Datenstrukturen und Algorithmen zum Rendern von OpenStreetMap-Karten

Jens Pönisch

2016-03-19

Motivation

- Karten für Radtourenbeschreibungen
  - einfacher Renderer in Tcl/Tk (tkosm)
  - Datenabfrage direkt vom OSM-Server
  - mit wachsender Datenmenge nicht mehr beherrschbar
- Wunsch für neue Deutschland-/Europakarte der CLT
- Erste Erfahrungen mit Mapnik
- Problem: PostGIS-Datenbank, Import $>$ 24 Stunden
- Vorstellung Spezialdatenbank Overpass-API, aber nicht von Mapnik aus nutzbar
- erneute Versuche mit eigenem Renderer
- Erfolgreich Deutschland- und Europakarten erstellt
- Karten für Pilgerführer (großer Maßstab), Ergänzungen

Ich will aber nur Karten rendern!

Vortrag beschreibt Algorithmen, kein fertiges Produkt!

OSM-Renderer:

Maperative .NET-Anwendung (closed source), kleine Karten, arbeitet mit OSM-XML-Datendatei und SRTM-Daten

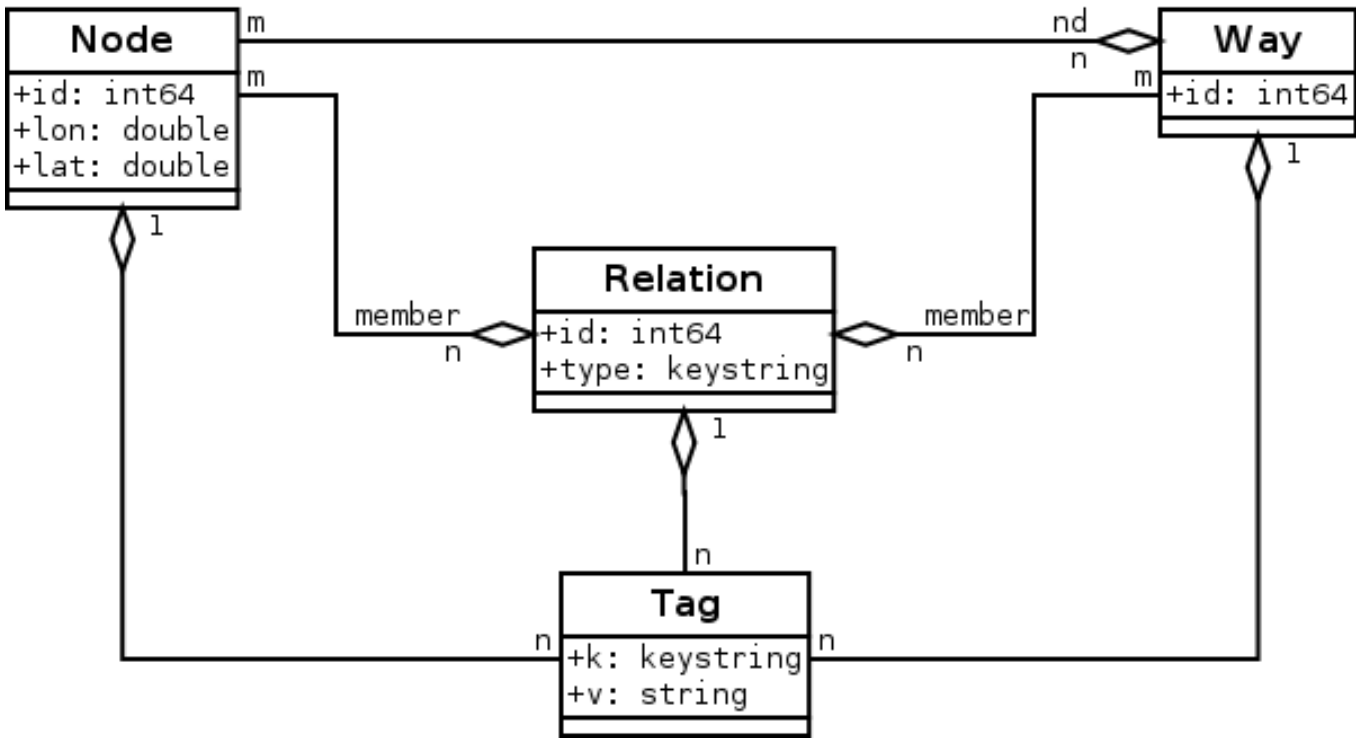
Mapnik C++/Python-Anwendung, benötigt PostGIS-Datenbank (Ressourcen!) oder Shapefiles, große Karten, OSM-Kacheln für Online-Karten

Tilemill/Mapbox Studio Frontend für Mapnik

Inhalt

OpenStreetMap-Datenmodell 1

Beschränkung auf die für das Rendern benötigten Objekte (keine History, Erstellerangaben, hochgeladene Tracks)



OpenStreetMap-Datenmodell 2

Eigenschaften:

- Geographische Koordinaten stehen nur in Knoten, alle anderen Objekte referenzieren Knoten für Positionsangaben.
- Bedeutung der Objekte wird über Tags (Key-Value-Paare) spezifiziert.
- IDs werden getrennt für Knoten, Wege und Relationen vergeben $\Longrightarrow$ nur innerhalb der Klasse eindeutig!
- Kein Flächentyp. Realisierung über geschlossene Wege mit speziellen Tags (`area: yes`, `landuse: forest`, ...) oder Relationen (Multipolygone).
- Knoten können einzeln genutzt werden oder in mehrere Wege und Relationen eingehen (z. B. Wehr im Flusslauf)
- Wege können können ebenfalls einzeln auftreten und Teil einer Relation (z. B. Wanderroute) sein.
- Knoten sind oft reine Positionsangaben $\Longrightarrow$ keine Tags.

Datenquellen

API Online-Zugriff auf OSM-Datenbank, zum Bearbeiten der Daten, aber nicht zur Abfrage großer Gebiete!
Format: XML

Planetfile Wöchentlicher Dump der Datenbank zum Download, tägliche und stündliche Diff-Dateien.
Formate: XML (komprimiert) und PBF.

Planet-Extracts von geofabrik.de. Dump der Einzelkontinente und -länder.
Format: PBF.

Overpass-API Online-Datenbank für Abfrage aus größerer Datenmengen.
Formate: XML und JSON.

Für große Karten eigene Datenquelle erforderlich.

Eigene Datenquelle

Direktes Lesen der Dump-Datei langsam, Datenmenge zu groß für Hauptspeicher.

Datenbank erforderlich:

- PostgreSQL/PostGIS-Datenbank,
- Overpass-API-Datenbank,
- ...

Befüllen der Datenquelle aus OpenStreetMap-Datendump.
Online-Updates mittels täglicher/stündlicher Diffs sind möglich.

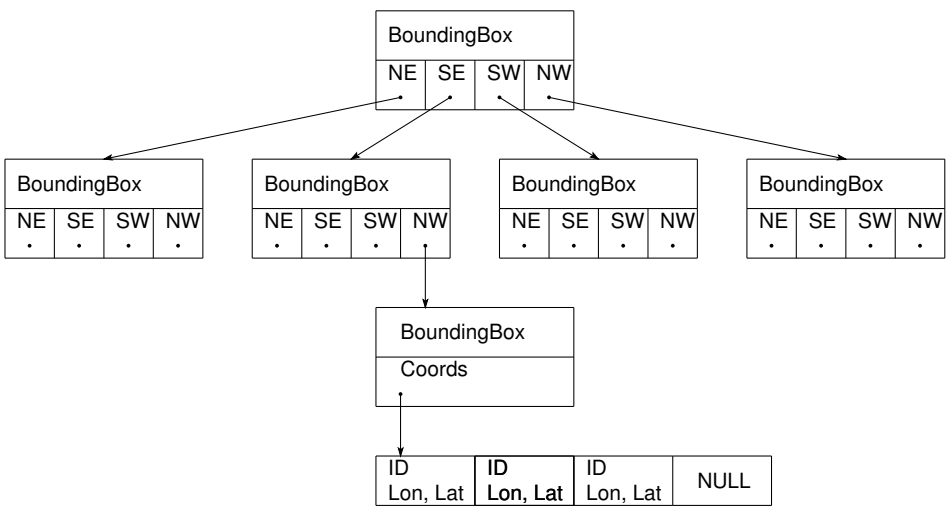
Aufwand (Europa-Dump):

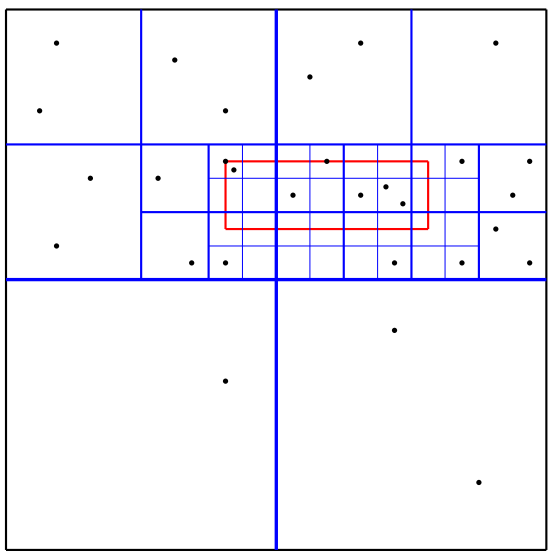
- PostgreSQL: mehrere Tage.
- Overpass-API: wenige Stunden.

Datenstruktur Overpass-API

Primäre Datenstruktur: *Quadtree*.

- Zerlege Gebiet rekursiv in 4 Rechtecke.
- Jeder Knoten fällt in eine der 4 Regionen.
- Befinden sich in Region zu viele Knoten, wird weiter unterteilt.
- Suche aller Knoten eines Gebietes in logarithmischer Zeit.





Quadtile-Index

Eindimensionaler Index für Quadtree:

- Umskalieren der Koordinaten auf ganzzahlige Werte (Multiplikation mit 10^7) $\Longrightarrow$ zwei 32-Bit-Zahlen
- Bitweises Verschränken von Höhen- und Breitenwert zu 64-Bit-Zahl:

$$x_{31}\ y_{31}\ x_{30}\ y_{30}\ \ldots\ x_1\ y_1\ x_0\ y_0$$

- Bitpaare von links nach rechts spezifizieren Kachel und deren rekursive Unterteilungen.
- Gebietsabfrage wird so zu einigen eindimensionalen Intervallabfragen im Quadtile-Index.

Abfrage Overpass-API 1

Interfaces:

- CGI-Script und zugehöriger Daemon, Kommunikation über shared memory
- Kommandozeilenprogramm
  Abfrage über `stdin`, Ergebnis auf `stdout`

Abfragesprachen:

- Overpass XML (ältere Variante)
- Overpass QL (kompakter, neuere Variante)

Datenformat Ergebnisse:

- OSM-XML
- JSON (neuere Variante)

Abfrage Overpass-API 2

Beispiel: Fast alle Objekte eines Gebietes abfragen.

Overpass Query Language

```
(                              # Vereinigung aller Ausgaben
  node($S, $W, $N, $E);        # Alle Knoten der Region nach _
  rel(bn)->.x;                 # Relationen, die Knoten enthalten,
                               # nach x, _ bleibt
  way(bn);                     # Wege, die Knoten enthalten, nach _
  rel(bw);                     # Relationen, die vorige Wege enthalten
);
out qt;                        # Ausgabe in Quadtile-Sortierung
```

Overpass XML

```
<osm-script>
  <union>
    <bbox-query w='$W' s='$S' e='$E' n='$N'/>
    <recurse type='node-relation' into='x'/>
    <recurse type='node-way'/>
    <recurse type='way-relation'/>
  </union>
  <print order='quadtile'/>
</osm-script>
```

Anmerkung: Die Relationen sind nicht vollständig abgefragt.

Parsen der OpenStreetMap-Daten

Umwandlung der angeforderten Daten in interne Datenstrukturen (Listen, Dictionaries) für Weiterverarbeitung.

- Datenformat: XML.
- Sehr flacher DOM-Baum.
- Beliebiger SAX-Parser verwendbar, derzeit wird `libexpat` verwendet.
- DOM-Parser aufgrund Datenmenge für große Karten indiskutabel!

Datenmenge und Speicherbedarf

OpenStreetMap-Statistik (Januar 2016).

- ca. 3 Mrd. Knoten weltweit
- ca. 1.6 Mrd. Knoten in Europa
- ca. 300 Mio. Wege weltweit

Knoten werden von anderen Objekten für Koordinaten referenziert
$\Longrightarrow$ schneller Zugriff erforderlich.

Speichern der Knoten 1

- Knoten sinnvoll im Hauptspeicher verwalten: Koordinaten für alle Zeichenoperationen benötigt
- Tags der Knoten nicht benötigt (bei Bedarf aus DB abfragen)
- Pro Knoten mindestens:
  - ID (64 Bit)
  - Länge (double)
  - Breite (double)

  Summe: 192 Bit = 24 Byte.
- Für Dictionary Verwaltungsdaten erforderlich:
  - Hash: ca. 50 Byte pro Knoten,
  - Binärbaum: 3 Pointer, Balanceinformation, ca. 30 Byte pro Knoten
- Bedarf: Europa mind. 80 GB.
- 64-Bit-Adressraum erforderlich!

Speicherbedarf verringerbar?

Speichern der Knoten 2

Speicherbedarf Koordinaten verringern:

$$-180° < \lambda < 180° \qquad -90° < \varphi < 90°$$

Werte mit 10^7 durchmultiplizieren und Nachkommastellen abschneiden:

- Wertebereich von `int32` wird nicht verlassen.
- Genauer als `float`.
- Genauigkeitsverlust $10^{-7°} < 12\,\text{cm}$.

Speicherplatz Koordinaten halbiert.

Quadtile-Index realisierbar (in Overpass-API genutzt).

Speichern der Knoten 3

Verringern der Verwaltungsinformation und implizites Speichern der ID: *Google Sparsetable*:

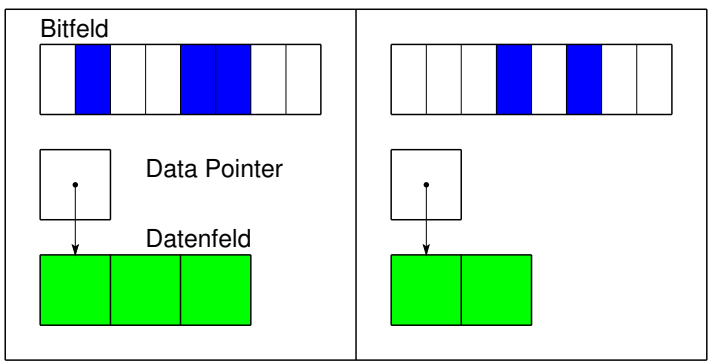
Knoten-ID ist gleichzeitig Feldindex.

Zeitaufwand Knotensuche: $O(1)$ wie Hash

Sparsetable kann vorberechnet und in Datei abgelegt werden.

Speicherbedarf (bei 48 Bit pro Block):

- Bitfeld bis zu maximaler ID (ca. 4 Mrd.): 512 MB
- je Block (84 Mio.) ein Pointer: 670 MB
- je Koordinate (tatsächlich benötigte Knotenzahl) 8 Byte
- Damit Speicherbedarf Europa (1,6 Mrd. Knoten): 13 GB



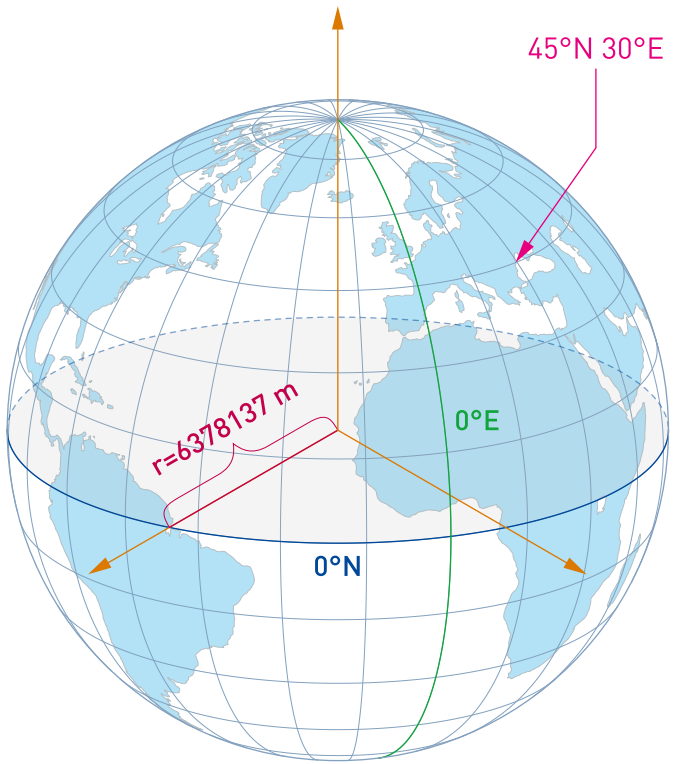
Weitere Datenstrukturen

Datenmenge überschaubar, wenn nur die für den aktuellen Zeichenschritt benötigte Daten geladen und anschließend freigegeben werden.

- Tags der Elemente (Key-Value-Paare): je ein *Dictionary* für Knoten, Wege, Relationen mit ID als Schlüssel und *Dictionary* oder *Paarliste* (Key-Value) als Wert.
- Knotenliste (IDs) jedes Weges: *List*, alle Wege: *Dictionary* der Weglisten mit Weg-ID als Schlüssel.
- Analog: Knoten-/Wegliste einer Relation.
- Rückwärtsreferenzen: zu welchen Wegen gehört ein Knoten? (z. B. für Zeichenrichtung Bahnhof).
  Derzeit nicht realisiert.
- Analog für Relationen.

Positionsangaben auf der Erde – Gradnetz

- Breitengrade (Latitude, φ): parallel zum Äquator
  Numerierung bis 90° nach Nord bzw. Süd
- Längengrade (Longitude, λ): Großkreise durch die Pole
  Numerierung bis 180° nach Ost bzw. West
  Längengrad 0: Greenwich
- Südliche Breiten und westliche Längen: negatives Vorzeichen
- Referenzsystem: WGS 84



Quelle: René Schwarz (CC-BY-NC-SA 3.0)

Probleme und Ziele

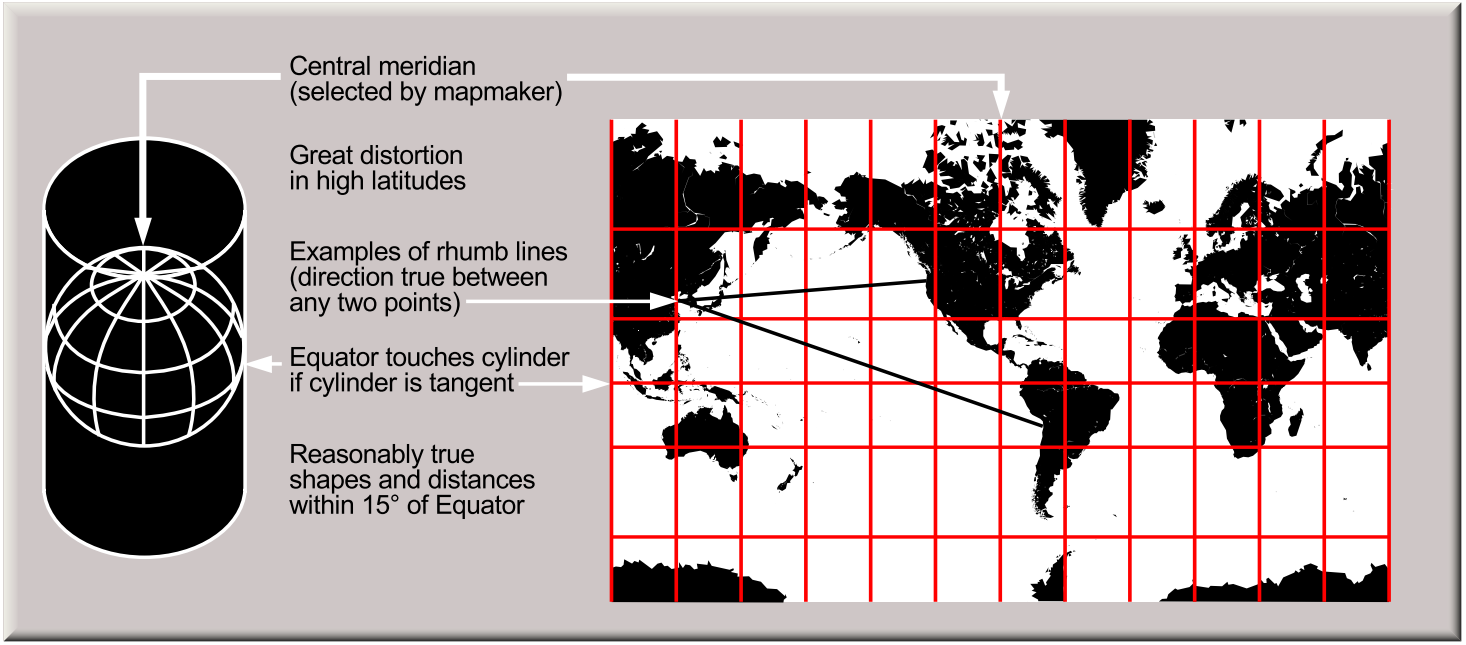
Betrachten im weiteren die Erde als ideale Kugel.

Probleme:

- Kugel lässt sich nicht in die Ebene abwickeln
- *Projektion* erforderlich, dabei möglichst kleine Fehler erzeugen
- Unterschiedliche Ziele nicht gleichzeitig realisierbar:
  - Richtungstreue für Navigation: Entfernungen verfälscht
  - Flächentreue: Richtungen, Krümmungen verfälscht

Beschränken uns auf jeweils eine Projektion pro Ziel.

Mercator-Projektion 1



Wikimedia Commons

- Zylinder in Nord-Süd-Richtung über die Erde stülpen
- Strahlen vom Erdmittelpunkt durch die Erdoberfläche auf Zylinder
- Zylinder aufschneiden und flach ausbreiten
- geeignet in Nord-Süd-Richtung strecken

Mercator-Projektion 2

Fläche verzerren:

- Himmelsrichtungen erhalten
- Form kleiner Gebiete erhalten

$$x = s \cdot \lambda$$

$$y = s \cdot \ln\left(\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right)$$

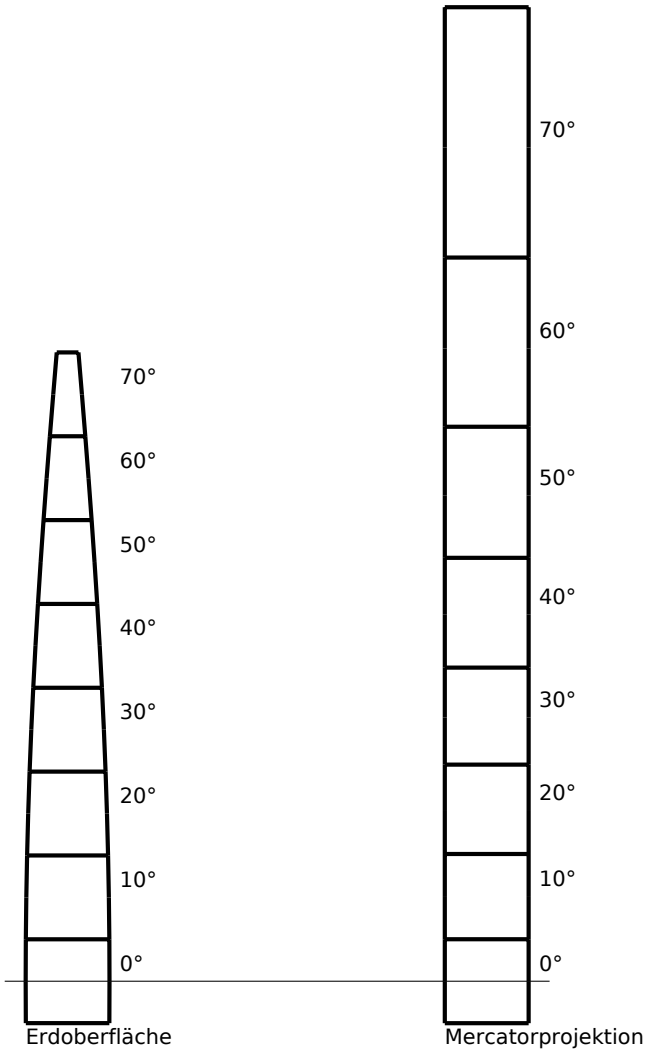
$$= s \cdot \ln\left(\tan\varphi + \frac{1}{\cos\varphi}\right)$$

λ – Längengrad in Radiant

φ – Breitengrad in Radiant

s – Skalierungsfaktor

$$\text{Radiant} = \frac{\pi}{180^0} \cdot \text{Grad}$$



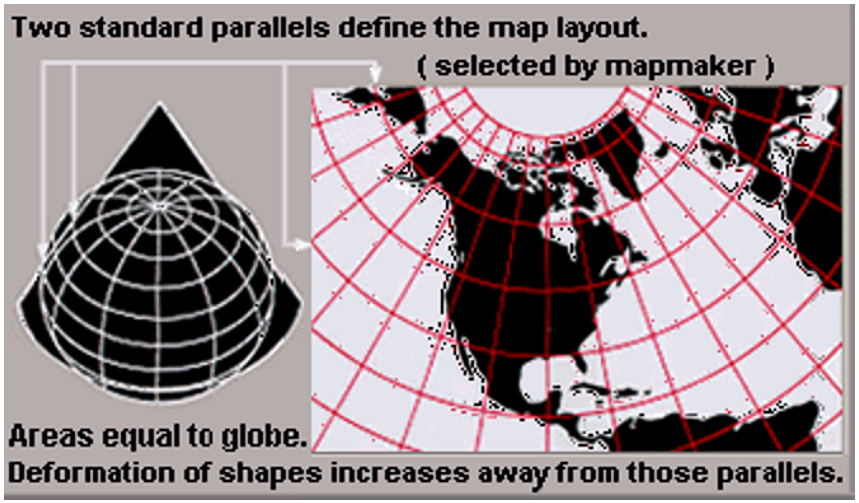
Mercatorprojektion – Eigenschaften

- winkel- und richtungstreu
- Form und Lage kleiner Regionen bleibt erhalten
- nicht flächentreu
- Maßstab vom Breitengrad abhängig
- Darstellung nur bis zum 82-ten Breitengrad sinnvoll möglich. Aber:
  - Anchorage: 61°N
  - Nordkapp: 71°N
  - Longyearbyen (Spitzbergen): 78°N
  - Navarino (Feuerland): 55°S

Gut für kleinere Karten (Wanderkarten, Stadtpläne) geeignet.

Albers-Projektion 1

Genauer: Albers' flächentreue Kegelprojektion mit zwei längentreuen Parallelkreisen.



Wikimedia Commons

- Kegel überstülpen, der zwei definierte Breitengrade schneidet.
- Kegel verzerren, so dass:
  Breitenkreise auf Kegel und Kugel gleiche Länge haben,
  durch Breitenkreise begrenzte Flächen auf Kegel und Kugel gleich sind.
- Kegel an Kante aufschneiden und abrollen.

Albers-Projektion 2

Umrechnung: λ_0, φ_0 sollen der Mittelpunkt in der Projektion werden, φ_1 und φ_2 sind die beiden Bezugsbreitenkreise ($\varphi_1 < \varphi_2$).

$$x = r \cdot \rho \cdot \sin\theta, \qquad y = r \cdot (\rho_0 - \rho\cos\theta)$$

mit

$$n = \frac{\sin\varphi_1 + \sin\varphi_2}{2}$$
$$\theta = n(\lambda - \lambda_0)$$
$$C = \cos^2\varphi_1 + 2n\sin\varphi_1$$
$$\rho = \frac{\sqrt{C - 2n\sin\varphi}}{n}$$
$$\rho_0 = \frac{\sqrt{C - 2n\sin\varphi_0}}{n}$$

Es muss gelten: $\varphi_1 \neq -\varphi_2$ (sonst entsteht kein Kegel)!

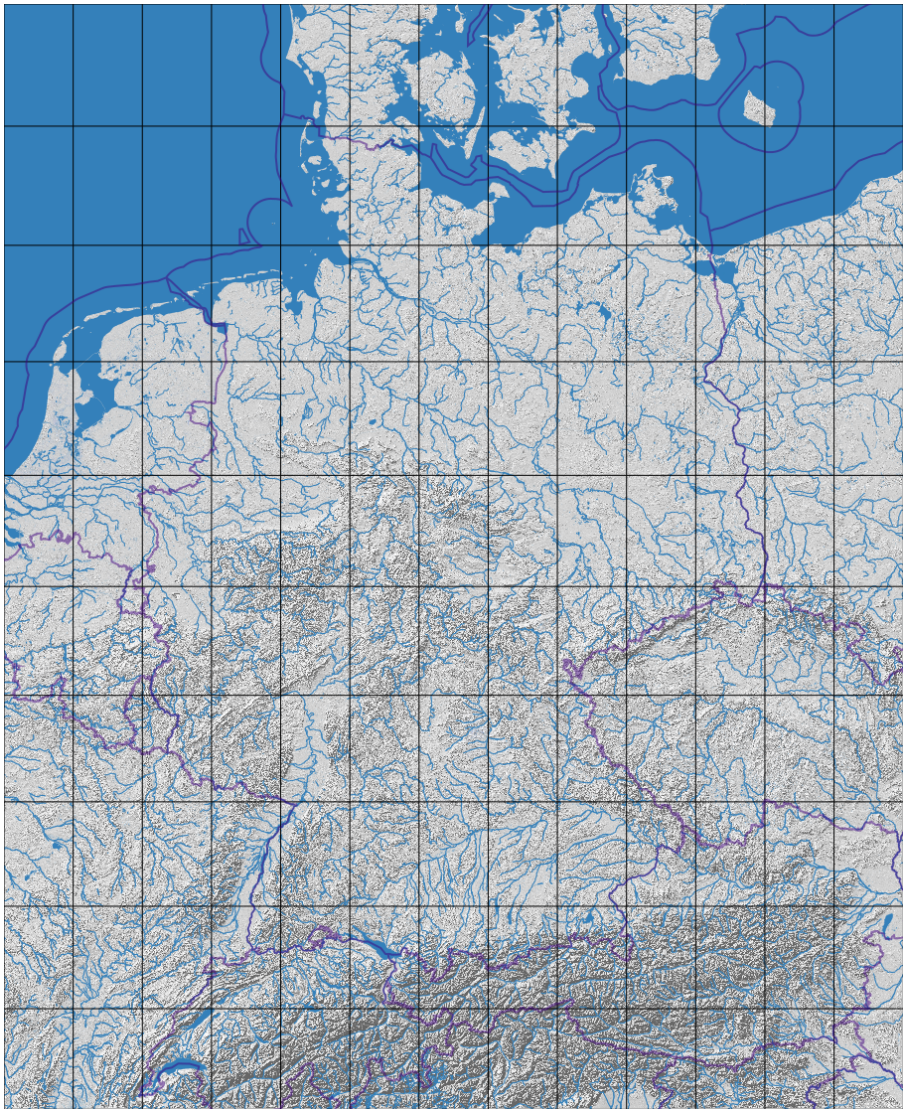
Herleitung: [12]

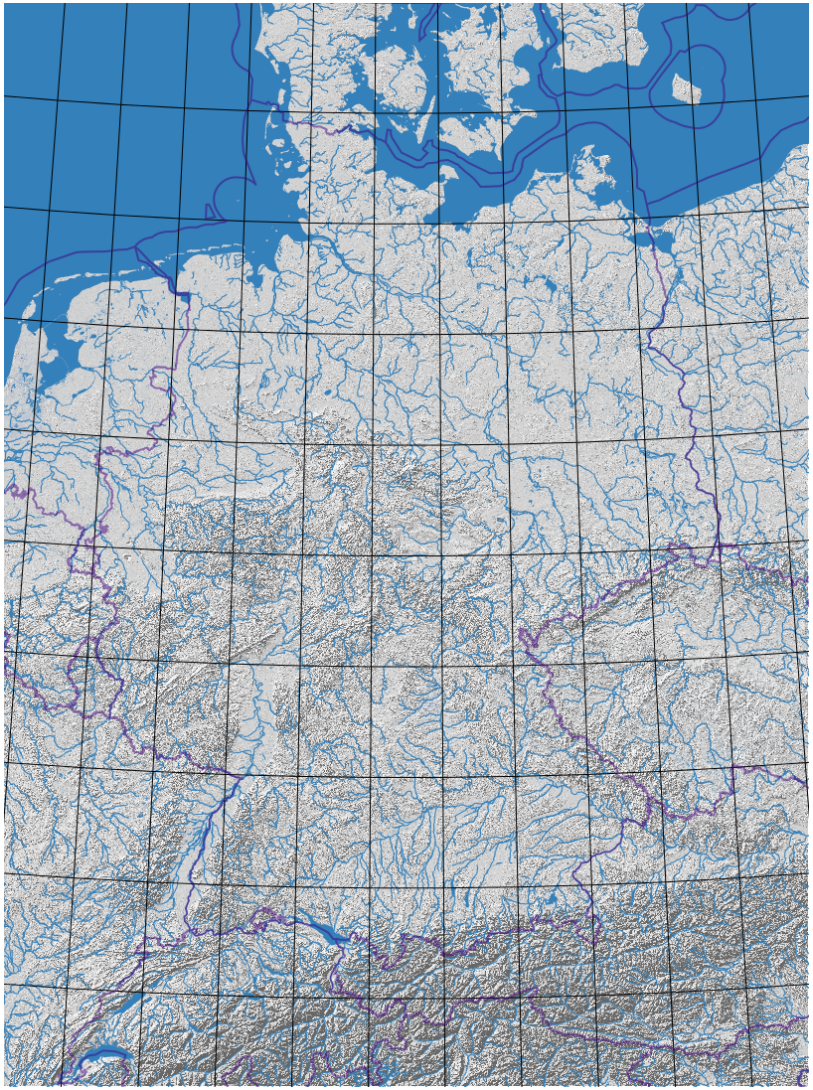
Eigenschaften der Albers-Projektion

- Für großflächige Karten, auch in Polnähe geeignet.
- Flächeninhalte bleiben gleich.
- Meridiane (Längengradlinien) sind Geraden, haben zueinander gleiche Abstände, schneiden sich in Kegelspitze.
- Breitenkreise sind Kreisbögen, senkrecht zu den Meridianen, parallel zueinander, unterschiedliche Abstände.
- In der Nähe der Bezugskreise nahezu längentreu.
- Weit außerhalb der Bezugskreise starke Formverzerrungen.

Wahl der Bezugskreise: Nähe des unteren und oberen gewünschten Kartenrandes sinnvoll.

Vergleich Mercator- vs. Albersprojektion





Koordinatentransformation 1

Zwei Schritte:

1. Umrechnung WGS84 in gewünschte Projektion
2. Umrechnung der Projektionsdaten in Koordinate der Zeichenfläche (Maßstab)

Zeichenfläche ist meist linksorientiertes Koordinatensystem, Ursprung in der linken, oberen Ecke.

Koordinatentransformation 2

Projektionsbereich (Mercator):

$$[-\pi, -\pi] \times [\pi, \pi]$$

(y-Wertebereich vergrößert)

Darzustellendes Gebiet:

$$[x_0, y_0] \times [x_1, y_1]$$

Bildkoordinaten x', y', s gemäß gewünschter Bildgröße (Maßstab):

$$x' = (x - x_0) \cdot s, \quad y' = (y_0 - y) \cdot s$$

Kartengröße in Pixel:

$$w' = s \cdot (x_1 - x_0), \quad h' = s \cdot (y_1 - y_0)$$

Skalierung s ergibt sich aus gewünschtem Maßstab.

Bestimmen Skalierung

Maßstab m in Pixel je Kilometer wird festgelegt.

Bestimmen den Abstand zwischen zwei Punkten p_1, p_2, die etwa in die Kartenmitte fallen, sowohl auf der Erde (d_e) als auch in der Projektion (d_p).

Skalierung:

$$s = m \cdot \frac{d_e}{d_p}$$

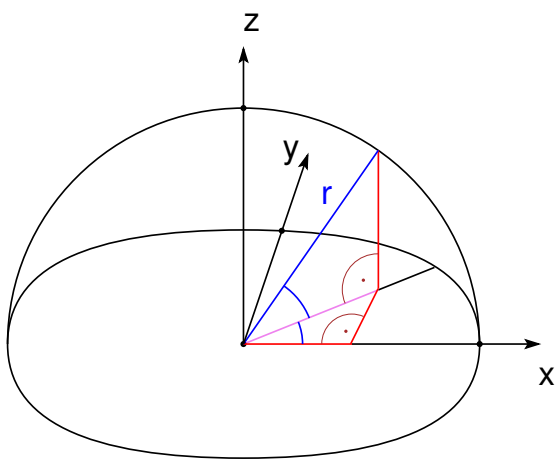
Projektionsebene: Euklidischer Abstand.

$$d_p = \sqrt{(x_2' - x_1')^2 + (y_2' - y_1')^2}$$

Abstand auf der Erde

Abstand zwischen zwei Punkten in WGS84-Koordinaten.
Umrechnung in kartesische Koordinaten:

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r \cdot \cos\varphi \cos\lambda \\ r \cdot \cos\varphi \sin\lambda \\ r \cdot \sin\varphi \end{pmatrix}$$



Skalarprodukt:

$$\begin{aligned} \langle \vec{p}_1, \vec{p}_2 \rangle &= x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 \\ &= \|\vec{p}_1\| \cdot \|\vec{p}_2\| \cdot \cos \angle(\vec{p}_1, \vec{p}_2) \\ &= r^2 \cdot \cos \angle(\vec{p}_1, \vec{p}_2) \end{aligned}$$

Abstand:

$$d_e = r \cdot \angle(\vec{p}_1, \vec{p}_2) = r \cdot \arccos\left(\frac{\langle \vec{p}_1, \vec{p}_2 \rangle}{r^2}\right)$$

Graphikbibliothek

Anforderungen:

- Unterstützung des benötigten Ausgabeformats (PNG, SVG, PDF, ...)
- Zeichnen der benötigten Formen, insbesondere *Multipolygone*.
- Unicode-Font-Unterstützung.
- Zeichnen von Icons aus Dateien.
- Kompositionsoperatoren.
- Gedrehter Text, Text entlang von Pfaden.

Beliebte Bibliotheken:

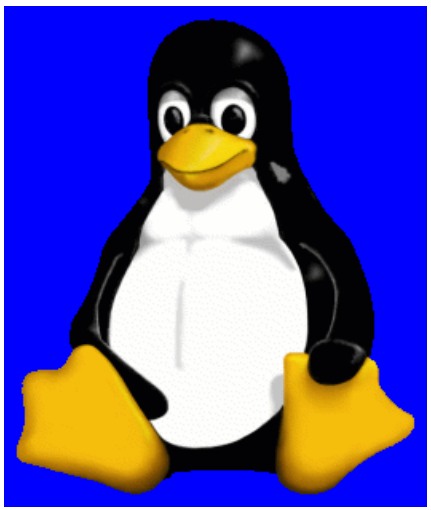
- AGG (C++, Nur Basisfunktionalität, von Mapnik genutzt)
- Cairo (C) [1], Schnittstellen zu Python, Lua, ...
- QT-Graphikbibliothek (C++, sehr schnell)

Anmerkung: Bei SVG-Ausgabe wird keinerlei Graphikbibliothek benötigt $\Longrightarrow$ Darstellung durch SVG-Renderer.

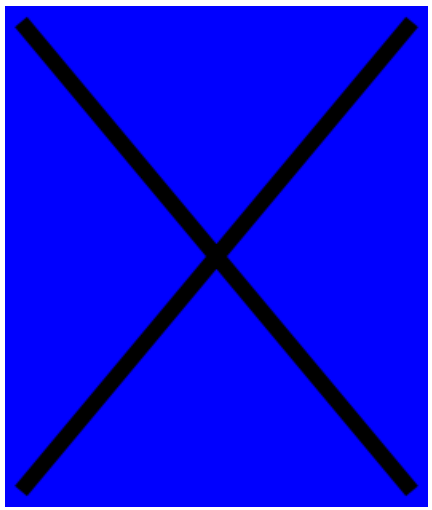
Cairo-Zeichenprinzip

Zeichenoperation besteht aus:

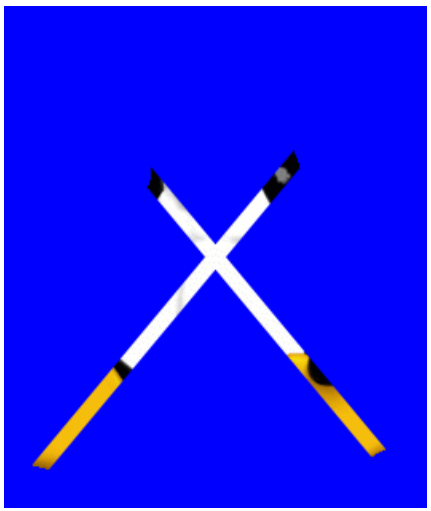
- Vorlage (Source): Farbe, Transparenz, Farbverlauf, Bild, Muster in Bildgröße.
- Maske (Pfad), aus der gewünschte Figur ausgestanzt wird.
- Vorlage wird durch die Maske auf das Bild übertragen.
- Von der Maske verdeckter Bildbereich bleibt unverändert.
- Neue Fläche überdeckt (`OPERATOR_OVER`) oder entfernt (`OPERATOR_CLEAR`) vorherige Fläche.



Source



Pfad



Zeichenergebnis

Zeichenreihenfolge

In Ebenen (einzelnen Bildern) zeichnen, diese überlagern (wegen Composite-Operationen).

Reihenfolge:

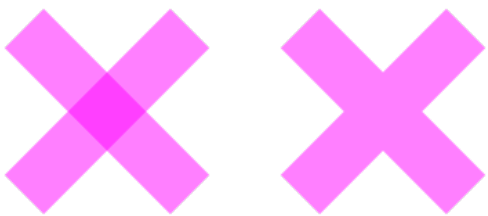
1. Flächen mit Höhenschattierung (Wälder, Wiesen, Parks)
2. Höhenschattierung, -linien
3. Ozeane (müssen fehlerhafte Höhen überdecken)
4. Wasserflächen (eben $\Longrightarrow$ keine Höhenschattierung)
5. Wege in Ebenen (normalerweise über Wasser)
6. Grenzen, Hochspannungsleitungen, Routen, Tracks
7. Symbole und Beschriftungen

Reihenfolge sollte konfigurierbar sein.

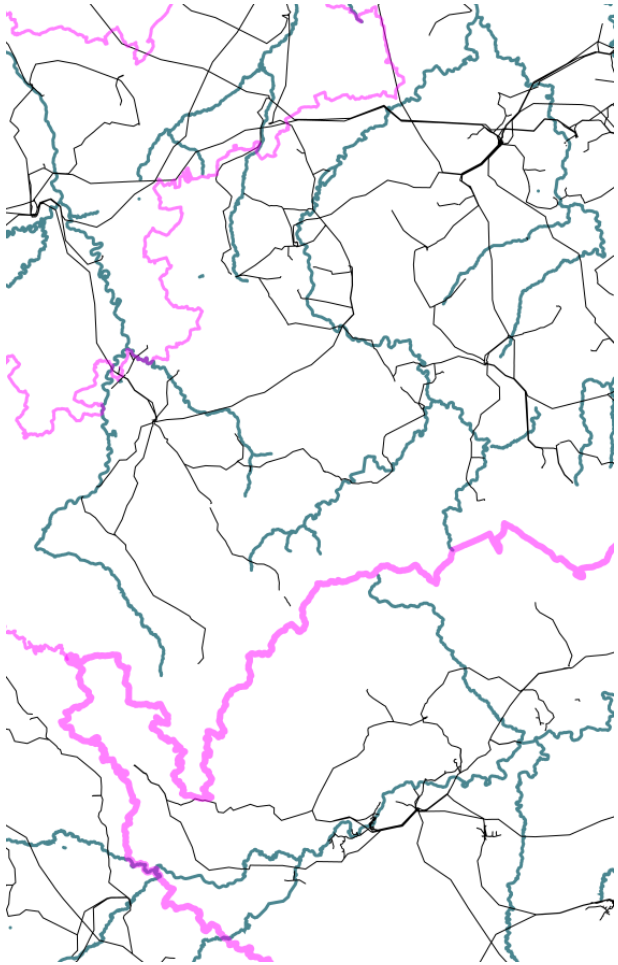
Einfache Wege

Grenzen, Flussläufe, Hochspannungsleitungen, Tracks, ...

- Linie, Strichlinie der gewünschten Breite.
- Kleine Maßstäben: Knoten fallen fast zusammen, Linienverdickungen $\Longrightarrow$ ausdünnen.
- Halbtransparente Linien: Darunter liegende Objekte bleiben sichtbar: Bei Überschneidungen dunkler: als gemeinsamen Pfad mit einer Zeichenoperation zeichnen.



- GPX-Tracks ebenso behandeln.
- Symbole (Pfeile) enlang von Wegen? (gleicher Abstand, korrekte Richtung)



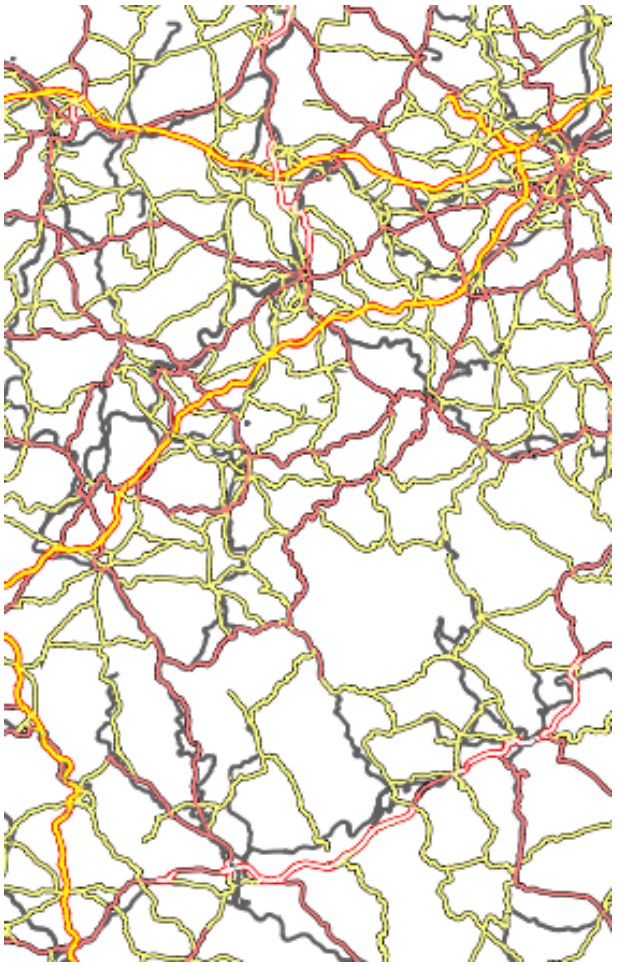
50° N 12° E – 51° N 13° E
Flüsse, Grenzen, Hochspannungsleitungen

Zusammengesetzte Wege

Besonderheiten gegenüber einfachen Linien:

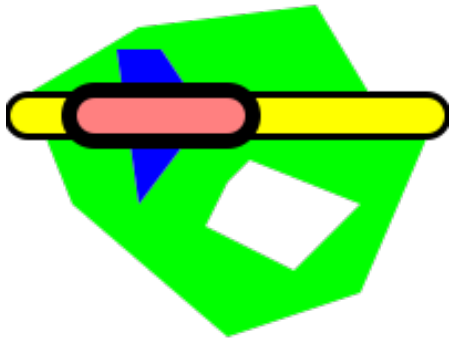
- Rand (*Casing*) und Füllung (*Core*) übereinander zeichnen.
- Kreuzung oder Brücke $\Longrightarrow$ Tag `layer` gleich oder verschieden.
- Kreuzung: Straße höherer Priorität durchzeichnen:
  Zuerst *beide* Casings, dann Cores in gewünscher Reihenfolge.

- Bahnübergang: Casing Bahn über Core der Straße.

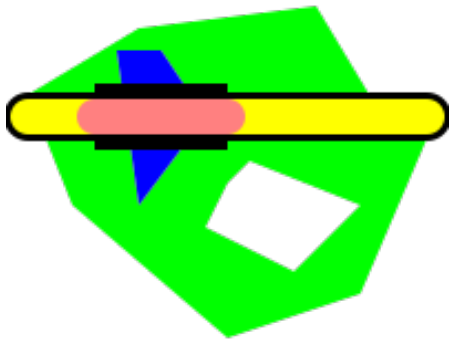


Brücken und Tunnel

Brücken: höherer Layer $\Longrightarrow$ Casing wird später gezeichnet
Problem: Linienenden.

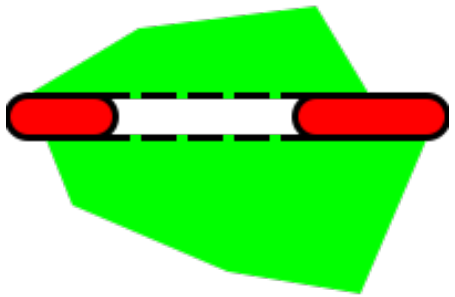


Abhilfe: Casing mit flachen (`LINE_CAP_BUTT`), Core mit runden Linienenden

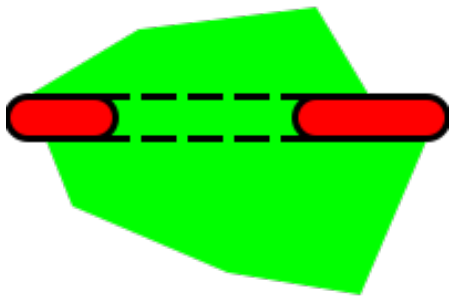


Casing breiter als sonstige Straße, Core gleiche Breite und Farbe

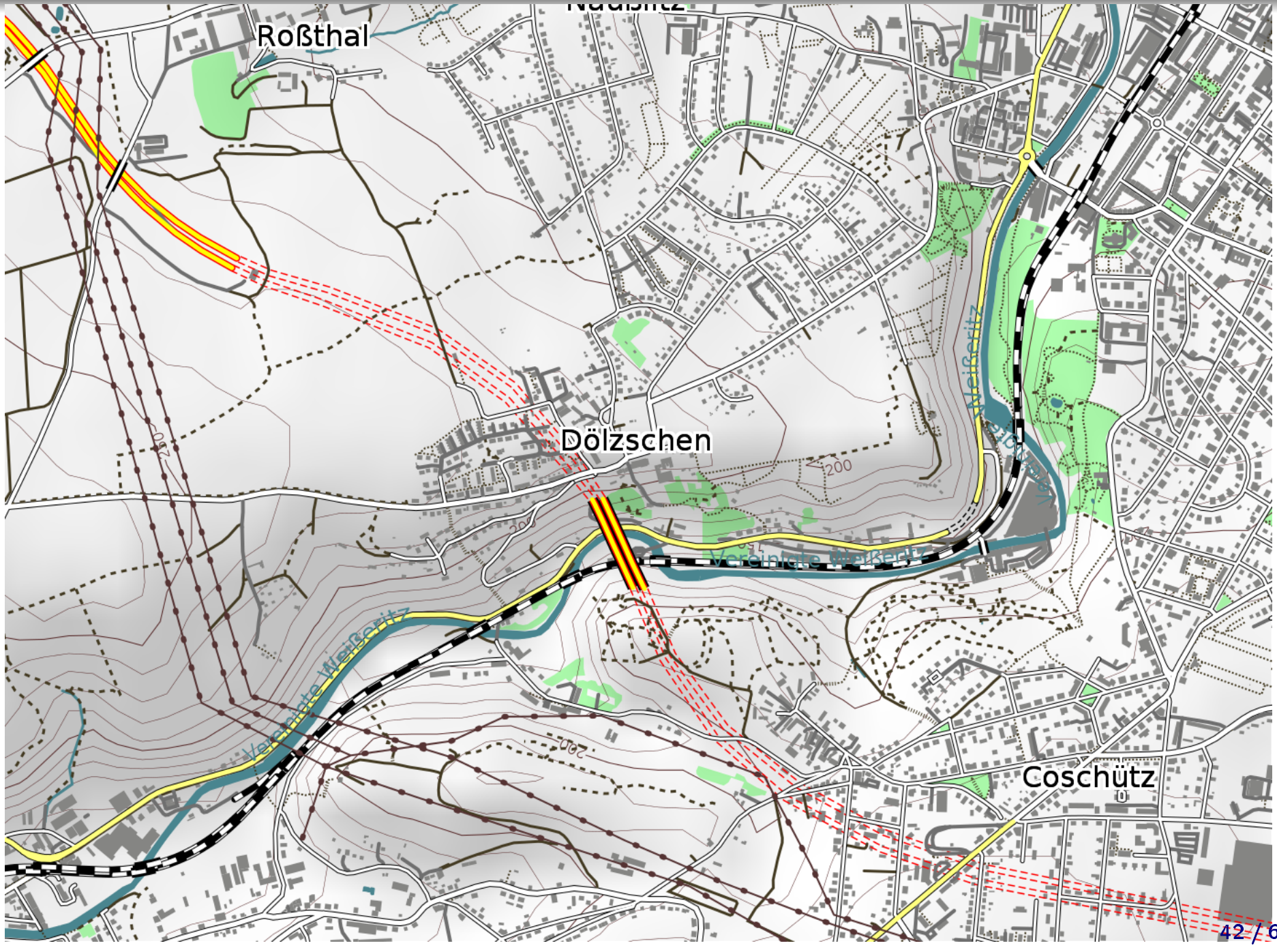
Tunnel: niedrigerer Layer, werden vorher gezeichnet.
Casing zeichnen, Core entfernen (`OPERATOR_CLEAR`).
Für Casing Strichlinie verwenden.
Achtung: Operator entfernt alle vorher gezeichneten Daten:



Abhilfe: Eigenes Teilbild für Tunnel-Layer, anschließend über Flächen legen.

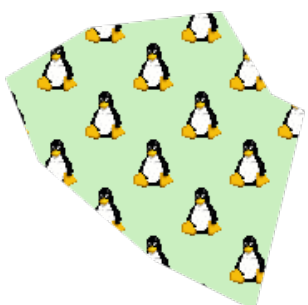


Beispiel: Brücken und Tunnel

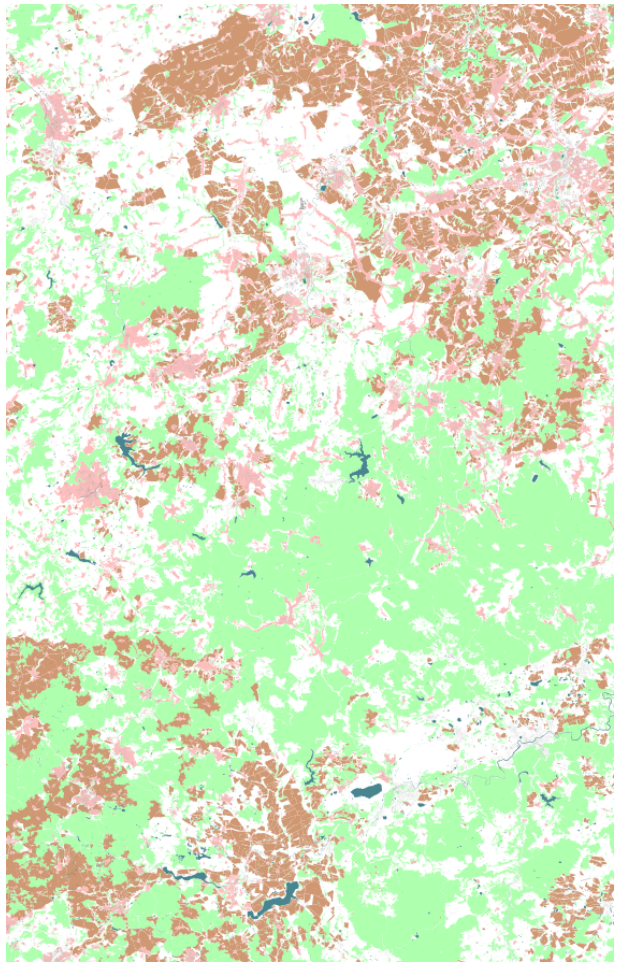


Einfache Flächen

- Im OSM-Datenmodell geschlossener Linienzug.
- Tags definieren Fläche statt Rundweg (`landuse: residential`, `area: yes`, `natural: wood`, ...)
- Zeichnen als Polygon: letzten Punkt entfernen und Pfad schließen.
- Statt Rand zeichnen (`stroke`) Fläche füllen (`fill`).
- Vorlage (`source`) ist Farbe oder Muster.



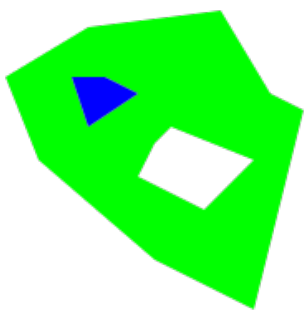
Tux-Friedhof



Wälder, Agrar- Wasser- und Wohnflächen

Multipolygone

- Polygone mit Aussparungen (Wald mit See, Lichtung) oder Rand aus mehreren Wegen zusammengesetzt ($>$ 2000 Punkte)
- Darstellung: Relation (`type: multipolygon`) aus Wegen (`role: outer`, `role: inner`).
- Art des Polygons (Wald, See, ...) durch Tags der Relation festgelegt.
- Innere und äußere Ränder geschlossen, überschneidungsfrei.
- Reihenfolge und Richtung der Wege beliebig.



Multipolygon mit innerem ausgezeichneten Polygon

Zeichnen von Multipolygonen

Cairo:

- Zeichnen und Schließen (`close_path()`) der äußeren Ränder.
- Sequentielles Zeichnen und Schließen der inneren Ränder.
- Füllen des Pfades.

Alle äußeren Ränder müssen die gleiche, alle inneren Ränder die entgegengesetzte Orientierung haben.

Zeichenalgorithmus:

- Zusammensetzen der äußeren Ränder:
  Nicht verarbeiteter Weg startet neuen Rand
  Suchen/Anhängen des nächsten Weges, bis Pfad geschlossen ist.
- Orientierung des Polygons bestimmen, eventuell umdrehen.
- Verketten der inneren Ränder und entgegengesetzt orientieren.
- Pfad aus den einzelnen Polygonen zusammensetzen und füllen.

Hinweis: Bei großen Polygonen Sortieren der Endpunkte der Wege, beschleunigt Zusammensetzen (binäre Suche des Folgeweges).

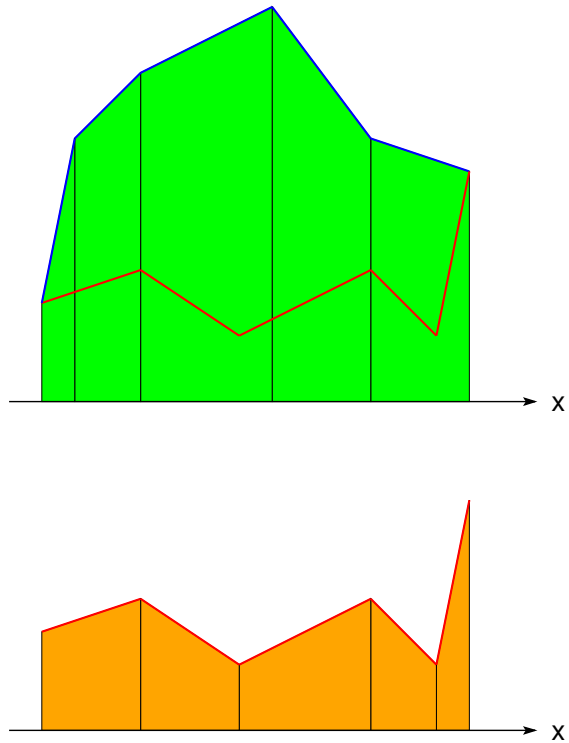
Orientierung von Flächen

Idee: Flächenintegral mit Trapezregel:

$$A_{\text{orient}} = -\frac{1}{2}\sum_{i=0}^{n-1}(y_{i\oplus 1} + y_i)\cdot(x_{i\oplus 1} - x_i)$$

mit $i \oplus 1 = i + 1 \mod n$

Vorzeichen der Fläche gibt die Orientierung an.



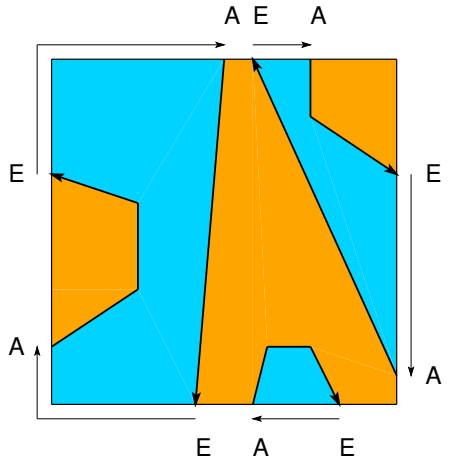
Küstenlinien

- Keine Relation, sondern Einzelwege
  Tag: `natural: coastline`
- Landseite stets in Normalenrichtung (links bzgl. Linienrichtung).
- Mehrere Linienzüge, innerhalb der Karte geschlossen bzw. Schnitt mit Kartenrand

Problem: Polygone müssen geeignet geschlossen werden, um Wasserflächen zu füllen.

Schließen der Küstenlinien

- Schnittpunkte Küstenline mit Rand:
  Entlang des Randes wechseln sich Anfangs- und Endpunkte ab.
- Randabschnitt von End- zu Anfangspunkt gehört zu Wasserfläche.



Algorithmus:

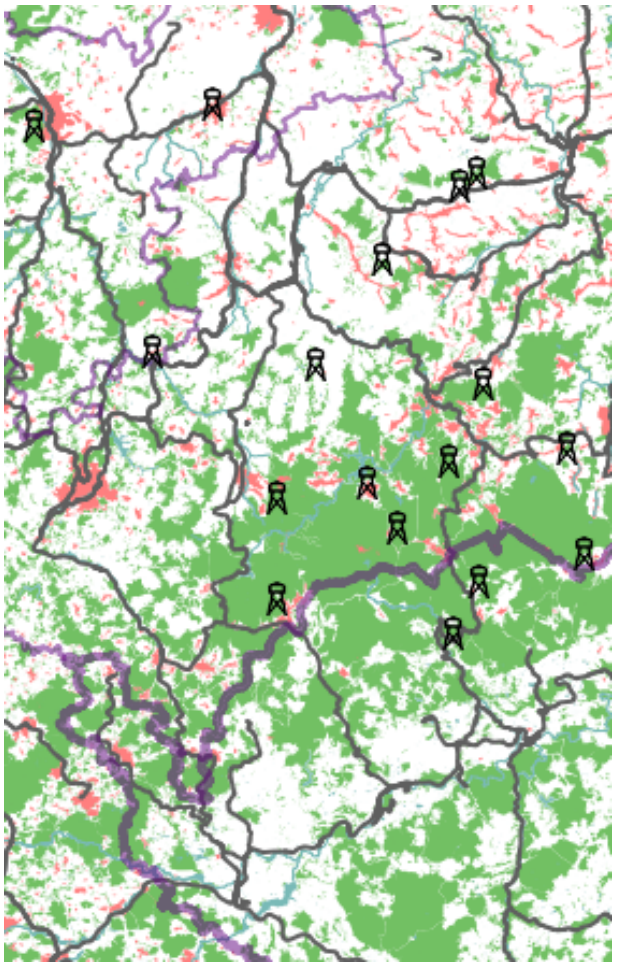
- Zusammensetzen der Küstenlinien aus Einzelwegen
- Geschlossener Weg: Insel.
- Offener Weg: Schnittpunkte mit Kartenrand bestimmen.
  Punkte entlang Kartenrand im Urzeigersinn sortieren (über Winkel zur Kartenmitte)
  Benachbarte End-/Anfangspunkte verbinden $\Longrightarrow$ äußere Polygone.
- Füllen der äußeren Polygone mit Wasserfarbe.
- Herausschneiden der Inseln (`OPERATOR_CLEAR`)
  alternativ: als Multipolygone zeichnen.

POIs

- Markierung von Berggipfeln, Aussichtspunkten, Bahnhöfen, . . . durch Symbole
- Auch Flächen können mit Symbolen versehen werden (Spielplatz, Firmengelände, Campingplatz)
  Positionieren auf Schwerpunkt:

$$\vec{m} = \frac{1}{n} \sum_{i=0}^{n-1} \vec{p}_i$$

  Problem: Schwerpunkt außerhalb der Fläche, bessere Heuristik z. B. *Medial Axes, Straight Skeleton*
- Symbol als eingelesenes Rasterbild, SVG wird von Cairo nicht direkt unterstützt.
- Als Pfade im Renderer hart codierte Symbole (Stern, Dreieck), skalierbar.



Beschriftungen

Zwei Probleme:

Zu wenig Kontrast:

Herausstellen des Textes durch Rand mit kontrastreicher Farbe:

- Text in Pfad umwandeln, Rand mit Hintergrund-, Fläche mit Vordergrundfarbe.
- Text in jeder Richtung 1 Pixel verschoben mit Hintergrund-, in Zielposition mit Vordergrundfarbe.

Textüberdeckung:

- Berechnen der *Bounding Box* des Textes und speichern.
- Nächsten Text nur bei genügend Abstand zu bisherigen Boxen ausgeben.

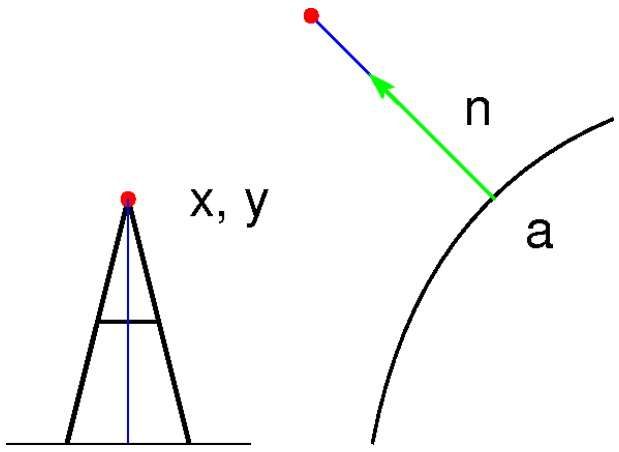
Wichtigste Texte zuerst ausgeben.

Beschriften von Pfaden 1 (Staßen, Flussläufe)

Theoretisches Vorgehen:

- Kurve in *natürliche Parametrisierung* überführen: $a \mapsto P(a) = (p_x(a), p_y(a))$ so, dass Kurvenlänge $P(0)$ bis $P(a)$ gerade a ist.
- Normalenvektoren $\vec{n}(a)$ der Kurve bestimmen.
- Text in Pfad umwandeln.
- Bestimme Parameter Textanfang $a_0 = \frac{1}{2}$ (Kurvenlänge − Textlänge).
- Pfadkoordinaten (x, y) transformieren:

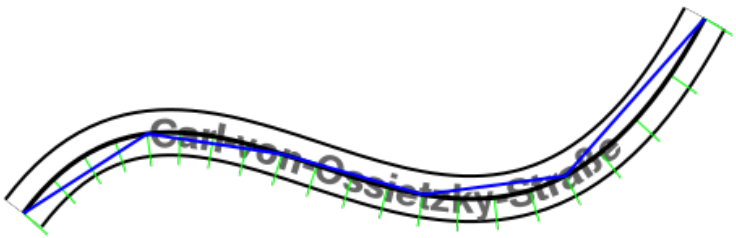
$$a = a_0 + x$$
$$x' = p_x(a) + y \cdot n_x(a)$$
$$y' = p_y(a) + y \cdot n_y(a)$$



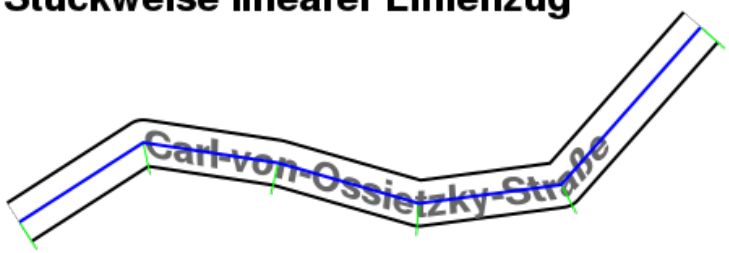
Beschriften von Pfaden 2

Praktisches Vorgehen:

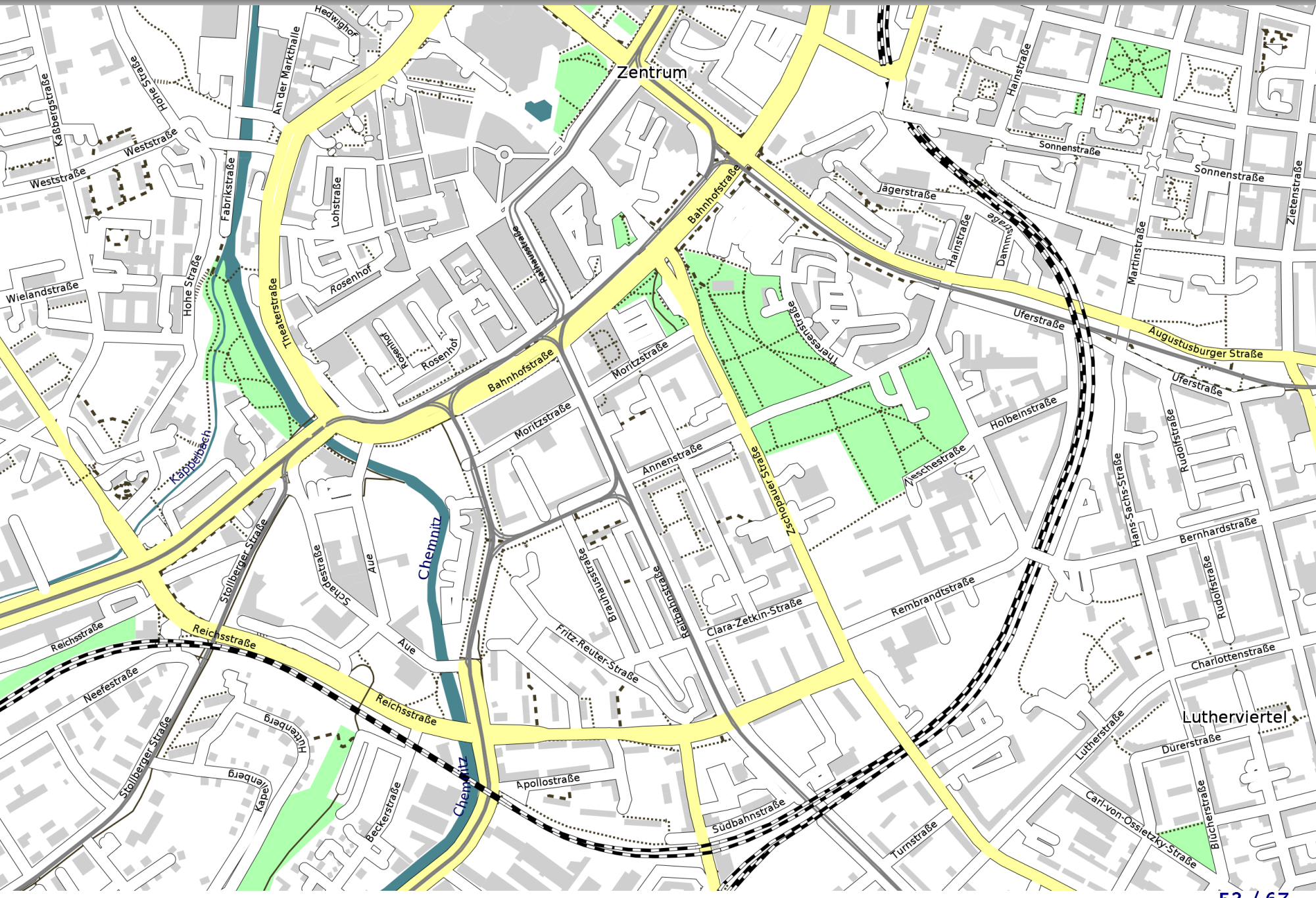
- Natürliche Parametrisierung kann nur approximiert werden $\Longrightarrow$ Kurve linearisieren.
- Oder: Kurve liegt bereits als Linienzug vor.
- Normalen in den Knickpunkten: Mittelwert der Normalen der angrenzenden Linien
- Stetige Veränderung der Normalen über Segmentlänge
- Richtung des Weges evt. umdrehen, damit Text aufrecht steht.



Stückweise linearer Linienzug



Beispiel: Beschriftungen

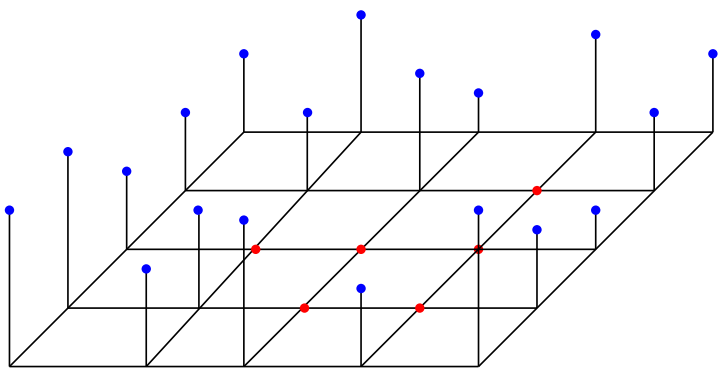


SRTM-Daten

- Vermessung der Erdöberfläche aus dem Weltraum durch die NASA im Jahr 2000
- Space Shuttle Endeavour, STS-99
- Kartierung der Höhendaten zwischen 60°N und 58°S
- Rasterabstand $\frac{1}{1200}^\circ = 3$ Bogensekunden (ca. 70×90 m²)
- Public Domain
- `http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/`
- Kacheln 1°×1° mit 1201×1201 Datenpunkten (ganzzahlig, 16 Bit)
- Kacheln überschneiden sich 1 Pixel.

Probleme mit SRTM-Daten

- Fehlende Datenpunkte (durch Wert -32768 gekennzeichnet). Kommerzielle Korrekturdaten verfügbar, aber Lizenz verbietet oft freie Nutzung.
- Daten nördlich des 60. Breitengrades fehlen. `http://www.viewfinderpanoramas.org/dem3.html` (Lizenz unklar).
- Meere haben keine einheitliche Höhe von 0 Metern.

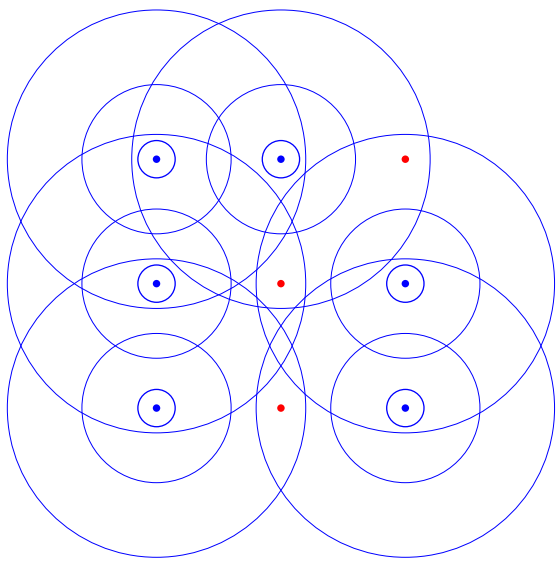


Interpolation der fehlenden Höhenpunkte

Radial Basis Functions [6]

$\vec{x}_1 \ldots \vec{x}_n$ sind Punkte mit vorhandenen Höhen $h(\vec{x}_i)$, definieren Abstandsfunktion $\Phi(r)$.
Interpolationsfunktion:

$$h(\vec{x}) = \sum_{i=1}^{n} \lambda_i \cdot \Phi(\|\vec{x} - \vec{x}_i\|)$$



Einsetzen bekannter Punkte $\vec{x}_i$ und Höhen $h(\vec{x}_i)$
$\Longrightarrow$ LGS mit n Unbekannten λ_i.
Bei geeigneten Funktionen $\Phi(r)$ immer lösbar.

$\Phi(r) = r$ (linear), $\quad \Phi(r) = e^{-\alpha r^2}$, $\alpha > 0$ (Gauß)

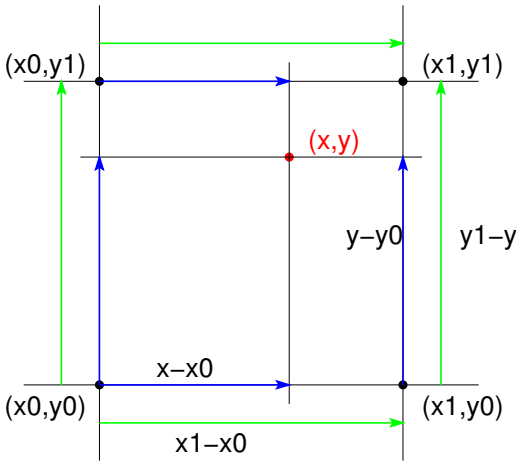
LGS für ganze Höhenkachel hat ca. 1 Mio. Unbekannte $\Longrightarrow$ Interpolation für gleitendes Fenster (z. B. 5×5 Punkte).

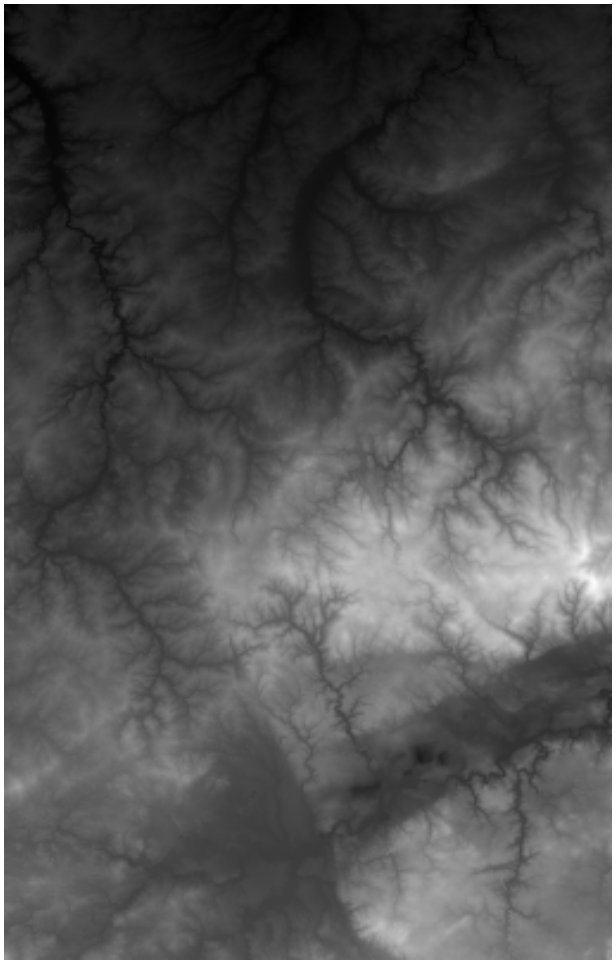
Höhenapproximation

Höhenangaben nur auf Raster, Werte an beliebigen Koordinaten benötigt.

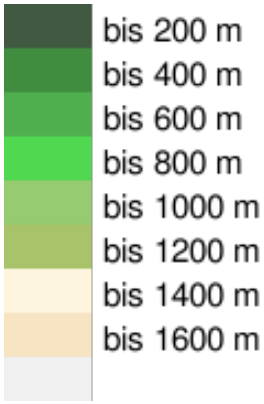
- Radial Basis Functions sind nutzbar.
- Hier Bilinearform verwendet.

$$t_x = \frac{x - x_0}{x_1 - x_0}, \quad t_y = \frac{y - y_0}{y_1 - y_0}$$
$$h_0 = (1 - t_x) \cdot h(x_0, y_0) + t_x \cdot h(x_1, y_0)$$
$$h_1 = (1 - t_x) \cdot h(x_0, y_1) + t_x \cdot h(x_1, y_1)$$
$$h(x, y) = (1 - t_y) \cdot h_0 + t_y \cdot h_1$$

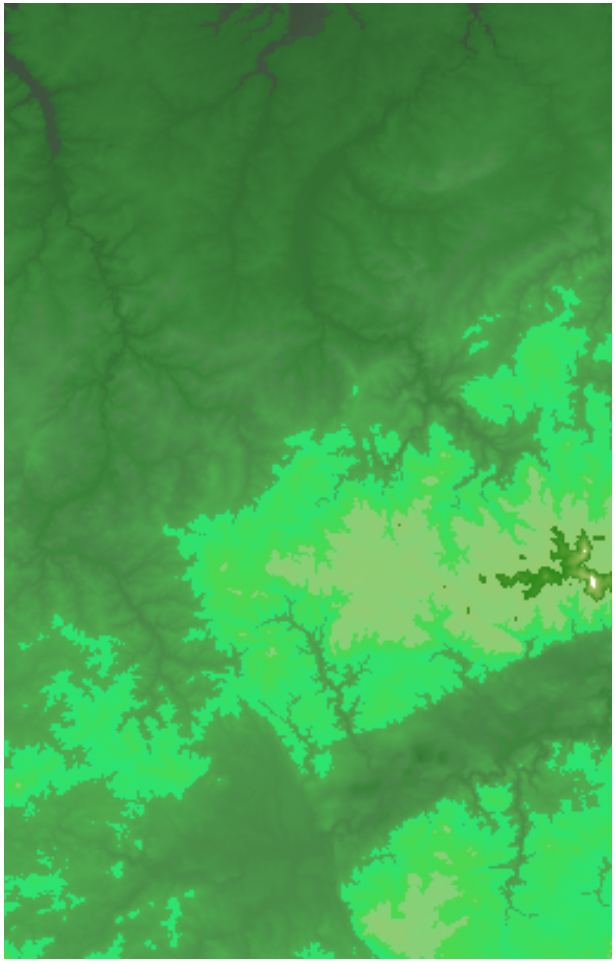




Höhenfärbung



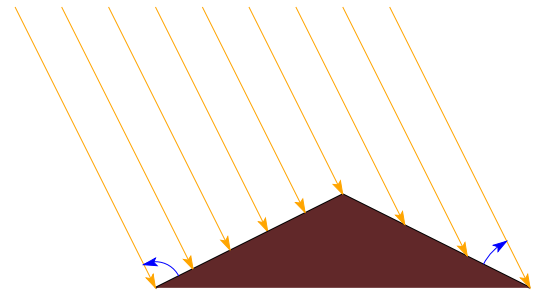
- Ordnen jedem Höhenintervall Farbwert zu.
- Färben Kartenpixel passend ein.
- Interpolation (besser im HSV-Farbmodell) verbessert Ergebnis.
- Mangels Bedarf nicht weiter untersucht.



Schattierung

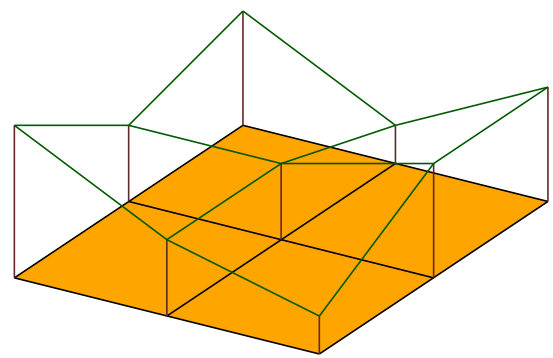
Denken uns weit entfernte Lichtquelle und betrachten Lichtmenge, die auf eine Fläche fällt.

- Abhängig von Neigung gegen Lichtquelle.
- Benötigen Richtung der Fläche: Flächennormale:



$$\vec{m}_x = \begin{pmatrix} 1 \\ 0 \\ \frac{\partial z}{\partial x} \end{pmatrix}, \quad \vec{m}_y = \begin{pmatrix} 0 \\ 1 \\ \frac{\partial z}{\partial y} \end{pmatrix}$$

$$\vec{n} = \frac{\vec{m}_x \times \vec{m}_y}{\|\vec{m}_x \times \vec{m}_y\|}$$

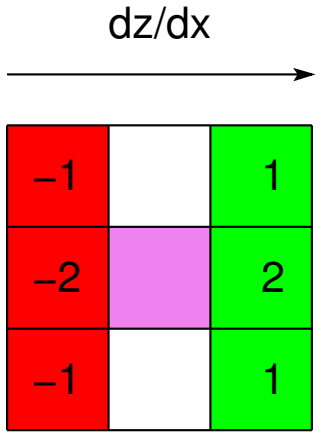


Berechnung der partiellen Ableitungen

Näherung nach Sobel [2] (Rasterabstand = 1):

$$\frac{\Delta z}{\Delta x}(x_i, y_i) = \frac{1}{8} \cdot \Big(z(x_{i+1}, y_{i+1}) - z(x_{i-1}, y_{i+1}) + 2 \cdot \big(z(x_{i+1}, y_i) - z(x_{i-1}, y_i) \big) + z(x_{i+1}, y_{i-1}) - z(x_{i-1}, y_{i-1}) \Big)$$

$$\frac{\Delta z}{\Delta y}(x_i, y_i) = \frac{1}{8} \cdot \Big(z(x_{i+1}, y_{i+1}) - z(x_{i+1}, y_{i-1}) + 2 \cdot \big(z(x_i, y_{i+1}) - z(x_i, y_{i-1}) \big) + z(x_{i-1}, y_{i+1}) - z(x_{i-1}, y_{i-1}) \Big)$$



Achtung: Flächen- und Höhenkoordinaten sind unterschiedlich skaliert (Bogensekungen vs. Meter) $\Longrightarrow$ passende Umskalierung.

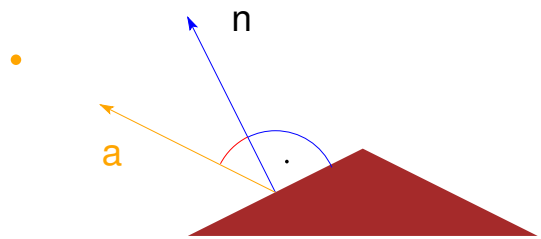
Phong-Beleuchtungsmodell

Helligkeit der Fläche setzt sich zusammen aus:

- Ambientlicht (Grundhelligkeit),
- Diffus reflektiertem Licht,
- Spiegeleffekten (nicht benötigt).

$$I = I_{\text{ambient}} + k \cdot I_{\text{diffus}}$$

$I_{\text{ambient}} + k = 255$ maximale Helligkeit

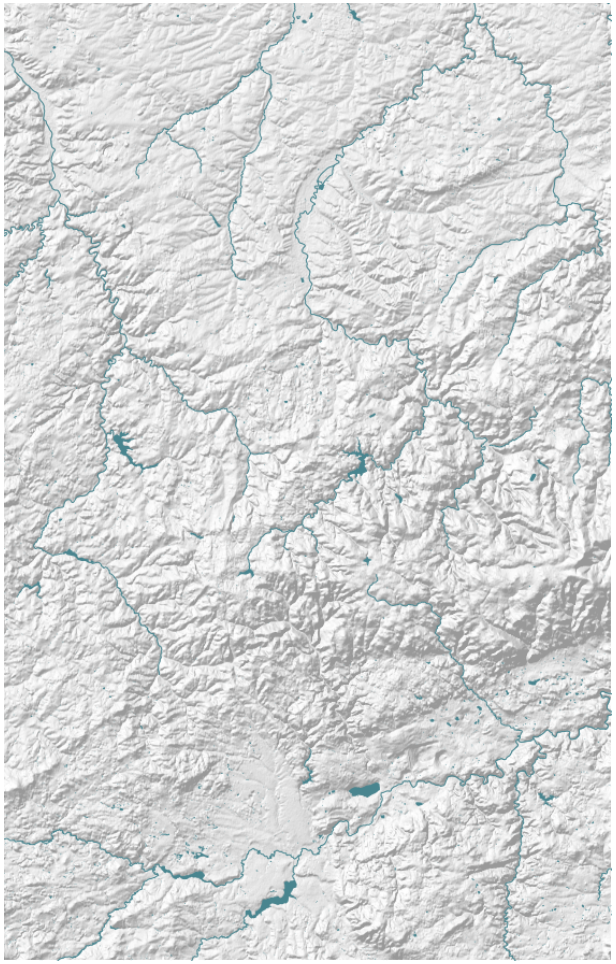


Helligkeit ist umso schwächer, je mehr Flächennormale von Lichtquellenrichtung abweicht:

$$I_{\text{diffus}} = \cos \angle(\vec{n}, \vec{a}) = \left\langle \vec{n}, \frac{\vec{a}}{\|\vec{a}\|} \right\rangle$$

Ergebnis Schattierung

- Lichtquelle in Richtung NW, 45° über Horizont
- Ambientanteil 40 %
- Schattierung als transparenter Layer über bereits gezeichnete Flächen
- Wasserläufe, -flächen nach Schattierung zeichnen!

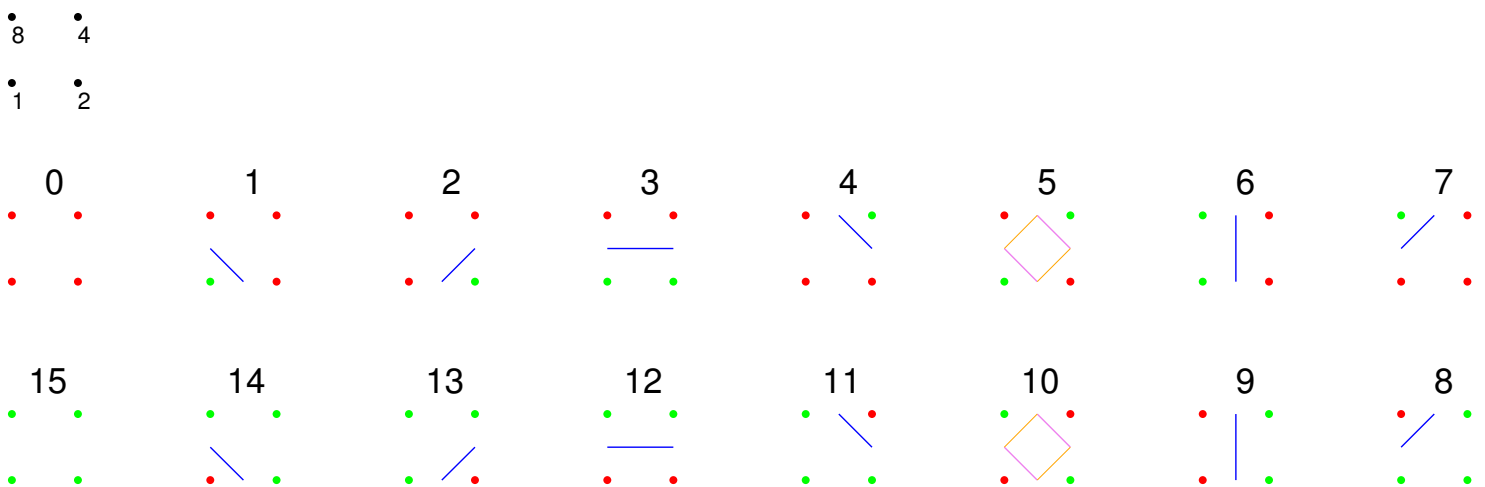


Höhenlinien

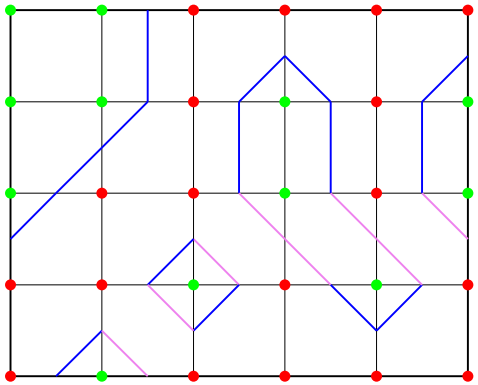
Problem: Höhendaten nur in den Rasterpunkten, benötigen (approximierte) Potentiallinien der gewünschten Höhen.

Marching-Square-Algorithmus

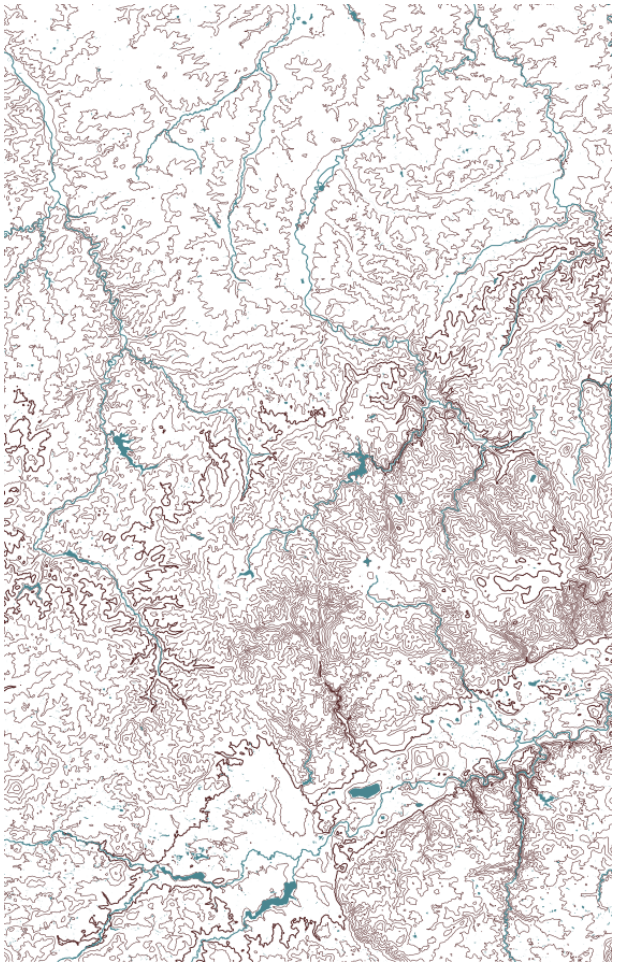
- Jede Potentiallinie (Höhe) wird getrennt behandelt
- Ordnen den vier Ecken einer Zelle die Zahlen 1, 2, 4, 8 zu
- Summieren die Werte der Ecken, deren Höhe höher als Potential ist
- 16 mögliche Fälle.



Marching Square-Algorithmus 2



- Lineare Approximation der Potentiallinie innerhalb der Zelle.
- Fälle 5 und 10 (magenta) nicht eindeutig $\Longrightarrow$ Richtung einheitlich festlegen.
- Richtung Potentialline erhalten: Normalenrichtung = ansteigende Höhe ($\Longrightarrow$ Beschriftungsrichtung)
- Abschnitte zusammensetzen (Sortieren, binäre Suche nach Folgesegment)



Abstand 50 m
Hauptlinien 500 m

Quellen I

[1] cairo 2D graphics library (Web site)
http://www.cairographics.org/

[2] Gonzales, Woods: Digital Image Processing. Person 2008.

[3] Google Sparsetable.
https://github.com/sparsehash/sparsehash

[4] Maple: Geometric Design and Space Planning Using the Marching Squares and Marching Cube Algorithms. GMAC 2003.

[5] Olbricht: Overpass API. FOSSGIS Tagungsband 2012.

[6] Powell: Five Lectures on Radial Basis Functions. Technical University of Denmark 2005.
http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3600

Quellen II

[7] Ramm, Topf: OpenStreetMap : die freie Weltkarte nutzen und mitgestalten. Lehmanns Media 2010.

[8] Robinson et al: Elements of Cartography. 6th ed. Wiley 1995.

[9] Schneider, Eberly: Geometric Tools for Computer Graphics. Morgan Kaufmann 2003.

[10] Ueberschär, Winter: Visualisieren von Geodaten mit SVG im Internet. Band 1. Wichmann 2006.

[11] Wolfram MathWorld: Albers Equal-Area Conic Projection. http://mathworld.wolfram.com/AlbersEqual-AreaConicProjection.html

[12] Zöppritz: Leitfaden der Kartenentwurfslehre, Erster Teil: Die Projektionslehre. Teubner 1912. https://archive.org/details/leitfadenderkart01zpuoft