Henry Fock
443068

Aufgabenteil 1 Linesimplification

Meine Idee war es erst einmal den Douglas-Peuker Algorithmus zu implementieren.
Als Hilfe habe ich die Java-Klassen Point2D, Line2D und Path2D verwendet. Line2D und Path2D jedoch nur zur Topologie Erhaltung.

Mein Ansatz zur Topologie Erhaltung war, dass am irgendwie die Topologie von vorher mit nachher vergleichen muss. Line2D stellt eine Funktion Line2D.relativeCCW(Point2D) zur Verfügung, welche einen Wert in Abhängigkeit zur Richtung des liegenden Punktes zurückgibt. Also einen Wert für die Topologie, den man mit vorher und nachher vergleichen kann. Der einzige Ort wo die Topologie zerstört werden kann ist der letzte Teil des Algorithmus, bei dem nur Start- und Endpunkt beibehalten werden. Daher muss man sich nur auf diesen Teil konzentrieren.

Wenn der Fall eintritt, dass der Pfad vereinfacht wird, konvertiere ich als erstes die Punkte, die den Pfad ausmachen, in Line2D Objekte und daraus, um den Algorithmus zu vereinfachen und schneller zu machen, ein Path2D Objekt.
Als nächstes suche ich die Landmarken, welche von der Topologie betroffen sein könnten. dafür benutze ich die Funktion Path2D.contains(Point2D). Diese Funktion schließt den Pfad zu einem Polygon und guckt ob sich der Punkt innerhalb des Polygons befindet. Mein Hintergedanke dabei ist, dass alle Punkte, die außerhalb dieses Polygons liegen, keinen Konflikt erzeugen können, da nur Punkte innerhalb des Polygons weggeschnitten werden können. Dieser Filter dürfte einen gewaltigen Unterschied in der Laufzeit und im Speicheraufwand haben, sollte es viele Landmarken geben.

```
} else { //delete points within threshold
    ArrayList<Point2D> marksInBound = new ArrayList<Point2D>();
    int[][] topo = new int[0][0];
    ArrayList<Line2D> lines = new ArrayList<Line2D>();

    if(keepTopology) {
        Path2D path2d = new Path2D.Double();
        for (int i = 0; i < path.size()-1; i++) {
            lines.add(new Line2D.Double(path.get(i), path.get(i+1)));
            path2d.append(lines.get(i), true);
        }
        lines.trimToSize();

        //Rectangle2D bbox = path2d.getBounds2D();
        for (Point2D mark : landmarks) {
            if (path2d.contains(mark.getX(), mark.getY())) {
                marksInBound.add(mark);
            }
        }
        marksInBound.trimToSize();

        //Für jede Linie wird die Topologie für jede Landmarke gespeichert, die sich im Bereich
        des Teilpfades befinden
        topo = new int[lines.size()][marksInBound.size()];
        for (int i = 0; i < lines.size(); i++) {
            for (int j = 0; j < marksInBound.size(); j++) {
                topo[i][j] = lines.get(i).relativeCCW(marksInBound.get(j));
            }
        }
    }
```

Abbildung 1 Erster Teil der Topologie Erhaltung

Dann erstelle ich eine (n x m) Matrix, wobei n = die Anzahl der einzelnen Linien des Pfades und m = Anzahl der gefundenen Landmarken ist.
Jetzt speichere ich die Topologie von jedem Liniensegment zu jeder Landmarke in dieser Matrix.

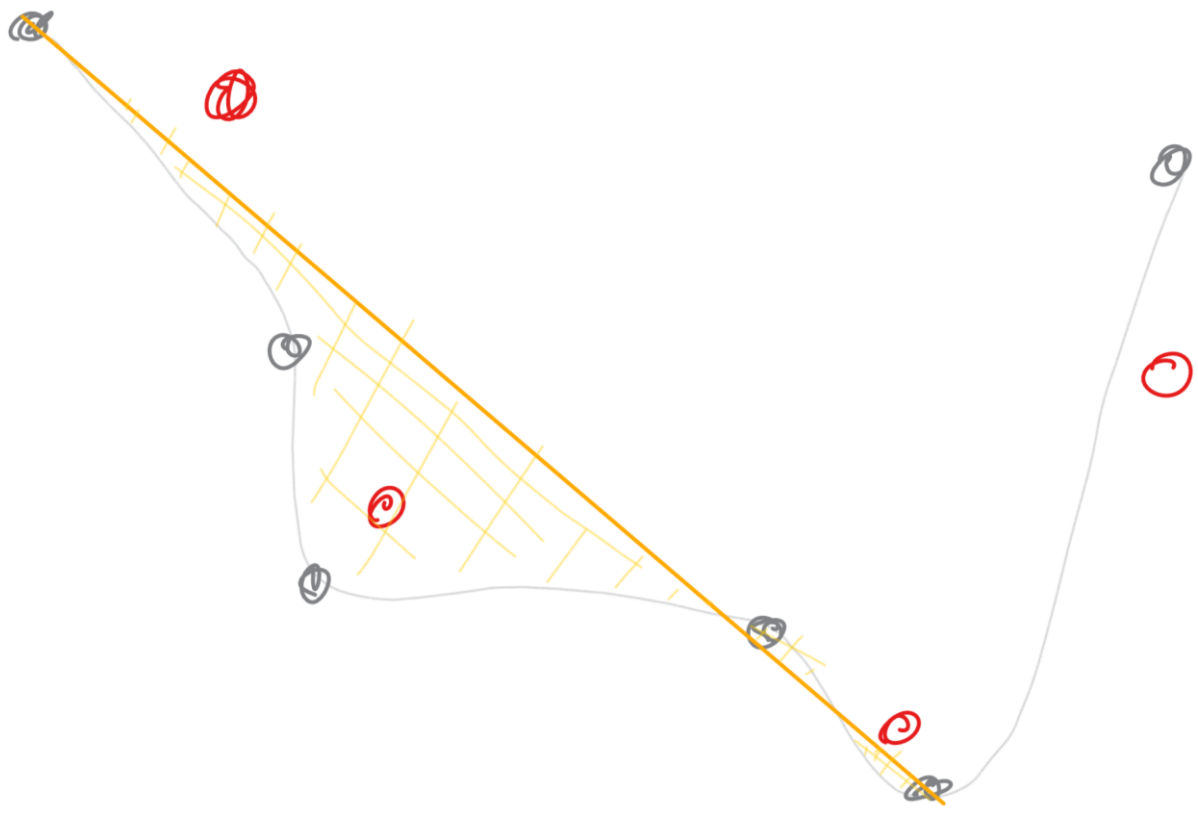
Nun wird der Pfad zu einer Linie vereinfacht.

Jetzt wird wieder die Topologie von jeder Landmarke zur neuen Linie berechnet.
Wenn die Topologie nicht zerstört wurde, dann wäre jetzt der Wert in jedem Punkt gleich.
Wenn dies jedoch nicht der Fall ist, so wird die vereinfachte Linie verworfen und die Funktion erneut rekursiv aufgerufen mit dem ursprünglichen Pfad, jedoch wird jetzt der Grenzwert verringert, um im Bereich des Konflikts eine weniger vereinfachte Linie zu erzeugen.

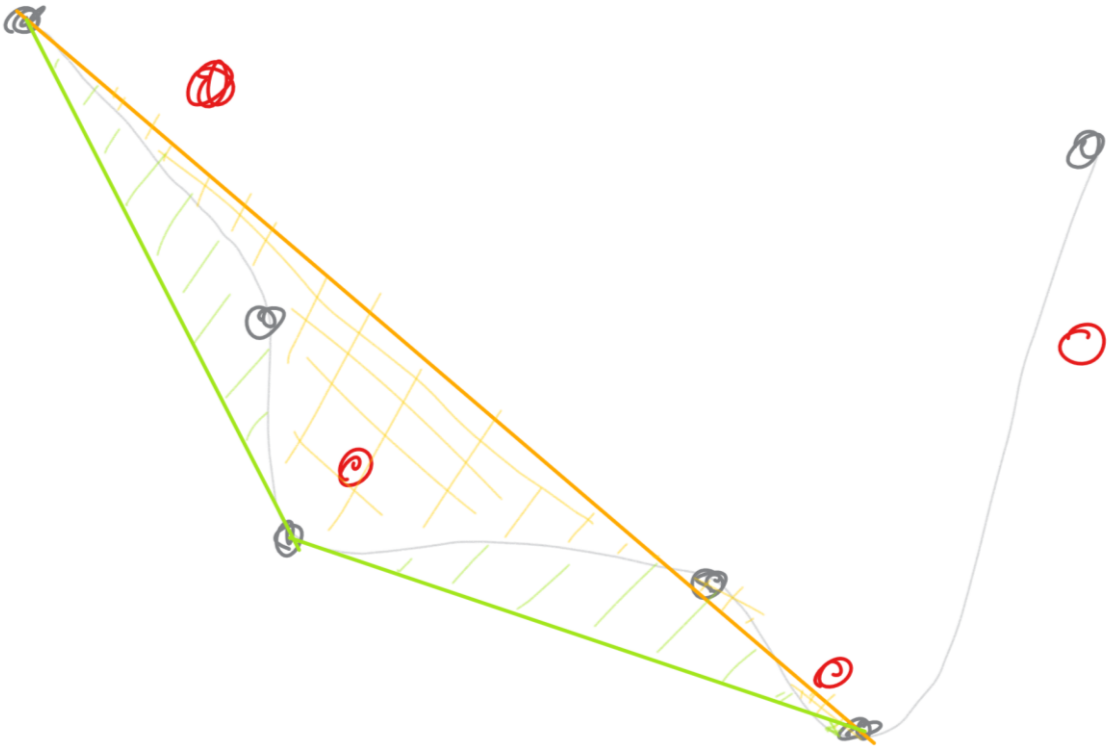
```
ArrayList<Point2D> ret = new ArrayList<Point2D>();
ret.add(path.get(0));
ret.add(path.get(path.size()-1));
res = ret;

if (keepTopology) {
   Line2D newLine = new Line2D.Double(ret.get(0), ret.get(1));
   int[] newTopo = new int[marksInBound.size()];
   for (int i = 0; i < marksInBound.size(); i++) {
        newTopo[i] = newLine.relativeCCW(marksInBound.get(i));
   }

   boolean flag = false;
   for (int i = 0; i < topo.length && !flag; i++) {
       for (int j = 0; j < topo[0].length; j++) {
          if (topo[i][j] != newTopo[j]) {
              flag = true;
          }
       }
   }

   if (flag && tolerance>0) {
       System.out.println("Topologie Conflict \n");
        res = topoPathSimplification(path, landmarks, tolerance-0.5, keepTopology);
   }
```

Abbildung 2 zweiter Teil der Topologie Erhaltung

Beim genaueren überlegen habe ich einen noch kürzeren Weg gefunden. Da die Contains Methode alle Landmarken zwischen der zukünftig neu erstellten Linie und der ursprünglichen Linie findet, müssen diese Punkte automatisch einen Konflikt auslösen. Damit kann man sich das Berechnen der Topologie sparen.
Zur Verdeutlichung dieses Bild:



Der Rote Punt wird von der contains Methode erfasst und ist würde als einziger hier einen Konflikt auslösen. Wenn also ein Konflikt ausgelöst wird, so wird der Algorithmus mit geringerer Toleranz erneut ausgeführt. Das erhoffte Ergebnis wäre dann wie folgt:



(Ergebnis in Grün)
Hier ist kein Punkt innerhalb der Fläche, somit gibt es auch keinen Topologie Konflikt.

Da ich mir meiner Theorie aber nicht 100% sicher bin, habe ich beide Ergebnisse eingefügt (mit und ohne …Simple.java)

```
} else { //delete points within threshold
    boolean flag = false;
    if(keepTopology) {
        Path2D path2d = new Path2D.Double();
        for (int i = 0; i < path.size()-1; i++) {
            Line2D line = new Line2D.Double(path.get(i), path.get(i+1));
            path2d.append(line, true);
        }

        for (int i = 0; i < landmarks.size() && !flag; i++) {
            flag = path2d.contains(mark.getX(), mark.getY());
        }
    }

    if (flag && tolerance>0) {
        System.out.println("Topologie Conflict \n");
        res = topoPathSimplification(path, landmarks, tolerance-0.5, keepTopology);
    } else {
        ArrayList<Point2D> ret = new ArrayList<Point2D>();
        ret.add(path.get(0));
        ret.add(path.get(path.size()-1));
        res = ret;

        System.out.println("gekuerzt: " + ret);
        System.out.println("\n");
    }
}
```

Abbildung 3 oben beschriebene Änderung am Algorithmus aus Abb1+2

https://github.com/HenFo/Geodaten-DouglasPeuker