Schnittpunkte zwischen Strecken, *„revisited, asymptotically fast“*

Plane Sweep Algorithmen

Plane Sweep: Paradigma zum Entwurf von Algorithmen (z.B. hier: Line Sweep Algorithmus)

n-dimensionales Problem aufspalten in

- (n-1)-dimensionales Problem
- Sortierter Durchlauf („Sweep“) in der verbleibenden Dimension („Zeit“)

Line Sweep Algorithmus

Problem: Schnittpunkte von Strecken ermitteln

Komplexität:

- Brute-Force: $O(n^2)$ (n: Anzahl der Strecken)

Geht das nicht effizienter?

- Im schlimmsten Fall n(n-1)/2 Schnittpunkte
- Aber: meist viel weniger!!!

Schön wäre ein Algorithmus, dessen Komplexität von der Ausgabe abhängt (Output-sensitiv)

Line Sweep Algorithmus

Voraussetzung:

- x-Koordinaten der Schnitt- und Endpunkte sind paarweise verschieden
- Länge der Segmente > 0
- nur echte Schnittpunkte
- keine Linien parallel zur y-Achse
- keine Mehrfachschnittpunkte
- keine überlappenden Segmente

Geht auch ohne diese Voraussetzungen, aber es wird komplizierter…

Bartuschka, Mehlhorn, Näher: *A Robust and Efficient Implementation of a Sweep Line Algorithm for the Straight Line Segment Intersection Problem* (impl. in LEDA)

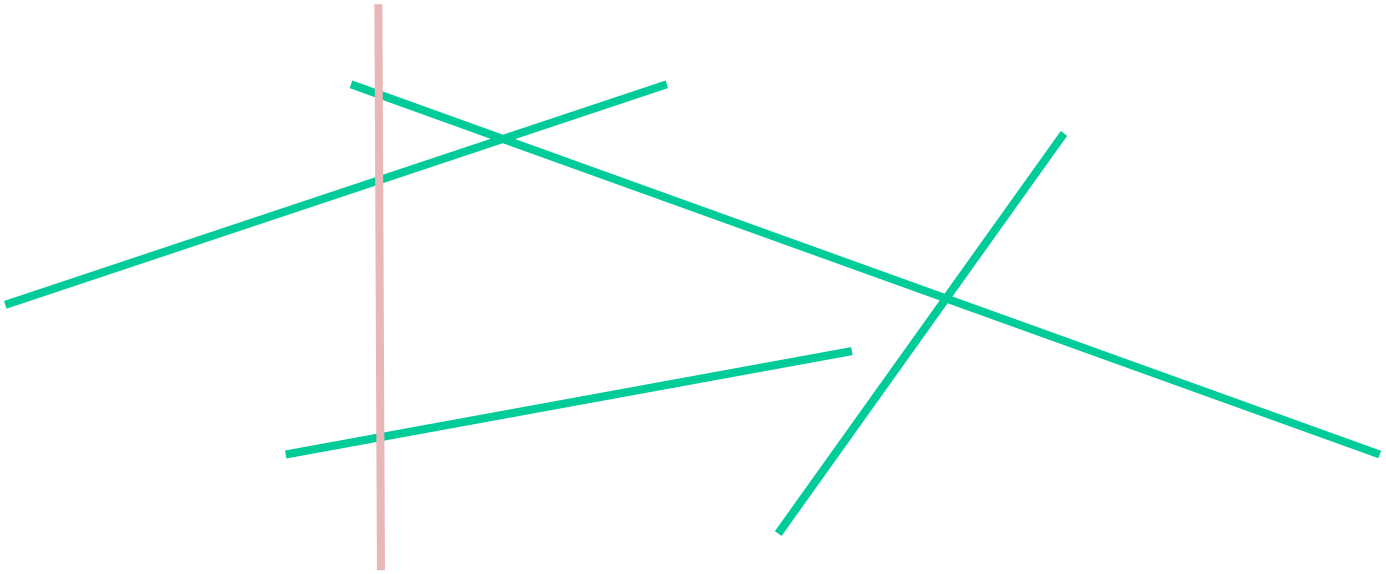
http://www.dsi.unive.it/~wae97/proceedings/ONLY_PAPERS/pap13.ps.gz

Line Sweep Algorithmus

Bentley-Ottmann Line Sweep Algorithmus

http://people.scs.carleton.ca/~michiel/lecturenotes/ALGGEOM/bentley-ottmann.pdf

- Wir laufen (*sweep*) mit einer imaginären, vertikalen Linie (***sweep line***) von links nach rechts
- Schnittpunkte können nur zwischen benachbarten Segmenten auf der Linie auftreten

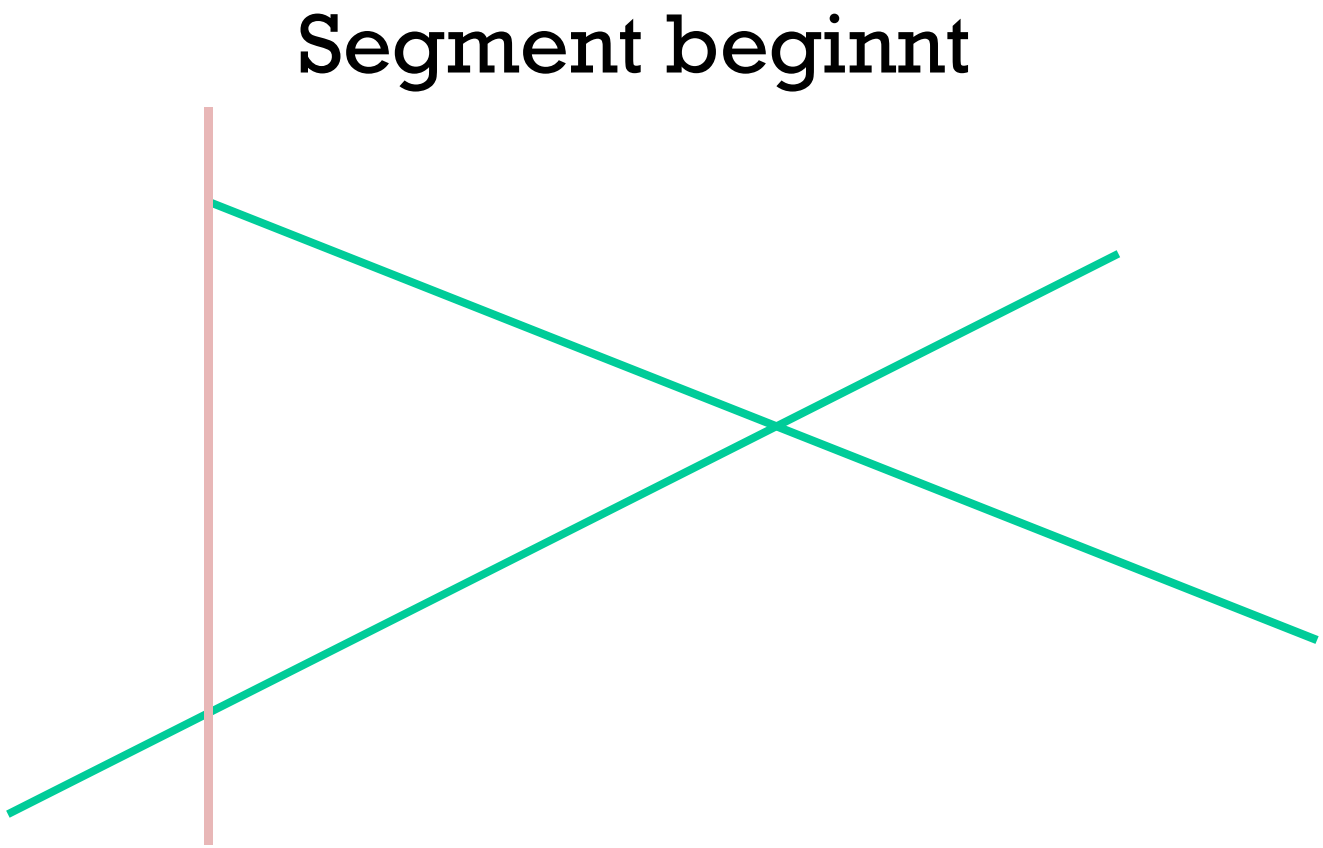


Line Sweep Algorithmus

- Während des Durchlaufs der Linie treten Ereignisse (***events***) auf:
  - Segment beginnt
  - Segment endet
  - Schnittpunkt zwischen Segmenten
- Datenstruktur für diese Ereignisse erstellen (***event queue, X-Struktur***)
- Datenstruktur erstellen, die die Schnittpunkte der Segmente mit der Linie verwaltet (***sweep line, Y-Struktur***)
- Schnittpunkte zwischen Segmenten treten auf, wenn sich die Reihenfolge der Schnittpunkte der Segmente mit der ***sweep line*** ändert (also bei obigen Ereignissen)
- Also: Bei jedem Ereignis ***sweep line*** updaten und ggf. Schnittpunkte ermitteln

Line Sweep Algorithmus

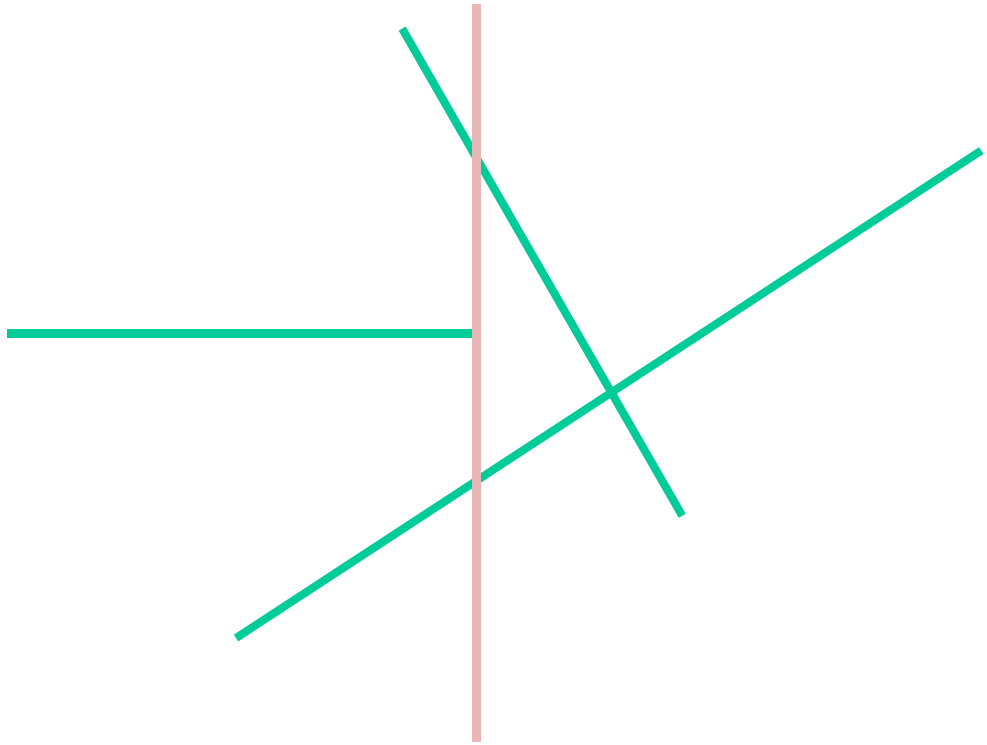
Ereignisse



Line Sweep Algorithmus

Ereignisse

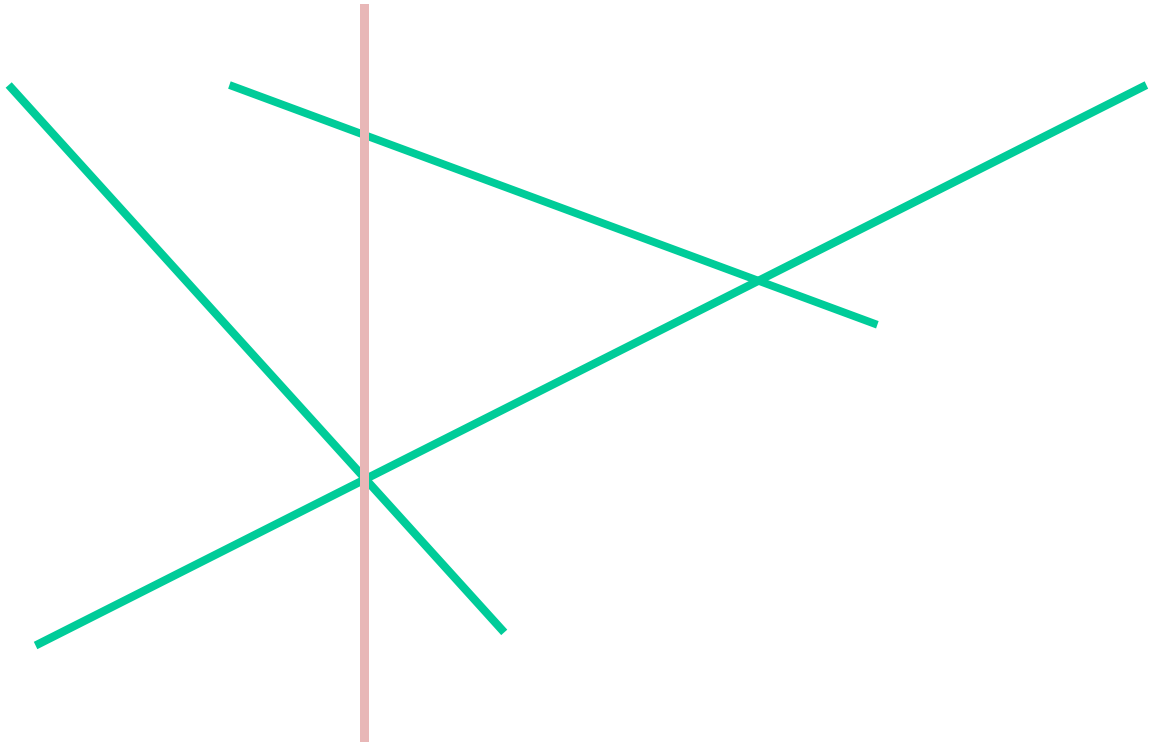
Segment endet



Line Sweep Algorithmus

Ereignisse

Schnittpunkt



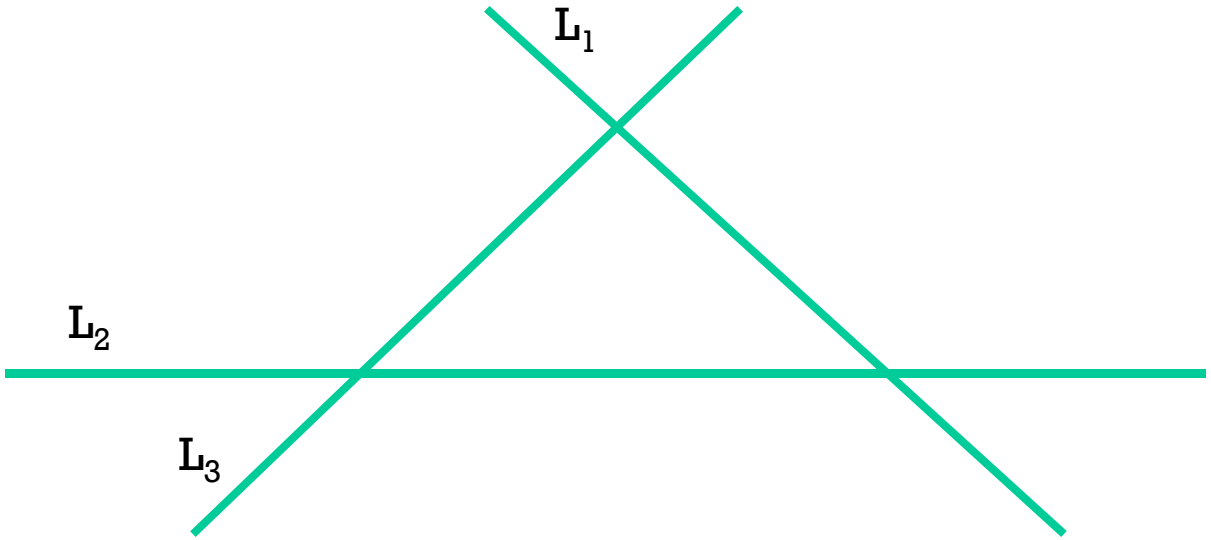
Line Sweep Algorithmus

Skizze des Line Sweep Algorithmus

- Initialisiere Segmentreihenfolge und Ereignisse
- Für alle Ereignisse:
  - Prüfe, ob sich Segmentreihenfolge ändert
  - Neue Nachbarsegmente auf Schnittpunkte testen
  - Im Falle eines Schnittpunkts:
    - diesen ausgeben und
    - in die Ereignisliste einfügen

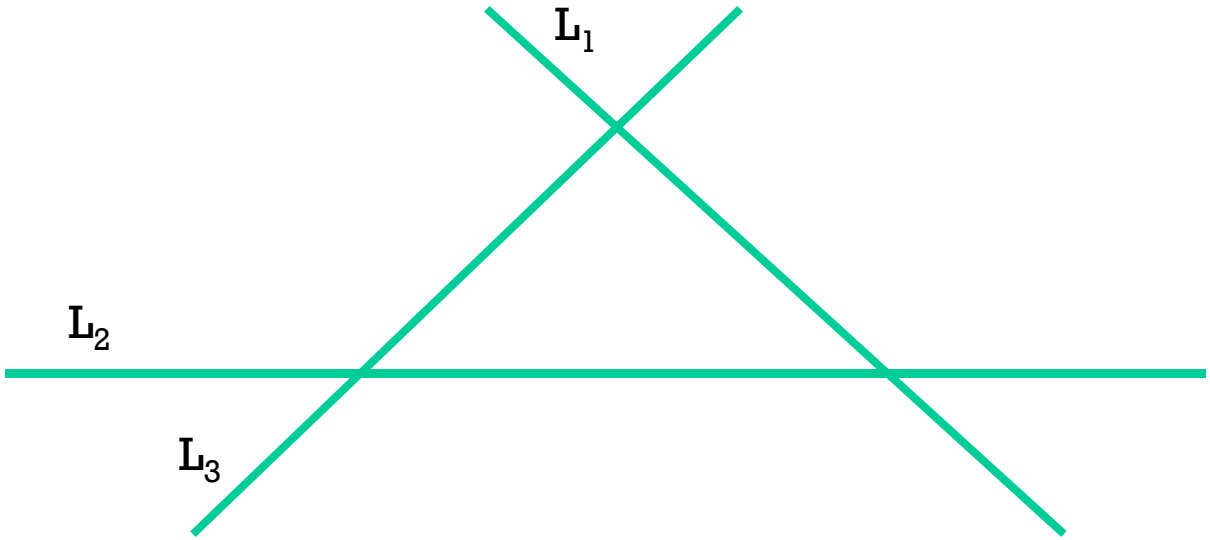
Line Sweep Algorithmus

©: Anlu Wang, CMU



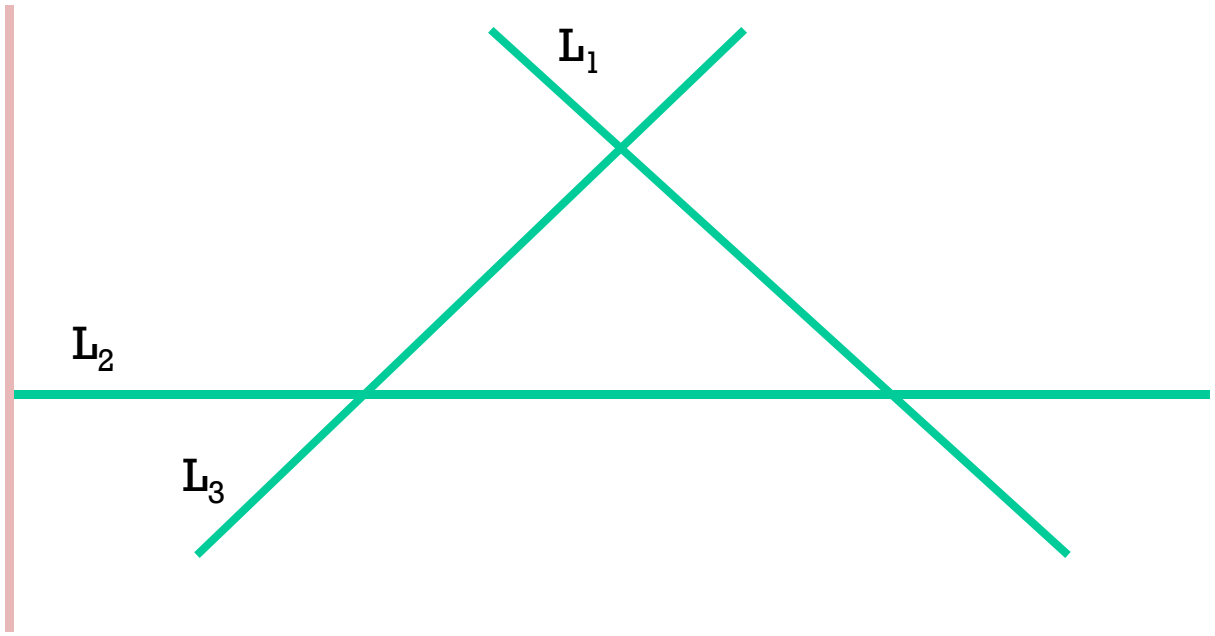
Ereignis Segmentreihenfolge Event Queue Ausgabe

Line Sweep Algorithmus



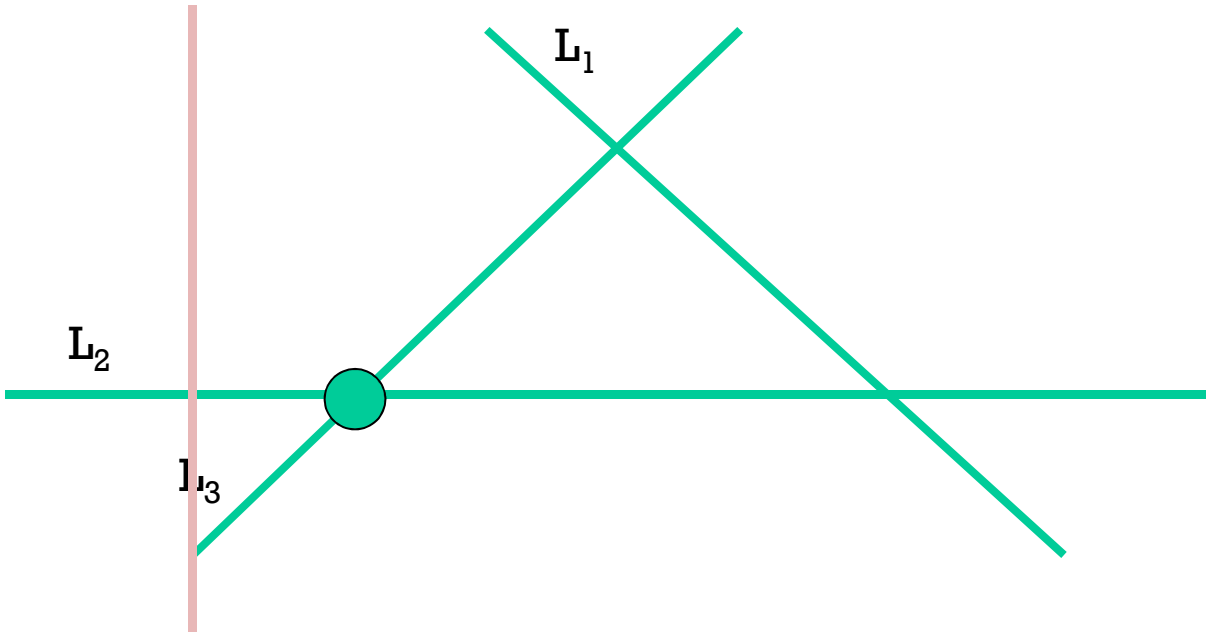
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
		$S_2, S_3, S_1, E_3, E_1, E_2$	

Line Sweep Algorithmus



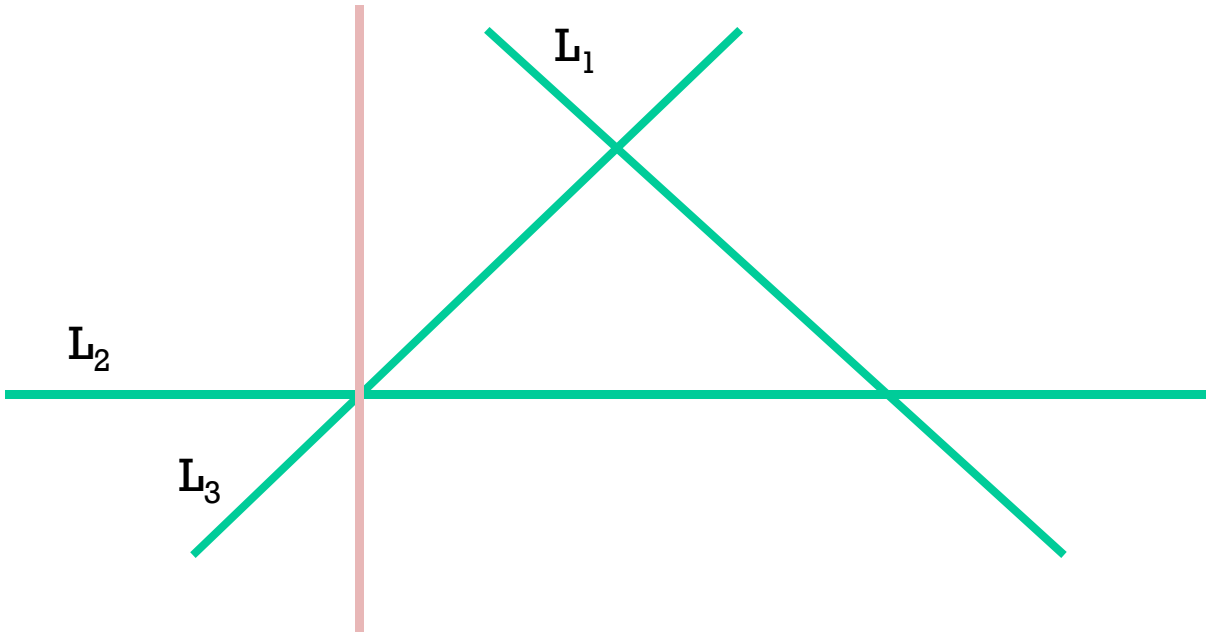
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
S_2	L_2	S_3, S_1, E_3, E_1, E_2	

Line Sweep Algorithmus



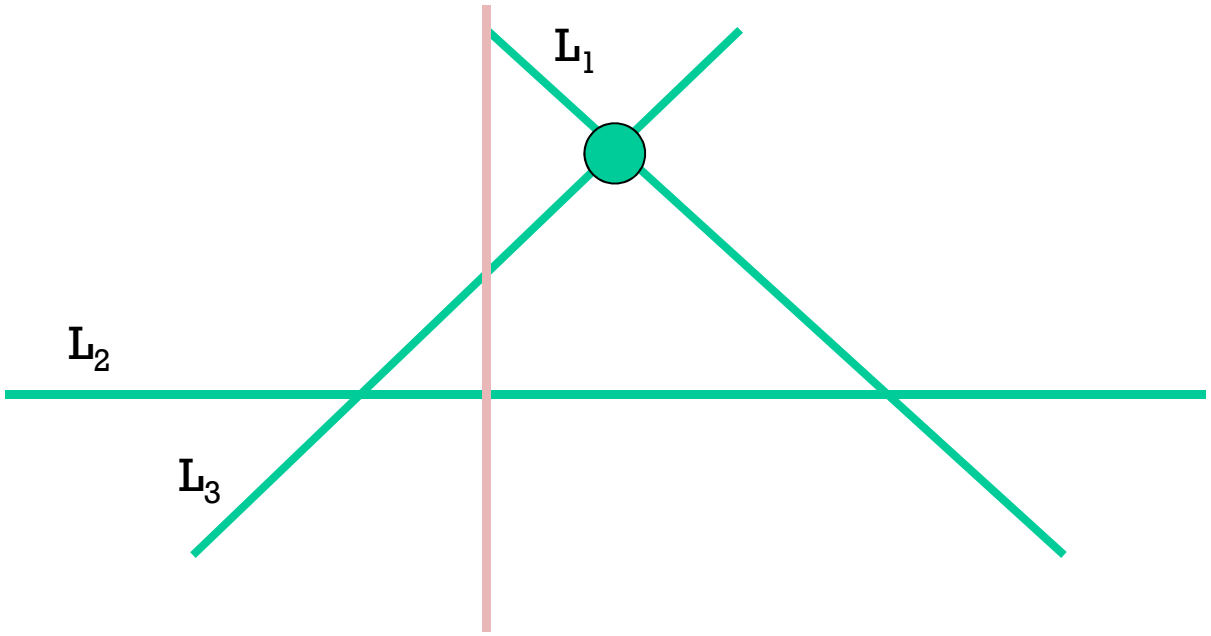
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
S_3	L_2 L_3	$I_{23}, S_1, E_3, E_1, E_2$	I_{23}

Line Sweep Algorithmus



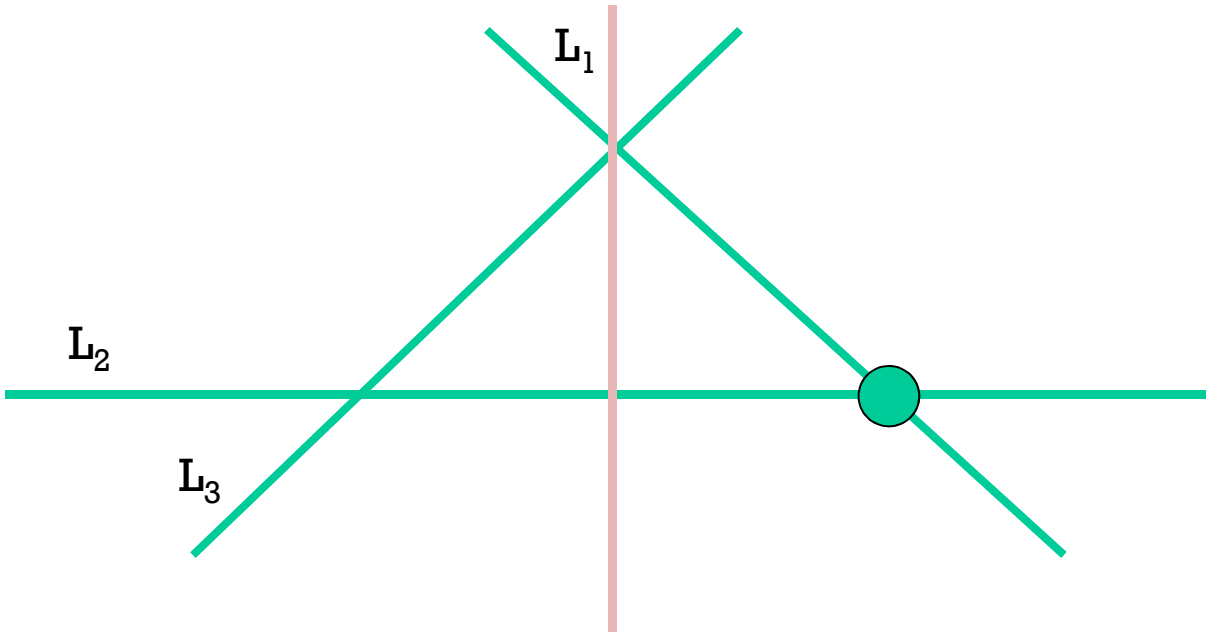
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
I_{23}	L_3 L_2	S_1, E_3, E_1, E_2	

Line Sweep Algorithmus



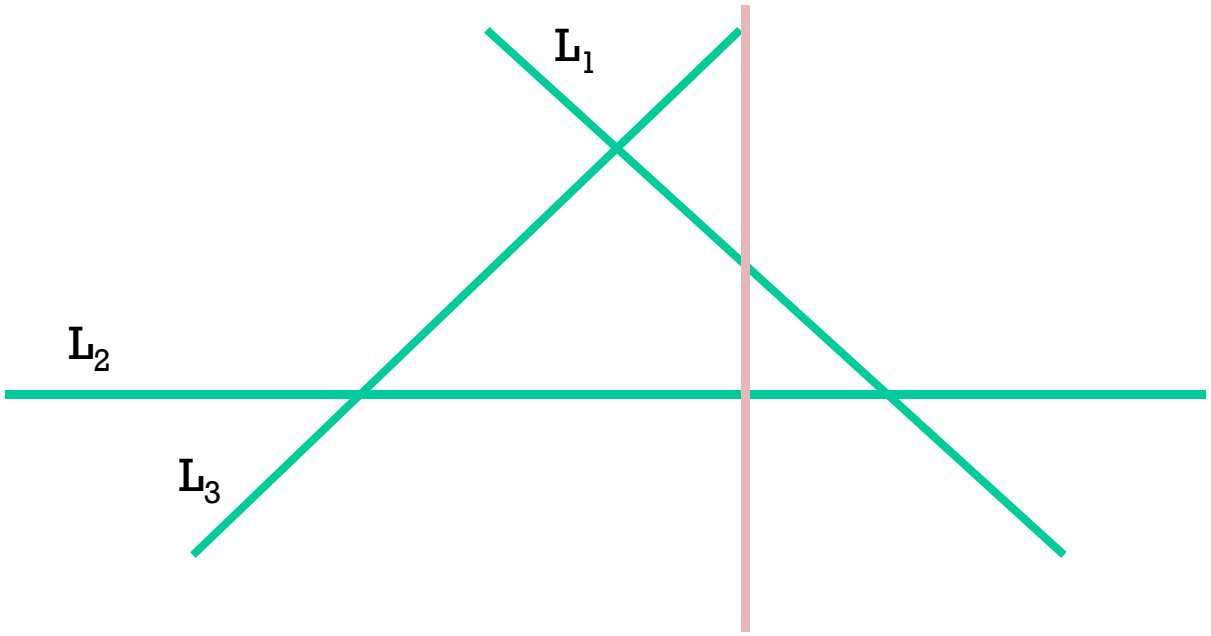
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
S_1	L_1 L_3 L_2	I_{13}, E_3,E_1, E_2	I_{13}

Line Sweep Algorithmus



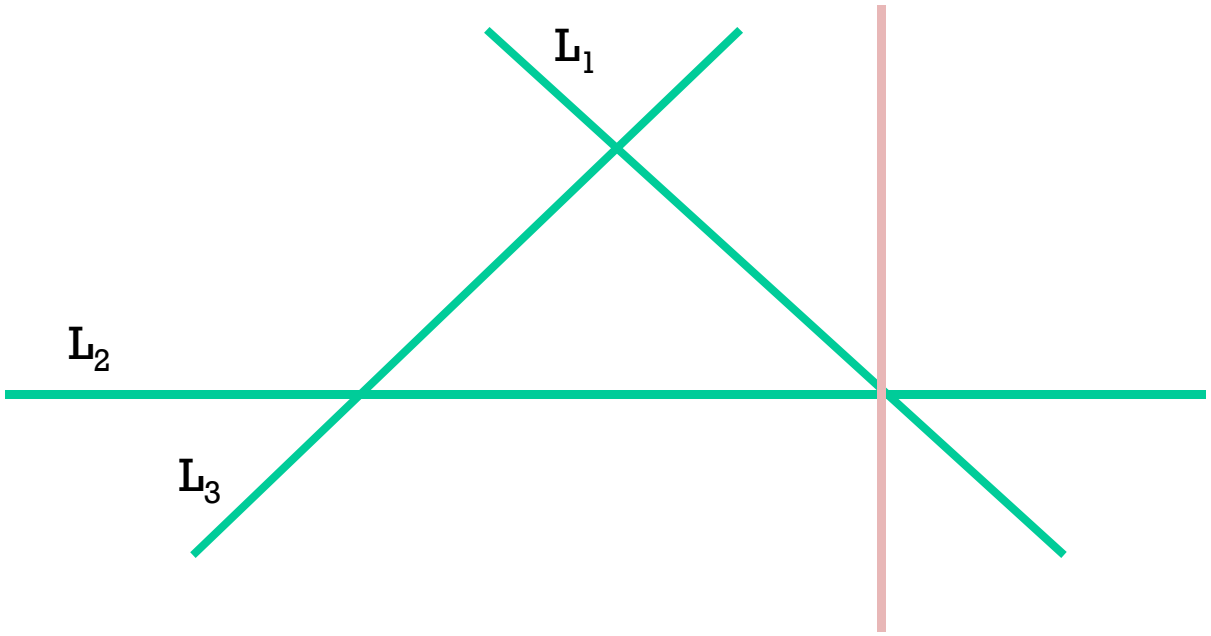
Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
I_{13}	$L_3\ L_1\ L_2$	E_3, I_{12}, E_1, E_2	I_{12}

Line Sweep Algorithmus



Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
E_3	$L_1\ L_2$	I_{12}, E_1, E_2	

Line Sweep Algorithmus



Ereignis	Segmentreihenfolge	Event Queue	Ausgabe
I_{12}	$L_2\ L_1$	E_1, E_2	

Line Sweep Algorithmus

Für Event Queue und Sweep Line:

- Datenstruktur, die die Operationen
  - Insert
  - Delete
  - Find next neighbors

  in O(log n) unterstützt
- z.B. balanzierter Binärbaum, AVL-Baum (nach Adelson-Velsky, Landis)
  - Bei jeder Operation wird sichergestellt, dass der kürzeste und längste Pfad von der Wurzel zu allen Blättern sich um maximal 1 unterscheiden
- meist als Container verfügbar

Line Sweep Algorithmus

Pseudo-Code:

```
Initialize event queue x = all segment endpoints;
    Sort x by increasing x and y;
    Initialize sweep line SL to be empty;
    Initialize output intersection list L to be empty;

    While (x is nonempty) {
        Let E = the next event from x;
        If (E is a left endpoint) {
            TreatLeftEndpoint(); // Add new segment to SL
        }
        Else If (E is a right endpoint) {
            TreatRightEndpoint();  // Remove segment from SL
        }
        Else {  // E is an intersection event
            TreatIntersection();    // swap intersecting segments in SL
        }
        remove E from x;
    }
    return L;
}
```

Line Sweep Algorithmus

Pseudo-Code:

```
TreatLeftEndpoint() {
    Let segE = E's segment;
    Add segE to SL;
    Let segA = the segment above segE in SL;
    Let segB = the segment below segE in SL;
    If (I = Intersect( segE with segA) exists)
        Insert I into x;
    If (I = Intersect( segE with segB) exists)
        Insert I into x;
}


TreatRightEndpoint() {
    Let segE = E's segment;
    Let segA = the segment above segE in SL;
    Let segB = the segment below segE in SL;
    Remove segE from SL;
    If (I = Intersect( segA with segB) exists)
        If (I is not in x already) Insert I into x;
}
```

Line Sweep Algorithmus

Pseudo-Code:

```
TreatIntersection() {  // E is an intersection event
    Add E to the output list L;
    Let segE1 above segE2 be E's intersecting segments in SL;
    Swap their positions so that segE2 is now above segE1;
    Let segA = the segment above segE2 in SL;
    Let segB = the segment below segE1 in SL;
    If (I = Intersect(segE2 with segA) exists)
        If (I is not in x already) Insert I into x;
    If (I = Intersect(segE1 with segB) exists)
        If (I is not in x already) Insert I into x;
}
```

Line Sweep Algorithmus

Komplexität:

Annahme:

- n Segmente (2n Endpunkte),
- k Schnittpunkte (Output!!!)

Analyse:

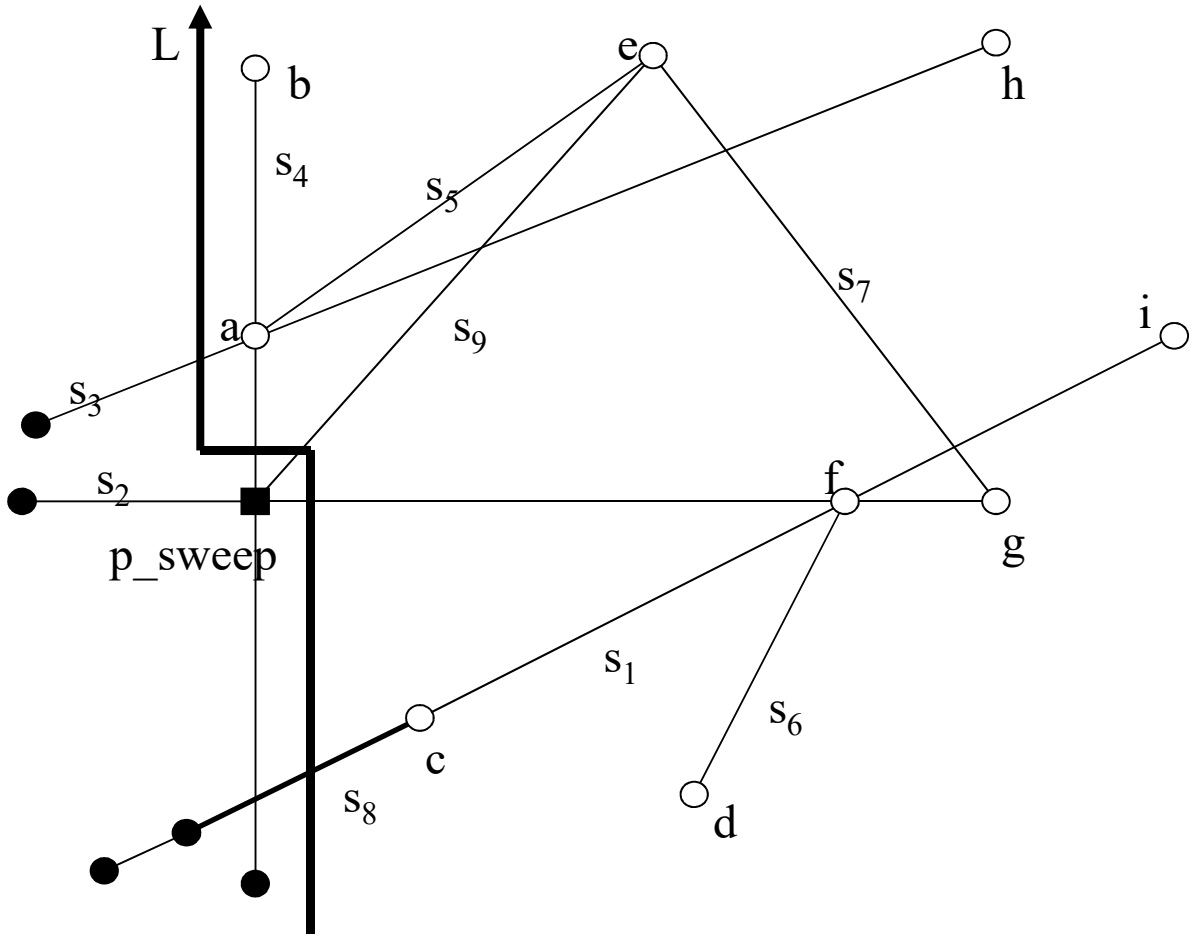
- Sortieren der n Segmente in einer Dimension: $O(n \log n)$
- Operationen entlang der Sweep-Line: 2n + k
- Pro Operation: Löschen, Einfügen in der ScanLine Struktur: $O(\log n)$

Somit:

$$O(n \log n) + (2n + k)\, O(\log n) = O((n + k) \log n)$$

Line Sweep Algorithmus

Idee robuste Implementierung (LEDA, CGAL)



Line Sweep Algorithmus

Ähnliche (Plane Sweep) Algorithmen:

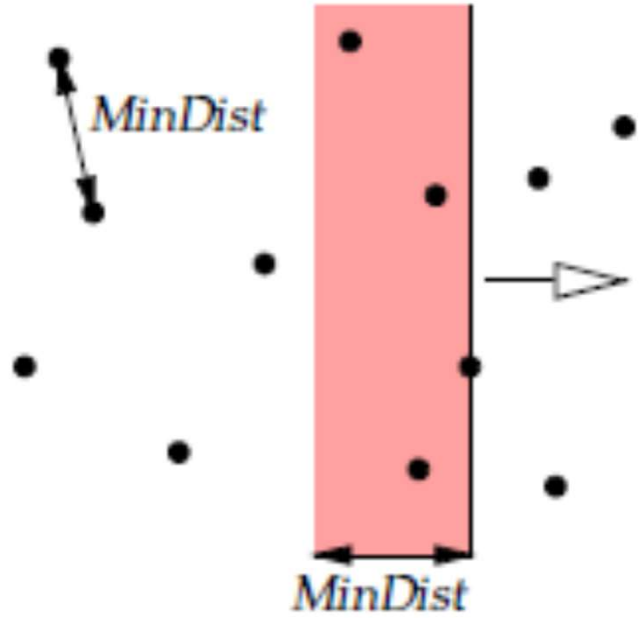
- Schnitt konvexer Polygone
- Closest Pair

Closest Pair

Voraussetzungen:

HS: Horizontalstruktur, Punkte nach x-Koord sortiert

VS: Vertikalstruktur, alle Punkte im Streifen MinDist links von der SweepLine, nach y-Koord sortiert



Closest Pair

Algorithmus 5.1 Bestimmung des dichtesten Punktepaares in der Ebene

CLOSESTPAIR($p_1, \ldots, p_n$)

Input: n Punkte $p_1, \ldots, p_n \in \mathbb{E}^2$

{Initialisierungsschritt}

1 Sortiere die Punkte $p_1, \ldots, p_n$ nach aufsteigenden x-Koordinaten und füge sie ins Feld *HS* ein

2 $VS \leftarrow$ AVL-Baum, initialisiert mit HS_1 und HS_2

3 $MinDist \leftarrow |HS_1\, HS_2|$

4 $MinPair \leftarrow (1, 2)$

5 $links \leftarrow 1$ {HS_{links} ist der am weitesten links liegende Punkt im Streifen}

6 $rechts \leftarrow 3$ {HS_{rechts} ist der aktuelle Punkt auf der Sweep line}

Closest Pair

```
   {Sweep}
 7 while rechts ≤ n do
 8   if HS_links.x + MinDist ≤ HS_rechts.x then
 9     Entferne HS_links aus VS
10     links ← links + 1
11   else
12     Bestimme den minimalen Abstand D = |HS_rechts HS_min|
       von HS_rechts zu allen Punkten in VS
13     if D < MinDist then
14       MinDist ← D
15       MinPair ← (rechts, min)
16     end if
17     Füge HS_rechts in VS ein
18     rechts ← rechts + 1
19   end if
20 end while
```

Output: Dichtestes Punktepaar *MinPair* mit minimalem Abstand *MinDist*

„Streifen“ updaten: alle Punkte, die zu weit links liegen, entfernen

Bestimme MinDist aus Punkt auf der SweepLine und allen Punkten des Streifens

SweepLine vorrücken