Computational Geometry - Abgabe 5

1st Bartolovic Eduard
Hochschule München
München, Deutschland
eduard.bartolovic0@hm.edu

I. Inkreis eines Polygons

Für den Inkreis wird der Mittelpunkt und der Radius benötigt.

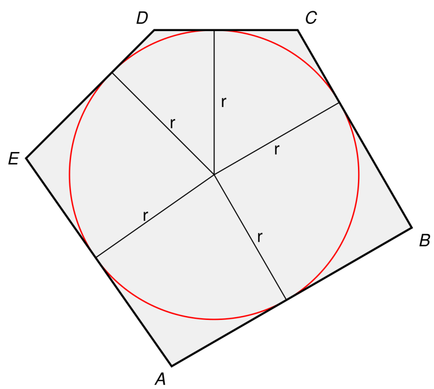

Abbildung 1. Darstellung eines Inkreis [1]

II. Idee 1:

A. Mittelpunkt des Inkreis

Um die Mittelpunkt eines Polygon zu Berechnen wird der durchschnitt aller Eckpunkte berechnet. Die Komplexität zum Berechnen des Mittelpunkt des Inkreis beträgt $\mathcal{O}(n)$.

B. Radius des Inkreis

Sobald der Mittelpunkt des Inkreises bekannt ist muss man nur noch den Radius berechnen. Dieser soll Maximal sein aber nicht über eine Kante des Polygons gehen. Hierfür berechnet man alle Distanzen zu allen Kanten des Polygons. Die kürzeste Distanz ist der Radius des Inkreis.
Zu Berechnung der Distanz vom Mittelpunkt zu einer Kante des Polygons wird die Flächenformel des Dreiecks $A = \frac{1}{2} * b * h$ zu $h = \frac{2*A}{b}$ umgestellt. Alles zusammen gesetzt ergibt die Formel zu Berechnung des Abstands von Punkt P_0 zur Strecke P_1P_2:

$$d(P_0, P_1, P_2) = \frac{|(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

C. Fazit

Die Komplexität zum Berechnen des Radius des Inkreis beträgt $\mathcal{O}(n)$.
Leider führt diese Idee nicht immer zu richtigen Lösung. In der Abbildung 2 ist einer dieser Fälle dargestellt.
Um das Problem zu lösen müssten mehrere Mittelpunkte

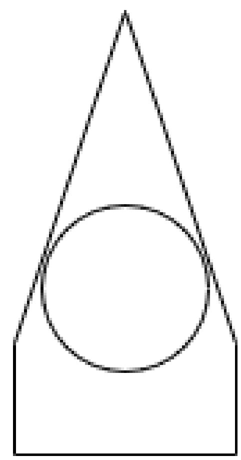

Abbildung 2. Problem des Verfahrens

getestet werden. Hierfür müsste man den voraussichtlich den Druchschnittspunkt aller Eckpunktkombinationen testen. Dies wären in einem naiven Ansatz $n!$ verschiedene Kombinationen. Da aber ein Durchschnitt mit drei Punkten erst sinn macht könnte man den Aufwand etwa reduzieren. Es bleibt aber trotzdem noch bei einem großen aufwand. Die Technik macht nur Sinn bei einem regulärem Polygon.

III. Lineare Programmierung

Es ist möglich die Lösung mithilfe der Linearen Optimierung zu lösen. So gibt es eine Matrix $A \in \mathbb{R}^{m,n}$, ein Vektor $\vec{b} \in \mathbb{R}^{m,1}$ und ein Vektor $\vec{c} \in \mathbb{R}^{1,n}$.
Es wird ein zulässiger Lösungsvektor $\vec{x} \in \mathbb{R}^n$ gesucht der

$$\vec{c}\vec{x} = c_1x_1 + c_2x_2 + ... + c_nx_n$$

maximiert. Das Optimierungsproblem kann so dargestellt werden:

$$\max\{\vec{c}\vec{x} | Ax \leq b, x \geq 0\}$$

[2] Unser Vektor $\vec{x}$ enthält den Mittelpunkt und Radius des Inkreises:

$$\vec{x} = \begin{pmatrix} x \\ y \\ r \end{pmatrix}$$

Die Zielfunktion ist:

$$\vec{c}\vec{x} = c_1 * x + c_2 * y + c_3 * r = -r$$

-r??????????0 0 -1????????????????????????
Der Ergebnisraum wird durch die Nebenbedingungen, die durch die Kanten des Polygons definiert werden. Bei n Kanten gibt es die n Nebenbedingungen. Für die Kante g_n entsteht die Ungleichung:

$$d(\begin{pmatrix} x \\ y \end{pmatrix}, g_n) \geq r$$

Dies kann umformuliert werden:

$$d(\begin{pmatrix} x \\ y \end{pmatrix}, g_n) = \vec{c}\vec{x} + a$$

$$c_1 * x + c_2 * y - a \geq r$$

$$c_1 * x + c_2 * y - r \geq a$$

$$-c_1 * x - c_2 * y + r \leq -a$$

Um auf c_1 und c_2 ist wie folgt definiert:

$$c_1 = \frac{p_2}{\sqrt{p_1^2 + p_2^2}}$$

$$c_1 = \frac{-p_1}{\sqrt{p_1^2 + p_2^2}}$$

p_1 und p_2 ist der Vektor der der Kante entspricht.

Zum Lösen dieses Problems wird eine Bibliotheksmethode verwendet.

Beispiel für ein Quadrat:
Das Quadrat ist definiert mit den Punkten:

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 10 \\ 0 \end{pmatrix}, \begin{pmatrix} 10 \\ 10 \end{pmatrix}, \begin{pmatrix} 0 \\ 10 \end{pmatrix}$$

Die resultierenden Nebenbedingungen sind:

$$c_2 * y + r \leq 0$$

$$-c_1 * x + r \leq -10$$

$$-c_2 * y + r \leq -10$$

$$c_1 * x + *r \leq 0$$

Der *linprog* Solver aus Matlab kommt auf das Ergebnis: $x = 5, y = 5, r = -5$.
Der Radius muss natürlich Positiv sein.

LITERATUR

[1] https://de.wikipedia.org/wiki/Inkreis#/media/Datei:Pentagon-inscribed-circle.svg
[2] https://de.wikipedia.org/wiki/Lineare_Optimierung

IV. ANHANG

Berechnung des Mittelpunkts des Polygons aus Idee 1

```
public Point getMiddle(){
  final double avgX = cords.stream().mapToDouble(p->
        p.getX()).sum()/(cords.size()-1);
  final double avgY = cords.stream().mapToDouble(p->
        p.getY()).sum()/(cords.size()-1);
  return new Point(avgX, avgY);
}
```

Berechnung des Inkreis des Polygons aus Idee 1

```
public Circle getLargestInscribedCircle(){
  //if(!Konvex) throw new IllegalArgument;
  final Point middle = getMiddle();

  double minDistance = Double.POSITIVE_INFINITY;
  for(int counter = 1; counter < cords.size() ; counter++){
    final Segment edge = new Segment(cords.get(counter-1), cords.get(counter));
    final double distance = edge.distanceToPoint(middle);
    if(distance < minDistance)
    minDistance = distance;
  }

  return new Circle(middle, minDistance);
}
```

Berechnung des Inkreis des Polygons mit Matlab und LP

```
C = [[0 0] ,[10 0], [10 10], [0 10], [0 0]];
x = reshape(C,2,length(C)/2);
x = transpose(x);

A = zeros(length(x)-1,3);
b = zeros(length(x)-1,1);
for i=1:length(x)-1
  first = x(i,:);
  second = x(i+1,:);

  p1 = second(1)-first(1);
  p2 = second(2)-first(2);

  n1 = p2 / sqrt(p1^2 + p2^2);
  n2 = -p1 / sqrt(p1^2 + p2^2);
  A(i,:) = [-n1,-n2,1];
  b(i) = -((n1)*first(1)+(n2)*first(2));
end

disp(A)
disp(b)
X = linprog([0 0 -1],A,b);
disp(X)
```