

HOCHSCHULE MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Computational Geometry

Praktikumsaufgabe 1

Team: Christopher Hinz, Tobias Gruber

1 Aufgabenstellung

In dem Tar-File 'strecken.tgz' (s.u.) befinden sich Dateien mit jeweils 4 Koordinaten pro Zeile. Diese stellen jeweils die x- und y-Koordinaten eines Start- bzw. Endpunkts einer Strecke dar. Lesen Sie jeweils eine Datei ein und ermitteln Sie die Anzahl der sich schneidenden (d.h. mindestens ein gemeinsamer Punkt) Strecken, indem Sie jedes Paar von Strecken gegeneinander testen. Messen Sie die pro Datei aufgewendete Zeit. Begründen Sie nachvollziehbar, warum die Anzahl der von Ihrem Programm jeweils gefundenen Schnittpunkte korrekt ist.

2 Vorüberlegung

Um einschätzen zu können inwiefern die Ergebnisse sinnvoll sind sollen an dieser Stelle einige Vorüberlegungen getroffen werden.

Welche Anzahl an Schnitten sind zu erwarten bzw. in welchem Wertebereich sollten sich die Ergebnisse bewegen?

Die minimal zu erwartende Anzahl an Schnitten wäre für 1.000, 10.000 und 100.000 Strecken jeweils null. Dies ist somit die untere Grenze des zu erwartenden Wertebereichs. Die maximale Anzahl an Schnitten würden in dem Fall auftreten, dass sich alle Strecken gegenseitig schneiden. Dann wäre nämlich für jede Strecke ein Schnittpunkt mit allen anderen Strecken gegeben. Damit wären für 1.000 Strecken maximal $1.000 * 999 = 999.000$ (für 10.000 maximal $10.000 * 9.999 = 99.990.000$, für 100.000 maximal $100.000 * 99.999 = 9.999.900.000$) zu erwarten. Dies ist somit die obere Grenze des zu erwartenden Wertebereichs.

3 Umsetzung

3.1 Programm zur Aufgabenstellung

```
1 #include <fstream>
2 #include <iostream>
3 #include <string>
4 #include <vector>
5
6
7 struct point{
8     float x;
9     float y;
10 };
11
12 struct line{
13     point p1;
14     point p2;
15 };
```

```
16
17
18 void read_dat(char* filename, unsigned int N, std::vector<float>& v){
19     std::ifstream file;
20     file.open(filename);
21     float data;
22     for(int i = 0; i < N; ++i){
23         file >> data;
24         v.push_back(data);
25     }
26     file.close();
27 }
28
29
30 void pack_koords(std::vector<float>& source, std::vector<line>& target){
31     line temp;
32     for(unsigned int i = 0; i < source.size()/4; ++i){
33         temp.p1.x = source[0+i];
34         temp.p1.y = source[1+i];
35         temp.p2.x = source[2+i];
36         temp.p2.y = source[3+i];
37         target.push_back(temp);
38     }
39 }
40
41 float ccw(point p, point q, point r){
42     float res = (p.x*q.y - p.y*q.x) + (q.x*r.y - q.y*r.x) + (p.y*r.x - p.x
43     *r.y);
44     return res;
45 }
46
47 bool line_intersect_check(line l1, line l2){
48     bool retval = false;
49     float ccw_res1 = ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2);
50     float ccw_res2 = ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2);
51     if(ccw_res1 <= 0.0f && ccw_res2 <= 0.0f){
52         retval = true;
53         if(ccw_res1 == 0.0f && ccw_res2 == 0.0f){
54             float lambda1 = (l1.p1.x-l2.p1.x)/(l2.p2.x-l2.p1.x);
55             float lambda2 = (l1.p2.x-l2.p1.x)/(l2.p2.x-l2.p1.x);
56             if ((lambda1 < 0 || lambda1 > 1) && (lambda2 < 0 || lambda2 >
57             1)) retval = false;
58         }
59     }
60     return retval;
61 }
```

Listing 1: p1_lib.h: Bibliotheksfunktionen

```
1 #include "p1_lib.h"
2 #include <chrono>
3
4
```

```
5 int main(){
6     auto start = std::chrono::steady_clock::now();
7
8
9     std::vector<float> vec;
10    std::vector<line> lines_vec;
11    read_dat((char*)"strecken/s_1000_1.dat", 1000, vec);
12    pack_koords(vec, lines_vec);
13
14
15    int intersect_counter = 0;
16    for(unsigned int i = 0; i < lines_vec.size(); ++i){
17        for(unsigned int j = 0; j < lines_vec.size(); ++j){
18            if(i!=j){
19                if(line_intersect_check(lines_vec[i], lines_vec[j])){
20                    ++intersect_counter;
21                }
22            }
23        }
24    }
25
26    std::cout << "Strecken insgesamt: " << vec.size() << "\n" << "Schnitte
    zweier Strecken: " << intersect_counter << "\n";
27
28    auto end = std::chrono::steady_clock::now();
29    std::cout << "Runtime: "
30    << (float)std::chrono::duration_cast<std::chrono::microseconds>(end -
    start).count()/1000
31    + std::chrono::duration_cast<std::chrono::milliseconds>(end -
    start).count() << " ms\n";
32
33    return 0;
34 }
```

Listing 2: strecken.cpp: Aufruf der Bibliotheksfunktionen

Wie unten zu erkennen ist liegt die Anzahl der Schnitte für alle Fälle (1.000, 10.000, 100.000) im zu erwartenden Wertebereich. Schnittpunkte:

- 1.000 Strecken: $0 \leq 62.250 \leq 999.000$
- 10.000 Strecken: $0 \leq 6.247.500 \leq 99.990.000$
- 100.000 Strecken: $0 \leq 624.975.000 \leq 9.999.900.000$

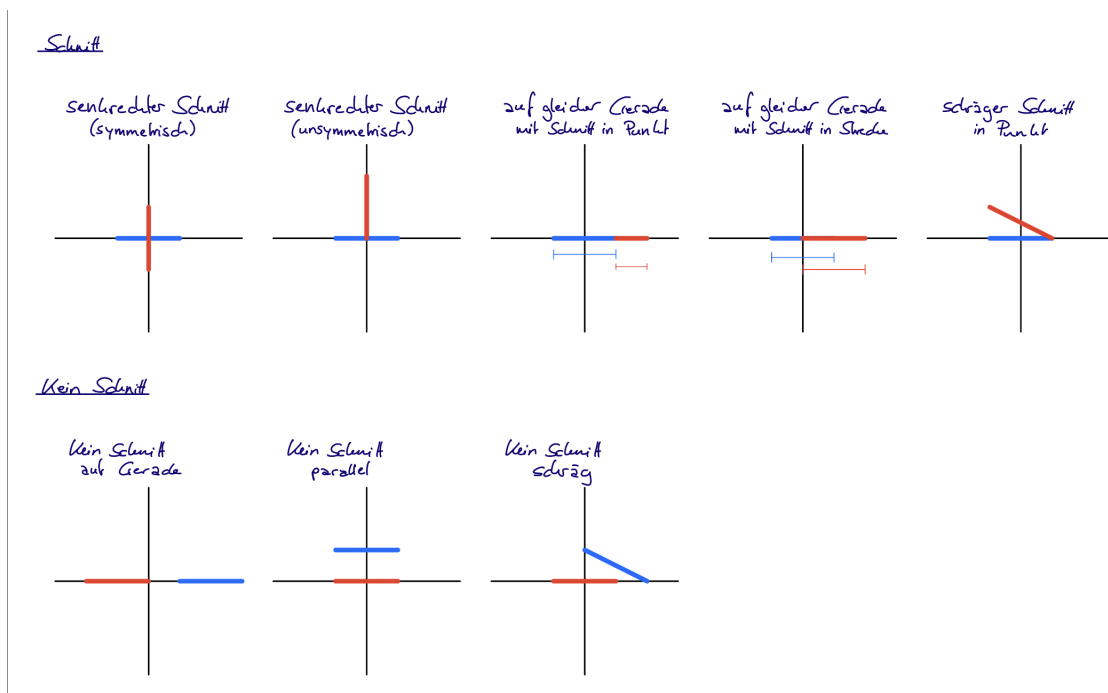
```
# s_1000_1.dat
Strecken insgesamt: 1000
Schnitte zweier Strecken: 62250
Runtime: 14.408 ms
#s_10000_1.dat
Strecken insgesamt: 10000
Schnitte zweier Strecken: 6247500
Runtime: 580.073 ms
```

```
#s_100000_1.dat
Strecken insgesamt: 100000
Schnitte zweier Strecken: 624975000
Runtime: 55100.2 ms
```

Listing 3: testing.cpp: Ausgabe Konsole

3.2 Programm zum Testen der Funktion

Neben den getroffenen Vorüberlegungen soll das Programm zusätzlich mit einigen definierten Testfällen validiert werden. Zum Sicherstellen der korrekten Funktion des Programms sollen verschiedene Streckenanordnungen getestet werden. Zum Veranschaulichen der getesteten Strecken dient das nachfolgende Bild.



Für jedes Streckenpaar ist zu berechnen, ob sie sich schneiden oder nicht.

```
1 #include "p1_lib.h"
2
3 void print_result(line l1, line l2, bool b_soll){
4     std::cout << std::boolalpha << "Soll: " << b_soll << ", Ist: " <<
5     line.intersect_check(l1, l2) << "\n";
6 }
7
8 int main(){
9
10     // Verschiedene Schnitte
11
12     // Strecken mit senkrechtem Schnitt (symmetrisch)
```

```

13 // p1 = (0,1) nach p2 = (0,-1)
14 // p1 = (1,0) nach p2 = (-1,0)
15 print_result(line{.p1.x = 0, .p1.y = 1, .p2.x = 0, .p2.y = -1}, line{.
p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 0}, true);
16
17 // Strecken mit senkrechtem Schnitt (unsymmetrisch)
18 // p1 = (0,2) nach p2 = (0,0)
19 // p1 = (1,0) nach p2 = (-1,0)
20 print_result(line{.p1.x = 0, .p1.y = 2, .p2.x = 0, .p2.y = 0}, line{.
p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 0}, true);
21
22 // Strecken auf gleicher Gerade mit Schnitt in einem Punkt
23 // p1 = (1,0) nach p2 = (2,0)
24 // p1 = (1,0) nach p2 = (-1,0)
25 print_result(line{.p1.x = 1, .p1.y = 0, .p2.x = 2, .p2.y = 0}, line{.
p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 0}, true);
26
27 // Strecken auf gleicher Gerade mit Schnitt in einer Strecke
28 // p1 = (0,0) nach p2 = (2,0)
29 // p1 = (1,0) nach p2 = (-1,0)
30 print_result(line{.p1.x = 0, .p1.y = 0, .p2.x = 2, .p2.y = 0}, line{.
p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 0}, true);
31
32 // Strecken auf gleicher Gerade mit Schnitt in einer Strecke
33 // p1 = (0,0) nach p2 = (2,0)
34 // p1 = (1,0) nach p2 = (-1,0)
35 print_result(line{.p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 1}, line{.
p1.x = 1, .p1.y = 0, .p2.x = -1, .p2.y = 0}, true);
36
37
38 // Verschiedene ohne Schnitt
39
40 // Strecken ohne Schnitt auf Gerade
41 // p1 = (-2,0) nach p2 = (0,0)
42 // p1 = (1,0) nach p2 = (3,0)
43 print_result(line{.p1.x = -2, .p1.y = 0, .p2.x = 0, .p2.y = 0}, line{.
p1.x = 1, .p1.y = 0, .p2.x = 3, .p2.y = 0}, false);
44
45 // Strecken ohne Schnitt parallel
46 // p1 = (-2,0) nach p2 = (0,0)
47 // p1 = (-2,1) nach p2 = (0,2)
48 print_result(line{.p1.x = -2, .p1.y = 0, .p2.x = 0, .p2.y = 0}, line{.
p1.x = -2, .p1.y = 1, .p2.x = 0, .p2.y = 2}, false);
49
50 // Strecken ohne Schnitt schraeg
51 // p1 = (-2,0) nach p2 = (0,0)
52 // p1 = (0,1) nach p2 = (2,0)
53 print_result(line{.p1.x = -2, .p1.y = 0, .p2.x = 0, .p2.y = 0}, line{.
p1.x = 0, .p1.y = 1, .p2.x = 2, .p2.y = 0}, false);
54
55 return 0;

```

56 }

Listing 4: testing.cpp: Testen der Bibliotheksfunktionen

Wie untenstehend gezeigt wird für alle Streckenpaare korrekt berechnet ob sie sich schneiden oder nicht.

```
Soll: true , Ist: true  
Soll: true , Ist: true  
Soll: true , Ist: true  
Soll: true , Ist: true  
Soll: true , Ist: true  
Soll: false , Ist: false  
Soll: false , Ist: false  
Soll: false , Ist: false
```

Listing 5: testing.cpp: Ausgabe Konsole