Algorithmische Geometrie

Prof. Dr. Sibylle Schwarz
HTWK Leipzig, Fakultät IM
Gustav-Freytag-Str. 42a, 04277 Leipzig
Zimmer Z 411 (Zuse-Bau)
https://informatik.htwk-leipzig.de/schwarz
sibylle.schwarz@htwk-leipzig.de

Sommersemester 2017

Nächste Tankstelle

Ziel:
Finden der (von der aktuellen Position aus) nächsten Tankstelle

bekannte Information:

- Karte mit eingezeichneten Tankstellen
- aktuelle Position in der Karte

Idee:
Hilfreich wäre eine Einteilung der Karte in Gebiete um jede Tankstelle
(Gebiet enthält alle Punkte, für die diese Tankstelle die nächste ist)

AG-Thema: Voronoi-Diagramme

Seenot

Ziel:
Zuordnung jedes Schiffes zu dem ihm am nächsten positionierten Schiff

bekannte Informationen:

- Karte mit aktuellen Positionen aller Schiffe

Idee: Für jedes Schiff

1. den Abstand zu jedem anderen Schiff berechnen und
2. das Schiff mit dem kleinsten Abstand auswählen.

Abstandsberechnung für jedes Paar von Schiffen wenigstens einmal, also Laufzeit wenigstens $\in O(|\text{Schiffe}|^2)$
Geht das besser (schneller)?

AG-Thema: nearest neighbour problem

Wächterproblem

Ziel:
Überwachung eines (verwinkelten) Raumes mit möglichst wenigen Wächtern (Kameras)
Bestimmung der Positionen der Wächter

bekannte Information:

- Grundriss des Raumes

AG-Thema: Triangulierungen

Hindernisse vermeiden

Ziel:
Beschädigungsfreie Navigation eines Haushaltroboters durch eine Wohnung

bekannte Information:

- Grundriss der Wohnung mit eingezeichneten Möbeln (Hindernissen)

AG-Thema: Wegplanung

Brückenplanung

Ziel:
Wo müssen für einen neu zu bauende Bahnstrecke Brücken über Flüsse und Straßen gebaut werden?

bekannte Information:

- mehrere Karten vom selben Gebiet
  (z.B. Wasserwege, Straßen, Planungsvarianten)

AG-Thema: Map Overlays
(Kombination der Information aus mehreren Karten)

Algorithmische Geometrie

beschäftigt sich mit:

Entwurf und Analyse von

Datenstrukturen und effizienten Algorithmen
speziell für geometrische Probleme,
d.h. Berechnungen mit geometrischen Objekten

Bestimmung der Komplexität (Ressourcenbedarf, Schranken) von

- Algorithmen
- geometrischen Problemen

Einsatzgebiete

Bildverarbeitung Objekterkennung, Features

Mustererkennung Schrift, Gestensteuerung

Computergraphik Ray tracing

Visualisierung Graph-Layout

Hardware-Entwurf VLSI-Design

CAD-Systeme Berechnung von Schnitt, Vereinigung, Interpolationsflächen

Geoinformationssysteme Navigationssysteme, Interpolation von Gebirgen zwischen Messpunkten

Robotik Bewegungsplanung

Produktionsplanung Werkstoffzuschnitt und -verbrauch

Datenanalyse , . . .

Geometrische Objekte

geometrischer Raum meist euklidischer Raum $\mathbb{R}^d$, oft mit $d \in \{2, 3\}$

Punkt $x \in \mathbb{R}^d$

Punktmenge $P \subseteq \mathbb{R}^d$ (endlich, unendlich)

Gerade (im $\mathbb{R}^2$) $\{x \in \mathbb{R}^2 \mid a_0 + a_1 x_1 + a_2 x_2 = 0\}$

Halbebene (im $\mathbb{R}^2$) $\{x \in \mathbb{R}^2 \mid a_0 + a_1 x_1 + a_2 x_2 \leq 0\}$

Strecke $\overline{pq} = \{x \in \mathbb{R}^d \mid \exists l \in [0, 1] : p + l(q - p)\}$

Polygon $P \subseteq \mathbb{R}^2$ (durch endlich viele zusammenhängende Strecken begrenzt)

Teilraum (Hyperebene) in $\mathbb{R}^d$

Halbraum in $\mathbb{R}^d$

Polyeder , Polytop

Viele geometrische Objekte sind unendliche Mengen (von Punkten).
Zur maschinellen Verarbeitung ist eine endliche Repräsentation dieser unendlichen Mengen notwendig.

Einordnung der LV Algorithmische Geometrie

Informatik Lehre von der Darstellung und Verarbeitung von Information durch Algorithmen

Teilgebiete der Informatik:

theoretisch
- Sprachen zur Darstellung von Information,
- Möglichkeiten und Grenzen der Berechenbarkeit

technisch
- maschinelle Darstellung von Information
- Mittel zur Ausführung von Algorithmen

Rechnerarchitektur, Hardware-Entwurf, Netzwerk, ...

praktisch Entwurf und Implementierung von Algorithmen: Algorithmen und Datenstrukturen, Betriebssysteme, Software-Entwicklung, Compilerbau, ...

angewandt Anwendung von Algorithmen: Text- und Bildverarbeitung, Datenbanken, KI, Medizin-, Bioinformatik, Geoinformationssysteme, Robotik, ...

Lernziele

aus Modulbeschreibung

http://portal.imn.htwk-leipzig.de/studierende/ordnungen-und-modulkataloge/dokumente/ordnungensaechshsg/ordnungen_in_abws2013/inb/MH-INB_09_2013.pdf

Die Studierenden

- können praktische Probleme auf geometrische Fragestellungen zurückführen. Sie können geometrische Probleme beurteilen und geeignete Datenstrukturen und Algorithmen zu deren Lösung einsetzen.
- sind in der Lage, sich anhand aktueller Fachliteratur eigenständig in Anwendungen auf verschiedenen Gebieten einzuarbeiten, ihre Erkenntnisse zu strukturieren und für eine Präsentation aufzubereiten.

Inhalt der Vorlesung

- konvexe Hülle endlicher Punktmengen
- Sweep-line-Verfahren
- Schnittpunkte endlicher Mengen von Strecken
- Triangulierung von Polygonen
- Punktsuche
- Voronoi-Diagramme endlicher Punktmengen
- Delauny-Triangulierungen endlicher Punktmengen
- effiziente geometrische Suchstrukturen (Bäume)

jeweils

- Anwendungen
- Datenstrukturen und Algorithmen
- Komplexität

Literatur

Folien zur Vorlesung,
jeweils nach der Vorlesung veröffentlicht

Bücher:

- M. de Berg et al:
  Computational Geometry, Algorithms and Application,
  Springer, 2008.
- R. Klein:
  Algorithmische Geometrie, Springer, 2005
- M. Joswig, T. Theobald:
  Algorithmische Geometrie, Vieweg, 2007
- F. P. Preparata, M. I. Shamos:
  Computational Geometry, Springer, Springer 1993.

Organisation der Lehrveranstaltung

Termine, aktuelle Änderungen, ... unter
`www.imn.htwk-leipzig.de/~schwarz/lehre/ss16/ag`

laut Modulbeschreibung:

- 60 h Präsenzstudium
- 30 h Vor- und Nachbereitung
- 60 h Projekt

Lehrveranstaltungen für jeden Studenten:

- wöchentlich zwei Termine (60 SWS)
- Vorlesungen, gelegentlich eine Übung (Prüfungsvorbereitung)
- Praktikum entsprechend Zeitplan

Prüfung

Klausur (120 min) im Juli 2016

Aufgabentypen wie Übungsaufgaben

Prüfungsvorleistungen:
Praktikum-Aufgaben entsprechend Zeitplan
als Prüfungsvorleistung + evtl. Noten-Bonus

Projekt: Zeitplan / Aufgaben

1. Mi 6.4.16 Festlegung Gruppen / Themen (Liste)
2. erste beide Wochen: Recherche
3. Mi 20.4.16 Präsentation (je 10 min):
   Vorstellung von
   - Wettbewerb, Ligen, Zielgruppen
   - Spielfeld, typische Aufgaben
   - Vorgaben (HW, SW, Team), wichtigste Regeln (kurz)
   - Lösungs-Ideen, -Ansätze
4. praktischer Teil:
   Erprobung verschiedener Lösungsansätze
   Dokumentation, Protokollieren von Ansätzen, Einstellungen, Schwierigkeiten usw.
5. Auswertung:
   - Präsentation der Lösungsansätze
   - Diskussion ähnlicher Aufgaben
   - Vergleich der Lösungsansätze und Realisierungen
   - Entwicklung anpassbarere Lösungen / Algorithmen
   - Abgabe Programme + Dokumentation
6. Vorbereitung / Betreuung Schüler-Aktionen

Geometrische Grundbegriffe

geometrischer Raum , meist euklidischer Raum $\mathbb{R}^d$

Punkt $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ (hier meist $d \in \{2, 3\}$)

Punktmenge $P \subseteq \mathbb{R}^d$

Beispiele für Punktmengen im $\mathbb{R}^2$:

- $\mathbb{R}^2, \emptyset$
- $\{(3,1),(0,2),(1,0)\} \subseteq \mathbb{R}^2$
- $\{x \in \mathbb{R}^2 \mid 2x_1 - x_2 - 1 = 0\}$ (Gerade)
- $\{x \in \mathbb{R}^2 \mid 2x_1 - x_2 - 1 \leq 0\}$ (Halbebene)
- $\{x \in \mathbb{R}^2 \mid 2x_1 - x_2 - 1 = 0 \wedge x_1 + x_2 = 2\} = \{(1,1)\}$
- $\{x \in \mathbb{R}^2 \mid 2x_1 - x_2 - 1 \leq 0 \wedge x_1 + x_2 = 2\}$ (Strahl)
- $\{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leq 2\}$ (Kreisfläche)
- $\{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leq 2 \wedge x_1 + x_2 = 0\}$ (Strecke)
- $\{(x,y) \mid x^2 + y^2 \geq 1\}$

Wiederholung: Abstand zwischen Punkten

Abstandsfunktion:

Funktion $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ mit allen folgenden Eigenschaften

- $\forall x, y \in \mathbb{R}^d : d(x, y) = 0$ gdw. $x = y$
- $\forall x, y \in \mathbb{R}^d : d(x, y) = d(y, x)$ (kommutativ)
- $\forall x, y, z \in \mathbb{R}^d : d(x, y) + d(y, z) \geq d(x, z)$ (Dreiecksungleichung)

prominente Abstandsfunktionen:

für alle $x = (x_1, \ldots, x_d) \in \mathbb{R}^d, y = (y_1, \ldots, y_d) \in \mathbb{R}^d$:

$$d_k(x, y) = \sqrt[k]{\sum_{i=1}^{m} (x_i - y_i)^k}$$

- $d_2(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$ (euklidisch)
- $d_1(x, y) = \sum_{i=1}^{d} |x_i - y_i|$ (Manhattan)
- $d_\infty(x, y) = \max\{|x_i - y_i| \mid i \in \{1, \ldots, d\}\}$ (Maximum)

Eigenschaften von Punktmengen

Punktmenge $P \subseteq \mathbb{R}^d$ heißt

endlich gdw. $P = \{p_1, \ldots, p_n\}$ für ein $n \in \mathbb{N}$

beschränkt gdw. $\exists c \in \mathbb{R}\ \forall p, q \in P : d(p, q) \leq c$

konvex gdw.
$\forall p, q, r \in \mathbb{R}^n\,(p \in P \wedge q \in P \wedge r \in \overline{pq} \rightarrow r \in P)$
P für jedes Paar von Punkten $p, q \in P$ alle Punkte auf der Strecke $\overline{pq}$ enthält

Lagebeziehungen im $\mathbb{R}^2$

Für Punkte $x, y \in \mathbb{R}^2$

Gerade durch x und y: (Hyperebene)

$$g(x,y) = \left\{ z \in \mathbb{R}^2 \mid \frac{x_2 - z_2}{x_1 - z_1} = \frac{x_2 - y_2}{x_1 - y_1} \right\}$$

Parameterform: $\overline{xy} = \{x + a(y - x) \mid a \in \mathbb{R}\}$

Halbgerade (Strahl) von $x \in \mathbb{R}^d$ durch $y \in \mathbb{R}^d$:

Parameterform: $\{x + a(y - x) \mid a \geq 0\}$

Strecke $\overline{xy}$ zwischen zwei Punkten $x, y \in \mathbb{R}^d$:

Parameterform: $\overline{xy} = \{x + a(y - x) \mid 0 \leq a \leq 1\}$

Polygon : Menge von Strecken (Kanten)
$\{\overline{p_1 p_2}, \overline{p_2 p_3}, \ldots, \overline{p_{n-1} p_n}\}$
meist repräsentiert durch Folge von Punkten (Ecken)
$(p_1, \ldots, p_n) \in (\mathbb{R}^d)^*$

Drei Punkte $x, y, z \in \mathbb{R}^d$ heißen kollinear gdw.
alle auf der selben Geraden liegen (z.B. $z \in g(x,y)$)

Polygone – endliche Repräsentation

Polygon: i.A. unendliche zusammenhängende Menge $P \subseteq \mathbb{R}^2$
Darstellungsformen eines Polygons $P \subseteq \mathbb{R}^2$ durch endliche

Menge seiner Kanten $P \in 2^{((\mathbb{R}^2)^2)}$

(Abschnitte der begrenzenden Geraden)

Folge seiner Eckpunkte $(q_1, \ldots, q_m) \in (\mathbb{R}^2)^*$

Ein Polygon $P \in \mathbb{R}^2$ ist genau dann konvex, wenn P ein Schnitt endlich vieler abgeschlossener Halbräume $H_1, \ldots, H_n \subseteq \mathbb{R}^2$ mit $H_k = \{x \in \mathbb{R}^d \mid a_{k,0} + \sum_{j=1}^{2} a_{k,j} x_{k,j} \leq 0\}$ ist, also

$$P = \bigcap_{k=1}^{n} H_k = \bigcap_{k=1}^{n} \left\{ x \in \mathbb{R}^d \mid a_{k,0} + \sum_{j=1}^{2} a_{k,j} x_{k,j} \leq 0 \right\}$$

Jedes konvexe Polygone ist durch endlich viele Koeffizienten $a_{k,j} \in \mathbb{R}$ also durch eine Matrix in $\mathbb{R}^{(d+1)\times n}$ eindeutig definiert.
nicht-konvexe Polygone:
endliche Vereinigungen konvexer Polygone

Beispiele

1. $[(1,2),(2,3),(4,2),(0,3),(1,2)]$
   ist nichtkonvexes Polygon (Liste der Ecken)
2. $\{((1,2),(0,3)),((2,3),(4,2)),((2,3),(1,2))\}$
   ist konvexes Polygon (Menge der Kanten)
3. $\{x \in \mathbb{R}_2 \mid x_1 + x_2 \leq 1 \wedge x_1 - x_2 \leq 1 \wedge x_2 \geq 0\}$
   ist konvexes Polygon (Fläche)
4. $\{x \in \mathbb{R}_2 \mid x_1 + x_2 \leq 1 \wedge x_1 - x_2 \leq 1\}$
   ist konvexes Polygon (Fläche)
5. $\{(x,y) \mid x^2 + y^2 \geq 1\}$ ist konvex, aber kein Polygon
6. $\{(x,y) \mid x^2 + y^2 \geq 1 \wedge x_1 - x_2 = 0\}$ ist konvexes Polygon

Maschinelle Repräsentation

Koordinaten $\in \mathbb{R}$

genaue maschinelle Darstellung reeller Zahlen nicht möglich, also: Gleitkommazahlen

Achtung: Rundungsfehler unvermeidbar

Punkte $\in \mathbb{R}^n$: Tupel von Koordinaten

endliche Punktmengen $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^n$:

eigentlich endliche Menge

praktisch meist endliche Liste von Punkten, da oft alle Punkte nacheinander (in einer bestimmten Reihenfolge) verarbeitet werden

Polygone $\subseteq \mathbb{R}^n$:

unendlich, daher explizite Angabe aller enthaltenen Punkte nicht möglich

endliche Repräsentation durch Angabe einer Reihenfolge aller Eckpunkte des Polygons

Beispiel: Polygone mit den Eckpunkte-Reihenfolgen $(0,0),(0,1),(1,1),(1,0)$ und $(0,0),(1,1),(0,1),(1,0)$ sind verschieden

Konvexe Hülle

gegeben: endliche Punktmenge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$

Definition (konvexe Hülle)

Für jede endliche Punktmenge $P \subseteq \mathbb{R}^d$ ist die konvexe Hülle $\text{conv}(P)$ von P die kleinste konvexe Punktmenge P' mit $P \subseteq P'$.

(kleinste bzgl. $\subseteq$)

Es gilt

$$\text{conv}(P) = \bigcap \{P' \mid P' \text{ konvex und } \ P \subseteq P'\}$$

Probleme für maschinelle Darstellung:

- Es gibt unendlich viele konvexe Punktmengen P' mit $P \subseteq P'$.
- $\text{conv}(P)$ ist eine unendliche Punktmenge.

Konvexe Hülle im $\mathbb{R}^2$ – Beobachtung

Für jede endliche Punktmenge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ ist die konvexe Hülle $\text{conv}(P)$ ein konvexes Polygon
(durch endlich viele zusammenhängende Strecken begrenzt).

anschaulich:
Gummiband um (endliche) Menge von Nadeln auf Pinnwand

also endliche Repräsentation von $\text{conv}(P)$ möglich

1. als Schnitt einer endlichen Menge von Halbebenen
   $\{x \in \mathbb{R}^2 \mid a_0 + a_1 x_1 + a_2 x_2 \leq 0\}$
   (jeweils Koeffizienten a_0, a_1, a_2)
2. durch die endliche Menge seiner Kanten $\overline{q_i q_j}$
   (jeweils repräsentiert durch ein Paar von Eckpunkten)
3. durch die endliche Folge seiner Eckpunkte $H = [q_0, \ldots, q_{m-1}]$
   Vereinbarung: im Uhrzeigersinn angeordnet)

Konvexe Hülle im $\mathbb{R}^2$ – mehr Beobachtungen

- Alle Ecken des Polygons $\text{conv}(P)$ sind Punkte aus P ($\{q_0, \ldots, q_{m-1}\} \subseteq P$).
- $\text{conv}(P)$ ist das (eindeutig bestimmte) konvexe Polygon,
  1. welches alle Punkte aus P enthält ($P \subseteq \text{conv}(P)$) und
  2. dessen Ecken $\{q_0, \ldots, q_{m-1}\}$ alle in P enthalten sind ($\{q_0, \ldots, q_{m-1}\} \subseteq P$)
- Bei Anordnung aller Ecken $H = [q_0, \ldots, q_{m-1}]$ von $\text{conv}(P)$ im Uhrzeigersinn gilt:
  Alle Punkte in P liegen auf oder rechts jeder Seite $\overline{q_i q_{((i+1) \mod m)}}$

Berechnung der konvexen Hülle – Idee

gegeben: endliche Punktmenge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

gesucht: $\mathrm{conv}(P)$, repräsentiert durch Folge $H = [q_0, \ldots, q_{m-1}]$ seiner Eckpunkte (im Uhrzeigersinn)

verwendetes Wissen:

1. $\{q_0, \ldots, q_{m-1}\} \subseteq P$,
2. alle Punkte in P liegen auf oder rechts jeder Kante $\overline{q_i q_{((i+1) \bmod m)}}$,
3. wenn für zwei Punkte $p_i, p_j \in P$ alle Punkte aus P rechts von der Strecke $\overline{p_i p_j}$ liegen, dann ist $\overline{p_i p_j}$ eine Kante von $\mathrm{conv}(P)$

Idee:

- Beginn mit leerer Kantenmenge von $\mathrm{conv}(P)$
- Für jedes Paar p_i, p_j verschiedener Punkte aus P testen, ob alle Punkte aus P rechts von der Strecke $\overline{p_i p_j}$ liegen
  falls ja, Strecke in Kantenmenge von $\mathrm{conv}(P)$ einfügen
- Kanten sortieren, Eckenfolge ablesen

Konvexe Hülle im $\mathbb{R}^2$ – Algorithmus

Algorithmus: conv-1

Eingabe: $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

Ausgabe: $H = [q_0, \ldots, q_{m-1}]$ (Eckenfolge von conv(P))

$K \leftarrow \emptyset$ (Kantenmenge)

für jedes *Paar* $(i, j) \in \{1, \ldots, n\}^2$ *mit* $p_i \neq p_j$ **:**

- Alle-rechts-von$(p_i, p_j) \leftarrow$ wahr
- **für jedes** $k \in \{1, \ldots, n\}$ *mit* $p_i \neq p_k$ *und* $p_j \neq p_k$ **:**
  - **wenn** p_k *links der Strecke* $\overline{p_i p_j}$, **dann**
    - Alle-rechts-von$(p_i, p_j) \leftarrow$ falsch
- **wenn** *Alle-rechts-von*$(p_i, p_j) =$ *wahr* **dann**
  - $K \leftarrow K \cup \{(p_i, p_j)\}$

Kanten in K sortieren und Folge $H = [q_0, \ldots, q_{m-1}]$ ablesen

Laufzeit:

$L(\text{conv-1}) \in O((n)^3 L(\text{rechts?}) + L(\text{Kanten-sortieren}))$

Konvexe Hülle – Rechts-von-Test

Wie kann man feststellen, ob ein Punkt $p \in \mathbb{R}^2$ rechts von einer Strecke $\overline{qr}$ liegt?

Berechnung der Determinante Laufzeit $O(1)$

$$\begin{vmatrix} p_1 & p_2 & 1 \\ q_1 & q_2 & 1 \\ r_1 & r_2 & 1 \end{vmatrix}$$

< 0: p liegt rechts von $\overline{qr}$
> 0: p liegt links von $\overline{qr}$

- Was geschieht bei kollinearen p, q, r ?
- Wie wirken Rundungsfehler?
  (gleichzeitig r rechts von $\overline{pq}$ und p rechts von $\overline{rq}$ und q rechts von $\overline{pr}$ oder nichts davon)

Konvexe Hülle – Kanten-sortieren

Algorithmus: Kanten-sortieren

Eingabe: Menge $K = \left\{(p_i, q_i) \in \left(\mathbb{R}^2\right)^2 \mid i \in \{1, \ldots, n\}\right\}$

Ausgabe: Folge $H = [q_0, \ldots, q_{m-1}] \in \left(\mathbb{R}^2\right)^*$

beliebige Kante $(p_i, q_i) \in K$ auswählen
$K \leftarrow K \setminus \{(p_i, q_i)\}$
$H \leftarrow [p_i]$
solange $K \neq \emptyset$ **:**
  Kante $\left(q_{|H|}, q'\right) \in K$ auswählen ($q_{|H|}$ = letzer Punkt in H)
  $K \leftarrow K \setminus \{(q_{|H|}, q')\}$
  $H \leftarrow H \circ q'$

Laufzeit: $O(m^2)$ (naiv), $O(m(\log m))$ möglich, $m \leq n$

Gesamtlaufzeit zur Berechnung der konvexen Hüllle mit conv-1:
$L(\text{conv} -1) \in O((n)^3 + n(\log n)) = O(n^3)$

Algorithmus conv-1 – Probleme

- Laufzeit $O(n^3)$ sehr hoch
- kaum robust
  - in Spezialfällen und
  - gegenüber Rundungsregeln

Was bisher geschah

- Motivation, Beispiele
- geometrische Objekte im $\mathbb{R}^2$:
  Punkt, Gerade, Halbebene, Strecke, Polygon
- maschinelle Repräsentation geometrischer Objekte
- konvexe Mengen im $\mathbb{R}^d$
- konvexe Hülle $\text{conv}(P)$ endlicher Punktmengen $P \subseteq \mathbb{R}^d$
- Beobachtungen im Spezialfall $\mathbb{R}^2$:
  - $\text{conv}(P)$ ist ein Polygon (darstellbar durch Eckenfolge $H = [q_0, \ldots, q_{m-1}]$ im Uhrzeigersinn)
  - $\forall p, q \in P$: $\overline{pq}$ Kante von $\text{conv}(P)$ gdw. $\forall r \in P$: r liegt rechts von $\overline{pq}$

Algorithmus conv-1

- Entwurf folgt direkt aus Definition von $\text{conv}(P)$
- Nachteile:
  - Laufzeit $O(n^3)$ sehr hoch
  - kaum robust
    - in Spezialfällen und
    - gegenüber Rundungsfehlern

Konvexe Hülle – bessere Idee

inkrementeller Algorithmus:

- Durchlaufen der Punkte in P
  - in einer sinnvollen Reihenfolge und
  - gleichzeitige Erzeugung / Verwaltung von Zusatzinformationen
- Repräsentation (und Verarbeitung) von P als geordnete Menge (Folge) $[p_1, \ldots, p_m]$
- Ordnung aufsteigend nach x-Koordinate
  (Punkte werden von links nach rechts verarbeitet)
- Berechnung der konvexen Hülle in zwei Durchläufen, oberen und unteren Teil

Obere und untere konvexe Hülle

gegeben:
$P = [p_1, \ldots, p_n]$ aufsteigend geordnet nach x-Koordinate
$H_o = [q_0, \ldots, q_k]$ ist obere konvexe Hülle von P gdw.

1. $[q_0, \ldots, q_k]$ zusammenhängende Teilfolge der (evtl. zyklisch verschobenen) Eckenfolge von $\mathrm{conv}(P)$,
2. $\mathrm{head}\ (H_o) = q_0 = p_1$
3. $\mathrm{head}\ (\ \mathrm{reverse}\ (H_o)) = q_k = p_n$

analog $H_u = [q_0, \ldots, q_k]$ ist untere konvexe Hülle von P gdw.

1. $[q_0, \ldots, q_k]$ zusammenhängende Teilfolge der (evtl. zyklisch verschobenen) Eckenfolge von $\mathrm{conv}(P)$,
2. $\mathrm{head}\ (H_u) = q_0 = p_n$
3. $\mathrm{head}\ (\ \mathrm{reverse}\ (H_u)) = q_k = p_1$

Eckenfolge der konvexe Hülle von P:

$$\mathrm{conv}(P) = H_o \circ \mathrm{tail}(H_u)$$

Beobachtung: H_o und H_u enthalten keine „Linkskurven“

Berechnung der oberen konvexen Hülle

gegeben:
$P = [p_1, \ldots, p_n]$, aufsteigend nach x-Koordinate geordnet

Idee: Linksabbiegen verboten (konvex)

schrittweise Berechnung der oberen konvexe Hülle H_0:
in jedem Schritt k:

- Punkte $[p_1, \ldots, p_k]$ sind abgearbeitet,
  dabei wurde $H_o = [q_1, \ldots, q_i]$ berechnet
  (mitgeführte Zusatzinformation)
- nächsten Kandidaten p_{k+1} an H_o anhängen ($H_o \leftarrow H_o \circ p_{k+1}$)

  - falls p_{k+1} nicht rechts von $\overline{q_{i-1}q_i}$,
    Korrektur der Folge $H_o = [q_1, \ldots, q_{i-1}, q_i, p_{k+1}]$ notwendig:
    solange p_{k+1} nicht rechts von $\overline{q_{i-1}q_i}$:
    q_i aus H_o entfernen und Rechts-Test für $i \leftarrow i - 1$ wiederholen
  - sonst ok

Warum „nicht rechts von“ statt „links von“?

Konvexe Hülle – Algorithmus 2

Algorithmus: conv-2

Eingabe: $P \subseteq \mathbb{R}^2$ endlich
Ausgabe: $H = [q_0, \ldots, q_{m-1}]$ (Eckenfolge von conv(P))

$[p_1, \ldots, p_n] \leftarrow P$ aufsteigend nach x-Koordinate sortieren
$H_o \leftarrow [p_1, p_2]$ (Berechnung obere konvexe Hülle)
für jedes $k \leftarrow 3, \ldots, n$ **:**
  $H_o \leftarrow H_o \circ p_k$
  solange $|H_o| \geq 2$ *und* $q_{|H_o|}$ *nicht rechts von* $(q_{|H_o|-2}, q_{|H_o|-1})$ **:**
    $q_{|H_o|-1}$ aus H_o entfernen (vorletzter Punkt)

$H_u \leftarrow [p_n, p_{n-1}]$ (Berechnung untere konvexe Hülle)
für jedes $k \leftarrow n-2, \ldots, 1$ **:**
  $H_u \leftarrow H_u \circ p_k$
  solange $|H_u| \geq 2$ *und* $q_{|H_u|}$ *nicht rechts von* $(q_{|H_u|-2}, q_{|H_u|-1})$ **:**
    $q_{|H_u|-1}$ aus H_u entfernen (vorletzter Punkt)

$q_{|H_o|}$ aus H_o entfernen
$q_{|H_u|}$ aus H_u entfernen
$H = H_o \circ H_u$

Korrektheit des Algorithmus conv-2

(unter idealen Bedingungen, keine Rundungsfehler)

Behauptung:
Die von conv-2 für eine endliche Punktmenge $P \subseteq \mathbb{R}^2$ berechnete Folge H ist eine Eckenfolge von conv(P).

Nachweis für H_o und H_u getrennt (analog Berechnung)

Ausgangspunkt:
$[p_1, \ldots, p_n]$ ist die korrekt (lexikographisch) nach x-Koordinaten aufsteigend geordnete Folge aller Punkte in P
(durch korrektes Sortierverfahren bestimmt)

zu zeigen: für jedes $i \in \{2, \ldots, n\}$ gilt:
Der nach Verarbeitung des Punktes p_i berechnete Teil von H_o ist die obere konvexe Hülle der Punkte $[p_1, \ldots, p_i]$.

Idee: Induktion über n

Laufzeit und Robustheit

Berechnung von H_o und H_u jeweils $\in O(n)$. Warum?
Sortieren von n Punkten $\in O(n \log n)$
Laufzeit:

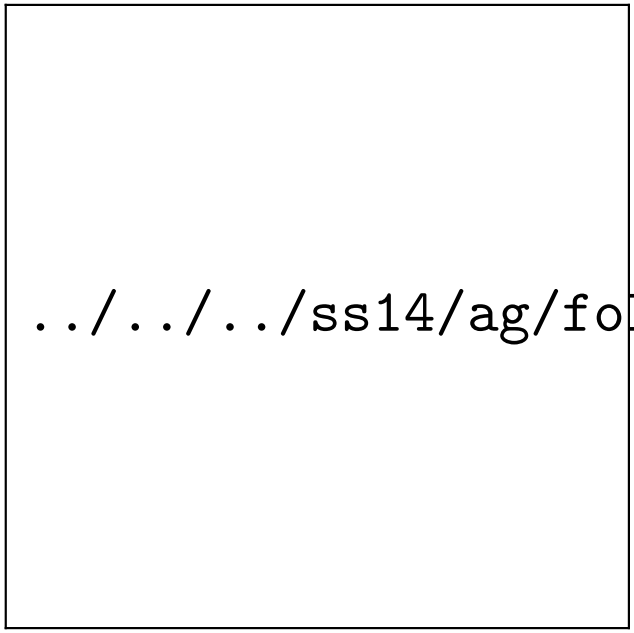
$$\begin{aligned} L(\text{conv} - 2) &= L(\text{Punkte} - \text{sortieren}) + L(H_o) + L(H_u) + L(\circ) \\ &= O(n \log n) + 2O(n) + O(1) \in O(n \log n) \end{aligned}$$

mögliche Probleme:

- P kann Punkte mit denselben x-Koordinaten enthalten
  Lösung: lexikographische Ordnung (erst x, dann y)
- P kann kollineare Punkte enthalten
  kein Problem wegen Test auf „nicht rechts von “
- Rundungsfehler führen nicht zu strukturellen, sondern höchstens zu kleinen lokalen Fehlern (für die meisten Anwendungen harmlos)
  lässt sich durch stärkeres Runden vermeiden, welches Punkte mit geringem Abstand zu einem zusammenfügt

Konvexe Hülle – Anwendungen

- Abschätzung von Kollisionsrisiken
- Hindernisumfahrung (Wegplanung)
- Mustererkennung
  - Objekterkennung, Segmentierung in digitalen Bildern
  - Texterkennung
  - Bestimmung von Bounding-Boxen
  - Gestenerkennung
    (Volumenverhältnis von Objekt und konvexer Hülle)
- Bioinformatik



- Medizininformatik (Ausdehnung von Krankheitsherden)

Schritte beim Entwurf geometrischer Algorithmen

1. Problemanalyse / Spezifikation:
   - Eingaben
   - Ausgaben
   - (mathematischer) Zusammenhang zwischen Ein- und Ausgaben
2. maschinelle Darstellung der Ein- und Ausgaben
3. Entwurf eines Lösungsalgorithmus (zunächst Abstraktion von Spezialfällen und Rundungsfehlern)
4. Analyse von Robustheit und Ressourcenverbrauch des entworfenen Algorithmus
5. evtl. weitere Problemanalyse und Entwurf besserer Algorithmen
6. Implementierung mit Fehlerabschätzung

Was bisher geschah

- Motivation, Beispiele
- geometrische Objekte im $\mathbb{R}^2$:
  Punkt, Gerade, Halbebene, Strecke, Polygon
- maschinelle Repräsentation geometrischer Objekte

effiziente Algorithmen und Datenstrukturen zur Berechnung

- der konvexen Hülle $\text{conv}(P)$ endlicher Punktmengen $P \subseteq \mathbb{R}^2$

Motivation

Art Gallery Problem (Wächterproblem):

Überwachung eines Museums (evtl. mit stark verwinkelten Räumen) durch möglichst wenige Kameras (omnidirektional)

- gegeben: Grundriss des Museums
  begrenzt durch einfachen geschlossenen Polygonzug
- gesucht: Menge (minimaler Mächtigkeit) von Positionen der Überwachungskameras

Abschätzung der notwendigen Kameraanzahl

Beobachtung:
Jedes Dreieck lässt sich durch eine Kamera überwachen

Idee:

- Zerlegung des Polygons in Dreiecke
- $|\text{Kameras}| \leq |\text{Dreiecke}|$

AG-Aufgabe Triangulierung

Triangulierung einfacher Polygone

einfaches Polygon P:
durch einen geschlossenen Polygonzug, deren nicht benachbarte Kanten keinen gemeinsamen Punkte enthalten, begrenztes Gebiet im R^2
Repräsentation als Graph $P = (V, E)$ (Kreis)

Diagonale in einem einfachen Polygon P:
vollständig in P enthaltene Strecke zwischen zwei Ecken von P

Triangulierung eines einfachen Polygons $P = (V, E)$:
Zerlegung von P in Dreiecke durch maximale Menge (bzgl. $\subseteq$) von Diagonalen, die einander paarweise nicht schneiden
(Graph $T = (V', E')$ mit $V' = V$ und $E \subseteq E'$)

Maschinelle Repräsentation

Begrenzung von Polygonen und Triangulierungen sind planare Graphen $P = (V, E)$.

Begrenzung von Polygonen: Kreise

maschinelle Repräsentation von Kreisen (V, E) als

- einfach verkettete Liste von Ecken (oder Kanten)
  z.B. als Ergebnis der Berechnung der konvexen Hülle
  (Angabe der Durchquerungsrichtung wichtig)
- doppelt verkettete Liste von Ecken oder Kanten
  (Angabe der Durchquerungsrichtung wichtig)

Vereinbarung: Kante $e_i = (v_i, v_{i+1})$
Transformation der einfach in eine doppelt verkettete Liste in $O(|V|)$ möglich.

maschinelle Repräsentation planarer Graphen als
doppelt verkettete Liste von Kanten

Anzahl der Dreiecke in Triangulierungen

Satz

- *Jedes einfache Polygon hat wenigstens eine Triangulierung.*
- *Jede Triangulierung eines einfachen Polygons mit $n > 2$ Ecken besteht aus $n - 2$ Dreiecken.*

Beweis durch Induktion über n

Folgerung:
Jedes einfache Polygon mit n Ecken kann mit $\leq n - 2$ Kameras überwacht werden.

Kamerapositionen in Triangulierungen

Beobachtungen:

- Jede Kamera auf einer Diagonalen überwacht (wenigstens) zwei Dreiecke.
- Jede Kamera in einer Ecke p des Polygons überwacht (wenigstens) alle Dreiecke mit Ecke p.

Bestimmung der Kamerapositionen bei gegebener Triangulierung T von P:

1. 3-Färbung der Ecken der Triangulierung $T = (V', E')$
   Warum für jede Triangulierung einfacher Polygone möglich ?
2. Bestimmung der am seltensten vorkommenden Farbe c
   (höchstens $\lfloor n/3 \rfloor$ Ecken haben Farbe c)
3. Kamerapositionen: Ecken mit Farbe c

Anzahl der Kameras $\leq \lfloor n/3 \rfloor$ (optimal)

Triangulierung – Schritte

- gegeben: einfaches Polygon P
- gesucht: Triangulierung T von P

Beobachtung:
Konvexe Polygone lassen sich einfach triangulieren.
Wie? Laufzeit?

Idee:
Triangulierung beliebiger einfacher Polygone P in zwei Schritten

1. Zerlegung von P in Menge P' einfach triangulierbarer Polygone
2. (einfache) Triangulierung aller Polygone in P'

Problem: Zerlegung in konvexe Polygone aufwendig

Monotone Polygone

Polygon P heißt monoton (y-monoton) gdw.
$\forall y \in \mathbb{R}$: $P \cap \{(x, y) \mid x \in \mathbb{R}\}$ (Schnitt von P mit horizontaler Linie) ist zusammenhängend (also Strecke, Punkt oder leer)

Idee: Triangulierung von P in zwei Schritten

1. Zerlegung von P in Menge P' monotone Polygone
2. (einfache) Triangulierung aller monotonen Polygone in P'

Triangulierung monotoner Polygone

Idee:
falls möglich, Diagonalen zwischen Paaren aus je einer Ecke auf dem rechten Rand und dem linken Rand einfügen

Problem bei Ecken mit Innenwinkeln $> \pi$

Sweep-Line-Algorithmus (absteigende y-Koordinate)

- Ereignisse:
  geordnete Liste aller Ecken des Polygons
- Zustand:
  Verwaltung der Ecken oberhalb der Sweep-line, die noch Eckpunkte neuer Diagonalen werden könnten, als Stack

Triangulierung monotoner Polygone

Algorithmus: Triangulierung-monoton

Eingabe: monotones Polygon $P = (V_P, E_P)$ (als doppelt verkettete Liste)
Ausgabe: Triangulierung T von P (als doppelt verkettete Liste)

$T \leftarrow P$, $[v_1, \ldots, v_n] \leftarrow V_p$ (sortierte Liste oder PQ, Ereignismenge)
$S \leftarrow \emptyset$, $S \leftarrow \text{push}(S, v_1)$, $S \leftarrow \text{push}(S, v_2)$ (Zustand, Stack)
für jedes $j \leftarrow 3, \ldots, n-1$ **:**
  wenn v_j *und* $top(S)$ *auf verschiedenen Randseiten* **dann**
    solange $|S| > 1$ **:**
      $T \leftarrow T \cup (\text{top}(S), v_j)$ (Diagonalen hinzufügen)
      $S \leftarrow \text{pop}(S)$
    $S \leftarrow \text{pop}(S)$, $S \leftarrow \text{push}(v_{j-1})$, $S \leftarrow \text{push}(v_j)$
  sonst
    $p \leftarrow \text{top}(S)$, $S \leftarrow \text{pop}(S)$
    solange $S \neq \emptyset$ *und Diagonale* $(v_j, top(S))$ *innerhalb* P **:**
      $p \leftarrow \text{top}(S)$, $S \leftarrow \text{pop}(S)$
      $T \leftarrow T \cup (p, v_j)$ (Diagonalen hinzufügen)
    $S \leftarrow \text{push}(p)$, $S \leftarrow \text{push}(v_j)$ S
$S \leftarrow \text{pop}(S)$
solange $pop(S) \neq \emptyset$ **:**
  $p \leftarrow \text{top}(S)$, $S \leftarrow \text{pop}(S)$, $T \leftarrow T \cup (p, v_n)$ (Diagonalen hinzufügen)

Zerlegung in y-monotone Polygone

Beobachtung:
In nicht y-monotonen Polygonen P
(dargestellt als Kreis (V, E)) existieren turn points,
d.h. Ecken $q \in V$, in denen sich die Richtung zu beiden Nachbarn ändert (beide nach oben oder beide nach unten)

Idee zur Zerlegung beliebiger einfacher Polygone P in y-monotone Polygone:
Zerlegung der Winkel an den turn points in V durch Hinzufügen geeigneter Diagonalen

Zerlegung in y-monotone Polygone

Klassifikation der turn points q in einem Polygon $P = (V, E)$ abhängig von

- y-Koordinaten von q und seinen Nachbar-Ecken p, r
- Winkel φ zwischen $\overline{pq}$ und $\overline{qr}$ innerhalb des Polygons

Klassen von turn points

start : p, r beide unterhalb q und $\varphi < \pi$

end : p, r beide oberhalb q und $\varphi < \pi$

split : p, r beide unterhalb q und $\varphi > \pi$

merge : p, r beide oberhalb q und $\varphi > \pi$

alle übrigen Ecken heißen regulär

Problem nur bei turn poins mit $\varphi > \pi$

Idee: Hinzufügen geeigneter Diagonalen an split- und merge-points

Sweep-line-Algorithmus

Idee: Sweep-line-Algorithmus (fallende y-Koordinate) mit

Ereignisse: geordnete Menge aller Ecken von P
(PQ oder sortierte Liste)

Zustand: nach x-Koordinate geordnete Menge von Paaren (e, h_e) mit

- Kante e aus P (schneidet die Sweep-line)
- Helfer-Punkt h_e zur Kante e:
  zuletzt gefundener nächster Punkt links von der Seite s im Inneren des Polygons,
  (potentieller Endpunkt einer Diagonalen zur Trennung eines turn points oberhalb oder unterhalb der aktuellen Sweep-line)

Zustandsänderungen:

- Ersetzen von h_e
- Einfügen von (e, h_e) am oberen Ende von e
- Entfernen von (e, h_e) am unteren Ende von e

Zerlegung in monotone Polygone

Algorithmus: Mono

Eingabe: Polygon P als doppelt verkettete Liste von Kanten

Ausgabe: Zerlegung P' von P als doppelt verkettete Liste von Kanten

$P' \leftarrow P$ (doppelt verkettete Liste von Kanten)
$E \leftarrow \emptyset$ (Ereignisse: Priority queue oder sortierte Liste)
$Z \leftarrow \emptyset$ (Zustand: binärer Suchbaum)
für jedes $e = \overline{pq} \in P$ **:**
  $E \leftarrow E \cup \{p, q\}$ mit y-Koordinate als Priorität
solange $E \neq \emptyset$ **:**
  $p \leftarrow \text{next}(E)$
  $E \leftarrow E \setminus \{p\}$
  $(E, Z) \leftarrow$ Mono-Ereignisbehandlung(p, E, Z)

Mono-Ereignisbehandlung

Mono-Ereignisbehandlung(p, E, Z)
abhängig vom Eckentyp des Ereignisses $p = v_i$

start : Kante e_i $(= (v_i, v_{i+1}))$ mit Helfer $h_{e_i} = v_i$ in Z einfügen

end : Kante e_{i-1} $(= (v_{i-1}, v_i))$ aus Z löschen
falls $h_{e_{i-1}}$ merge-Punkt: Diagonale $(v_i, h_{e_{i-1}})$ in P' einfügen

split : Kante e_j direkt rechts von v_i in Z finden
Diagonale (v_i, h_{e_j}) in P' einfügen
$h_{e_j} \leftarrow v_i$ (Helfer für Kante e_j aktualisieren)
Kante e_i mit $h_{e_i} = v_i$ in Z einfügen

merge : Kante e_{i-1} aus Z löschen
falls $h_{e_{i-1}}$ merge: Diagonale $(v_i, h_{e_{i-1}})$ in P' einfügen
Kante e_j direkt rechts von v_i in Z finden
falls h_{e_j} merge: Diagonale (v_i, h_{e_j}) in P' einfügen
$h_{e_j} \leftarrow v_i$ (Helfer für Kante e_j aktualisieren)

regulär : falls Inneres von P links von v_i:
{ falls $h_{e_{i-1}}$ merge: {Diagonale $(v_i, h_{e_{i-1}})$ in P' einfügen},
e_{i-1} aus Z löschen und e_i mit Helfer $h_{e_i} \leftarrow v_i$ in Z einfügen}
sonst (Inneres von P rechts von v_i):
{Kante e_j direkt rechts von v_i in Z finden;
falls h_{e_j} merge: { Diagonale (v_i, h_{e_j}) in P' einfügen}
$h_{e_j} \leftarrow v_i$ (Helfer für Kante e_j aktualisieren)}

Triangulierung – Laufzeit

Triangulierung beliebiger einfacher Polygone P mit n Ecken in zwei Schritten:

1. Zerlegung von P in Menge P' monotoner Polygone
   $O(n \log n)$ (für Sortieren / Verwaltung der Ereignismenge)
2. (einfache) Triangulierung jedes monotonen Polygonen $P_i \in P'$ (mit m_i Ecken)
   $O(m_i)$ je Polygon $P_i \in P'$
   (insgesamt $O(n) = \sum_{P_i \in P'} O(m_i)$
   Ordnung der Ereignisse (Ecken in P') kann aus vorigem Schritt übernommen werden (kein erneutes Sortieren notwendig)

Gesamtlaufzeit $O(n \log n)$

Was bisher geschah

- Motivation, Beispiele
- geometrische Objekte im $\mathbb{R}^2$:
  Punkt, Gerade, Halbebene, Strecke, Polygon,
  ebene Zerlegung in Regionen (planare Graphen)
- maschinelle Repräsentation geometrischer Objekte

Algorithmen zur Bestimmung

- der konvexe Hülle endlicher Punktmengen $P \subseteq \mathbb{R}^2$
- einer Triangulierung eines Polygons $P \subseteq \mathbb{R}^2$
  in zwei Schritten
  1. Zerlegung in monotone Polygone
  2. Triangulierung der monotonen Teilpolygone

Punktsuche – Motivation

- In welchem Land befindet sich ein gegebener Ort (Koordinaten)?
- Befindet sich ein Schiff in der Nähe gefährlicher Stellen?
- Interaktive Geoinformationssysteme:
  Welches Land einer elektronisch angezeigten Karte wurde angeklickt?
- Befindet sich der Fußball-Roboter in der eigenen oder gegnerischen Spielfeld-Hälfte?
- Auf welcher Fahrspur (in welchem Land, welcher Parklücke) befindet sich ein autonomes Fahrzeug?

AG-Aufgaben:
Entwurf einer Datenstruktur, welche

- effiziente Suche nach der Region in einer Zerlgung M, in der ein gegebener Punkt p liegt, ermöglicht ($O(\log n)$)
- effizient herstell-, modifizier- und speicherbar ist

Punktsuche im $\mathbb{R}^1$

gegeben: Zerlegung M im $\mathbb{R}^1$
(in Intervalle durch endliche Menge von Punkten),
Punkt $p \in \mathbb{R}^1$

gesucht: Intervall in M, in welchem p liegt

Idee:

- Speichern der Intervallgrenzen (mit Verweis auf das rechts benachbarte Intervall) in einem binären Suchbaum
- Finden der Region, welche p enthält:
  $O(\log n)$ für Anzahl n der Regionen
- Finden, Hinzufügen, Löschen von Grenzpunkten: $O(\log n)$
- initiale Konstruktion der Datenstruktur: $O(n \log n)$

Verallgemeinerung auf $\mathbb{R}^2$

gegeben:	Karte M im $\mathbb{R}^2$
	(Zerlegung in Regionen durch planaren Graphen)
	Punkt $p \in \mathbb{R}^2$ (Koordinaten)
gesucht:	Region (Fläche) in M, in welcher p liegt

einfacher Fall:
Unterteilung der Ebene durch achsenparallele Geraden

Punktsuche:

1. binäre Suche nach x-Koordinate (Streifen)
2. binäre Suche nach y-Koordinate (Rechteck, Region)

Streifen-Zerlegungen von Karten im $\mathbb{R}^2$

Beobachtungen:

- Senkrechte Geraden an den x-Koordinaten aller Ecken der Zerlegung M zerlegen M in vertikale Streifen:
- Inneres jedes Streifens enthält keine Ecken aus M: Zerlegung von M in Trapeze (und Dreiecke $=$ entartete Trapeze)
- Trapeze bilden Verfeinerung der Zerlegung M. Jedes Trapez gehört zu genau einer Fläche in M.
- Alle Trapeze innerhalb eines Streifens lassen sich in y-Richtung ordnen (Array).
- Punktsuche innerhalb eines Streifens in $O(\log n)$ mit $n =$Anzahl der Trapeze im Streifen möglich (binäre Suche)

Punktsuche in Streifen-Zerlegungen

Repräsentation der

Trapezkanten :

- (Verweis auf) Kante e
- (Verweis auf) Fläche f direkt oberhalb e

Streifen : sortiertes Array von Trapezkanten (nach y-Koordinate)

Streifen-Zerlegung : sortiertes Array von Streifen (nach x-Koordinate des Punktes auf der rechten Streifengrenze)

Idee des Algorithmus:

1. Finden des Streifens S, welches p_x enthält
   $O(\log n)$ (binäre Suche in Streifen-Array)
2. Suche nach Kante e in S, so dass p auf e oder in der Region direkt oberhalb e liegt
   $O(\log n)$ (binäre Suche im Streifen)

\+ : Laufzeit für Punktsuche $O(\log n)$

\- : Speicherplatz für Streifen-Zerlegung $O(n^2)$

Trapez-Zerlegungen

gegeben: Karte M (ebene Zerlegung, planarer Graph),
Punkt $p \in \mathbb{R}^2$ (Koordinaten)
gesucht: Fläche (Kante, Ecke) in M, in der p liegt

Idee: nur die zu einem Punkt p in M benachbarten Flächen werden an p_x vertikal unterteilt

- obere vertikale Unterteilung (der Fläche direkt oberhalb p) bis zur oberen Begrenzung der oberen Fläche
- untere vertikale Unterteilung (der Fläche direkt unterhalb p bis zur unteren Begrenzung der unteren Fläche

unschön: Sonderbehandlung der unbegrenzten Region außerhalb M

deshalb: Einfügen einer rechteckigen Begrenzung R (Polygon, bounding box), die M vollständig enthält
(repräsentiert die unbegrenzte Region außerhalb M)
Punkt außerhalb R (einfacher Koordinatentest) liegen ohnehin in der unbegrenzten Region außerhalb M

Eigenschaften von Trapez-Zerlegungen

Jede Fläche der Trapez-Zerlegung von M

- ist konvex
- hat je eine (evtl. zu einem Punkt entartete) vertikale linke und rechte Seite, die durch je eine Ecke p (Achtung: einschränkende Annahme) aus M definiert oder die Teil des Randes R ist (5 verschiedene Fälle), (leftp, rightp)
- hat je eine obere und untere (nicht-vertikale) Seite, die Teil einer Kante in M oder des Randes R ist, (top, bottom)
- ist vollständig in einer Fläche in M enthalten (Fläche direkt über der unteren Begrenzungskante), Trapez-Zerlegung von M ist also Verfeinerung von M.
- Jedes Trapez hat 1,2,3, oder 4 Nachbarn (unter der einschränkenden Annahme oben)
- Die Trapez-Zerlegung einer Karte M it n Kanten hat höchstens $6n + 4$ Ecken und $3n + 1$ Flächen

Repräsentation von Trapez-Zerlegungen

Ecke : Punkt $p \in \mathbb{R}^2$ (Koordinaten)

Kante :

- Strecke $s = (s_a, s_e) \in \left(\mathbb{R}^2\right)^2$ in M
  (Endpunkte nach x-Koordinate aufsteigend geordnet)
  und
- Verweis auf (oder Name der) Fläche in M direkt über s

Fläche :

top : Verweis auf obere Begrenzungskante aus M

bottom : Verweis auf untere Begrenzungskante aus M

leftp : Verweis auf Ecke aus M, der die linke Begrenzung definiert

rightp : Verweis auf Ecke aus M, der die rechte Begrenzung definiert

Nachbarn : Verweis auf die $1, \ldots, 4$ Nachbar-Flächen (Flächen mit gemeinsamer vertikaler Kante)

Konstruktion von Trapez-Zerlegungen

gegeben: endliche Menge S einander nicht schneidender Strecken im $\mathbb{R}^2$
Rand R

gesucht: Trapez-Zerlegung T zu S und R
Suchstruktur D mit Trapezen in T als Blättern

Idee: randomisierter Algorithmus

- Beginn: Zerlegung enthält nur Rand R,
- Strecken s aus S in zufälliger Reihenfolge einfügen
  dabei jeweils betroffene Regionen in T teilen und D aktualisieren:
  1. alle Flächen $f_1, \ldots, f_n$, die s schneidet, bestimmen
     (Punktsuche nach s_a, dann Übergang zu Nachbar-Flächen)
  2. $f_1, \ldots, f_n$ aus T und D entfernen,
  3. Jede Fläche $f_i \in \{f_1, \ldots, f_n\}$ entlang s zerlegen / zusammenfügen
  4. jeweils bei Verarbeitung von f_i neu entstandene Trapeze
     - in T einfügen
     - in D Blatt f_i durch neuen Teilgraphen ersetzen (Tafel)

Suchstruktur zur Punktsuche

gerichteter azyklischer Graph (DAG) D:
gerichteter Graph mit

- eindeutige Wurzel (keine eingehenden Kanten)
- je ein Blatt (keine ausgehenden Kanten) für jede Fläche der Trapez-Zerlegung
- innere Knoten haben genau zwei ausgehende Kanten
  zwei Typen innerer Knoten:
  - x-Knoten : markiert mit Ednpunkt einer Strecke aus M
  - y-Knoten : markiert mit Strecke aus M

Suche der Fläche für gegebenen Punkt p:

- Durchlauf von D von Wurzel zu einem Blatt
- Entscheidung an inneren Knoten: falls
  - x-Knoten (Streckenendpunkt e):
    p rechts oder links von e?
  - y-Knoten (Strecke s):
    p ober- oder unterhalb von e?

Konstruktion der Trapez-Zerlegung – Laufzeit

Konstruierte Suchstruktur D hängt von der zufälligen Einfüge-Reihenfolge der Strecken in S ab.

- Es existieren $n!$ verschiedene zufällige Einfüge-Reihenfolgen der n Strecken.
- Größe der Zerlegung T ist immer $O(n)$.
- Erwartete Größe der Suchstruktur D ist Mittel der $n!$ Größen: $O(n)$
- Erwartete Laufzeit des Algorithmus zur Konstruktion der Trapez-Zerlegung ist Mittel der $n!$ Laufzeiten: $O(n \log n)$
- Erwartete Laufzeit der Punktsuche in D ist Mittel der $n!$ Laufzeiten: $O(\log n)$

Konstruktion der Trapez-Zerlegung – Robustheit

bisherige Annahmen:

- Keine Punkte mit gleichen x-Koordinaten.
- Keine Punkte (außer den definierenden Ecken) auf den vertikalen Trapez-Kanten.
- Keine Punkte auf den Kanten aus S.

Entfernung dieser Annahmen durch
gerinfügige Koordinatentransformation
(z.B. Rotation oder Scherung)

Was bisher geschah

- Motivation, Beispiele
- maschinelle Repräsentation geometrischer Objekte im $\mathbb{R}^2$: Punkt, Gerade, Halbebene, Strecke, Polygon,
- spezielle Datenstrukturen zur (mehrfachen) effizienten Suche

Algorithmen zur

- Bestimmung der konvexe Hülle endlicher Punktmengen $P \subseteq \mathbb{R}^2$
- Bestimmung von Triangulierungen von Polygonen $P \subseteq \mathbb{R}^2$
- Punktsuche in Regionen
  Trapezzerlegung (und passende Suchstruktur)

Wegplanung im $\mathbb{R}^2$

Anwendung z.B. in

- Robotik
- Routen-Planung

gegeben:	Startposition $s \in \mathbb{R}^2$,
	Zielposition $z \in \mathbb{R}^2$,
	Karte der Umgebung (Hindernisse)
	(evtl.) Ausdehnung und Form des Objektes
gesucht:	Weg (endlicher Polygonzug) von s zu z

Anforderungen an den Weg:

- vollständig innerhalb der Fläche, die s und z enthält,
- hinreichend breit für kollisionsfreie Objekt-Bewegung

Annahme: statische Umgebung (keine bewegten Hindernisse)

Repräsentation der Konfigurationen

statisch:

Umgebung polygonale Zerlegung des $\mathbb{R}^2$,
Menge $H = \{h_1, \ldots, h_k\}$ aller Hindernisse
(Polygone $h_i \subseteq \mathbb{R}^2$)

Form des Objekts: (konvexes) Polygon $R \subseteq \mathbb{R}^2$,
enthält meist den Referenzpunkt $(0, 0) \in \mathbb{R}^2$

Start-Position $s \in \mathbb{R}^2$

Ziel-Position $z \in \mathbb{R}^2$

dynamisch:

aktuelle Position des Objekts $a \in \mathbb{R}^2$
(Position des Referenzpunktes)

Richtung des Objekts (Winkel $\varphi \in [0, 2\pi)$ gegen Uhr)

Bewegung des Objektes

geometrische Transformationen:

Verschiebung $V_a : \mathbb{R}^2 \to \mathbb{R}^2$ um einen Vektor $a = (a_x, a_y) \in \mathbb{R}^2$:

$$\forall p \in \mathbb{R}^2 : V_a(x) = (p_x + a_x, p_y + a_y)$$

Drehung $D_\varphi : \mathbb{R}^2 \to \mathbb{R}^2$ (um Ursprung $(0, 0)$)
um einen Winkel $\varphi \in [0, 2\pi)$ im mathematischen Drehsinn (entgegen Uhrzeigersinn)
$\forall p \in \mathbb{R}^2$:

$$D_\varphi(p) = (p_x \cos\varphi - p_y \sin\varphi, p_x \sin\varphi + p_y \cos\varphi)$$

Kollisionsfreie Positionen

Kollision zwischen Hindernis und Objekt unter der (dynamischen) Konfiguration $(a, \varphi) \in \mathbb{R}^2 \times [0, 2\pi)$ gdw.
$\exists h_i \in H : V_a(D_\varphi(R)) \cap h_i \neq \emptyset$

Position $(a, \varphi) \in \mathbb{R}^2 \times [0, 2\pi)$ ist kollisionsfrei gdw.

$$V_a(D_\varphi(R)) \cap \bigcup_{h_i \in H} h_i = \emptyset$$

$K \subseteq \mathbb{R}^2 \times [0, 2\pi)$: Bereich (Menge) aller kollisionsfreien Positionen

- R befindet sich immer im Bereich K, kann sich nur innerhalb K bewegen
- Wegplanung nur innerhalb K

Bewegung von Punkten

Idee:

- Repräsentation des freien Bereiches $F \subseteq \mathbb{R}^2$ in einer zur Wegsuche zwischen beliebigen Punkten aus F geeigneten Datenstruktur
  (Vorbereitung, einmal)
- Definition eines Streckennetzes innerhalb F
  (in F eingebetteter planarer Graph G_F)
  durch geeignete (Hilfs-)Punkte in F (Ecken von G_F)
- Wegsuche:
  Finden einer Verbindung zwischen zwei beliebigen Punkten $s, z \in F$ über einen Weg im Graphen G_F
  (für verschiedene s und z, häufig wiederholt)

Vorbereitung der Datenstuktur

zur Repräsentation des freien Bereiches F

- Trapez-Zerlegung der Karte $O(n \log n)$
  Menge der Hindernisse $H = \{h_1, \ldots, h_k\}$ mit insgesamt n Ecken
  (mit Bounding-Box = zusätzliches äußeres Hindernis)
- Trapeze innerhalb der Hindernisse (überflüssig) entfernen
  (einfach, Verweise von Trapezkanten auf Flächen h_i)
- Graph G_F erzeugen:

  Ecken von G_F:
  - Mittelpunkte aller senkrechten Trapez-Kanten
  - Mittelpunkte aller Trapeze

  Kanten von G_F: Strecken
  zwischen Mittelpunkt jedes Trapezes und allen Hilfspunkten auf dessen (senkrechten) Seiten

Wegplanung für Punkte

- Bestimmung des Trapezes t_s, welches den Startpunkt enthält (Punktsuche)
- Bestimmung des Trapezes t_z, welches den Zielpunkt enthält (Punktsuche)
- falls $t_s = t_z$: Weg = Strecke $\overline{s, z}$
  sonst: Konstruktion des Weges als Polygonzug aus
  - Strecke zwischen s und Mittelpunkt von t_s,
  - Pfad im Graphen G_F von Mittelpunkt von t_s zum Mittelpunkt von t_z,
    (Wegsuche im Graphen G_F, z.B. Breitensuche)
  - Strecke zwischen z und Mittelpunkt von t_z

Korrektheit der Wegplanung für Punkte

- Jeder so konstruierte Polygonzug ist ein kollisionsfreier Weg, der s und z verbindet.
  Warum?
  - Trapeze sind konvex.
  - Jede Strecke des Polygonzuges liegt innerhalb eines Trapezes.
- Wenn in F ein kollisionsfreier Weg zwischen s und z existiert, konstruiert auch der Algorithmus einen (evtl. anderen) kollisionsfreien Weg zwischen s und z.

Laufzeit zur Wegplanung für ein Paar $(s, z) \in (\mathbb{R}^2)^2$: $O(n)$

Wegplanung für nicht punktförmige Objekte

Idee:
Wegplanung für Bewegung eines Punktes in einer Umgebung mit geeignet vergrößerten (dann evtl. überlappenden und miteinander verschmelzenden) Hindernissen

Minkowski-Summe zweier Mengen $M, N \subseteq \mathbb{R}^2$:

$$M \oplus N = \{(x + x', y + y') \mid (x, y) \in M \wedge (x', y') \in N\}$$

Beobachtung:
Für beliebige konvexe Polygone M, N mit m und n Ecken gilt:
$M \oplus N$

- ist ein konvexes Polygon,
- hat höchstens $m + n$ Ecken,
- Extrema in $M \oplus N$ sind Summe der Extrema in M und N (in jede Richtung)

Berechnung der Minkowski-Summe konvexer Polygone

naive Berechnung von $P \oplus Q$ für konvexe Polygone P, Q:

1. Berechnung der paarweisen Summen $r_{i,j} = p_i + q_j$ aller Ecken $p_i \in P$, $q_j \in Q$: $R = \{p_i + q_j \mid p_i \in P \wedge q_j \in Q\}$
2. Berechnung der konvexen Hülle der Menge aller dieser $r_{i,j}$: $P \oplus Q = \text{conv}(R)$

Laufzeit $O(mn)$
bessere Idee: verschränkte Berechnung der

- Summen der Extrema in P und Q in die selbe Richtung
- konvexen Hülle der Menge dieser Summen

Laufzeit $O(m + n)$
Laufzeit der Berechnung der Minkowski-Summe zweier Polygone M, N mit m und n Ecken:

- falls M, N beide konvex $O(m + n)$
- falls M oder N konvex $O(mn)$
- sonst $O(m^2 n^2)$

Wegplanung für Verschiebung ohne Drehung

Wegplanung für nicht drehbare Objekte
(in jeder Konfiguration (a, φ) gilt $\varphi = 0$)

Drehung einer Menge $M \subseteq \mathbb{R}^2$ um π (Spiegelung am Ursprung):

$$-M = \{(-x, -y) \mid (x, y) \in M\}$$

- Vorbereitung:
  Erzeugen der Karte F' mit (vereinigten) vergrößerten Hindernissen $H' = \{h'_1, \ldots, h'_{k'}\}$ wobei h'_i genau die zusammenhängenden Gebiete von $H \oplus R = \bigcup_{h_i \in H}(h_i \oplus R)$ sind,
  und Berechnung des Graphen $G_{F'}$ zu F'
  Berechnung der Minkowski-Summe $O(n \log^2 n)$
- Wegfindung: $O(n)$
  Weg für Punktobjekt in F' ist kollisionsfreier Weg für R in F
  (Warum kollisionsfrei?)

Was bisher geschah

- Motivation, Beispiele
- maschinelle Repräsentation geometrischer Objekte im $\mathbb{R}^2$: Punkt, Gerade, Halbebene, Strecke, Polygon
- spezielle Datenstrukturen zur (mehrfachen) effizienten Suche

Algorithmen zur

- Bestimmung der konvexe Hülle endlicher Punktmengen $P \subseteq \mathbb{R}^2$
- Bestimmung von Triangulierungen von Polygonen $P \subseteq \mathbb{R}^2$
- Punktsuche in Regionen
  1. Konstruktion der Suchstruktur (einmal)
  2. Punktsuche darin (mehrmals)
- Wegplanung in Regionen (mit Hindernissen)

Motivation Voronoi-Diagramme

Post Office Problem:
Bestimmung der Einzugsgebiete (Anzahl potentieller Kunden) aller Postämter in einem Gebiet

vereinfachende Annahmen:

- Jedes Postamt bietet dieselben Leistungen (zum selben Preis).
- Kunden besuchen immer das (in Luftlinie) nächstgelegene Postamt.
- Die Bevölkerungsdichte ist überall gleich.

Modellierung der Aufgabe:

gegeben: endliche Menge von Punkten $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

gesucht: Zerlegung der Ebene $\mathbb{R}^2$ in Gebiete (Punktmengen) mit gemeinsamem nächsten Punkt $p_i \in P$

Wiederholung: Abstand zwischen Punkten

Abstandsfunktion:

Funktion $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ mit allen folgenden Eigenschaften

- $\forall x, y \in \mathbb{R}^d : d(x, y) = 0$ gdw. $x = y$
- $\forall x, y \in \mathbb{R}^d : d(x, y) = d(y, x)$ (kommutativ)
- $\forall x, y, z \in \mathbb{R}^d : d(x, y) + d(y, z) \geq d(x, z)$ (Dreiecksungleichung)

prominente Abstandsfunktionen:

für alle $x = (x_1, \ldots, x_d) \in \mathbb{R}^d, y = (y_1, \ldots, y_d) \in \mathbb{R}^d$:

$$d_k(x, y) = \sqrt[k]{\sum_{i=1}^{d} (x_i - y_i)^k}$$

- $d_2(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$ (euklidisch)
- $d_1(x, y) = \sum_{i=1}^{d} |x_i - y_i|$ (Manhattan)
- $d_\infty(x, y) = \max\{|x_i - y_i| \mid i \in \{1, \ldots, d\}\}$ (Maximum)

Vorberachtungen

gegeben: endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

gesucht: Zerlegung $Q = \{Q_1, \ldots, Q_n\}$ des $\mathbb{R}^2$, so dass für jeden Punkt $p_i \in P$ und die Menge $Q_i \subseteq \mathbb{R}^2$ gilt $\forall q \in Q_i : d(q, p_i) \leq \min\{d(q, p) \mid p \in P\}$

Beobachtungen:

- zwei verschiedene Punkte: $P = \{p_1, p_2\} \subseteq \mathbb{R}^2$: $p_1 \in P$ ist nächster Punkt zu $q \in \mathbb{R}^2$ gdw. $d(q, p_1) < d(q, p_2)$
- zwei verschiedene Punkte $p_1, p_2 \in \mathbb{R}^2$ teilen $\mathbb{R}^2$ in zwei Halbebenen

$$\begin{aligned} h(p_1, p_2) &= \{q \in \mathbb{R}^2 \mid d(q, p_1) < d(q, p_2)\} \\ h(p_2, p_1) &= \{q \in \mathbb{R}^2 \mid d(q, p_2) < d(q, p_1)\} \end{aligned}$$

Trenngerade $g(p_1, p_2) = \{q \in \mathbb{R}^2 \mid d(q, p_1) = d(q, p_2)\}$
(Mittelsenkrechte zur Strecke $\overline{p_1 p_2}$)

Voronoi-Zellen

Menge aller zu einem Punkt $p_i \in P$ nächsten Punkte

$$\text{vor}(P, p_i) = \bigcap_{p_j \in P \setminus \{p_i\}} h(p_i, p_j)$$

Voronoi-Zelle des Punktes $p_i \in P$

Voronoi-Zellen für endliche Punktmengen sind immer Schnitt endlich vieler Halbebenen,
also konvexes Polygon (evtl. unbeschränkt)

Voronoi-Diagramme

gegeben: endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

gesucht: Karte M im $\mathbb{R}^2$ (zusammenhängende Menge von Strecken, planarer Graph)

Voronoi-Zelle von $p_i \in P$ in P:

$$\begin{aligned} \mathrm{vor}(P, p_i) &= \left\{ q \in \mathbb{R}^2 \mid d(q, p_i) \leq \min\{d(q, p) \mid p \in P\} \right\} \\ &= \bigcap_{p_j \in P \setminus \{p_i\}} h(p_i, p_j) \end{aligned}$$

Voronoi-Diagramm $\mathrm{vor}(P)$ von P: Menge aller Voronoi-Zellen, Zerlegung des $\mathbb{R}^2$, repräsentiert durch planaren Graphen mit

- Voronoi-Kanten:

$$\left\{ q \in \mathbb{R}^2 \mid \exists p_i, p_j \in P : \begin{array}{l} d(q, p_i) = d(q, p_j) \wedge \\ \forall p_k \in P \setminus \{p_i, p_j\} : d(q, p_i) < d(q, p_k) \end{array} \right\}$$

- Voronoi-Ecken:

$$\left\{ q \in \mathbb{R}^2 \mid \exists p_i, p_j, p_k \in P : \begin{array}{l} d(q, p_i) = d(q, p_j) = d(q, p_k) \wedge \\ \forall p_k \in P \setminus \{p_i, p_j, p_k\} : d(q, p_i) < d(q, p_k) \end{array} \right\}$$

Voronoi-Diagramme – Beispiele

- für zwei Punkte $P = \{p_1, p_2\}$:
  ist vor(P) eine Gerade (Mittelsenkrechte auf $\overline{p_1 p_2}$)
- alle $p_i \in P$ kollinear: vor(P) Menge von Geraden
- drei nicht-kollineare Punkte $P = \{p_1, p_2, p_3\}$: vor(P) enthält drei unbeschränkte Polygone
  (begrenzt durch je zwei Halbgeraden)
- alle $p_i \in P$ auf einem Kreis: vor(P) Menge von Halbgeraden vom Mittelpunkt des Kreises
  (Winkelhalbierende zwischen benachbarten Punkten)
- vier nicht-kollineare Punkte $P = \{p_1, p_2, p_3, p_4\}$

Anzahlen in Voronoi-Diagrammen

gegeben: endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$

gesucht: Voronoi-Diagramm vor(P) (als planarer Graph (V, E))

Voronoi-Zelle vor(P, p_i): Anzahl der

Ecken $\leq n - 1$

Kanten $\leq n - 1$

Voronoi-Diagramm vor(P): Anzahl der

Ecken $\leq \sum_{p_i \in P}(n-1)$ $(\in O(n^2))$

Kanten $\leq \sum_{p_i \in P}(n-1)$ $(\in O(n^2))$

Flächen $= n$ (eine für jeden Punkt $p_i \in P$)

bessere Abschätzungen:

Satz

Für jede endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ mit $n \geq 3$ existieren in vor(P)

- *höchstens $2n - 5$ Kanten und*
- *höchstens $3n - 6$ Ecken.*

Berechnung des Voronoi-Diagrammes

gegeben: endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$
gesucht: Voronoi-Diagramm $\text{vor}(P) = (V, E)$

naiver Algorithmus:
Für jeden Punkt $p_i \in P$ Voronoi-Zelle $\text{vor}(P, p_i)$ berechnen:
Schnitt aller Halbebenen $h(p_i, p_j)$ für $j \neq i$

(Laufzeit je Punkt $O(n \log n)$)

Laufzeit zur Berechnung des Voronoi-Diagrammes: $O(n^2 \log n)$

Das geht aber besser.

Leere Kreise

gegeben: Voronoi-Diagramm vor(P) (als planarer Graph (V, E))
Ecke $q \in V$

größter leerer Kreis um $q \in \mathbb{R}^2$ in $P = \{p_1, \ldots, p_n\}$

$$C(P, q) = \{s \in \mathbb{R}^2 \mid d(q, s) = \min\{d(q, p_i) \mid p_i \in P\}\}$$

Satz

Für jede endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ *gilt im Voronoi-Diagramm* vor$(P) = (V, E)$*:*

- $\forall q \in \mathbb{R}^2$:
  $q \in V$ *(Voronoi-Ecke)* gdw. $|C(P, q) \cap P| \geq 3$
- $\forall (p_i, p_j) \in P^2$:
  Mittelsenkrechte $M(p_i, p_j)$ *enthält Voronoi-Kante* gdw.
  $\exists q \in M(p_i, p_j) : C(P, q) \cap P = \{p_i, p_j\}$

Sweep-line-Algorithmus

gegeben: endliche Menge $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$
gesucht: Voronoi-Diagramm $\text{vor}(P) = (V, E)$

Idee:

- Ereignisse:
  alle Punkte aus P geordnet nach fallender y-Koordinate
- Zustand:
  alle Schnittpunkte des Voronoi-Diagramms (Kanten) mit Sweep-line

Problem:

- Grenzen zwischen Voronoi-Zellen hängen auch von Punkten p_i noch unterhalb der Sweep-line ab
- Es gibt also Punkte oberhalb der Sweep-line, deren Zuordnung zu Voronoi-Zellen noch nicht bekannt ist.

Partielles Voronoi-Diagramm

Welcher Teil des Voronoi-Diagrammes ist bekannt (kann sich durch Punkte unterhalb der Sweep-line nicht mehr ändern), nachdem die Sweep-line einen Punkt $p_i \in P$ erreicht hat?

- Sweep-line bei p_i: $l = \{q \in \mathbb{R}^2 \mid q_y = p_{iy}\}$
- Abstand eines Punktes $q \in \mathbb{R}^2$ oberhalb der Sweep-line zur Sweep-line l: $d(q, l) = q_y - p_{iy}$
- Für Punkte $q \in \mathbb{R}$ oberhalb der Sweep-line mit $\exists p_j \in P : d(q, p_j) < d(q, l) = q_y - p_{iy}$ ändert sich die Zuordnung nicht mehr.
- Grenze zwischen $\{q \in \mathbb{R}^2 \mid d(q, p_j) < q_y - p_{iy}\}$ (wenn Sweep-line bei p_i) ist eine (nach oben geöffnete) Parabel, Funktion

$$f(x) = \frac{(x - p_{jx})^2}{2(p_{jy} - p_{iy})} + \frac{(p_{jy} + p_{iy})}{2}$$

Küste

- Grenze zwischen $\{q \in \mathbb{R}^2 \mid \min\{d(q, p_j) \mid p_j \in P\} < q_y - p_{iy}\}$ ist das Minimum einer endlichen Menge von (nach oben geöffneten) Parabeln (Küste)

$$f(x) = \min \left\{ \frac{(x - p_{jx})^2}{2(p_{jy} - p_{iy})} + \frac{(p_{jy} + p_{iy})}{2} \mid p_j \in P \right\}$$

- Schnittpunkte der Parabeln liegen immer auf Voronoi-Kanten (Warum ?)
- Punkt $p_i \in P$ auf der Sweepline gdw. neue (entartete) Parabel entsteht
- Küste kann mehrere Teile derselben Parabel enthalten (z.B. nach Entstehung einer neuen Parabel)
- Parabeln verschwinden (Nachbar-Parabeln verschmelzen) genau in Schnittpunkten von Voronoi-Kanten, also in Voronoi-Ecken

Idee: Zustand = Repräsentation der Küste

Ereignisse

Entstehen neuer Parabeln
Punkt-Ereignis $p_i \in P$

Verschwinden von Parabeln (Entdeckung von Voronoi-Ecken)
Kreis-Ereignis $v \in V$

Problem:
Voronoi-Ecken befinden sich bei Ihrer Entdeckung auf der Küste, also oberhalb der Sweep-line

Beobachtung:
Bei der Entdeckung einer Voronoi-Ecke r ist die Sweep-Line Tangente an den größten leeren Kreis $C_P(r)$ für die Voronoi-Ecke r mit $|C_P(r) \cap P| \geq 3$

Sweep-line-Algorithmus – Idee

Ereignisse : geordnet nach fallender y-Koordinate

- alle Punkte aus P (zu Beginn bekannt)
- minimale Positionen der leeren Kreise von je 3 Punkten aus P mit benachbarten Parabelteilen

Zustand : Küste als (geordnete) Menge von Parabel-Teilen

Repräsentation des Voronoi-Diagrams

Zerlegung des $\mathbb{R}^2$:
doppelt verkettete Kantenliste mit

Halbkanten : Anfangspunkt, next, prev

Ecken : Koordinaten und
geordnete Liste der inzidenten Halbkanten

Flächen : Verweis auf eine (Innen-)Kante

Problem: unbeschränkte Flächen

Lösung: bounding box um P

Datenstruktur für Ereignismenge

Ereignismenge Q:
priority queue mit y-Koordinate als Priorität
von zwei Typen von Ereignissen:

Punkt : Punkt $p_i \in P$

Kreis :

- Berührungspunkt der Sweep-line mit dem Kreis durch drei Punkte $p_i, p_j, p_k \in P$ oberhalb der Sweep-line (mit benachbarten Parabelteilen)
- Verweis auf verschwindende Parabel (Blatt in der Zustands-Datenstruktur)

Datenstruktur für Zustand

Zustand:

balancierter Suchbaum

- Blätter: (aktive) Parabelteile um Punkte aus P oberhalb der Sweep-line
  - $p_i \in P$ definierender Punkt der Parabel
  - Verweis auf (Kreis-)Ereignis in Q, an welchem die Parabel verschwindet (zunächst $\perp$)
- innere Knoten: Schnittpunkte der benachbarten Parabelteile
  - Punkte $p_i, p_j \in P^2$, welche benachbarte Parabelteile definieren
  - Verweis auf die Kante der Zerlegung, die den Schnittpunkt der Parabelteile enthält

Finden, Einfügen, Löschen in $O(\log n)$

Kreis-Ereignisse

Ausschnitt der Küste bei Sweep-line l bei y:
drei benachbarte Parabelteile a_{i-1}, a_i, a_{i+1} zu den Punkten p_i, p_j, p_k

Parabelteil a_i zu p_j verschwindet aus der Küste gdw.
$\exists q \in \mathbb{R}^2 : d(q, p_i) = d(q, p_j) = d(q, p_k)$

wenn Parabelteil a_i zu p_j verschwindet, gilt

- $|C(P, q) \cap P| \geq 3$
- $d(q, p_j) = d(q, l) = q_y - y$

Berechnung der Kreis-Ereignisse

Kreis-Ereignis:

- Position (y-Koordinate) der Sweep-line beim Zusammentreffen zweier Voronoi-Kanten in einem Punkt q oberhalb der Sweep-line
- Position der Sweep-line = kleinste y-Koordinate im leeren Kreis $C(P, q)$
- drei Punkte p_i, p_j, p_k auf dem Kreis $C(P, q)$ gegeben
  Kreis $C(P, q)$ eindeutig bestimmt
  (Radius r, Mittelpunkt (q_x, q_y) durch Gleichungssystem $(p_x - q_x)^2 + (p_y - q_y)^2 = r^2$ bestimmen)
  kleinste y-Koordinate im Kreis $C(P, q)$ eindeutig bestimmt $(q_y - r)$

Kreis-Ereignisse

- Kreis-Ereignisse können höchstens an vorher benachbarten Tripeln von Parabelteilen auftreten
- Berechnung potentieller Kreisereignisse, wenn neue benachbarte Tripel von Parabelteilen entstehen
  Wann geschieht das?
  - Punkt-Ereignis zerlegt einen Parabelteil a_i in zwei Teile a_i' und a_i'' und fügt neues Parabelteil b dazwischen ein,
    ≤ 2 neue Tripel $((a_{i-1}, a_i', b)$ und $(b, a_i'', a_{i+1}))$ zu paarweise verschiedenen Punkten aus P
    ≤ 2 potentielle Kreis-Ereignisse
  - Kreis-Ereignis: Parabelteil a_i verschwindet
    Nachbarn a_{i-1} und a_{i+1} des verschwundenen Parabelteils werden zueinander benachbart
    ≤ 2 neue Tripel $((a_{i-2}, a_{i-1}, a_{i+1})$ und $(a_{i-1}, a_{i+1}, a_{i+2}))$
    ≤ 2 potentielle Kreis-Ereignisse
- nicht jedes benachbarte Tripel von Parabelteilen führt zu einem Kreis-Ereignis (falscher Alarm, wird später aus der Ereignis-Queue gelöscht)

Punkt-Ereignis-Behandlung

Punkt-Ereignis: p_i

Aufgaben:

- Parabelteil a oberhalb p_i in Küste (Zustand) finden:
  - an inneren Knoten (p_l, p_{l+1}) Berechnung der aktuellen Parabel-Schnittpunkte notwendig
- Kreis-Ereignis, auf das a verweist, löschen (falscher Alarm)
- a in zwei Parabelteile a_0, a_2 teilen (zwei neue Blätter)
  neue (entartete) Parabel a_1 dazwischen (als Blatt) in Z einfügen
  neue innere Knoten mit Parabelpunkten einfügen,
- neue potentielle Kreis-Ereignisse berechnen und
  - in Ereignis-Queue einfügen,
  - mit den Parabelteilen verknüpfen

Kreis-Ereignis-Behandlung

Kreis-Ereignis:

- y-Koordinate
- Verweis auf Parabelteil (Blatt in Z), der durch das Ereignis verschwindet
- Mittelpunkt q

Aufgaben:

- verschwindenden Parabelteil a aus Z löschen
  (dabei innere Knoten aktualisieren)
- alle Kreis-Ereignisse, an denen a beteiligt ist, aus Q, löschen
  (über Verweise von Nachbarn von a in Z)
- Kreis-Ereignisse für neu entstandene Tripel benachbarter Parabelteile bestimmen und in Q einfügen
- Voronoi-Knoten q in Voronoi-Diagramm einfügen
- Kante zwischen $V(P, p_i)$ und $V(P, p_k)$ mit Ecke q im Voronoi-Diagramm anlegen (und Verweise erstellen)

Algorithmus

(Fortune's Sweep)

Algorithmus: Voronoi

Eingabe: $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{R}^2$

Ausgabe: Voronoi-Diagramm D (Zerlegung, doppelt verkettete Kantenliste)

$D \leftarrow \emptyset$ (Voronoi-Diagramm)
$Q \leftarrow \emptyset$ (Ereignisse)
$Z \leftarrow \emptyset$ (Zustand)
für jedes $i \leftarrow 1, \ldots, n$ **:**
  $Q \leftarrow Q \cup \{p_i\}$
solange $Q \neq \emptyset$ **:**
  $e \leftarrow \text{getmin}(Q)$, $Q \leftarrow \text{deletemin}(Q)$
  wenn e *Punkt-Ereignis* **dann**
    Punkt-Ereignisbehandlung(e, Z, Q, D)
  sonst
    Kreis-Ereignisbehandlung(e, Z, Q, D)
Nachbearbeitung(D)

Nachbearbeitung

Voronoi-Diagramm nach Ende der Ereignisbehandlung noch nicht vollständig:

Reststruktur Z nach Behandlung aller Ereignisse:
innere Knoten repräsentieren Voronoi-Kanten mit nur einer Voronoi-Ecke (Halbgeraden)

Idee: bounding box R um P

- Berechnung der Schnittpunkte der Halbgeraden mit R als Endpunkte
- D aktualisieren
  (Verweise zwischen Ecken, Kanten, Flächen)

Laufzeit

- Anzahl der Ereignisse
  - Punkt: $|P| = n$
  - Kreis: $\leq 2n - 5$

  insgesamt $O(n)$
- Laufzeit für Ereignis-Bearbeitung:
  - Punkt: $O(\log n)$
  - Kreis: $O(\log n)$

Gesamt-Laufzeit: $O(n \log n)$

Speicherbedarf: $O(n)$
(Eingabe, Ereignis-Queue, Voronoi-Diagramm)

Robustheit

- Punkte mit gleicher y-Koordinate:
  Kippen der Sweep-line
- Punkt- und Kreisereignis mit gleicher y-Koordinate:
  Kreis- vor Punktereignis bearbeiten
- Kreis-Ereignisse für < 3 Punkte aus P für je drei Punkte unabhängig voneinander verwalten,
  führen zu Kanten der Länge 0,
  Voronoi-Diagramm nachträglich bereinigen (um überflüssige Voronoi-Knoten und -Kanten)
- Punkt-Ereignis auf Parabel-Schnittpunkt:
  beliebigen der Parabelteile wählen

Delone-Triangulierung

(auch Delaunay-Triangulierung)

Delone-Graph einer endlichen Punktmenge $P \in \mathbb{R}^2$:
dualer Graph zum Voronoi-Diagramm von P

Delone-Zerlegung: Einbettung des dualen Graphen in R^2 mit

- Menge P von Ecken,
- Strecken zwischen benachbarten Voronoi-Zellen als Kanten

- Delone-Zerlegung ist planar (Kanten schneiden einander nicht)
- kann Flächen enthalten, die keine Dreiecke sind

Delone-Triangulierung:
Delone-Zerlegung mit zusätzlicher (geeigneter) Triangulierung der (konvexen) Flächen, die in der Delone-Zerlegung keine Dreiecke sind

Anwendung z.B. zur Interpolation von Höhenprofilen aus Messwerten

Bestandteile der Lehrveranstaltung:

- Vorlesung
- Übungsaufgaben
- Robotik-Praktikum

Vorlesung – Themen

- maschinelle Darstellungen von Polygonen im $\mathbb{R}^2$
- konvexe Hülle endlicher Punktmengen
- Berechnung der konvexen Hülle endlicher Punktmengen im $\mathbb{R}^2$ (sweep-line-Algorithmus)
- Wächterproblem
- Zerlegung in monotone Polygone (sweep-line-Algorithmus)
- Triangulierung monotoner Polygone (sweep-line-Algorithmus)
- Trapez-Zerlegungen (randomisierter Algorithmus)
- Punktsuche in Trapez-Zerlegungen
- Minkowski-Summe
- Pfadplanung
- Voronoi-Diagramme im $\mathbb{R}^2$ (sweep-line-Algorithmus)
- Delone-Triangulierungen

Übungsaufgaben

- konvexe Mengen
- konvexe Hülle im $\mathbb{R}^2$
- monotone Polygone
- Wächter-Problem
- Triangulierungen
- Trapez-Zerlegungen
- Minkowski-Summe
- Pfadplanung
- Voronoi-Diagramme, Delone-Zerlegungen

Prüfung

Klausur (120 min):
19. Juli 2016, 8:00 - 10:00 Uhr in G126

Aufgabentypen ähnlich Übungsaufgaben

zulässige Hilfsmittel:

- alles aus Papier,
- nichts elektronisches (TR, Telefon,Smartwatch,. . .)

Alle Studenten, die ihr Robotik-Projekt abgeschlossen und am 23.6.2016 vorgeführt haben, sind zur Prüfung zugelassen.