6.2.3 Algorithmus II: Divide & Conquer

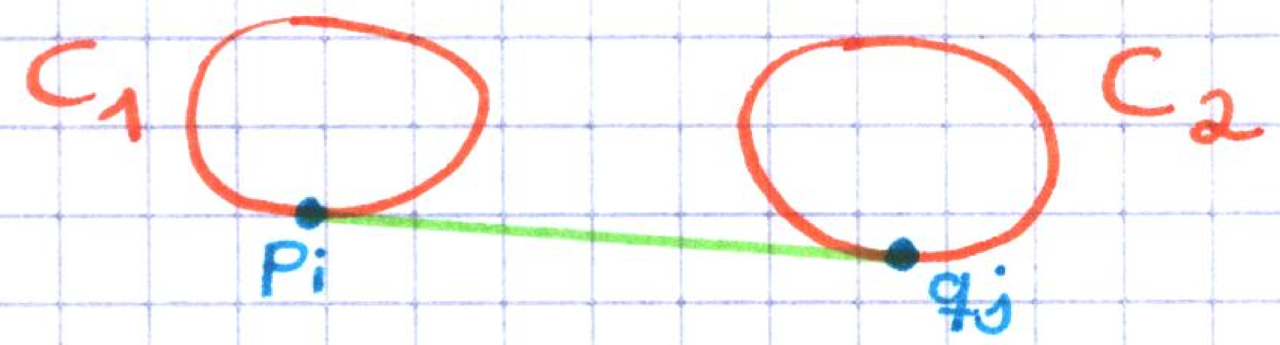
- **Algorithmus:**

```
CONVEX_HULL(S)
  if |S| == 1 then
      output S
  else
     · Teile S in 2 disjunkte Hälften gem xyz-lexikogr.     } Divide
       Sortierung ~> S1 und S2
     · C1 <- CONVEX_HULL(S1)                                 } Conquer
       C2 <- CONVEX_HULL(S2)                                   (rekursive Aufrufe)
     · Berechne CH(C1 ∪ C2)                                  } Mischschritt
  fi
```

$S_1 = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ $S_2 = \{p_{\lfloor n/2 \rfloor + 1}, \dots, p_n\}$

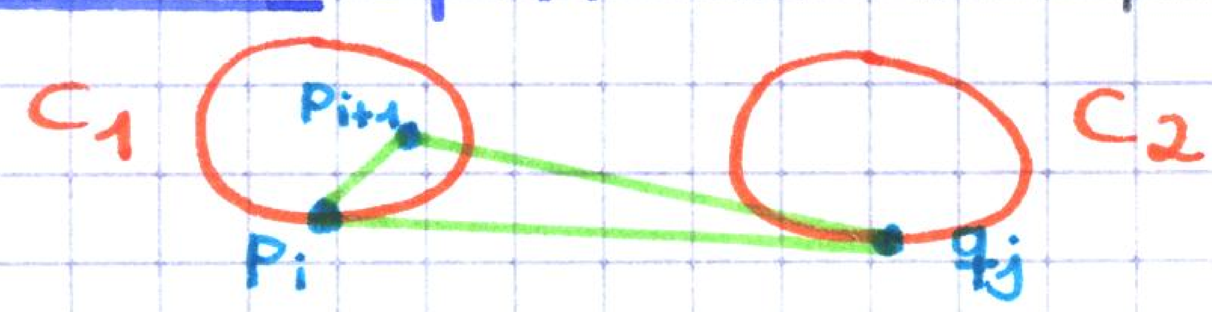
- **Mischschritt:**
  - Konstruktion des "Tangentialzylinders" durch Einwickeln mit einer Ebene.
  - Einwickeln der ganzen Pktmenge (siehe 2-dim "Gift-wrapping") kostet im schlechtesten Fall $O(n^2)$.
  - Wir wollen den Tangentialzylinder aber in Zeit $O(n)$ konstruieren. Dabei nutzen wir aus, dass wir 2 konvexe Polyeder C_1 und C_2 einwickeln.
  - Jeder Schritt verläuft analog zum 2-dim Fall, mit folgendem Unterschied:
    - 2 Pkte a und b, dh Kante $\overline{ab}$ ist fest. Man sucht Pkt c mit minimalem Winkel
    - Verwende 3-dim orientation Prädikat

- **Konstruktion des "Tangentialzylinders":**
  Sei $p_i \in C_1$, $q_j \in C_2$, sodass Kante $p_i q_j$ den aktuellen Rand des Einwickelzylinders darstellt.



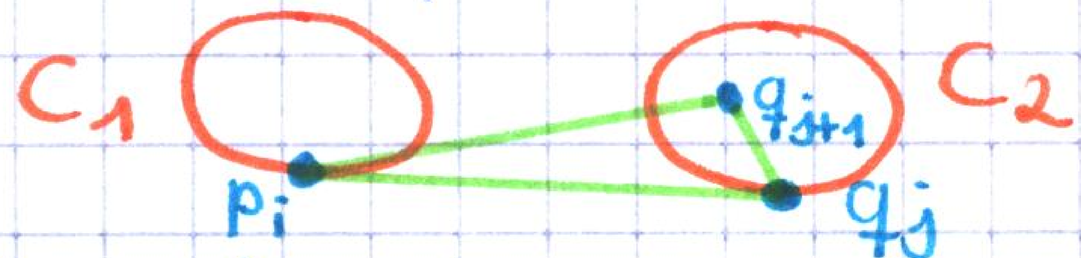
Initialisierung:
$\overline{p_1 q_1}$ durch Berechnung der Tangente in der Projektion (2-dim Problem)

  - **1. Fall:** p_{i+1} = nächster Berührpkt der Einwickelebene liegt in C_1:



$\Rightarrow$ Füge Dreieck (p_i, q_j, p_{i+1}) zum Zylinder hinzu. Die neue Drehachse der Ebene geht jetzt durch $\overline{p_{i+1} q_j}$

  - **2. Fall:** q_{j+1} = nächster Berührpkt der Einwickelebene liegt in C_2:



$\Rightarrow$ analog ...

Auf diese Weise entstehen auf jeder Seite 2 Berührkreise $p_1, \dots, p_k$ und $q_1, \dots, q_k$. (~> siehe nächster Pkt auf nächster Seite)
Während der Konstruktion des Zylinders müssen die abgedeckten Teile von C_1 und C_2 entfernt werden.
Diese Änderungen in der Datenstruktur bzw am Graphen kosten Zeit $O(n)$. Das ist o.k.