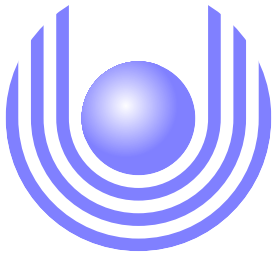
Ein SECONDO-basierter Routenplaner mit Berücksichtigung von Steigungen

Holger Hennings, Fabio Valdés

Fachpraktikum Erweiterbare Datenbanksysteme
(WS 2015/16)

Lehrgebiet Datenbanksysteme für neue Anwendungen



FernUniversität in Hagen

Motivation

Motivation

- **Ziel:**
  Finde den aus Sicht des Anwenders
  (unter Berücksichtigung von Steigungen) optimalen Weg
  zwischen zwei Punkten
- **Dazu werden benötigt:**
  - **Höhendaten**: In Form von Punktwolken
    (enthalten dreidimensionale Punkte mit
    Längen-, Breiten- und Höhenangabe)
  - **Straßennetzdaten**: OpenStreetMap (OSM)
- **Problem:**
  - Kombination dieser vorhandenen Informationen, die dann zur Berechnung des optimalen Weges benutzt werden kann
  - **Zwischenschritt:** Irreguläre Dreiecksnetze (TIN)

Motivation

- **Genauer:**
  - Die durch Punktwolken vorliegende Höheninformation wird in ein TIN überführt
  - Kombination des TINs mit den OSM Straßennetzdaten
  - Berechnung des optimalen Weges nach Adresseingabe
  - Benutzerfreundliche Darstellung und Interaktion

Aufgabenbeschreibung

Übersicht Aufgabenbeschreibung

1. Import von Höhendaten in Punktwolken
2. Erweiterung eines Straßennetzes um Höhendaten
3. Bestimmung des optimalen Weges
4. Erweiterung der SECONDO GUI

Import von Höhendaten in Punktwolken

Import von Höhendaten in Punktwolken

- Höhendaten liegen ursprünglich in Form von LAS-Dateien vor
  - LAS-Dateien in komprimierter Form: LAZ-Dateien
  - Entpacker: `laszip`
  - LAS Dateien enthalten LIDAR-Punktwolkendaten
  - LIDAR (Light Detection and Ranging) ist eine optische Fernerkundungstechnik

- Zu implementieren: Operator **importpointcloud**
  - Aufgabe: Einlesen von einer oder mehreren LAS-Dateien
  - Eingabe: Dateipfad oder Verzeichnis
  - Ausgabe: Strom von Punktwolken

- **importpointcloud**: $\{\underline{string}, \underline{text}\} \rightarrow \underline{stream}(\underline{pointcloud})$

Beispielhafte Anwendung: **importpointcloud**

```
let Pointclouds = importpointcloud("elevationdata/") consume
```

pointcloud-Objekte

- Noch zu realisieren: Attributdatentyp *pointcloud*
- Die einzelnen *pointcloud*-Objekte repräsentieren dabei disjunkte rechteckige Teilbereiche des ursprünglichen Bereiches aus den LAS-Dateien
  - enthalten bis zu 50.000 dreidimensionale Punkte
  - Interne Realisierung: FLOB
  - werden gemeinsam in einer neuen Relation (mit einem Attribut) abgelegt

pointcloud-Objekte

- Anlegen eines `R-Baums` über dieser neuen Relation, damit Teilbereiche schnell gefunden werden
  - R-Baum Implementierung in SECONDO vorhanden: `RTreeAlgebra`
  - `RTreeAlgebra` hat Operatoren zum Erstellen und Abfragen eines R-Baums
- Forderung:
  - Innerhalb der einzelnen Punktwolken: Verwaltung der Punkte in einem Gitter (statische oder dynamische Größe bis maximal 2.500 Zellen)
  - Zu realisieren: Zu einem Anfragerechteck (z.B. Bounding Box eines Straßenabschnitts) effizient die entsprechenden Punktmengen ermitteln

Erweiterung eines Straßennetzes um Höhendaten

Straßennetz-Import

Grundelemente von OSM-Daten: Nodes, Ways, Relations, Tags

Import:

1. Herunterladen der OSM-Daten von der Geofabrik-Seite (Version mit Endung: `osm.bz2`)
2. Entpacken der `bz2`-Datei
3. Import über SECONDO-Skript (*OrderedRelationGraphFromFullOSMImport.SEC*) in eine Datenbank (Relation `Edges`)

Relation Edges

- Linienzug , der den Straßenverlauf beschreibt: *sline*
  - Relation `Edges` → Attribut `Curve`: Typ *sline*
- Attribut `Altitude` zur Relation `Edges` hinzufügen:
  Soll passendes Höhenprofil zu jeder Straße beinhalten
- Zu implementieren: Operator **lcompose**:
  Zu jeder *sline* passendes *lreal* erzeugen
  - **lcompose**: $sline \times stream(pointcloud) \times real \rightarrow lreal$
  - Datentyp *lreal*: Kontinuierliche Höhenabbildung
- Also zunächst nötig:
  Kombination der `Edges` Relation mit Punktwolken

Der Weg zum **lcompose**-Operator Teil 1

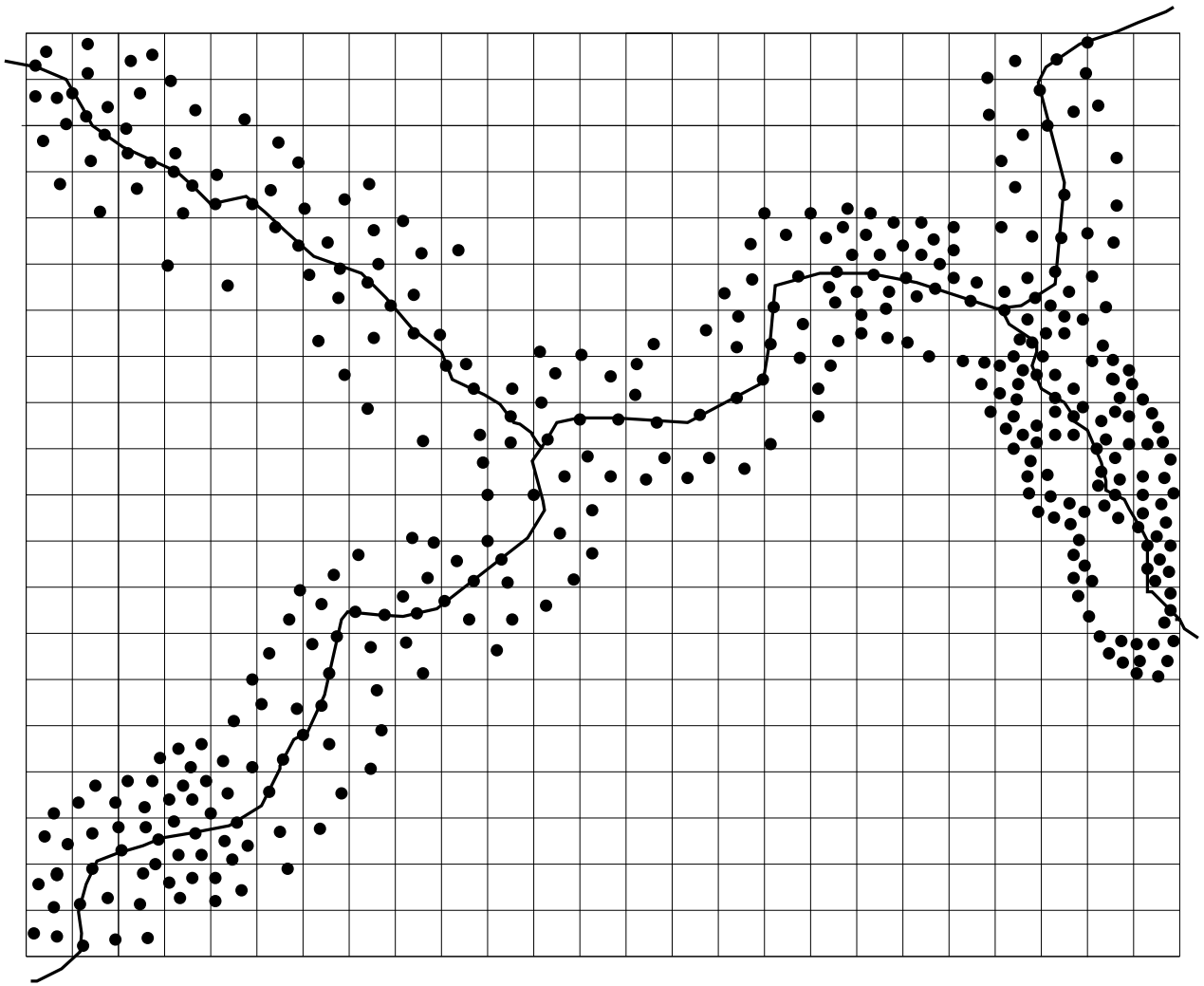
ZIELE:

1. Kombination der Relation `Edges` mit Punktwolken
2. Zu jeder *sline* passendes *lreal* als neues Attribut `Altitude` zur Relation `Edges` hinzufügen

Der Weg zum **lcompose**-Operator Teil 1

1. Kombination der Relation `Edges` mit Punktwolken
   - Anhängen des neuen Attributs mit **extend** für jedes Tupel
   - Punktwolken filtern mit R-Baum
   - Punkte in der Nähe (unter 20m) der *sline* aus einzelnen *pointcloud*-Objekten extrahieren

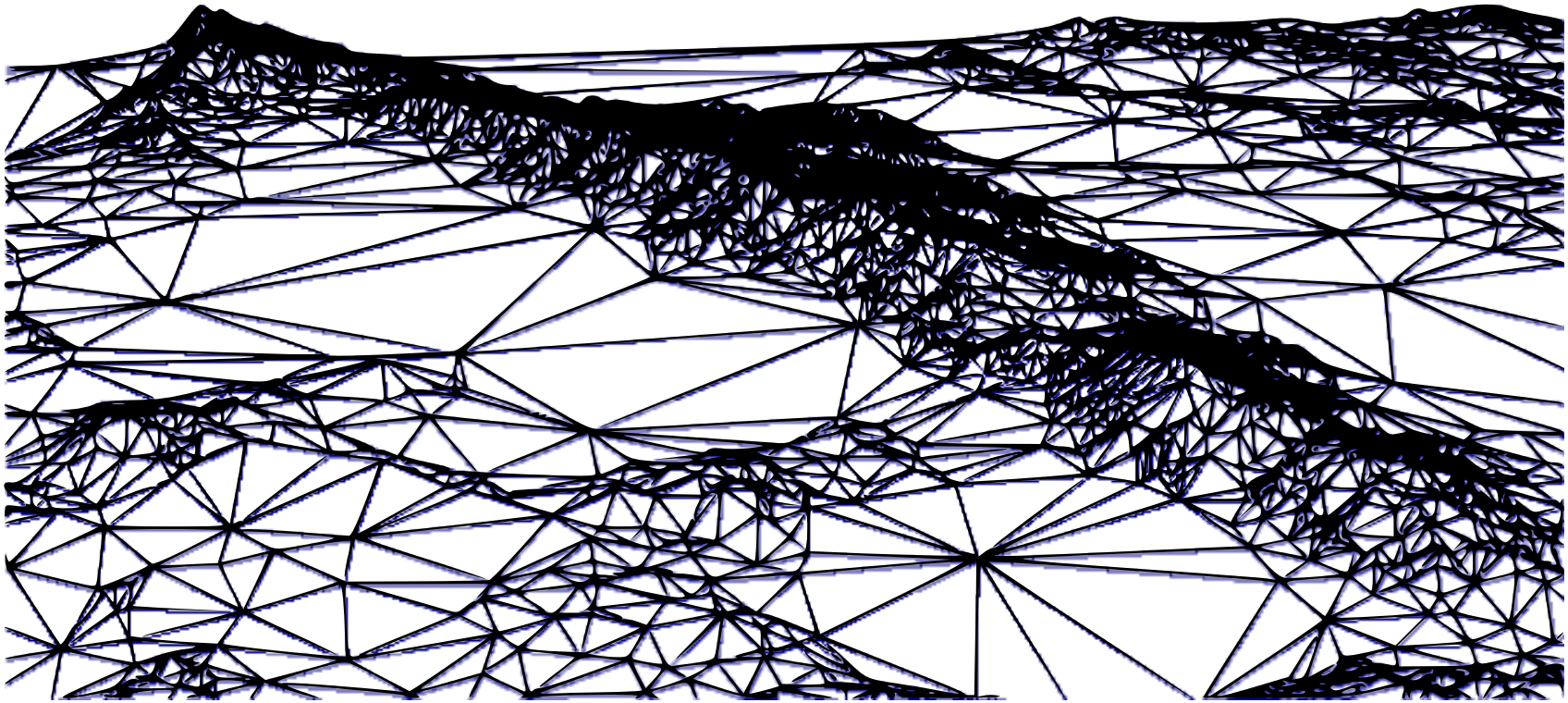
Punktwolken beschränkt auf eine Straßennetzumgebung:



Der Weg zum **lcompose**-Operator: Teil 1

1. Kombination der Relation `Edges` mit Punktwolken
   - ...
   - extrahierte Punktmengen im Hauptspeicher in irreguläres Dreiecksnetz (triangulated irregular network, TIN) überführen
   - TIN nicht als Datenbankobjekt speichern
   - Zusätzlicher Parameter (zwischen 0 und 1) in **lcompose** für die Präzision des TIN
   - zur Orientierung: `TinAlgebra` in SECONDO

Beispiel eines TINs:



Der Weg zum **lcompose**-Operator: Teil 2

1. Zu jeder *sline* passendes *lreal* als `Altitude`-Attribut in `Edges` erzeugen
   - hilfreich: Algebra `LineFunction`
   - deren Datentyp *lreal* ordnet Abschnitten eines Linienzugs lineare reelle Funktionen zu
   - Operator **atlocation** berechnet zu jedem Punkt des Linienzugs den reellen Funktionswert
   - Alternative: reellen Parameter für Genauigkeit der LineFunction verwenden

Beispielhafte Anwendung: **lcompose**-Operator

```
let Pointclouds_RTree = Pointclouds createrтree[Pointcloud]



let Edges2 = Edges
             extend[Altitude: .Curve
                    lcompose[Pointclouds_RTree Pointclouds
                    windowintersects[bbox(.Curve)], 0.75] ]
             oconsume
```

Bestimmung des optimalen Weges

geocode-Operator

- Erweiterung des vorhandenen Operators **geocode**
  - Bereits möglich:
    Ermittlung der Geokoordinaten bei Eingabe der Adresse (Straße, Hausnummer, PLZ, Ort) von Google Maps
  - Erweiterung:
    Liste von Vorschlägen bei unvollständiger Eingabe

shortestpath-Operator

- Ergebnis von **lcompose**: Höhenprofile (*lreal*)
- Ziel: Berechnung von optimalen Wegen abhängig von Benutzerpräferenzen
- Zu implementieren: Operator **shortestpath**
  - Zu berücksichtigen: Benutzerpräferenzen, falls spezifiziert
  - Zu verwenden: A*-Algorithmus
- **shortestpath**:
  $orel(tuple(X)) \times IDENT^2 \times point^2 \times \text{pref}) \rightarrow stream(sline)$

Beispielhafte Anwendung: **shortestpath**-Operator

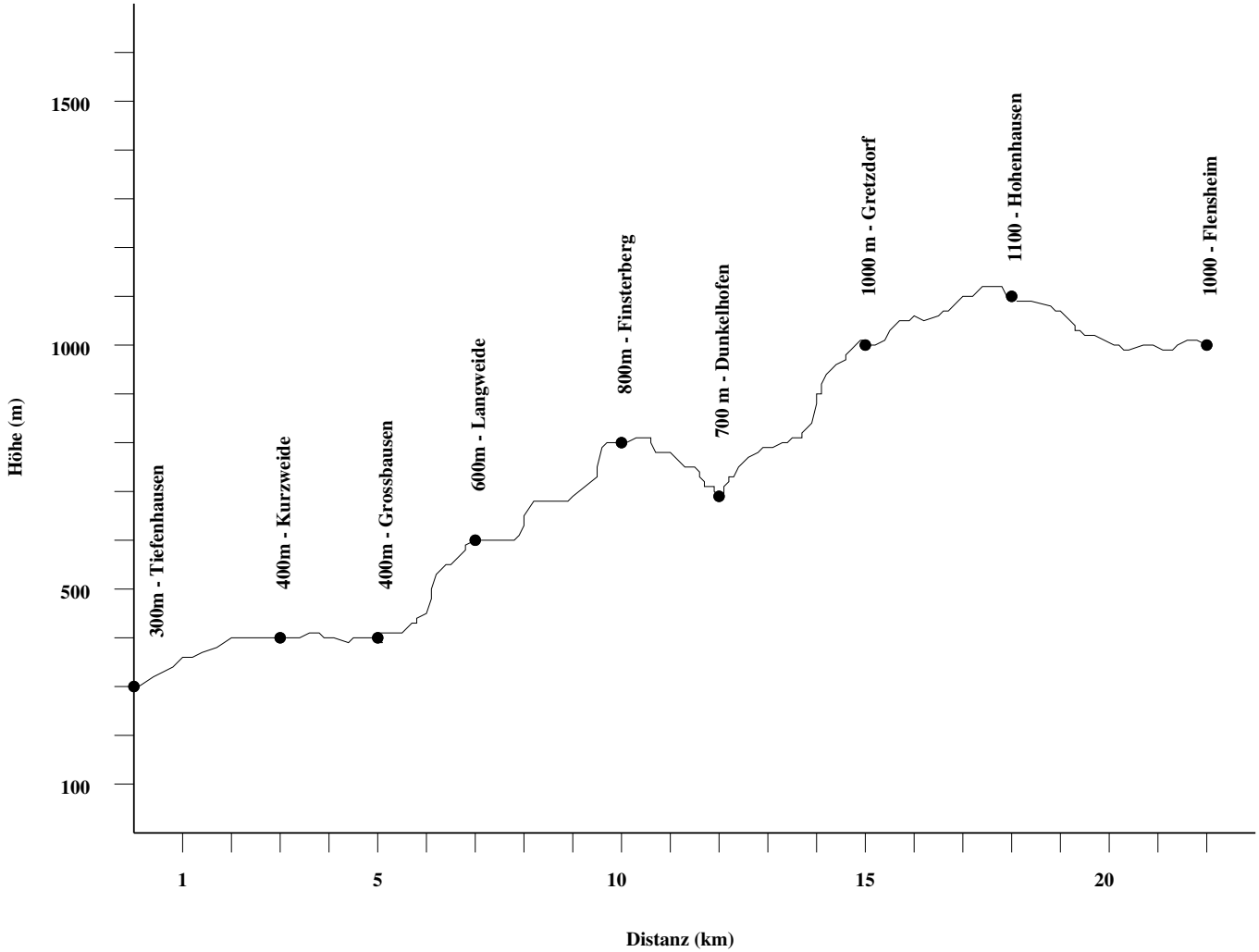
```
query Edges2
      shortestpath[Curve, LineFunction, start, end, pref]
      transformstream consume
```

Erweiterung der SECONDO GUI

Erweiterung SECONDO GUI

- durch **shortestpath** berechneten Weg im Höse-Viewer markieren/einfärben
- Implementierung Shortestpath-Viewer:
  - Eingabe von Höhenpräferenzen
    - Steigungsbereiche (0 bis 5%, 5 bis 10%, usw.) denen Präferenzen zugeordnet werden (z.B. von 0 bis 5)
  - Eingabe von Adressen
  - ggf. Auswahl der gewünschten Adresse aus einer Liste
- Separates Fenster zur Anzeige des Höhenverlaufs

Beispiel eines Höhenprofils:



Zusammenfassung

Übersicht der neuen Operatoren

- **importpointcloud**:
  $\{\underline{string}, \underline{text}\} \rightarrow \underline{stream}(\underline{pointcloud})$
- **lcompose**:
  $\underline{sline} \times \underline{stream}(\underline{pointcloud}) \times \underline{real} \rightarrow \underline{lreal}$
- **shortestpath**:
  $\underline{orel}(\underline{tuple}(X)) \times IDENT^2 \times \underline{point}^2 \times$ pref$\rightarrow \underline{stream}(\underline{sline})$

Danke für die Aufmerksamkeit