Sweepline-Algorithmen

Wolfgang Mulzer

1 Bentley-Ottman Algorithmus

Die Eingabe besteht aus einer Folge `s[1]`, `s[2]`, ..., `s[n]` von Strecken in "allgemeiner Lage", die jeweils durch den Start- und Endpunkt repräsentiert werden. Dabei liegt jeder Startpunkt links vom jeweiligen Endpunkt. Gesucht ist die Menge alle paarweisen Schnittpunkte der Strecken, sortiert von links nach rechts.

Der Algorithmus benutzt zwei Datenstrukturen: den *Event Point Schedule* (EPS) und den *Sweepline Status* (SLS). Beim Event Point Schedule handelt es sich um eine Prioritätswarteschlange, welche die Ereignispunkte von links nach rechts sortiert enthält. Der Sweepine Status ist ein geordnetes Wörterbuch, das die Strecken, welche die aktuelle Sweepline schneiden, von oben nach unten sortiert enthält (wie implementiert man das in Java?) Der Algorithmus geht folgendermaßen vor.

```
  SLS <- ()
  for i := 1 to n do
      EPS.insert(s[i].start)
      EPS.insert(s[i].end)
  while !EPS.empty() do
      q <- EPS.deleteMin()
      HandleEvent(q)

HandleEvent(q)
    if q is the startpoint of a line segment s[i] then
        SLS.insert(s[i])
        s' <- SLS.pred(s[i])
        s'' <= SLS.succ(s[i])
        if s[i] intersects s' or s'' to the right of q then
            compute the intersection point(s) and insert it/them into EPS
    if q is the endpoint of a line segment s[i] then
        s' <- SLS.pred(s[i])
        s'' <= SLS.succ(s[i])
        SLS.delete(s[i])
        if s' and s'' intersect to the right of q then
            compute the intersection point and insert it into EPS
    if q is the intersection of s' and s'' then
        output q
        switch s' and s'' in EPS
        if any of s' or s'' intersect any of its new neighbors to the right of q
          then
            compute the intersection point(s) and insert it/them into EPS
```

2 Triangulierung eines Polygons

Die Eingabe ist ein einfaches Polygon P in allgemeiner Lage, d.h., die y-Koordinaten aller Ecken von P sind paarweise verschieden. Die Ausgabe besteht aus einer Menge von Diagonalen, die P in y-monotone Polygone zerlegen.

Wir lösen das Problem mit einem Sweepline-Algorithmus. Der Event-Point-Schedule `EPS` enthält die Ecken von P, sortiert von oben nach unten. Er wird als Stack realisiert.

Der Sweepline-Status besteht aus der Folge der Intervalle, in denen die horizontale Sweepline das Polygon schneidet, von links nach rechts sortiert. Er wird als geordnetes Wörterbuch realisiert. Jedes Intervall I speichert eine linke und eine rechte Polygonkante, die Start- und die Endkante des Intervalls, I.`start` und I.`end`. Außerdem speichert I einen *Helfer* I.`helper`. Dies ist die niedrigste Ecke des Polygons in dem geschlossenen Viereck definiert durch die Sweepline, I.`start` und I.`end`. (bezüglich einer allgemeinen Sweepline, welche keine Ecke des Polygons schneidet, aber I als Intervall enthält). Beachte: I.`helper` ist entweder ein Endpunkt von I.`start` oder I.`end`, oder eine Verschmelzungsecke.

```
while !EPS.isEmpty() do
    q <- EPS.pop()
    handleVertex(q)

// q ist Startecke: die zu q inzidenten Kanten liegen unter q, der
//                  Innenwinkel bei q ist < 180 Grad
handleStartVertex(q)
    Erzeuge ein neues Intervall I mit den zu q inzidenten Kanten als
        Start- und  Endkante, und mit I.helper = q
    Fuege I in SLS ein

// q ist regulaere Ecke: eine zu q inzidente Kante liegt ueber q
//                       eine zu q inzidente Kante liegt unter q
handleRegularVertex(q)
    I <- Intervall, das q enthaelt
    if I.helper ist Verschmelzungsecke then
        Fuege Diagonale von q zu I.helper ein
    aktualisiere I.start bzw. I.end
    I.helper <- q

// q ist Trennungsecke: die zu q inzidenten Kanten liegen unter q,
//                      der Innenwinkel bei q ist > 180 Grad
handleSplitVertex(q)
    I <- Intervall, das q enthaelt
    Fuege Diagonale von q zu I.helper ein
    Teile I in zwei Intervalle I1 und I2
    I1.helper <- q; I2.helper <- q

// q ist Verschmelzungsecke: die zu q inzidenten Kanten liegen ueber q,
//                           der Innenwinkel bei q ist > 180 Grad
 handleMergeVertex(q)
    I1, I2 <- benachbarte Intervalle, die q enthalten
    if I1.helper ist Verschmelzungsecke then
        Fuege Diagonale von q zu I1.helper ein
    if I2.helper ist Verschmelzungsecke then
```

```
        Fuege Diagonale von q zu I2.helper ein
    Verschmelze I1 und I2 zu einem Intervall I
    I.helper <- q

// q ist Endecke: die zu q inzidenten Kanten liegen ueber q,
//                der Innenwinkel bei q ist < 180 Grad
 handleEndVertex(q)
    I <- Intervall, das q enthaelt
    if I.helper ist Verschmelzungsecke then
        Fuege Diagonale von q zu I.helper ein
    Loesche I
```

Das Sortieren der Ecken nach y-Koordinate benötigt $O(n \log n)$ Zeit. Danach gibt es n Ereignispunkte. Jeder Ereignispunkte kann in $O(\log n)$ Zeit bearbeitet werden, da man nur konstant viele Wörterbuchoperationen ausführen muss. Die Gesamtlaufzeit ist $O(n \log n)$.

Die Korrektheit des Algorithmus folgt, weil jede Trennungs- und Verschmelzungsecke durch eine Diagonale nach oben bzw. unten zerstört wird. Die Definition der Helfer stellt sicher, dass die eingefügten Diagonalen im Inneren des Polygons liegen und sich nicht echt schneiden. Die Korrektheit der Helfer im Algorithmus folgt durch Induktion nach der Anzahl der Durchläufe der `while`-Schleife und durch Inspektion der einzelnen Fälle.