Übungsblatt 10

Julius Auer, Thomas Tegethoff

Aufgabe 1 (Voronoi-Diagramme):

Letztes Semester habe ich bereits für eine andere Lehrveranstaltung (Algorithmische Geometrie) Fortune's Sweep implementiert. Genau genommen haben wir dort den Algo nur theoretisch behandelt - Prof. Alt hielt eine Implementierung im Rahmen des Übungsbetriebs für zu aufwändig ;)
Ich hatte aber trotzdem Lust dazu, was sich nun auszahlt :)

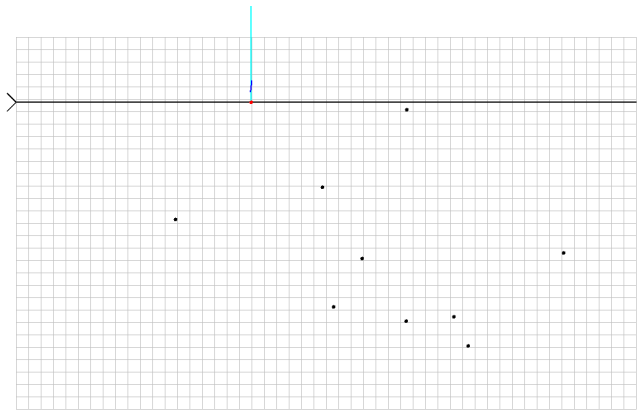
Der Code ist diesmal in Java. Die Implementierung benötigt eine Anzahl Hilfsklassen (Kreise, Strahlen, Strecken, etc.) weshalb ich die gesamte Codebasis aus Algorithmische Geometrie mit einreichen muss. Solltet Ihr den Code lesen wollen müsst Ihr deshalb etwas suchen, am besten in *geometry/algorithms/FortunesSweep*. Der Code ist mit im Jar.

Das abgegebene Jar ist ausführbar (erfordert Java 7) und visualisiert Fortune's Sweep für 80 zufällig generierte Punkte. Für Keyboard-Controls siehe stdout.

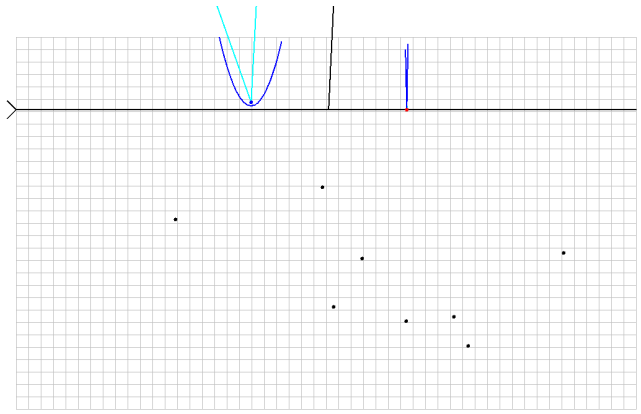
Für zehn zufällige Punkte zeigt Abbildung 1 die ersten vier Schritte des Algos und Abbildung 2 die letzten vier.

Die Plots sollten einigermaßen selbst-erklärend sein - ungewöhnlich sind vielleicht nur die Cyan-farbenen Linien, die für jedes Parabelsegment den Punkt visualisieren, der die zugehörige Parabel erzeugt hat. Man sieht so auch, wie viele Segmente einer Parabel noch "im Rennen" sind.

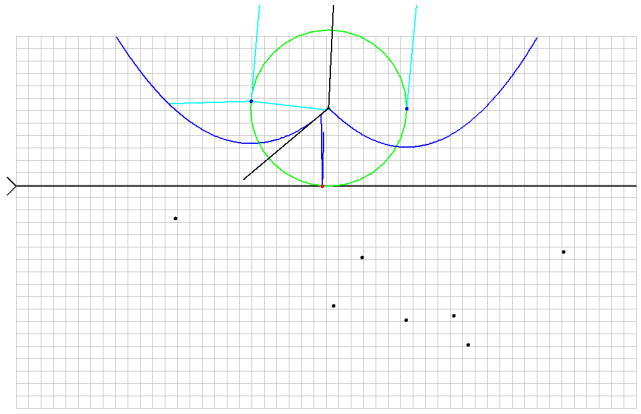
Es ist anzumerken, dass die Parabelsegmente hier nicht in einem Baum (sondern nur in einer Liste) gespeichert werden. Dadurch wird die worst-case Laufzeit quadratisch! Da hier die Visualisierung im Vordergrund stand und die Implementierung des Baums doch recht aufwändig gewesen wäre, haben wir das in Kauf genommen.



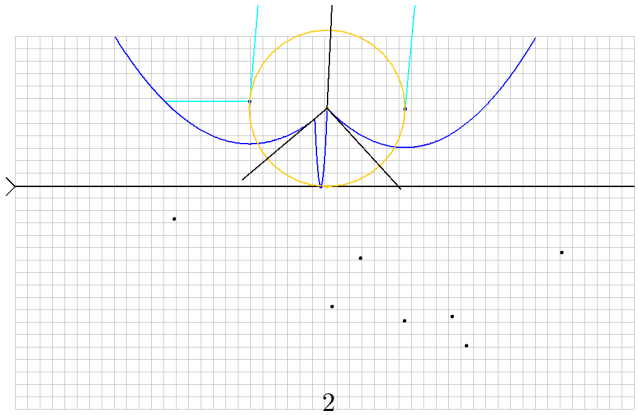
(a) Schritt 1



(b) Schritt 2

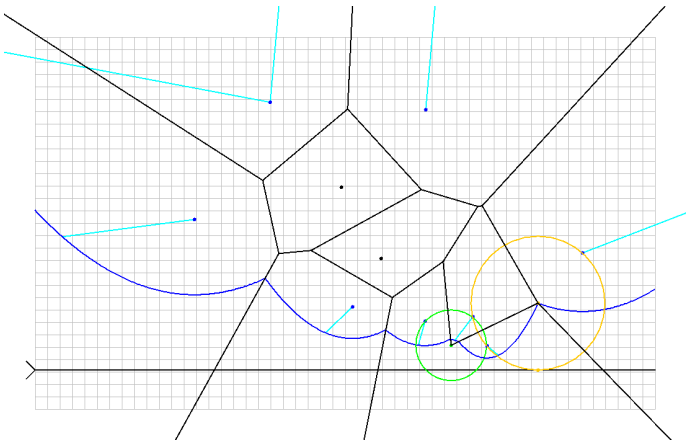


(c) Schritt 3

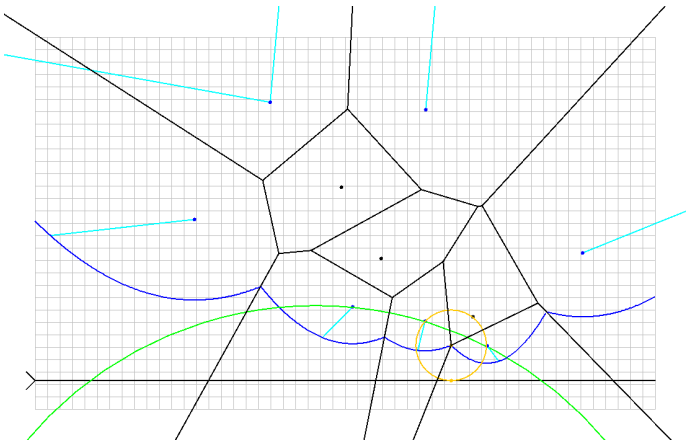


(d) Schritt 4

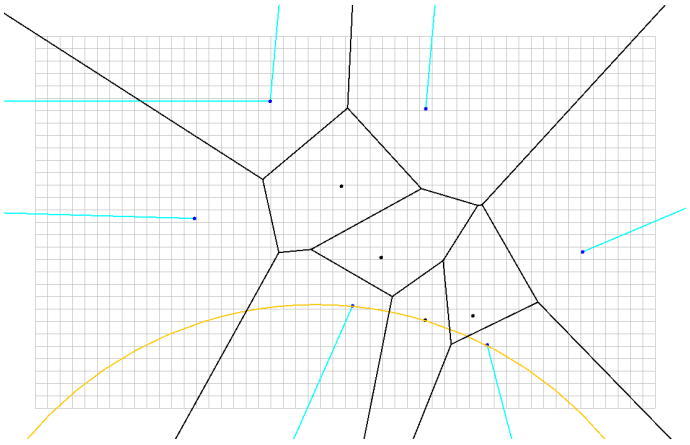
Abbildung 1: Die ersten Schritte von Fortune's Sweep



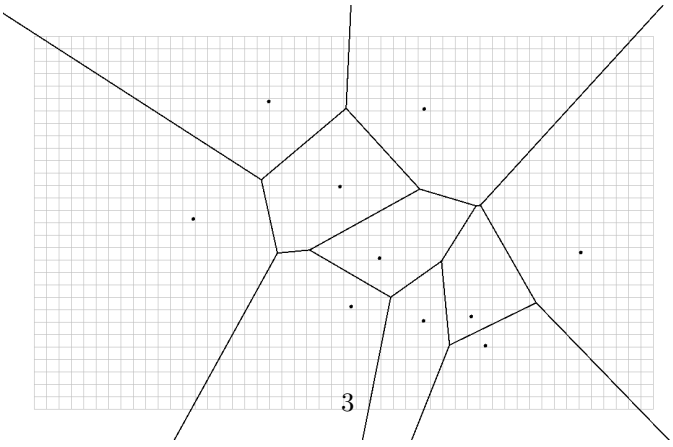
(a) Schritt n-3



(b) Schritt n-2



(c) Schritt n-1



(d) Schritt n

Abbildung 2: Die letzten Schritte von Fortune's Sweep

Aufgabe 2 (Potentialfelder):

Potentialfelder sind eine sehr flexible Waffe im Kampf gegen die Wirren der Pfadfindung im zweidimensionalen Raum. In Ermangelung eines konkreten Anwendungsfalls implementieren wir sie völlig willkürlich und zwar wie folgt:

- Additive Felder können schwierig zu handeln sein (bspw. wenn ein Hinderniss nahe des Ziels dieses unerreichbar macht), unsere Felder sind deshalb exklusiv: an jedem Pixel wirkt nur die Kraft des einen Feldes, das dort am stärksten ist.

- Alle attraktiven Felder haben im Zentrum den Wert -1 und alle repulsiven Felder den Wert 1. Mit zunehmendem Abstand nähert sich die Stärke des Feldes dem Wert 0 und zwar linear! Ein Feld f ist somit eindeutig durch $f = (x, y, r)$ beschrieben, wobei (x, y) das Zentrum des Feldes ist und r der Abstand, bei dem unter Annahme einer linearen Abschwächung die Stärke des Feldes 0 ist.

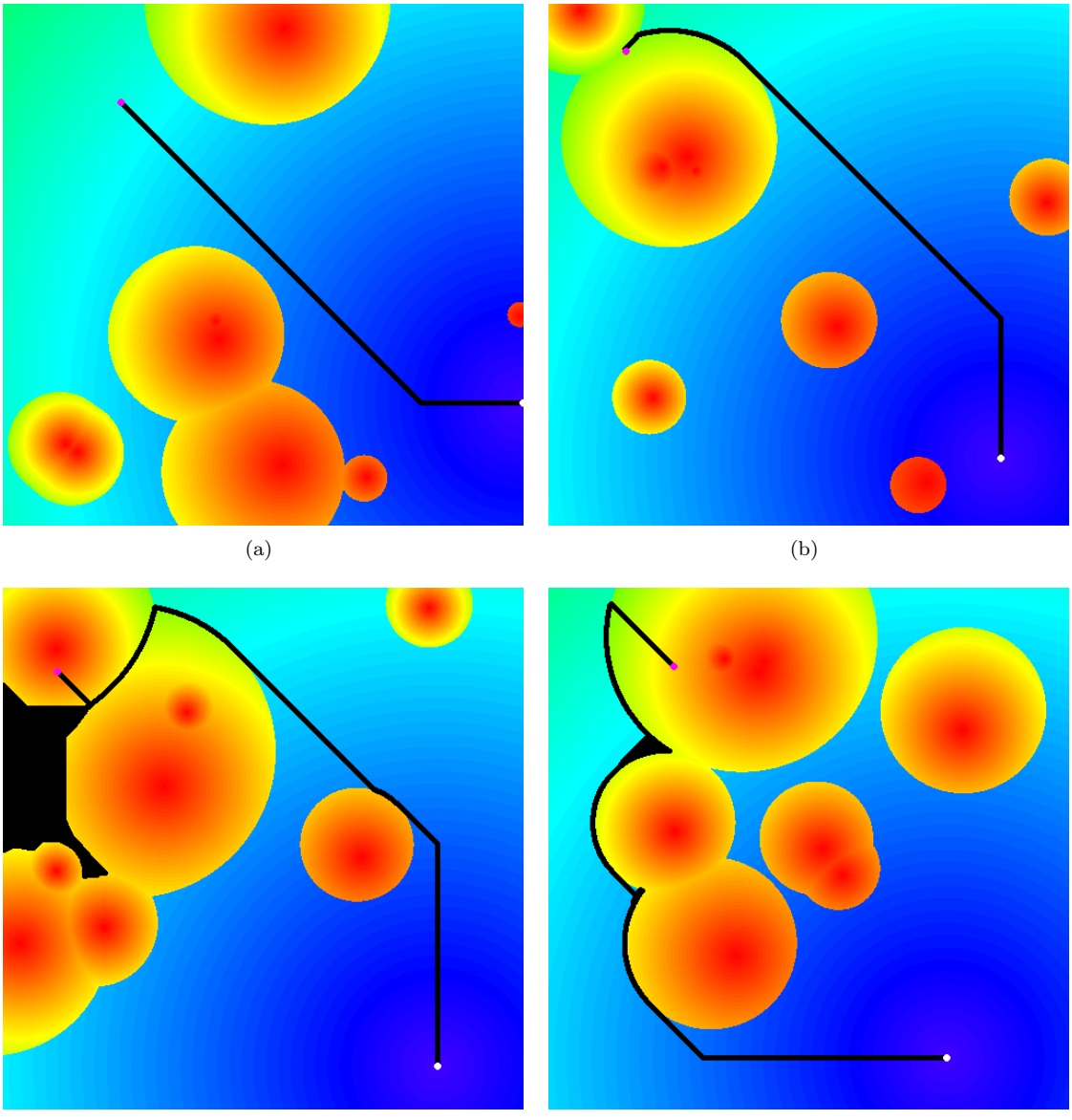
- Um aus lokalen Optima zu entkommen, weisen wir bei der Pfadsuche einem besuchten Pixel eine zusätzlich repulsive Kraft zu.

Im Experiment werden jeweils 8 repulsive Felder erzeugt, mit zufälligen x, y, r. Umso roter ein Pixel visualisiert ist, desto stärker ist dort die repulsive Kraft.

Außerdem wird ein Zielpunkt (weiss) zufällig (im unteren, rechten Quadranten) erzeugt und mit einem attraktiven Feld versehen. Umso blauer ein Pixel, desto stärker ist dort die attraktive Kraft.

Der Startpunkt ist lila und liegt zufällig im oberen linken Quadranten. Der gefundene Weg zwischen Start und Ziel ist schwarz eingezeichnet.

Abbildung 3 zeigt vier Instanzen des Experiments. Vor allem in (c) ist schön zu sehen, wie aus einem lokalen Optimum "entkommen" wird.



(a) (b)

(c) (d)

Abbildung 3: Potentialfelder