Aufgabe 1 Siedler

Teilnehmer-ID: ???

Bearbeiter dieser Aufgabe:

Daniel Hohmann

April 8, 2024

Inhaltsverzeichnis

1 Loesungsidee

Ich habe mir die Aufgabe durchgelesen, und dabei gemerkt, dass die Hauptaufgabe im Endefekt nichts anderes ist, wie die den Punkt im polygon zu finden, der wen ich mein radius von 85km platziere, die meisten Doerfer im abstand von 10km halten kann. Danach muss ich im endefekt nur noch die restlichen Doerfer im abstand von 20km platzieren.

2 Umsetzung

2.1 Erste schritte und fehler

2.1.1 raster algorithmus

ich habe viele probleme mit der umsetzung. Das groeste problem ist es, den optimalsten punkt innerhalb des polygons zu finden. So das ich moeglichst viele doerfer im abstand von 10km platzieren kann.

Meine erste idee war es, irgendwie ein raster ueber das polygon zu legen. So das ich nur noch schauen muss bei welchem rasterpunkt ich am meisten rasterpunkte im radius von 85km habe. Dass muestte dann auch der Beste platz sein, wo ich das Gesundheitszentrum platzieren kann.

Dafuer habe ich mir einfach den kleinsten x und y wert, und den groesten x und y wert gesucht. danach musste ich nur noch in meinem Gewuenschten abstand nach oben zaehlen, von den kleinsten werten aus bis zu den groesten. Damit hatte ich am ende ein raster von Punkten ueber das ganze Polygon. Damit ich nur die Punkte habe die auch wirklich im polygon liegen, hab ich einfach mit einem raycast algorithmus geschaut welcher punkt im polygon liegt und welcher nicht, der kommt auch in meinem finalen Programm zum einstaz heist ich erleutere ihn spaeter noch mal genau.

Der ansatz hat auch am Ende funktioniert aber das Problem ist das wenn ich den abstand zu den rasterpunkten bestimme, am anfang also als Bsp.5 oder aber auch 1, dann habe ich zwar da die punkte. Und ich kann die auch durchiterieren, aber ich wuerde genau wegen den rasterpunkten, nie den genauen punkt finden.
Je geringer ich den abstand der punkte mache, desto genauer ist logischerweise auch das ergebnis geworden. Aber um so laenger war auch die rechenzeit, die mein Rechner gebraucht hat. So das ich irgendwann bei einem Rasterpunkte abstand von 0.001 bei weit ueber einer stunde war. zudem muss ich nach dem rastern noch schauen, welcher der Punkte nun wirklich im Polygon lag und welcher eben nicht.

2.1.2 kreis algorihtmus

Ich habe versucht Bei meiner anderen umsetzung, den Besten platz mithilfe von Kreisen zu finden. Auf die idee bin ich gekommen als ich im Internet geschaut habe, was man alles beim polygon berechen kann. Dabei bin ich auf den Innenkreis des Polygons gestossen, der den groesten kreis innerhalb eines polygons darstellt.

wen ich es dann noch schaffen wuerde den mittelpunkt des kreises zu bestimmen habe ich gewonnen. Das ganze hat am ende nicht so gut funktioniert, da ein paar der Bsp. Polygone unregelmaessige polygone sind. wo es schwer bis gar unmoeglich ist den innernkreis zu berechen. Zudem ist mir als Problem aufgefallen das wenn ich versuche den mittelpunkt zu bestimme der punkt nie ganz genau ist.

2.2 Die End Umsetzung

2.2.1 Geometrie.h: Geometriesche funktionen klasse Point, Polygon

2.2.2 Klasse Point

Um alles auch ordentlich darzustellen im code habe ich eine Geometrie.h datei erstellt, die alle Funktionen fuer die verarbeitung und die darstellund von Polygonen und Punkten enthaelt
Die Datei enthaelt eine klasse *Point*, die Aufgabe dieser klasse ist es, einfach ein Punkt darzustellen also zwei werte vom typ double. zudem hat die klasse *Point* zwei funktionen, die Erste ist es den x-wert eines punktes wiederzugeben(*getX()*), die zweite funktion soll den y-wert eines punktes wiedergeben (*getY()*).

2.3 Klasse Polygon

2.3.1 Funktion: centroid

Die Klasse *Polygon* hat im Grunde die Aufgabe, ein Polygon darzustellen. wobei hier das Polygon im endefekt nichts anderes ist wie ein vector aus Point's. mit eine der wichtigsten Funktionen ist die Funktion *centroid* Bestimmt den schwerpunkt des Polygons das ich spaeter fuer den Hauptalgorithmus brauche die Funktion *centroid* macht im endefekt nicht anderes als als erstes die flaeche des polygons mithilfe der Gauss'schen Trapezformel zu Berechen. fuer die gilt:

$$\text{area} = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

Hierbei wird P_{n+1} als P_1 behandelt, d.h., $x_{n+1} = x_1$ und $y_{n+1} = y_1$.
Der Schwerpunkt (cx, cy) des Polygons wird dann berechnet als:

$$cx = \frac{1}{6 \cdot \text{area}} \sum_{i=0}^{n-1}(x_i + x_{i+1}) \cdot (x_i y_{i+1} - x_{i+1} y_i)$$

$$cy = \frac{1}{6 \cdot \text{area}} \sum_{i=0}^{n-1}(y_i + y_{i+1}) \cdot (x_i y_{i+1} - x_{i+1} y_i)$$

Hierbei wird P_n als P_0 behandelt, d.h., $x_n = x_0$ und $y_n = y_0$.

2.3.2 Funktion isInsidePolygon

Die naechste wichtige funktion der Klasse Polygon ist die Funktion *isInsidePolygon*. Diese funktion schaut ob ein punkt innerhalb des Polygons liegt. Dies tut die Funktion mithilfe eines raycastalgorithmus. Die funktions idee ist das ich ein punkt habe und von diesem punkt aus versende ich strahlen(rays), dann schaue ich wie oft dieser strahl die kanten meines polygons schneidet. wenn die Anzahl ungrade ist, liegt der Punkt innerhalb des Polygons. Ist die annzahl jedoch Grade liegt der Punkt ausserhalb des Polygons. Das heisst gegeben muessen sein, mein Punkt den ich pruefen will $p(x_p|y_p)$ und mein Polygon $\{(x_i, y_i)\}_{i=0}^{n-1}$, hier steht n fuer die anzahl der Eckpunkte.

als naechstes inzinalisiere ich mein counter intersectionCount mit 0.
Als naechstes lassen wir eine schleife ueber alle kanten des Polygons laufen, fuer jede kante des Polygons gilt $(P_i, P_i + 1)$, wobei $P_i = (x_i, y_i)$, $P_i + 1 = (x_i + 1, y_i + 1)$:
Jetzt ueberpruefen wir jede kante ob die horizontale linie durch p schneidet.
Danach Bestimmen wir die y-Koordinaten der Kantenenden: $y_i, y_i + 1$.
Nun ueberpruefen wir ob p zwischen P_i und $P_i + 1$ liegt oder andersherum $(y_i > y_p) \neq (y_i + 1 > y_p)$.
Wenn nun die kannte die Horizontale linie schneidet berechne ich den x-wert des Schnittpunkts der kante $x_{\text{intersect}}$ was die horizontale Linie durch p ist:
$x_{\text{intersect}} = x_i + (y_i + 1 - y_i)\frac{(x_i+1-x_i)*(y_p-y_i)}{y_i+1-y_i}$
Als naechstes schauen wir das wenn $x_{\text{intersect}} > x_p$ zaehlen wir unseren intersectionCount um eins hoch.
Zum schluss schauen wir dann wie hoch unser intersectionCount ist. wenn er ungrade ist liegt der Punkt im Polygon ist der Grade liegt der Punkt nicht im polygon.

2.3.3 Der Hauptalgorithmus und seine funktion

2.3.4 Erweiterung des Programms, der Plot des Polygons samt doerfern

3 Beispiele

4 Quellcode

4.1 Geometrie.h

```
#pragma once

#include <iostream>
#include <cmath>
#include <vector>
#include <sstream>
#include <fstream>

namespace Geometrie{

 // klasse point um ein punkt darzustellen
 class Point{
  private:
   double x,y;
  public:
          Point() : x(0.0), y(0.0){}
   Point(double _x, double _y) : x(_x), y(_y){}

   // funktionen zum returnen der x und y koordinaten
   double getX() const {return x;}
   double getY() const  {return y;}

 };

 // klasse Polygon
 class Polygon{
  private:
   std::vector<Point> points;
  public:
   Polygon(){}

   std::vector<Point> getPoints() const{
    return points;
   }

   const Point& getPoint(int index) const{
    return points[index];
   }

   // Funktion um ein Punkt in das Polygon hinzuzufuegen
   void addPoint(double x, double y){
    points.push_back(Point(x,y));
   }

   void addPointsFromVector(const std::vector<Point>& vec) {
```

```
  points.insert(points.end(), vec.begin(), vec.end());
 }

 // Funktion zum Bestimmen des Schwerpunktes oder auch Centroid des Polygons
 Point centroid() const {
         double cx = 0.0, cy = 0.0;
         double area = 0.0;

         for (int i = 0; i < points.size(); ++i) {
             double xi = points[i].getX();
             double yi = points[i].getY();

             double xi1 = points[(i + 1) % points.size()].getX();
             double yi1 = points[(i + 1) % points.size()].getY();

             double common = xi * yi1 - xi1 * yi;
             cx += (xi + xi1) * common;
             cy += (yi + yi1) * common;
             area += common;
         }

         area /= 2.0;
         cx /= (6 * area);
         cy /= (6 * area);

         return Point(cx, cy);
 }

// Funktion zum schauen ob der punkt innerhalb des polygons liegt oder nicht
 bool isInsidePolygon(const Point& p) const {
  int intersectioncount = 0;

  for(int i = 0; i < points.size(); ++i){
   double xi = points[i].getX();
   double yi = points[i].getY();
   double xi1 = points[(i + 1) % points.size()].getX();
   double yi1 = points[(i + 1) % points.size()].getY();

   if((yi > p.getY()) != (yi1 > p.getY()) && p.getX() < (xi1 - xi) *
    (p.getY() - yi) / (yi1 - yi) + xi){
    intersectioncount++;
   }
  }

  return intersectioncount % 2 == 1;
 }
// Funktion zum einlesen des Polygons von der file .txt
void readPointsFromFile(const std::string& filename) {
 std::ifstream file(filename);
if (!file.is_open()) {
 std::cerr << "Fehler beim Öffnen der Datei: " << filename << std::endl;
  return;
}
```

```
  std::string line;
  while (std::getline(file, line)) {
   std::stringstream ss(line);
    double x, y;
    if (ss >> x >> y) {
      addPoint(x, y);
    }
  }

  file.close();
}
 };
};
```

4.2 main.cpp

```
#include <iostream>
#include <cmath>
#include <vector>

#include <Geometrie.h>
#include <matplotlibcpp.h>

namespace plt = matplotlibcpp;


// Funktion um die punkte eines umfangs Berechen
std::vector<Geometrie::Point> umfang_punkte_eins(const Geometrie::Polygon& polygon,
 Geometrie::Point centroid, int radius){
 std::vector<Geometrie::Point> umfang_Punkte;
 double umfang = 2 * M_PI * radius;
 int anzahl_punkte = static_cast<int>(umfang/20);

 for(int i = 0; i < anzahl_punkte; ++i){
  double winkel = (2*M_PI*i)/anzahl_punkte;
  double x = centroid.getX() + radius * std::cos(winkel);
  double y = centroid.getY() + radius * std::sin(winkel);
  if(polygon.isInsidePolygon(Geometrie::Point(x,y))){
   umfang_Punkte.push_back(Geometrie::Point(x, y));
  }
 }
 return umfang_Punkte;
}
// Funktion um die letzten punkte im polygon zu bestimmen
std::vector<Geometrie::Point> last_points(const Geometrie::Polygon& polygon,
 const Geometrie::Point& center_point) {
 int radius = 100;
 std::vector<Geometrie::Point> end_points;
 std::vector<Geometrie::Point> points = umfang_punkte_eins(polygon, center_point, radius);
 // endet wenn der kreis so gros ist das kein punkt mehr im polygon platziert werden kann
 while(points.size() > 0){
  end_points.insert(end_points.end(), points.begin(), points.end());
  radius += 20;
  points = umfang_punkte_eins(polygon, center_point, radius);
 }
 return end_points;
}

// Funktion zum berechen der Punkte auf dem Umkreis eines Kreises in 10 radius abstaenden bis 85
std::vector<Geometrie::Point> circle_Points(int distance, Geometrie::Point centroid_circle,
 const Geometrie::Polygon& polygon){
 std::vector<Geometrie::Point> umfang_Punkte;

 for(int radius = 10; radius <= 85; radius += 10){
  double umfang = 2 * M_PI * radius;
  int anzahl_punkte = static_cast<int>(umfang/distance);

  for(int i = 0; i < anzahl_punkte; ++i){
   double winkel = (2*M_PI*i)/anzahl_punkte;
```

```
   double x = centroid_circle.getX() + radius * std::cos(winkel);
   double y = centroid_circle.getY() + radius * std::sin(winkel);
   if(polygon.isInsidePolygon(Geometrie::Point(x,y))){
    umfang_Punkte.push_back(Geometrie::Point(x, y));
   }
  }


 }
 return umfang_Punkte;
}
// Funktion zum bestimmen des Gesundheitszentrums
Geometrie::Point gesundheitszentrum (int distance, Geometrie::Point centroid_circle,
 const Geometrie::Polygon& polygon){
 std::vector<Geometrie::Point> points = circle_Points(distance, centroid_circle, polygon);
 Geometrie::Point end_point;
 int start_value = points.size();
 int end_value = 0;
 Geometrie::Point centroid = centroid_circle;
 // wir gehen gehen unsere 1 liste von punkten(doerfern) durch und schauen jeden punkt
 // durch ob er ein besserer centroid waere
 do{
  for(const Geometrie::Point& p: points){
   std::cout << "jetzt" << std::endl;
   std::vector<Geometrie::Point> test_points = circle_Points(distance, p, polygon);
   if(test_points.size() > start_value){
    start_value = test_points.size();
    centroid = p;
    }
    // wen der wert vom vorherigem durchlauf gleich dem 2 durchlauf ist returnen wir
    // den vector
   if(start_value == end_value){
    end_point = p;
    return p;
   }
   end_value = start_value;

  }
 }while(true);
}


int main() {
 // Erstellen eines Polygon Objektes
 Geometrie::Polygon polygon;

 // Punkte aus einer .txt file einlesen
 polygon.readPointsFromFile("../inputs/siedler1.txt");

 // bestimmen des centroid
 Geometrie::Point centroid = polygon.centroid();

 // bester punkt fuer das gesundheitszentrum
 Geometrie::Point gesundheitszent = gesundheitszentrum(10, centroid, polygon);
```

```
// vector mit den punkten innerhalb des radiuses von dem Gesundheitszentrum
std::vector<Geometrie::Point> gesundheitszentrum_Points = circle_Points(10, gesundheitszent, polygon);
// Restlichen punkte ausserhalb des radiuses des gesundheitszentrums
std::vector<Geometrie::Point> end_points = last_points(polygon, gesundheitszent);

//output
std::cout << "Gesundheitszentrum: " << "(" << gesundheitszent.getX() <<"|" << gesundheitszent.getY() <<
std::cout << "Doerfer innerhalb des Radiuses von dem Gesundheitszentrum: " << std::endl;
int counter = 1;
for (const Geometrie::Point p : gesundheitszentrum_Points) {
 std::cout << "Das " << counter << "Dorf liegt bei den koordinaten: " << "(" << p.getX() << "|"
 << p.getY() << ")" << std::endl;
 counter++;
}
std::cout << "Restliche doerfer ausserhalb des Radiuses des gesundheitszentrums: " << std::endl;
for (const Geometrie::Point p : end_points) {
 std::cout << "Das " << counter << "Dorf liegt bei den koordinaten: " << "(" << p.getX() << "|"
 << p.getY() << ")" << std::endl;
 counter++;
}



// Zustaz Plott erstellen mit matplotlibcpp
std::vector<double> _x_werte;
std::vector<double> _y_werte;
for(const Geometrie::Point p : end_points){
 _x_werte.push_back(p.getX());
 _y_werte.push_back(p.getY());
}

//std::cout << "groesse ohne algo: " << test.size() << std::endl;
std::vector<double>x_werte;
std::vector<double>y_werte;
for(Geometrie::Point p : gesundheitszentrum_Points){
 x_werte.push_back(p.getX());
 y_werte.push_back(p.getY());
}

// Polygon plotten
std::vector<double> x_points;
std::vector<double> y_points;
for(const Geometrie::Point point : polygon.getPoints()){
 x_points.push_back(point.getX());
 y_points.push_back(point.getY());
}
x_points.push_back(x_points[0]);
y_points.push_back(y_points[0]);

// Plot erstellen und Polygon plotten
plt::plot(x_points, y_points, "r-", x_werte, y_werte
,"bo", _x_werte, _y_werte, "ro");

// Optional: Fülle das Polygon
```

```
 plt::xlabel("X-Achse");
 plt::ylabel("Y-Achse");

 // Titel des Plots
 plt::title("Polygon mit matplotlibcpp");

 // Plot anzeigen
 plt::show();

 return 0;
}
```