COMP4202 Project
Student name: Xiaofeng Luo

Implementing line simplification algorithm: Douglas–Peucker algorithm

Abstract

It is easy to list a simple example application for line simplification algorithm. When zooming out a map in a navigation system (for example in google map), the digital display is unavoidable to simplified due to its limited pixels and also in order to save time and space. Therefore, how to simplify a geographic boundary that ensuring the shape of the curve is unchanged but the number of the vertex is minimized becomes an important problem. A classic solution for this question is using the Ramer-Douglas-Peucker algorithm. This is a line simplification algorithm that is widely used in 2D graphs and navigation systems. For this project, my topic is going to implement the classical Douglas-Peucker algorithm and a deformed version of the line simplification algorithm.

What is Douglas-Peucker algorithm

Douglas-Peucker algorithm (called the DP algorithm in follows) was proposed by David Douglas and Thomas Peucker in 1972 and then it was gradually improved in the next few years. This is an algorithm that approximates curves as a series of points and at the same time, it reduces redundant points with maximumly retaining curve skeletons due to the given threshold. Nowadays, it is widely used in GIS systems and it helps to delete redundant data, reduce the amount of data stored and save storage space, and speed up subsequent processing.

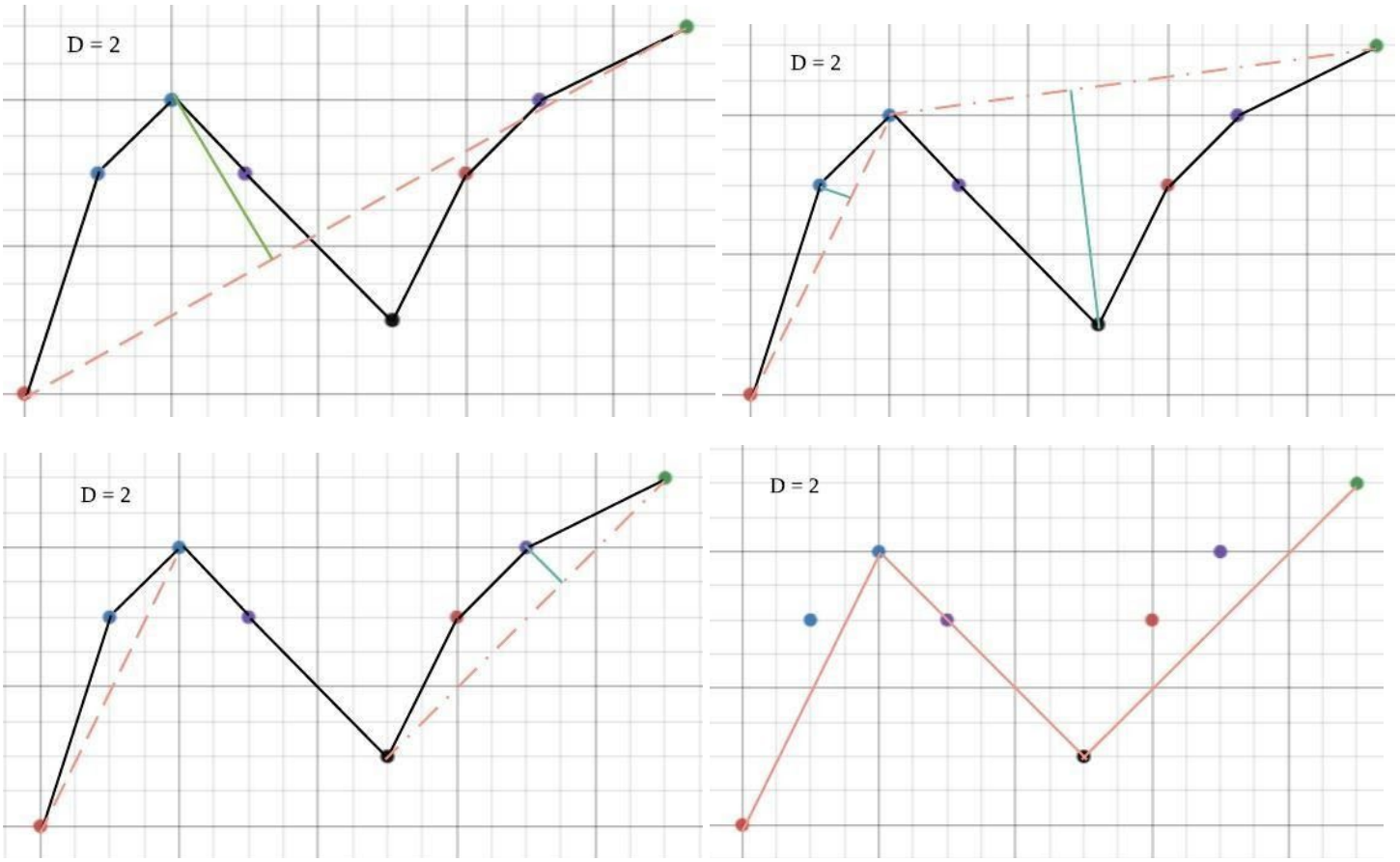
How DP-algorithm works

First of all, in the classic DP algorithm, there is a threshold epsilon named D that determines the simplified accuracy. Then connect the start point and endpoint on the curve (receive start-endpoint line), find all the distances from the rest of the points

to the start-endpoint line and retrieve the maximum one named dMax. And then compare dMax and D:

(1) If dMax < D, then remain start-endpoint line and delete all the points.

(2) If dMax >= D, then remain the coordinate point that corresponding to dMax and use this point as a boundary to divide the curve into two parts. Repeat the above steps for these two parts. This algorithm is implemented with recursing. Here show the basic flow below.

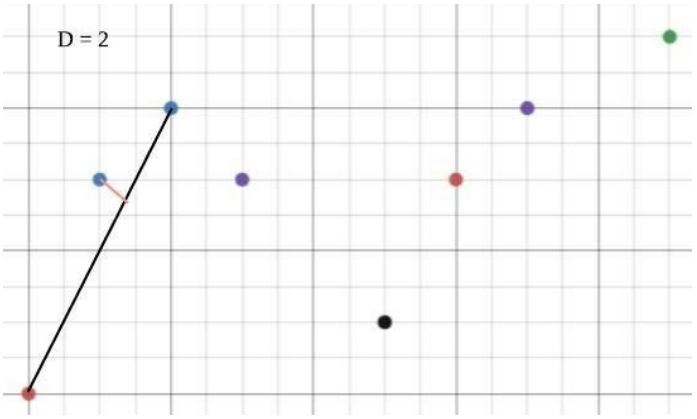


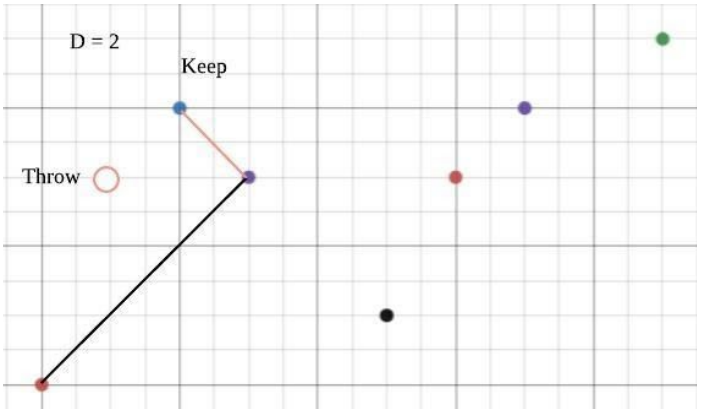
Secondly, there is another method of implementing the DP algorithm which is a little different from the classic one. This method is written in Chinese and could not find its exact English name for this method, hence, in this report, it will be called the vertical limit method(VL method). The implementation still needs a threshold epsilon(D). However, this time the algorithm will connect the 1st and 3rd points to construct a start-endpoint line, and see if the distance(d) from the 2nd point to the start-endpoint line is smaller or larger than threshold D:

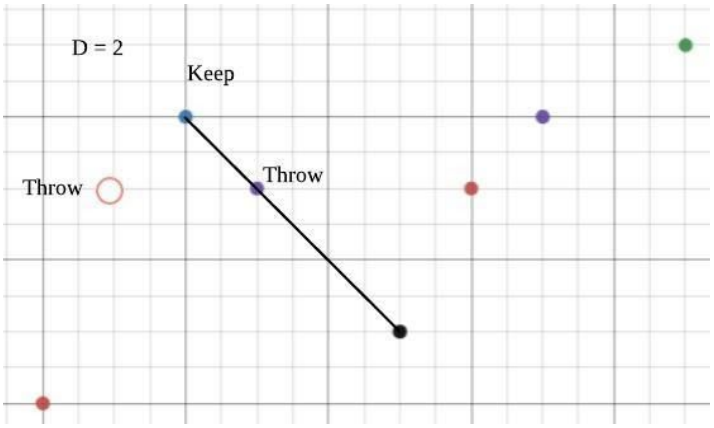
(1) If d<D, the 2nd point will be thrown, then 1st and 4th points will be connected to construct a start-endpoint line and distance between 3rd point and line will be compared with D.

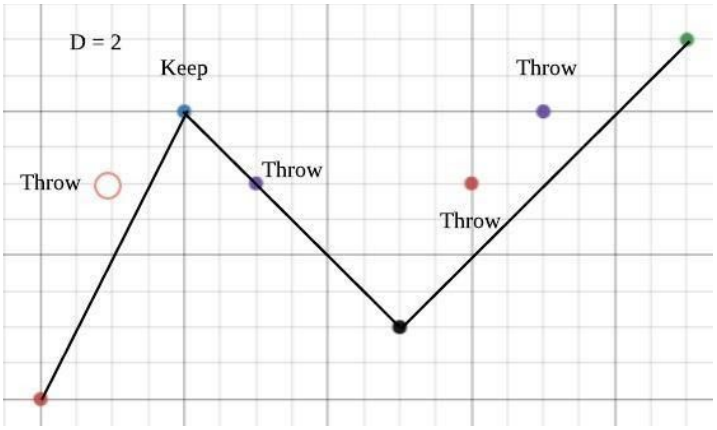
(2) If d>=D, then the 2nd point will be kept, then 2nd and 4th points will be connected to construct a start-endpoint line and distance between 3rd point and line will be compared with D.

Repeat the steps until all points have been checked. This algorithm can be implemented without recursing.









Implementing

Environment: Windows 7.

Language: JAVA.

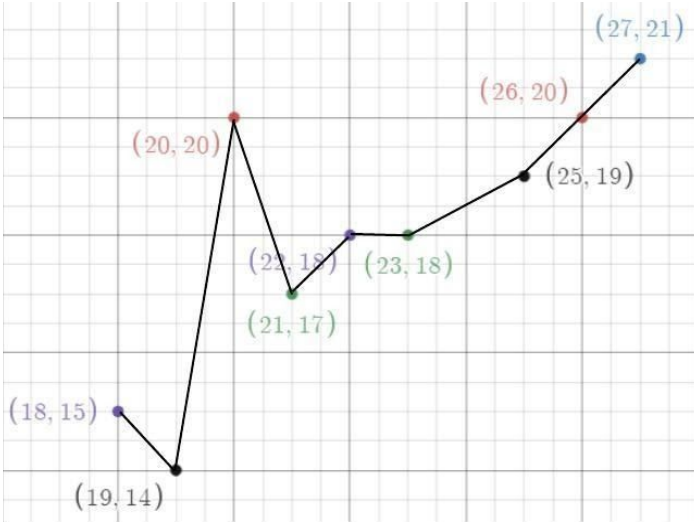
Input: coordinates. Output: coordinates.

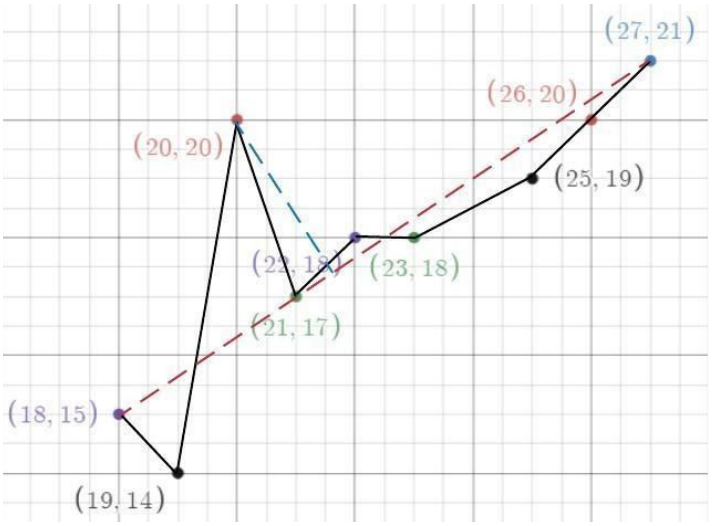
Firstly, in the classical DP algorithm implementation, it needs to set up a point class that has three attributes: X coordinate value, Y coordinate value and an index value that indicate pointers order(when the number is positive) or pointers need to

delete (when the number is -1). Then the program needs an ArrayList to store the source points. In the function named Douglas which takes two points(start point and endpoint) as parameters, for every element in source point(except the first and last one), do the calculation based on Heron's formula to calculate the distance between the point and start-endpoint line. The distance(d) will be stored in an ArrayList and then do a loop for finding the dMax and its corresponding points(set as middle point). If dMax is larger than or equal to D, then to pass the middle point to the Douglas function and keep calling the function recursively. Otherwise, take a loop and set all the indexes in points to -1. In the end, the main function will only output points which indexes are larger than -1.

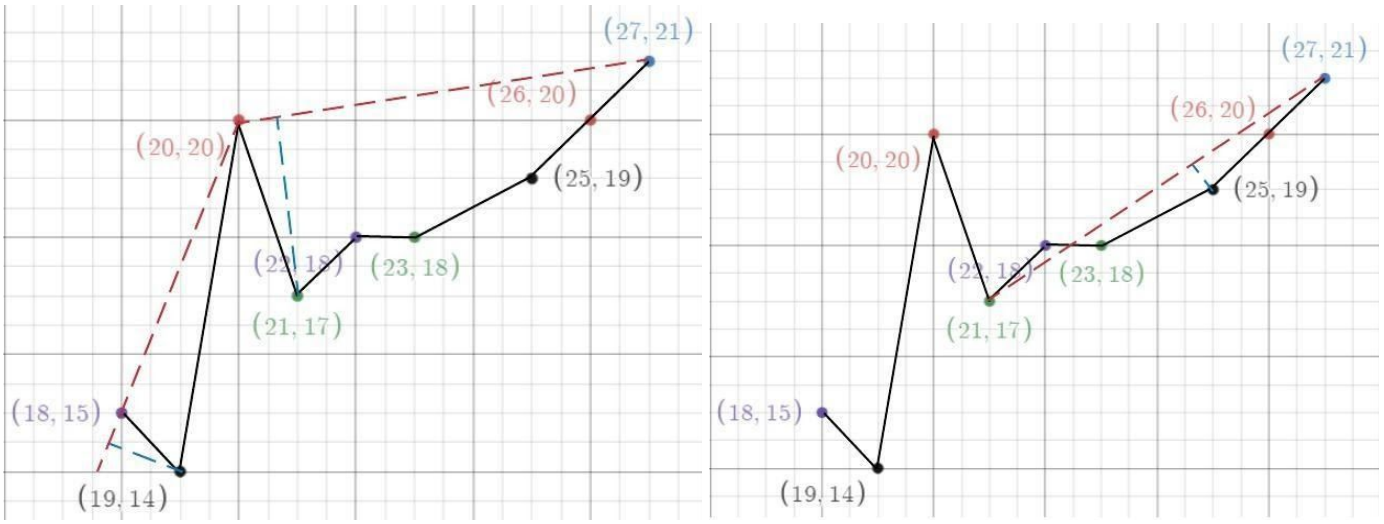
For the VL implementation, it will be much simple than before. Take a loop on the source and compare the distance (second points) with D and set the indexes of points that need to delete to -1. In the end, the main function will only output points which indexes are larger than -1.

In this section, I will take an example to do it step by step. Here is the input of the coordinates: (18,15),(19,14),(20,20),(21,17),(22,18),(23,18), (25,19),(26,20),(27,21), the epsilon D is 1. Here show some graphs about the classical DP algorithm.



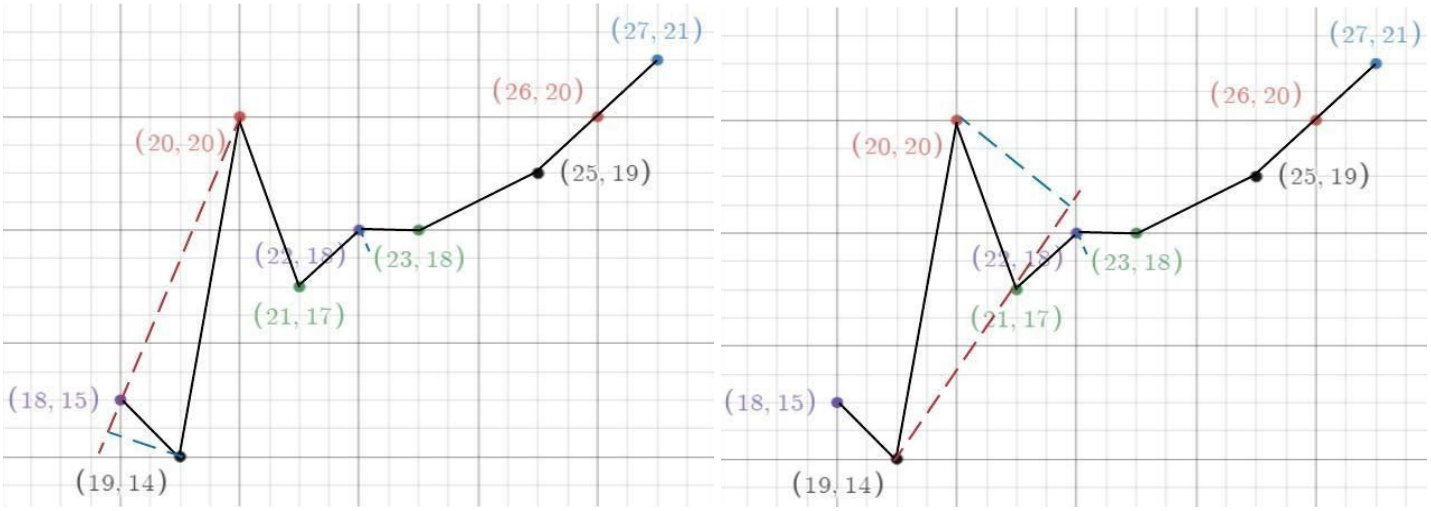


Therefore, keep point (20,20).

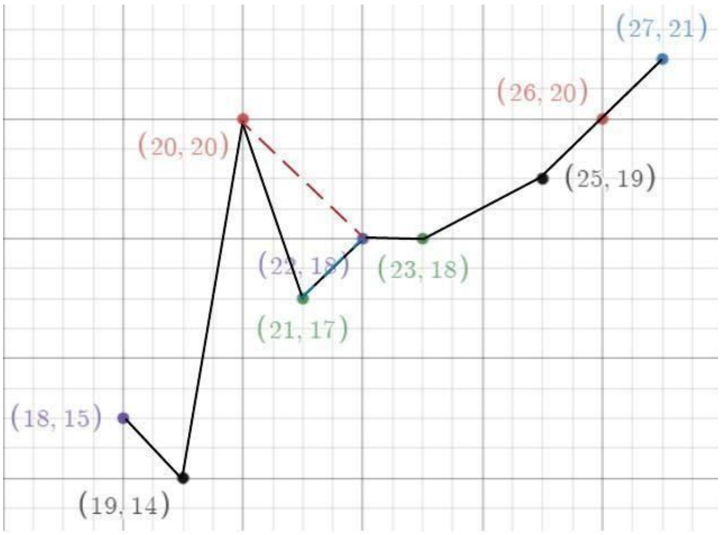


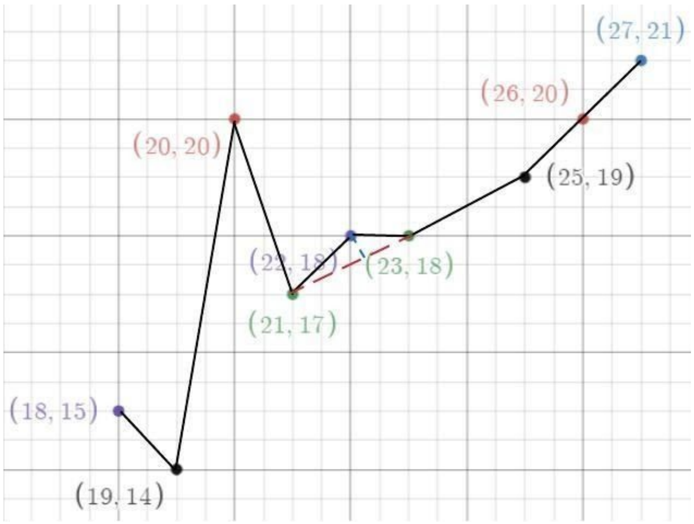
Therefore, keep points(19,14) and point(21,17). On the second graph, we can see that the distance between the point(25,19) and the start-endpoint line is small than D where is 1. Hence, in total the coordinates after simplification should be (18,15), (19,14), (20,20), (21,17), (27,21).

For VL algorithm, here are the graphs about the processing.

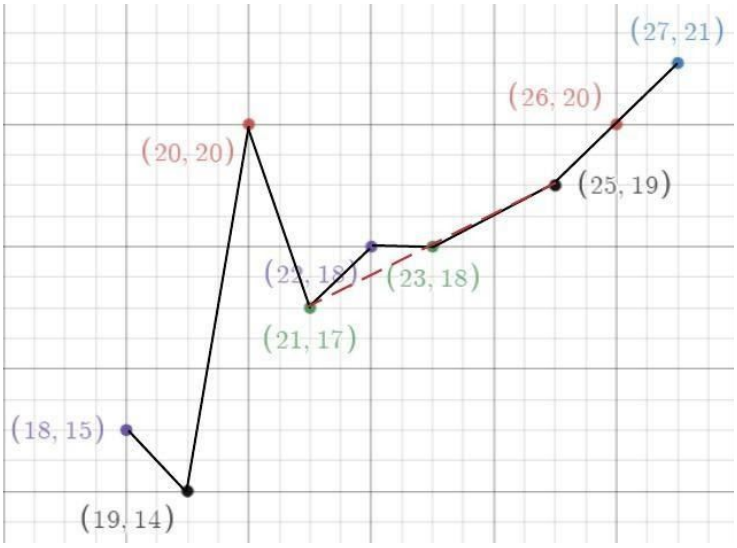


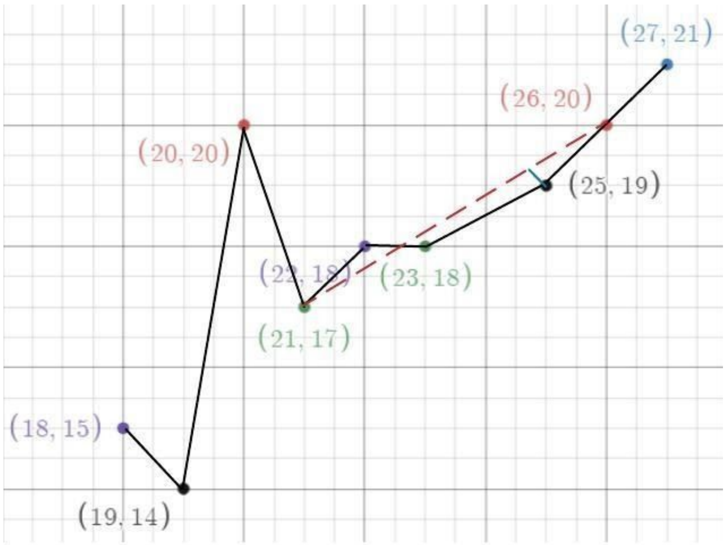
Obviously, points (19,14) and (20,20) can be kept because their distances are far larger than D.

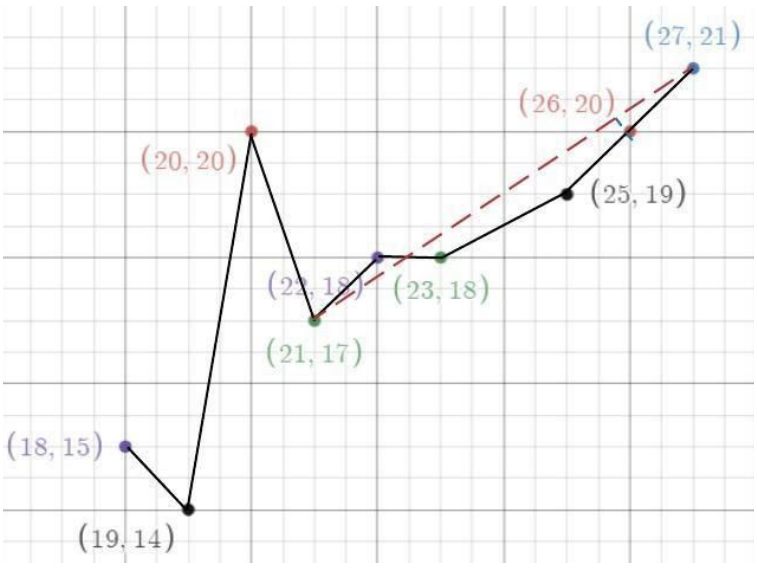




Hence, points (21,17) will be kept, however, point(23,18) will be abandoned because the d for it is smaller than D.







From the 3 graphs above we are easy to know that 3 points(23,18), (25,19),(26,20) should be reduced due to their distances. Therefore, the total coordinates after simplification should be (18,15), (19,14), (20,20), (21,17), (27,21).

Testing output of this example:

```
Before DP-compress
(18.0,15.0) (19.0,14.0) (20.0,20.0) (21.0,17.0) (22.0,18.0) (23.0,18.0) (25.0,19
.0) (26.0,20.0) (27.0,21.0)
After DP-compress
(18.0,15.0) (19.0,14.0) (20.0,20.0) (21.0,17.0) (27.0,21.0)
In total: 5

run time is 176192

Now using another type of compression
After another compress
(18.0,15.0) (19.0,14.0) (20.0,20.0) (21.0,17.0) (27.0,21.0)
In total: 5

run time is 31219
```

In order to compare these two algorithms, the experiment needs more data. Assume there are 3 levels of numbers of input coordinates(10, 20, 50).

Here is another example of testing(10 coordinates).

```
Before DP-compress
(1.0,5.0) (2.0,3.0) (3.0,6.0) (4.0,8.0) (5.0,8.0) (8.0,13.0) (10.0,-9.0) (12.0,1
1.0) (14.0,17.0) (17.0,16.0) (18.0,16.0)
After DP-compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (18.0,16.0)
In total: 7

run time is 434065

Now using another type of compression
After another compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (18.0,16.0)
In total: 7

run time is 116321
```

Here is another example of testing(20 coordinates).

```
Before DP-compress
(1.0,5.0) (2.0,3.0) (3.0,6.0) (4.0,8.0) (5.0,8.0) (8.0,13.0) (10.0,-9.0) (12.0,1
1.0) (14.0,17.0) (17.0,16.0) (18.0,16.0) (18.0,15.0) (19.0,14.0) (20.0,20.0) (21
.0,10.0) (21.0,17.0) (22.0,18.0) (23.0,18.0) (25.0,19.0) (26.0,20.0) (27.0,21.0)

After DP-compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (18.0,16.0) (
19.0,14.0) (20.0,20.0) (21.0,10.0) (21.0,17.0) (27.0,21.0)
In total: 12

run time is 536701

Now using another type of compression
After another compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (19.0,14.0) (
20.0,20.0) (21.0,10.0) (27.0,21.0)
In total: 10

run time is 62009
```

Here is another example of testing(50 coordinates).

```
Before DP-compress
(1.0,5.0) (2.0,3.0) (3.0,6.0) (4.0,8.0) (5.0,8.0) (8.0,13.0) (10.0,-9.0) (12.0,1
1.0) (14.0,17.0) (17.0,16.0) (18.0,16.0) (18.0,15.0) (19.0,14.0) (20.0,20.0) (21
.0,10.0) (21.0,17.0) (22.0,18.0) (23.0,18.0) (25.0,19.0) (26.0,20.0) (27.0,21.0)
 (30.0,22.0) (33.0,21.0) (35.0,22.0) (38.0,-13.0) (40.0,59.0) (41.0,58.0) (44.0,
60.0) (48.0,20.0) (50.0,21.0) (53.0,22.0) (55.0,27.0) (59.0,37.0) (71.0,12.0) (7
3.0,48.0) (82.0,28.0) (85.0,59.0) (89.0,20.0) (91.0,23.0) (94.0,73.0) (99.0,58.0
) (103.0,18.0) (111.0,59.0) (122.0,21.0) (123.0,48.0) (128.0,29.0) (132.0,75.0)
(140.0,79.0) (150.0,93.0) (157.0,38.0) (160.0,45.0)
After DP-compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (18.0,16.0) (
19.0,14.0) (20.0,20.0) (21.0,10.0) (22.0,18.0) (30.0,22.0) (35.0,22.0) (38.0,-13
.0) (40.0,59.0) (41.0,58.0) (44.0,60.0) (48.0,20.0) (53.0,22.0) (59.0,37.0) (71.
0,12.0) (73.0,48.0) (82.0,28.0) (85.0,59.0) (89.0,20.0) (91.0,23.0) (94.0,73.0)
(99.0,58.0) (103.0,18.0) (111.0,59.0) (122.0,21.0) (123.0,48.0) (128.0,29.0) (13
2.0,75.0) (140.0,79.0) (150.0,93.0) (157.0,38.0) (160.0,45.0)
In total: 38

run time is 1289794

Now using another type of compression
After another compress
(1.0,5.0) (2.0,3.0) (8.0,13.0) (10.0,-9.0) (12.0,11.0) (14.0,17.0) (19.0,14.0) (
20.0,20.0) (21.0,10.0) (30.0,22.0) (35.0,22.0) (38.0,-13.0) (40.0,59.0) (41.0,58
.0) (44.0,60.0) (48.0,20.0) (53.0,22.0) (59.0,37.0) (71.0,12.0) (73.0,48.0) (82.
0,28.0) (85.0,59.0) (89.0,20.0) (91.0,23.0) (94.0,73.0) (99.0,58.0) (103.0,18.0)
 (111.0,59.0) (122.0,21.0) (123.0,48.0) (128.0,29.0) (132.0,75.0) (140.0,79.0) (
150.0,93.0) (157.0,38.0) (160.0,45.0)
In total: 36

run time is 115038
```

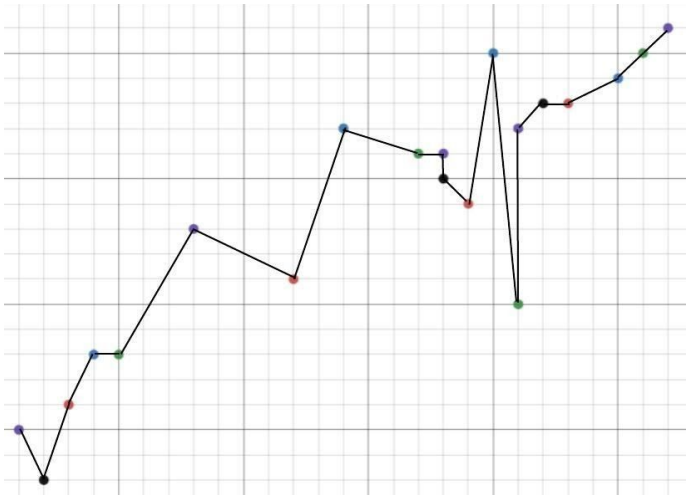
COMP4202 Project
Student name: Xiaofeng Luo

Number of coordinates	Remain points (Douglas)	Remain points (VL)	Runtime (Douglas) (in nano times)	Runtime (VL) (in nano times)
10	7	7	434065	116321
20	12	10	536701	62009
50	38	36	1289794	115038

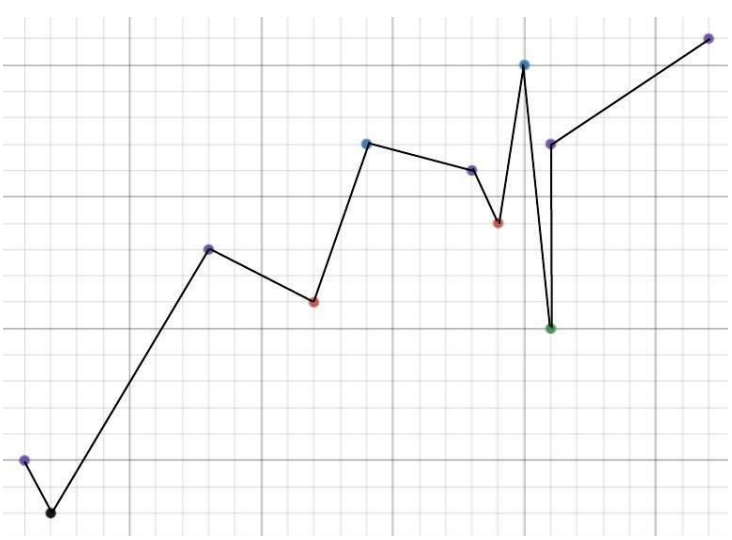
Analysis

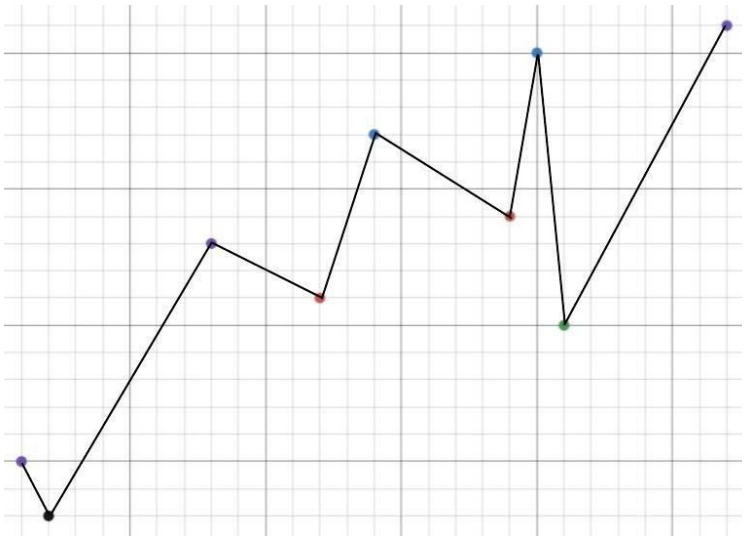
From the table, it is easy to observe that the most significant difference should be the runtime of these two methods. Obviously, the runtime for the classical Douglas algorithm is 4 to 11 times as long as the runtime for the VL method. It is because of the difference in the coding structure. In the classical method, the reducing point function keeps recursively calling itself which is the reason why the speed of processing will slow down. As for the VL method, it is simply only using one for loop to process all the points. It is not hard to predict when processing small data, the gap between 2 runtimes will not be so obvious. However, when the large data have been input, the gap will become larger and larger.

There is no doubt that the runtime performance of the VL method is better than the classical one and it is easy to implement. However, it does not mean the VL method is the most suitable one for the GIS system. Take a close look at the second column it shows that the remained coordinates for the VL method is less than the classical method. Here is the graph for the original curve.



After 2 processing, here are the graphs for classical and VL methods:





From the graph, it is clear that using the classical DP algorithm, this algorithm can have better fidelity, and it can compress a large amount of data while maintaining the original curve shape. Meanwhile, when using the VL method, the graph is missing some important turning points, which slightly affects the curve skeleton. The reason why causing this error is that when determining whether a point will be kept is based on the previous and latter points. If the latter point is an abandoned point, then reference value becomes less important and it means there are not different between the kept point and latter points, which causes a slight error.

Conclusion

In this report, we have implemented two different line simplification algorithm: the classical Douglas algorithm, which makes sure the fidelity of curve but it takes more runtime and larger storage for data; and the VL method makes sure simply coding and its fast runtime, however it is not as accurate as Douglas algorithm. Each algorithm has an advantage. When implementing it on a GIS system, the developers should consider carefully due to different requirements.

COMP4202 Project
Student name: Xiaofeng Luo

Reference:

https://rosettacode.org/wiki/Ramer-Douglas-Peucker_line_simplification#Java

https://www.sanfoundry.com/java-program-douglas-peucker-algorithm-implementation/

https://blog.csdn.net/morgerton/article/details/60878302