Computational Geometry

Praktikum 1

Titel: Bestimmung der Anzahl der sich schneidenden Strecken inklusive Zeitmessung.
Gruppe: Johannes Walter, Luca Biege
Datum: 07.05.2024

Einleitung

Im folgenden Praktikumsversuch geht es darum, einen Code zu entwickeln, welcher die Anzahl von Schnittstellen vieler Strecken berechnet und diese ausgibt. Die Strecken sind durch ein Punktpaar mit den Koordinaten $[x_1, y_1, x_2, x_2]$ definiert und in einer *.dat* Datei gespeichert. Hierbei entpricht jede Zeile einer Strecke mit dem Startpunkt *P1* und dem Endpunkt *P2*. Die drei Dateien beinhalten 1000, 10.000 und 100.000 Strecken.
Dabei soll die Dauer für die Ausführung des Programms gemessen und verglichen werden.
Als Programmiersprache wird *Rust* verwendet.

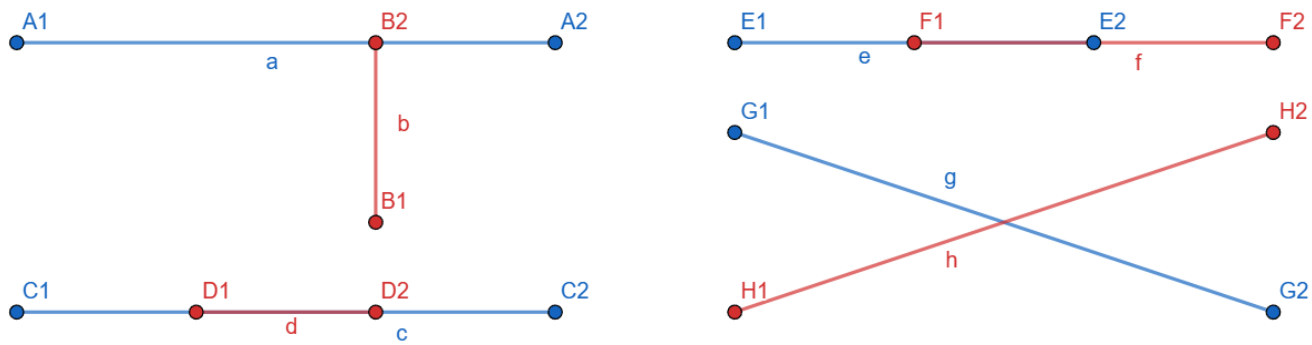
Grundlagen

Strecken unterscheiden sich von Geraden darin, dass sie einen klaren Start- und Endpunkt besitzen und damit eine endliche Länge haben. Eine Form zur Darstellung von Strecken mit dem Startpunkt P1 und dem Endpunkt P2 ist:

$$\vec{x} = \overrightarrow{P1} + t \cdot \vec{d} \quad | \, t \in \mathbb{R} \text{ mit } 0 \leq t \leq 1 \, |$$

$$\text{mit} \quad \vec{d} = \overrightarrow{P1P2} = \overrightarrow{P2} - \overrightarrow{P1}$$

Zwei Strecken haben einen gemeinsamen **Schnittpunkt**, wenn sie mindestens einen gemeinsamen Punkt beinhalten. Das gilt auch, wenn exakt der Endpunkt einer Strecke auf der anderen Strecke liegt (wie ein "T"), oder wenn die beiden Strecken auf derselben Geraden liegen und sich dabei berühren (teilweise oder komplett die eine Strecke innerhalb der anderen liegt). Die Abbildung zeigt exemplarisch Möglichkeiten für Streckenpaare mit (mindestens) einem Schnittpunkt:



Die **Determinante** einer 2x2 Matrix gibt eine Aussage über die Abhängigkeit der Vektoren einer Matrix. In diesem Fall wird sie verwendet um zu prüfen, wie zwei (Richtungs-) Vektoren zueinander stehen. Ist das Ergebnis der Berechnung $det(A) = 0$, so bedeutet dies, dass die Vektoren in dieselbe Richtung zeigen.

$$det(A) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = a \cdot d - c \cdot b$$

Das **Kreuzprodukt** zweier Vektoren ist ein Normalenvektor, also ein Vektor, welcher senkrecht auf diesen beiden Vektoren steht. Die Länge des Vektors gibt eine Aussage über die durch die beiden Vektoren aufgespannte Fläche. Sind die Vektoren parallel zueinander, spannen sie keine Fläche auf, d.h. das Kreuzprodukt $\vec{a} \times \vec{b} = 0$. Das Kreuzprodukt zweier 2-dimensionaler Vektoren lässt sich folgendermaßen berechnen:

$$\vec{a} \times \vec{b} = \begin{bmatrix} a1 \\ a2 \end{bmatrix} \times \begin{bmatrix} b1 \\ b2 \end{bmatrix} = a1 \cdot b2 - a2 \cdot b1$$

Ausführung

Zur besseren Übersicht sind die Koordinaten zeilenweise aus der Datei ausgelesen und einem (n x 4)-Vektor `points` als float-Werte gespeichert. Fehlerhafte Zeilen werden dabei übersprungen und vermerkt. Anschließend wird in einer verschachtelten for-Schleife jede Zeile von `points` ausgelesen und die Koordianten als `Point` P1 und P2, bzw. für die zweite Strecke als P3 und P4 gespeichert. Aus ihnen lassen sich die Richtungsvektoren D12 und D34 berechnen.

```
struct Point {
    x: f64,
    y: f64,
}
```

Im folgenden Abschnitt werden zwei gefundene Lösungsansätze für die weitere Berechnung erläutert und analysiert.

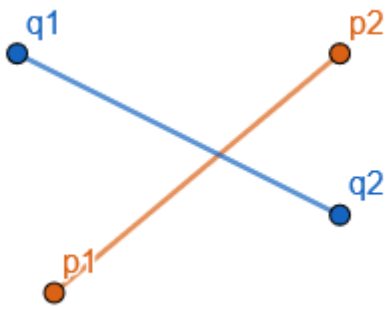
Ansatz geometrisch über Dreisatz

Ein elementarer Test, um herauszufinden, ob sich zwei Strecken schneiden, ist das Gleichsetzen der Parametergleichungen. Punkte entlang der Strecke werden dabei durch die Verwendung eines Parameters beschrieben. Beide Strecken werden durch eine Parametergleichung dargestellt.
Gegeben sind zwei Strecken:

$$\overline{P} := [p_1 p_2] \text{ und } \overline{Q} := [q_1 q_2] \text{ im } \mathbb{R}^2$$

Berechnen des Schnittpunkts durch Gleichsetzen:

$$p_1 + \lambda(p_2 - p_1) = q_1 + \mu(q_2 - q_1)$$



(Nach Prof. M. Fischer, HS München)

Die Parameter λ und μ repräsentieren Punkte entlang der Strecke. Sie nehmen immer Werte zwischen 0 und 1 an. So erhält man den Ausgangspunkt der Strecke für den Wert 0 und den Endpunkt der Strecke für den Wert 1. Löst man das obige Gleichungssystem und erhält für λ oder μ einen Wert außerhalb des Bereichs von 0 bis 1, bedeutet das, dass ein Punkt außerhalb der Strecken liegt und somit kein Schnittpunkt ist.

Dieser Ansatz lässt sich mit Stift und Papier leicht umsetzen. Für die Berechnung mit Code bietet sich jedoch eine Lösung mithilfe von Determinaten an, da dies in Code sehr viel lesbarer und anwendbarer umsetzen lässt.

Quelle: Vorlesungsunterlagen, 02Grundlagen, Prof. M. Fischer, HS München

Ansatz über Determinante

Um verschiedene Fälle möglichst Effizient abzuarbeiten ist es empfehlenswert, den common-case zuerst zu betrachten. So muss nicht bei jeder Berechnung zuerst alle Sonderfälle betrachtet werden, was unnötig Zeit kosten würde. Im Fall von zwei Strecken ist der common-case, dass die Ausrichtung dieser Strecken nicht parallel ist, sondern verschiedene Richtungen aufweisen.

Durch die Determinante lässt sich dies ermitteln, da im 2-dimensionalen die Determinante zweier Strecken $\overline{P}$, $\overline{Q}$ genau dann Null ergibt, wenn die Richtungsvektoren parallel sind.
Anschließend lässt sich über das Kreuzprodukt eines Richtungsverktors $\overrightarrow{u_P} = \overrightarrow{p_2} - \overrightarrow{p_1}$ zusammen mit dem Vektor zwischen den Ortsvektoren $\overrightarrow{p_1}$ und $\overrightarrow{q_1}$ beider Strecken ermitteln, ob die Linien zusätzlich kollinear sind. Der Quotient aus dem Kreuzprodukt und der Determinante ergibt ein Skalar t, über welches man den Schnittpunkt S durch einsetzen in die Geradengleichung der ersten Strecke erhält. Berechnet man nun mit diesem Schnittpunkt S das Skalar s der zweiten Strecke, so ergibt sich ein Schnittpunkt der Strecken, wenn:

$$0 \leq s, t \leq 1$$

Andernfalls kann überprüft werden, ob im Falle von Kollinearität eine partielle oder sogar vollständige Übereinstimmung der Geraden vorliegt. Hierzu wird lediglich die Reihenfolge der Start- und Endpunkte betrachtet und auf mindestens eine Überschneidung überprüft.

Validierung

Zur Validierung der Ergebnisse bietet sich das Prinzip von Unit-Tests an. Dabei lassen sich einfache Tests auf bestimmte Charakteristiken mit bekanntem Ergebnis, wie beim Prinzip des überwachten Lernens, erstellen und analysieren. So kann der Code auf sein Verhalten untersucht werden. Die Testfälle sind in der Abbildung im Kapitel Grundlagen bereits aufgegriffen. Weitere Edgecases lassen sich auf Basis dieser Fälle finden. Hierzu gehört beispielsweise der Schnitt im Endpunkt einer oder beider Strecken.

Auch unerwartete Daten müssen sichergestellt und korrekt interpretiert werden. Fehlerhafte Daten werden bereits beim Einlesen der *.dat* Datei herausgefiltert und als Fehler markiert. Diese sind glücklicherweise nicht vorgekommen.
Ein Sonderfall, welcher vorgekommen ist, ist eine Strecke der Länge Null. Hier lässt sich kein Richtungsvektor bilden, weshalb separat auf Punkt-auf-Strecke untersucht werden muss.

Um einen Unittest in Rust zu erstellen, wird das Attribut `#[test]` der Funktion vorgestellt. Wenn die zu testende Funktion mittels `cargo test` aufgerufen wird, wird eine *test runner binary*, also eine ausführbare Datei für automatisiertes Testing, gestartet. Der Return-Wert des Programms kann mit dem `assert_eq!()` Makro mit dem erwarteten Wert, in diesem Fall die Anzahl der sich schneidenden Strecken, verglichen werden und liefert bei Übereinstimmung (="equal") ein *Passed* als Testresultat. Der folgende Code zeigt beispielhaft einen kleinen Unittest für zwei Strecken, welche denselben Endpunkt besitzen.

```
#[cfg(test)]
mod tests {
    #[test]
    fn gleicher_endpunkt() {
        let points = &[
            (0.0, 0.0, 2.0, 2.0),
            (1.0, 3.0, 2.0, 2.0)];
        assert_eq!(calculate_intersections(points), 1);
    }
}
```

Ergebnis:

```
running 1 test
test tests::gleicher_endpunkt ... ok

successes:
    tests::gleicher_endpunkt

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 13 filtered out;
finished in 0.01s
```

Durch mehrere Tests an Edgecases lässt sich das Ergebnis der Berechnungen der drei gegebenen Dateien plausibilisieren.

Auswertung

Im vorliegenden Praktikum wurden drei Files mit einer unterschiedlichen Anzahl an Strecken ausgewertet.

Datei	Anzahl Strecken	Schnittpunkte	Berechnungsdauer
s_1000_1.dat	1000	11	18 ms
s_10000_1.dat	10.000	733	1.294 ms
s_100000_1.dat	100.000	77.171	124.300 ms

Die Laufzeit des Algorithmus liegt bei $O(n^2)$, da jede Strecke mit jeder anderen Strecke verglichen werden muss. Für die Laufzeiten der drei Dateien ergibt sich durch Teilen der Berechnungsdauer durch die quadrierte Anzahl an Strecken folgende Dauer:

Anzahl Strecken	1000	10.000	100.000
Dauer pro Vergleich (in ms)	1.8 e-5	1.3 e-5	1.2 e-5

Die Größenordnung stimmt bei allen drei Berechnungen überein und erscheint damit realistisch. Abweichungen kommen vor allem durch initiale Laufzeiten mit $O(1)$ zustande, welche bei längerer Laufzeit eine immer kleinere Rolle spielen.