Softwareprojekt Anwendung von Algorithmen

Punkte in allgemeiner Lage

Julian Dobmann, Frederik Feiten, Felix Herter, Johannes Sauer

22. April 2013

Bacon ipsum dolor sit amet meatloaf id ball tip anim sunt, corned beef ad pork loin prosciutto non velit ut nisi deserunt. Eu beef ex tri-tip meatball in pork chop, officia drumstick proident laborum. Aute short loin spare ribs corned beef. Proident labore kielbasa ham hock sint ut, fugiat nisi prosciutto chuck chicken. Pig fugiat bresaola, ut strip steak boudin proident officia. Elit andouille pork chop ut cow adipisicing short loin, in et laborum dolore in.

Inhaltsverzeichnis

1 Einleitung

2 Datenstrukturen

2.1 Einführung der arithmetischen Unschärfe

Bedingt durch die endliche Darstellung von Fließkommazahlen im Prozessor waren wir gezwungen eine arithmetische Unschärfe einzuführen.
Wird zum Beispiel der Ausdruck $0.3 - 0.1$ berechnet, so wird anstelle des exakten Ergebnisses von 0.2 als nächstbeste Näherung:

```
js> 0.3-0.1
0.19999999999999998
```

zurückgeliefert. Die Differenz des exakten Wertes von der Näherung beläuft sich auf

```
js> 0.2-(0.3-0.1)
2.7755575615628914e-17
```

, weicht also um die Größenordung 10^{-16} von der, der Operanden ab. Wir haben uns entschienden die arithmetische Unschärfe auf 10^{-10} zu setzen. ((**remove me) Warum?**).

2.2 Punkte und Vektoren

Als grundlegende Datenstruktur wurden Vektoren eingeführt. Diese haben ihre zwei kartesischen Koordinaten als Attribute und verfügen über die grundlegenden Eigenschaften und Funktionalitäten, welche von ihnen zu erwarten sind (Vektoraddition/-subtraktion, Multiplikation mit einem Skalar, Skalarmultiplikation, etc.). **(remove me) explizit dokumentieren**
Ohne in ihrer programmatischen Funktionalität von den Vektoren abzuweichen, wurden Punkte eingeführt. Es stellte sich heraus, dass beide Strukturen ihre Notwendigkeit hatten (der Abstand zweier Punkte sollte als Vekor und nicht als Punkt ausgedrückt werden, wohingegen ein Graph als Tupel von Verbindungskanten und Punkten, nicht Vektoren dargestellt wird). **(remove me) Was gibts noch zu sagen?**

2.3 Kanten

Die `Edge` Datenstruktur repräsentiert eine Strecke in einem kartesischen Koordinatensystem, als auch eine Kante in einem Graph. Wir wählten aufgrund ihrer arithmetischen Vorzüge die Geradendarstellung in *Hessescher Normalform*. In ihr wird eine Gerade g dargeschtellt durch ihren *Normaleneinheitsvektor* $\widehat{n}$, sowie durch ihren Abstand zum Koordinatenursprung. Jeder Punkt $p = (x, y)$ auf der Geraden erfüllt somit die Gelichung:

$$\widehat{n}\vec{p} - d = 0 \tag{1}$$

, wobei $\vec{p}$ der Ortsvektor von p bezeichne.
Zusätzlich zu $\widehat{n}$ und d speichert die `Edge` Datenstruktur noch ihre zwei Endpunkte als Attribute.

2.3.1 Funktionen

`reload()` Aktualisiert für eine Kante ihren Normaleneinheitsvektor, sowie ihren Abstand zum Koordinatenursprung. Wird z.B. nach Verschieben eines Endpunktes aufgerufen.

`length()` Liefert die Länge einer Kante zurück.

`getY(x)` Lifert die y-Koordinate einer Geraden an x-Koordinate `x`. **in fancy?**

`getLeft()` Liefert den Endpunkt einer Kante mit geringerer x-Koordinate. (Haben beide Endpunkte die selbe x-Koordinate so liefert `getLeft()` einen anderen Punkt zurück als `getRight()`).

`getRight()` Analog zu oben.

`projectionToLine(pt)` Liefert die Projektion eines Punktes `pt` auf die Geraden durch **Alle Kanten benennen!**

`projectionToEdge(pt)` Analog zu `ProjectionToLine`, leifert jedoch `null` sollte die Projektion nicht in **Alle Kanten benennen!** liegen.

`distanceToLine(pt)` Liefert den Abstand eines Punktes zu

`signedDistanceToLine(pt)`

`lineContains(pt)`

`contains(pt)`

`lineIntersection(edge)`

`edgeIntersection(egde)`

3 Aufbau / Architektur !?

3.1 Speicherung der Highscores

Blabla

4 Algorithmen

4.1 Algo 1

4.2 Algo 2

4.3 Punkte befinden sich auf Kreis

Ein weiteres Kriterium für die Bewertung der allgemeinen Lage von Punkten, welches im Rahmen dieses Projekts umgesetzt wurde, besteht darin zu berechnen, ob mehr als drei Punkte auf einem Kreis liegen. Da drei Punkte immer auf einem Kreis liegen ist hierdurch noch keine Besonderheit gegeben, was die Lage der Punkte im Koordinatensystem anbelangt. Da mehr als drei Punkte jedoch nicht immer zwingend auf einem Kreis liegen müssen, kann eine solche Anordnung so als Sonderfall betrachtet werden, die einer allgemeinen Lage der Punkte wiederspricht.
Um dieses Kriterium zu betrachten wurde kein mathematischer Ansatz gewählt, der auf der Berechnung von Kreisgleichungen basiert, sondern eher ein grafischer. Dieser soll im Folgenden genauer erläutert werden.
Um zu untersuchen, ob verschiedene Punkte auf einem Kreis liegen könnte ein Ansatz darin bestehen viele Kreise mit verschiedenen Radien und Positionen durchzuprobieren und jedes Mal zu zählen, wie viele Punkte sich auf einem solchen Kreis befinden. Dabei würde es jedoch eine große Anzahl an Kreisen geben, auf denen sich gar keine der Punkte befinden und die somit umsonst betrachtet werden würden. Aus diesem Grund wurde hier der genau umgekehrte Ansatz gewählt die Kreise mit verschiedenen Radien um die Punkte selber zu ziehen und dann zu schauen, wie viele der Kreise sich im selben Schnittpunkt treffen. Gibt es einen Schnittpunkt in dem sich zum Beispiel vier Kreise treffen, dann bedeutet dies, dass sich um diesen Schnittpunkt ein Kreis mit dem gleichen Radius ziehen lässt, der dann wiederum genau diese vier Punkte schneidet.

Es muss somit also nur geschaut werden, ob es bei verschiedenen Radien Schnittpunkte gibt, in denen sich mehr als drei Kreise treffen. Die jeweiligen Schnittpunkte und Radien werden dann vom Algorithmus zurückgegeben und grafisch dargestellt.
Der Algorithmus erstellt hierfür zunächst ein Array, welches von den Proportionen der Größe des

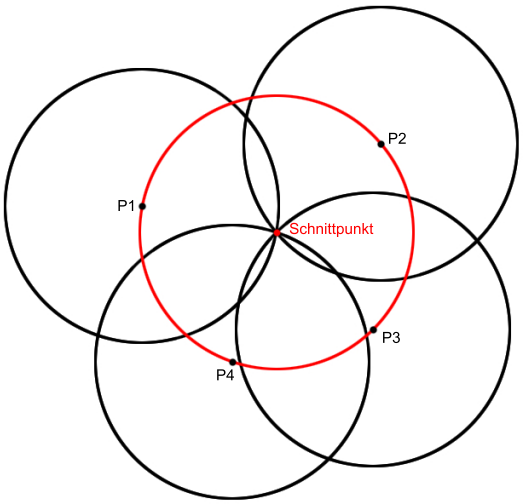

Abbildung 1: Umgekehrte Heransgehensweise um Punkte auf Kreis zu finden

Sichtbereiches entspricht. Die Kreise werden dann diskretisiert um die jeweiligen Punkte in das Array übertragen. Die Punkte sind hierbei vorher ebenfalls diskretisiert und auf die Größe des Arrays angepasst worden.
Die Kreise werden nicht einfach nur „gezeichnet“, sondern es wird im Array, welches vom Typ Integer ist, an jeder Position an der sich der Kreis befindet um Eins inkrementiert. Ein Kreis ist also zunächst durch eine diskrete Anordnung von Einsen im Array dargestellt. Wird nun ein Kreis um einen weiteren Punkt eingezeichnet, der mit dem vorherigen Kreis einen Schnittpunkt hat, so ergibt sich an dem Schnittpunkt im Array ein Wert von Zwei. Sind alle Kreise eingezeichnet worden muss so am Ende nur im Array nach allen Werten gesucht werden, die größer als drei sind. Hier befindet sich ein Schnittpunkt von mehr als drei Kreisen mit dem betrachteten Radius.

Die Genauigkeit mit der die Punkte auf einem Kreis liegen lassen sich im Algorithmus über die Größe des Arrays bestimmen. Je größer das Array, desto genauer liegen die Punkte am Ende auf dem Kreis. Hier wurde eine relativ kleine Größe des Array gewählt, sodass Punkte auch dann als auf dem Kreis liegend bewertet werden, wenn sie sich etwas daneben befinden.
Als Score ließe sich hier der Abstand der Punkte zu einer möglichen Lage auf einem Kreis angeben. In diesem Projekt wurde jedoch nur überprüft, ob sich mehr als drei Punkte mit einer gewissen Unschärfe auf einem Kreis befinden.

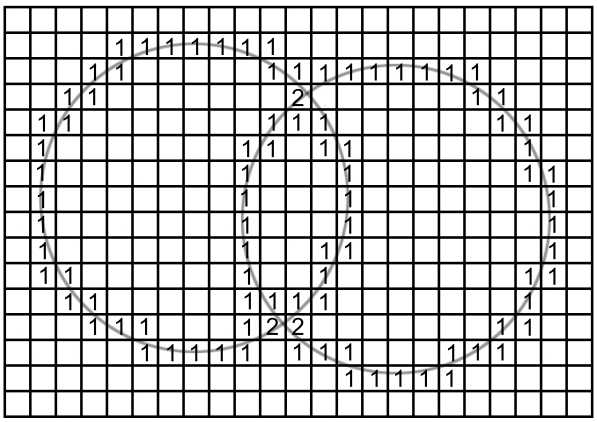

Abbildung 2: Überscheidung von zwei diskretisierten Kreisen im Array