1 Lernziele

Allgemein Es soll eine Problemstellung aus dem Bereich des parallelen Rechnens eigenständig in einer Gruppe bearbeitet werden. Dazu ist eine Einarbeitung in das Themengebiet und das Erstellen eines Projektplans notwendig. Die entwickelten Lösungen sollen implementiert und in einem konkreten Anwendungsfall analysiert werden. Dabei ist das Problem mit Fachwissen unter Anwendung geeigneter Methoden und Verfahren entsprechend des aktuellen Stands der Technik zu lösen, das Projekt zu planen, durchzuführen, abzuschließen und zu dokumentieren.

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden.

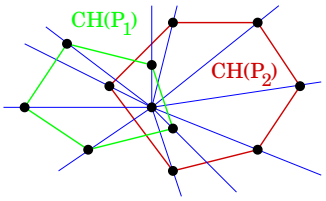
Speziell Ziel des Projekts ist das Kennenlernen sowie die Implementierung und Evaluation paralleler Algorithmen zur Berechnung der konvexen Hülle für eine Menge von 2D-Punkten. Es soll die Performance gegenüber sequentiellen Algorithmen verglichen und die Skalierbarkeit auf Multicore- oder verteilten Systemen analysiert werden.

Die konvexe Hülle einer Punktmenge ist die kleinste konvexe Menge, die alle Punkte enthält. Klassische Algorithmen sind Graham-Scan[1] mit Laufzeit $\mathcal{O}(n \log n)$, Jarvis-March[2] mit Laufzeit $\mathcal{O}(nh)$, wobei h die Anzahl der Punkte auf der Hülle ist, sowie ein Divide-and-Conquer-Ansatz mit Laufzeit $\mathcal{O}(n \log n)$. Für eine parallele Umsetzung bietet sich insbesondere der Divide-and-Conquer-Ansatz an, da sich die Punktmenge ganz natürlich in Teilmengen unterteilen lässt.

Geometrisches Divide-and-Conquer Zunächst wird die Punktmenge $P = \{p_1, \ldots, p_n\}$ aufgeteilt in zwei etwa gleich große Mengen $P_1 = \{p_1, \ldots, p_{\lceil n/2 \rceil}\}$ und $P_2 = \{p_{\lceil n/2 \rceil + 1}, \ldots, p_n\}$ (Divide). Dann werden die konvexen Hüllen $CH(P_1)$ und $CH(P_2)$ rekursiv berechnet (Conquer). Schließlich müssen die beiden konvexen Hüllen zu einer einzigen verschmolzen werden (Merge). Dazu bestimmen wir einen Punkt p innerhalb der konvexen Hülle $CH(P_1)$.

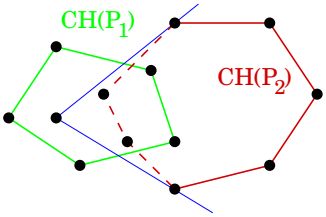
- falls p innerhalb von $CH(P_2)$ liegt:

  Mische $CH(P_1)$ und $CH(P_2)$ zu einer Sortierung nach Winkeln bzgl. eines Segments mit Endpunkt p.



- sonst liegt p außerhalb von $CH(P_2)$:

  Bestimme den Keil von p und q_1, q_2 aus $CH(P_2)$, sodass alle Punkte aus P_2 in diesem Keil liegen. Mische $CH(P_1)$ und die äußere Kontur von $CH(P_2)$.



[1] https://de.wikipedia.org/wiki/Graham_Scan
[2] https://de.wikipedia.org/wiki/Gift-Wrapping-Algorithmus

Auf die gemischte Liste wird schließlich Graham-Scan angewendet, um die finale konvexe Hülle zu berechnen.

2 Projekt

Zunächst ist ein sequentieller Algorithmus wie Graham-Scan zu implementieren, der als Grundlage für die Analyse der Effizienz und des Speedups dient. Dann sind mindestens zwei verschiedene parallele Algorithmen zu implementieren.

2.1 Chan's Algorithmus

Chan's Algorithmus[3] ist ein sogenannter ausgabesensitiver Algorithmus[4] zur Berechnung der konvexen Hülle einer Punktmenge in der Ebene oder im Raum. Er kombiniert bekannte Methoden wie den Graham-Scan und den Gift-Wrapping-Algorithmus (Jarvis March) in einem Divide-and-Conquer-Ansatz, um eine asymptotisch optimale Laufzeit von $\mathcal{O}(n \log k)$ zu erreichen, wobei n die Anzahl der Eingabepunkte und k die Anzahl der Punkte auf der konvexen Hülle ist.

Die Menge von n Punkten wird in etwa $\lceil n/m \rceil$ Teilmengen mit jeweils höchstens m Punkten aufgeteilt. Für jede dieser Teilmengen wird die konvexe Hülle mit dem Graham-Scan-Algorithmus berechnet.

Diese Teilhüllen werden dann zu einer globalen Lösung zusammengefügt, indem der Gift-Wrapping-Algorithmus auf die Teilhüllen angewendet wird. Dabei wird für jeden Punkt der aktuellen konvexen Hülle mithilfe binärer Suche der nächste mögliche Punkt aus jeder Teilhülle bestimmt, der den maximalen Winkel zum vorherigen Punkt aufweist. Der nächste Punkt auf der globalen konvexen Hülle ist derjenige aus allen Kandidaten mit dem größten Winkel.

Dieser Prozess wird bis zur vollständigen Umrundung der Hülle wiederholt. Da die Anzahl k der Punkte auf der Hülle nicht im Voraus bekannt ist, wird zunächst ein Wert für m gewählt und der Algorithmus wiederholt mit einem größeren Wert ausgeführt, bis die Hülle vollständig gefunden ist.

2.2 Quickhull-Algorithmus

Zunächst werden zwei Extrempunkte der Menge gefunden, zum Beispiel die Punkte mit der minimalen und maximalen x-Koordinate, siehe Abbildung 1b. Diese beiden Punkte sind garantiert Teil der konvexen Hülle.

Eine Gerade wird zwischen diesen beiden Punkten aufgespannt, die die Punktmenge in zwei Teilmengen aufteilt: Punkte rechts bzw. links dieser Gerade, siehe Abbildung 1c.

Für jede dieser Teilmengen wird rekursiv nach dem Punkt gesucht, der den größten Abstand zur Gerade hat, siehe Abbildung 1d. Dieser Punkt gehört ebenfalls zur konvexen Hülle und teilt die Teilmenge in kleinere Mengen auf.

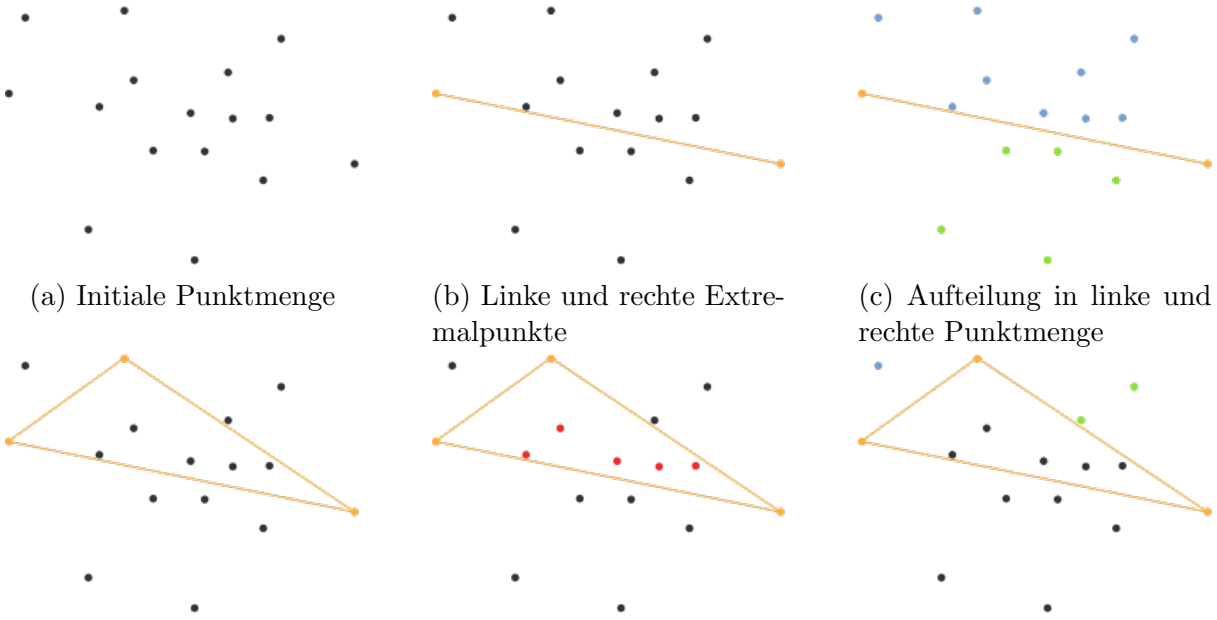
Der Vorgang wird für die jeweils neu gebildeten Teilmengen wiederholt, indem die Punkte außerhalb des Dreiecks, das durch die aktuelle Gerade und den gefundenen Punkt gebildet

[3] `https://de.wikipedia.org/wiki/Chans_Algorithmus`

[4] Ein ausgabesensitiver Algorithmus ist ein Algorithmus, dessen Laufzeit von der Größe der Ausgabe abhängt, anstatt oder zusätzlich von der Größe der Eingabe. `https://de.wikipedia.org/wiki/Ausgabesensitiver_Algorithmus`

wird, betrachtet werden. Punkte innerhalb dieses Dreiecks können nicht zur konvexen Hülle gehören und werden daher nicht weiter betrachtet, siehe Abbildung 1e.

Die Rekursion stoppt, wenn keine Punkte mehr außerhalb der aktuellen Geraden liegen oder die Punktmenge sehr klein ist (typischerweise drei oder weniger Punkte). Durch diese rekursive Teilung entsteht letztlich der geschlossene Polygonzug, der die konvexe Hülle bildet.



(a) Initiale Punktmenge

(b) Linke und rechte Extremalpunkte

(c) Aufteilung in linke und rechte Punktmenge

(d) Punkt mit maximaler Distanz von der Geraden

(e) Alle Punkte innerhalb des Dreiecks werden ignoriert

(f) Erneute Aufteilung und rekursiver Aufruf

Abbildung 1: Prinzipieller Ablauf des Algorithmus Quickhull. Quelle `https://de.wikipedia.org/wiki/QuickHull`

```
QuickHull(S):
  Finde Extrempunkte A und B (min und max x)
  Füge A und B zur konvexen Hülle hinzu
  Teile S in S1 (rechts von AB) und S2 (links von AB)
  FindHull(S1, A, B)
  FindHull(S2, B, A)

FindHull(Sk, P, Q):
  Finde Punkt C mit größtem Abstand zur Gerade PQ
  Füge C zur konvexen Hülle zwischen P und Q ein
  Teile Sk in S1 (rechts von PC) und S2 (rechts von CQ)
  FindHull(S1, P, C)
  FindHull(S2, C, Q)
```

2.3 Bewertung der Algorithmen

Um die parallelen Algorithmen bewerten zu können, muss eine Testumgebung erstellt werden, mit der gezielt verschiedene Einflussgrößen auf die Laufzeit und Effizienz analysiert werden können. Vor allem müssen Tests mit verschieden großen Punktmengen (z.B. 10^5 bis 10^8 Punkte) durchgeführt werden.

Die wesentliche Metrik ist natürlich die Laufzeit, da wir parallele Algorithmen vor allem dann einsetzen, wenn die sequentiellen Algorithmen zu langsam sind. Aber wir wollen auch andere Metriken betrachten:

- **Speedup und Skalierung** Der Speedup $S = {}^{T_{\text{seriell}}}/_{T_{\text{parallel}}}$ zeigt, wie effektiv die Parallelisierung ist und zeigt das Skalierungsverhalten bei wachsender Thread-/Prozessanzahl.
- **Ressourcenauslastung (CPU, RAM)** Messen Sie die Auslastung von Prozessor und Hauptspeicher während der Tests.

Weiterführende Literatur und Quellen

- Kirkpatrick, D.G., Seidel, R.: „The Ultimate Planar Convex Hull Algorithm?“, SIAM Journal on Computing 15(1), 1986.[5]
- Preparata, F.P., Hong, S.J.: "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", CACM, 1977.[6]
- Tzeng, S., Owens, J.D.: „Finding Convex Hulls Using Quickhull on the GPU“, UC Davis.[7]
- Die CGAL (Computational Geometry Algorithms Library) bietet effiziente Implementierungen für Vergleiche.[8]

Dieses Projekt sollte von drei bis fünf Studierenden bearbeitet werden.

[5] `https://www.ibr.cs.tu-bs.de/courses/ws2223/ag/papers/Ch2/KirkSeidel.pdf`
[6] `https://dl.acm.org/doi/pdf/10.1145/359423.359430`
[7] `https://arxiv.org/pdf/1201.2936`
[8] `https://www.cgal.org/`