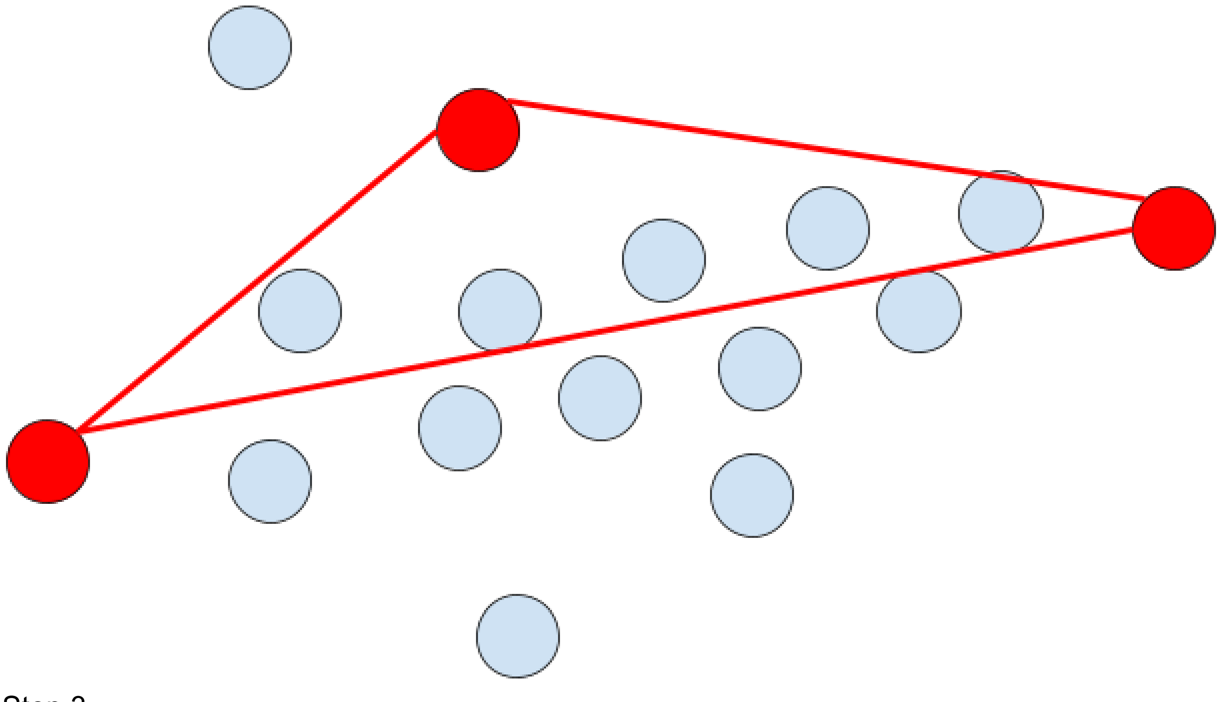
The main principle of quickhull is that it is a divide-and-conquer algorithm that utilises recursion. The algorithm creates a line segment between the minimum and maximum points. This line segment is used to create two subsets of the set of points. The farthest point to the line segment is then identified and connected, forming a triangle with the line segment. The points contained inside the triangle cannot be part of the convex hull so they are ignored. Instead, this process now creates more partitions and the process of creating a triangle from the furthest point is repeated recursively with each furthest point being added to the convex hull. This process is continued recursively until there are no more points outside of the convex hull. Below is a visual representation of the algorithm

Visual Representation

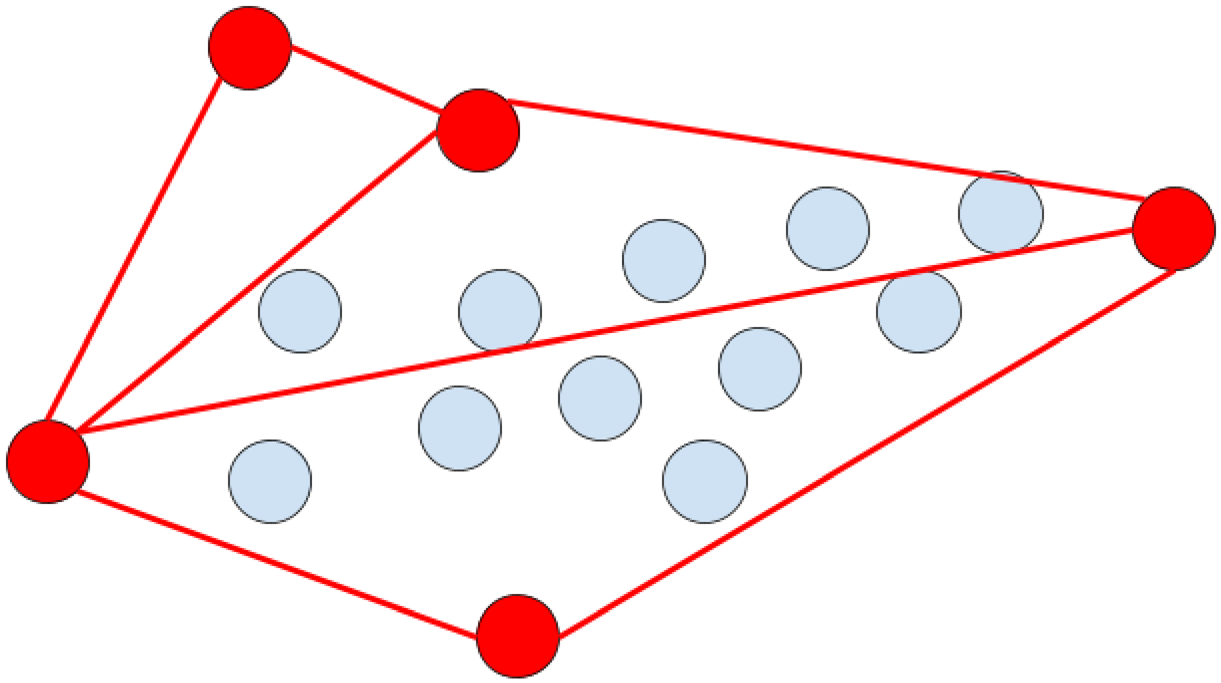
Step 1
A line segment between the two farthest points formed. These points are added as convex hull points.



Step 2
A triangle formed between the farthest point from the line segment and the line segment. This point is also added as a convex hull point.

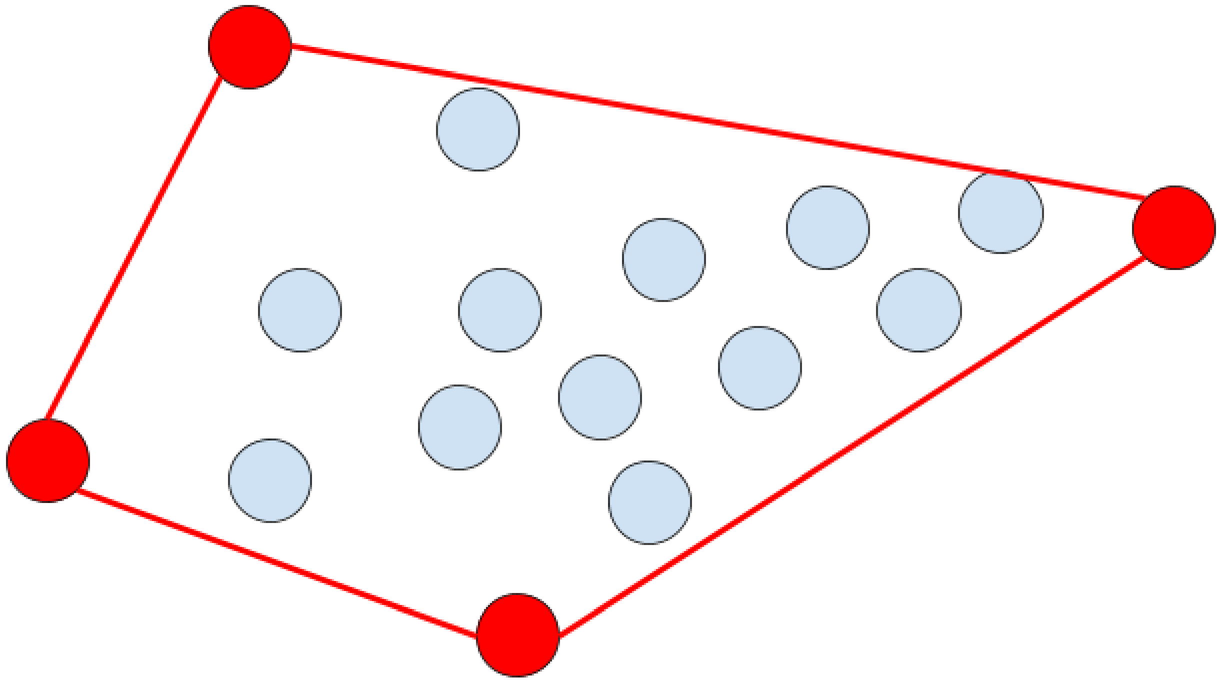


Step 3
More triangles are formed by repeating step 2 recursively until all the points are contained inside the convex hull or are convex hull points themselves.

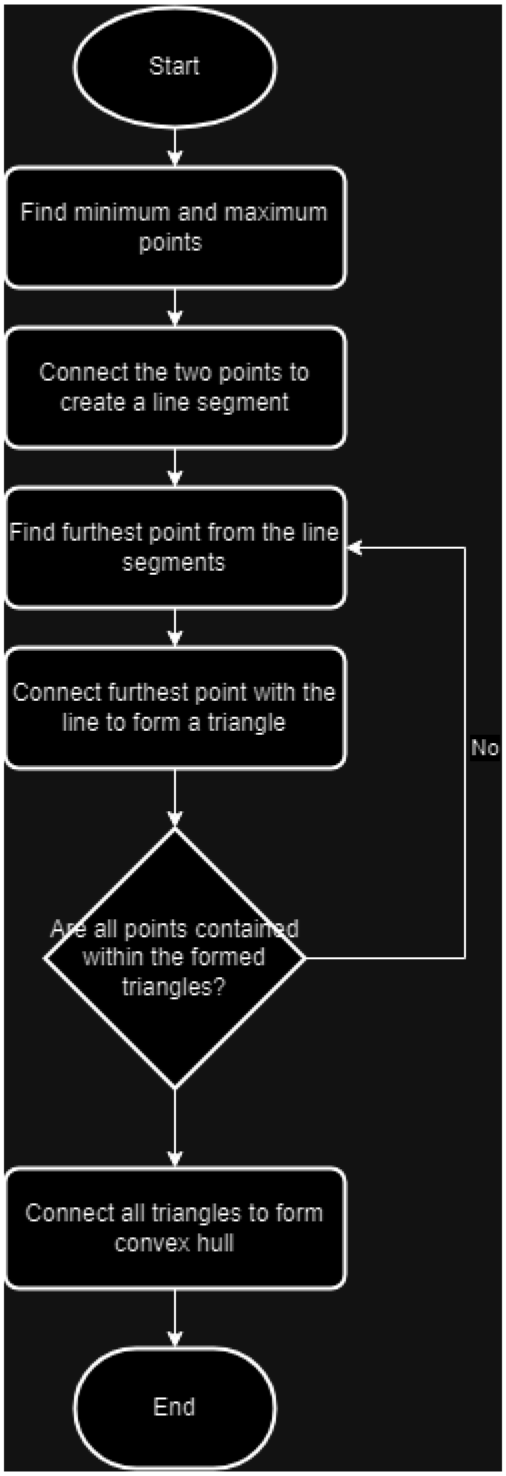


Step 4

A convex hull is formed by connecting all the triangles formed by the process



Flowchart


Start
Find minimum and maximum points
Connect the two points to create a line segment
Find furthest point from the line segments
Connect furthest point with the line to form a triangle
No
Are all points contained within the formed triangles?
Connect all triangles to form convex hull
End

Complexities

Time

On average, the Quickhull algorithm will have a time complexity of O(n) = n log(n). This is due to it being a divide and conquer algorithm which all perform very well on average, especially with randomly distributed data.

Worst case, it can reach O(n^2) in terms of time complexity. This occurs when there is a heavy imbalance in the partitions. For instance, one partition has 10 points whilst the other has 1.

Space

The space complexity of this algorithm is O(n) as it only needs memory to store the set of points given by the user to the algorithm. However, in practice, it can be much greater if implemented inefficiently due to the recursive nature of the algorithm.

Limitations

One of the limitations of this algorithm stated earlier is its worst-case time complexity reaching O(n^2). This is very inefficient due to the excessive number of recursive calls being made. Depending on the situation, many methods can be used to help reduce the chance of a worst-case scenario. For example, the median can be calculated and used when selecting minimum and maximum points to help increase the probability of both partitions being closer to balanced.

As stated earlier, another issue is the recursive nature of the algorithm. A poorly implemented recursive function or even a deeply nested one can easily cause stack overflow errors. Therefore, to prevent this, the programmer will need to ensure that their recursive functions are very well optimised and can handle the target loads without failure.

Application

As shown by the code provided, the Quickhull algorithm can be used in computational geometry to easily form a convex hull from a set of points. This has many real-world applications, one major application being the usage in sports analysis. For example, the algorithm can be utilised in football and basketball to analyse the effectiveness of a team's defence.

There are also many use cases in data analysis. The output from a Quickhull algorithm is a convex hull which is a very desirable shape in data analysis. The use of the convex hull can easily allow for outlier detection for example

The convex hull shape once again can be utilised in many different applications. Another one being creating collision meshes for use in computer graphics and even real-world application where the collision mesh can be used as a way to prevent traffic collisions of cars and aircrafts