

Bachelorarbeit

Dynamische Beschriftung von Straßen

Sergiy Pakholchak

3. Dezember 2012

Inhaltsverzeichnis

1 Einführung	3
1.1 Zielsetzung und Vorgehen	4
1.2 Literaturüberblick	5
1.3 Überblick über den hier vorgestellten Ansatz	7
2 Schriften und Rohdaten	8
2.1 Schriftarten, Schriftgrößen und Darstellung von Schrift	8
2.2 OpenStreetMap	9
3 Algorithmus	11
3.1 Kandidatengenerierung	11
3.2 Kandidatenauswahl	15
3.2.1 Feature Definition	16
3.2.2 Schnittpunktermittlung	16
3.2.3 Kandidatengenerierung	17
3.2.4 Kandidatenauswahl	19
3.2.5 Zeichnen	20
3.3 Berücksichtigung von POI-Beschriftungen	20
4 Experimente	25
5 Glossar	28
Literaturverzeichnis	29

1 Einführung

Seit Jahren entwickeln sich mobile Endgeräte zu einem ständigen Begleiter. Durch die technische Weiterentwicklung der Geräte ist auch ein sinnvoller Einsatz von mobilen Programmen (Apps) möglich. Gerade Navigations- und Kartensysteme sind ein wichtiger Bestandteil solcher Programme. Besonders im touristischen Bereich kann eine gute Integration und Darstellung von Karten einen echten Mehrwert für den Benutzer bedeuten. Im Fokus des Touristischen Bereichs spielt vor allem die Nutzung des Internets durch Apps eine wichtige Rolle. Die Darstellung von Karten sollte auch ohne Internet funktionieren, da gerade im Ausland oftmals hohe Roaming-Gebühren anfallen. Kann eine Darstellung auch ohne Internet funktionieren, so spricht man von Offline-Karten.

Wenn man Karten auf mobilen Endgerät darstellen will, dann gibt es drei Möglichkeiten. Im ersten Fall werden die Karten online als einzelne Pixel-Bilder (Bitmap) einem Server heruntergeladen. Der Vorteil ist, dass man zu jedem Zeitpunkt Karten von beliebigen Region unbegrenzt darstellen kann. Allerdings benötigt man einen permanenten Internetzugriff. Eine weitere Möglichkeit ist es, dass man alle Pixel-Bilder von einer bestimmten Kartenregion offline als Paket auf dem Gerät speichert. In diesem Fall kann man ohne Internetzugriff auf die gespeicherten Karten-Bilder zugreifen. Als großer Nachteil ist der sehr hohe Speicherbedarf und die dadurch begrenzte Größe des geografischen Bereichs zu nennen. Bereits kleine Gebiete können bei einem mittleren Detaillierungsgrad Hunderte von Megabyte bedeuten. Dies ist vor allem bei mobilen Endgeräten sehr ineffizient und problematisch. Die dritte Möglichkeit ist es, wie im zweiten Fall, die Karte von einer bestimmten Region komplett im Gerät zu speichern, allerdings in Form von rohen Geodaten (z.B. sql, xml). Aus diesen Rohdaten können die Karten auf dem Gerät gerendert werden. Das Rendern bezeichnet den Vorgang, bei dem aus der Datenbank Bilder erzeugt werden. Da die Rohdaten weitaus weniger Speicher benötigen als fertige Karten- Pixel-Bilder, spart man in diesem Fall viel Speicherplatz. Allerdings hängt die Dauer des Rendervorgangs von der Rechenleistungen des mobile Endgerätes ab. Ein Problemfaktor, der die Rendergeschwindigkeit stark beeinflusst, ist die Beschriftung von Straßen.

Es existieren eine Vielzahl kommerzielle Karten-Produkten, die ausgereifte Online-Darstellungen beinhalten. Dazugehören Google Maps, Microsoft Bing Maps und Yahoo Maps. Die meisten kommerziellen Anbieter stellen eine optimierte Darstellung für mobile Endgeräte zu Verfügung. Auch Möglichkeiten zur Offline-Darstellung werden einzeln von den Herstellern angeboten. Das derzeit wichtigste frei verfügbare Kartenmaterial ist OpenStreetMap (OSM). Hier existieren neben kommerziellen Anbieter auch OpenSource-Projekte, bei denen allerdings der Funktionsumfang und die Qualität der Darstellung dem kommerzieller Lösungen bisher noch nicht nachkommen. Der Grund liegt vermutlich darin, dass eine gute Darstellung von Karteninformation ein äußerst

komplexer Vorgang ist. Essentiell bei der sinnvollen Verwendung von Karten auf mobilen Endgeräten ist eine gute Darstellung von Schriften bzw. Bezeichnungen. Dazu gehören beispielsweise Straßen- und Ortsnamen.

Es gibt eine Reihe von Softwareprodukten, die OSM Karten rendern [Ope]. Einer der bekanntesten ist Mapnik [Mapa]. Mapnik ist ein freies in C++ geschriebenes Toolkit für die Entwicklung von Kartenanwendungen. Allerdings kann dieses Tool nur auf einem Desktop- PC und nicht auf mobilen Endgeräten sinnvoll ausgeführt werden.

Mapsforge [Mapb] ist einer der frei zugänglichen OSM-Renderer für die Java-Plattform Android. Mit dieser Toolbox kann man relativ einfach OSM-basierte Anwendungen mit Routing, Navigation, POI-Suche u.a. erstellen. Die Beschriftung der Straßen geschieht anhand von geraden Segmenten. Dieses Vorgehen wird im späteren Teil der Arbeit erläutert. Dabei werden Vor- und Nachteile aufgezeigt. Ein vernünftiges Rendern von dynamischen Schriften in Kurven ist bisher nicht vorgesehen. Ein Grund dafür ist vermutlich die Problem von Kollisionen und Überschneidungen. Auch die Performance dürfte eine Rolle spielen. Die Mapsforge-Bibliothek werde ich als Basis verwenden um die Ergebnisse der Arbeit auf der Android-Plattform praktisch zu testen.

Libosmscout [Lib] ist eine frei zugängliche Bibliothek, die in C++ geschrieben ist und welche Karten Renderer und Routing anbietet. Libosmscout hat einen plattformübergreifenden Ansatz. Für jede Plattform (Linux, Mac OS, MS Windows, Java, usw.) kann theoretisch ein sogenannter Painter entwickelt werden. Es existieren bereits Painter für Plattformen die auf C/C++ basieren. Für Java-Plattformen sind die Painter noch nicht ausgereift. Die Bibliothek ist in ständiger Entwicklung und der aktuelle Algorithmus für die Straßenbeschriftung ist bei libosmscout sehr einfach gehalten. Das führt zu Problemen, wie z.B. Kollisionen und Überschneidungen. Eine Portierung von libosmscout auf Apple iOS ist mit gewissem Zusatzaufwand möglich. Aus diesem Grund werde ich die Bibliothek verwenden um Algorithmen für die iOS Plattform zu implementieren und zu testen.

1.1 Zielsetzung und Vorgehen

In dieser Bachelorarbeit werde ich mich mit Straßenbeschriftung beschäftigen. Speziell mit der Offline-Darstellung von OSM (Daten). Dabei werden auch technische Hindernisse der mobilen Endgeräte berücksichtigt. Dazu gehören Displaygröße, Rechenleistung und Speicherplatz. Ziel ist es, die Probleme aufzuzeigen und Lösungen bzw. Lösungsansätze zu erarbeiten. Hierfür werden zunächst theoretische Grundlagen für die Beschriftung von Straßen aufgeführt. Anschließend werden die wichtigsten Abhängigkeiten erörtert, die das Rendern von Schriften beeinflussen. Dazu gehören beispielsweise Schriftart, Schriftgröße, Abstand der Buchstaben, usw. Auch der Umgang mit der Straßenlänge und die optimale Berücksichtigung von Zoomlevels werden aufgezeigt. Desweiteren werden Algorithmen bzw. Vorgehensweisen zur Vermeidung von Kollisionen und Überlappungen aufgeführt. Dabei wird vor allem darauf geachtet, dass die Lösungsansätze auch für

die Rechenleistungen von mobilen Endgeräten sinnvoll sind. Danach werden konkrete technische Umsetzungen der Beschriftung durch Pseudocode im Detail aufgezeigt. Ein anschließender praktischer Test soll den Einsatz auf den wichtigsten Plattformen untersuchen. Dabei wird ein Dummy-Programm für jeweils Apple iOS und Google Android implementiert. Es werden Renderdauer, Speicherverbrauch und die Möglichkeit von Hardwareunterstützung beim Rendern untersucht.

Die Problemstellung umfasst folgende Bereiche. Als Basis bzw. Input liegen OSM-Rohdaten in XML-Form vor. Dies sind in der Regel Polygone die durch GPS-Koordinaten definiert werden. Die OSM-Rohdaten werden im Abschnitt 2.2 genauer beschrieben. Diese Rohdaten müssen optimiert, verarbeitet und schließlich als Pixelgrafiken umgewandelt werden. Diese Pixelgrafiken können dann mit Smartphones o.ä. dargestellt werden. Für die Verarbeitung und Umwandlung in Pixelgrafiken werden Teile von libosmscout verwendet. In diesem Prozess ist es wichtig so viele Straßen wie möglich zu beschriften, ohne dass diese sich überschneiden. Darin liegt auch die Komplexität, da das Problem NP-schwer ist, welche in der Arbeit von Michael Formann and Frank Wagner bewiesen wird [FW91]. Für die Beschriftungen wird u.a. der 3-Regel-Algorithmus 1.2 zu Hilfe gezogen, der im folgenden Abschnitt genauer beschrieben wird.

1.2 Literaturüberblick

Wagner et al. [WWKS01] stellen einen graphbasierten Algorithmus zur Platzierung von Beschriftungen vor. Der Algorithmus kann zur Beschriftung von Punkten, Linien und Regionen angewendet werden. Dabei kann jedes zu beschriftende Objekt mehrere Label-Kandidaten haben. Beispielsweise kann man für einen Punkt mit quadratischem Label vier Kandidaten definieren: ausgehend vom zu beschriftenden Punkt links oben, rechts oben, links unten und rechts unten. Der Algorithmus besteht aus zwei Phasen. In Phase I werden für die einzelnen Label-Kandidaten aller Punkte drei Regeln durchlaufen, Abhängigkeiten untereinander überprüft sowie nicht benötigte Label-Kandidaten entfernt. Anschließend werden in Phase II die Label-Kandidaten pro Punkt auf höchstens einen reduziert. Im Folgendem nennen wir diesen Algorithmus der 3-Regel-Algorithmus.

Laut dem Paper ist der Algorithmus unabhängig von der Form der Label, d.h. Es können auch unregelmäßig geformte Labels verwendet werden. Dies ist sicherlich ein wichtiger Punkt zur Verwendung bei der dynamischen Beschriftung von Straßen. Schwierigkeiten sehe ich bei der Ermittlung von sinnvollen Label-Kandidaten einer Straße. Dies ist nach meinem Verständnis eine wichtige Voraussetzung für den Einsatz des Algorithmus. Die Performance soll grundsätzlich gut sein. Allerdings ist vermutlich das Prüfen von Überschneidungen bei unregelmäßig geformten Labels rechenaufwändiger, als bei rechteckigen Labels. Ich sehe in der Theorie grundsätzlich eine Einsatzmöglichkeit des Algorithmus bei der dynamischen Beschriftung von Straßen. Im Detail müsste man es aber näher untersuchen und vor allem praktische Test bzgl. der Performance durchführen.

In seiner Dissertation [Str01] befasst sich Strijk mit verschiedenen Problemen beim Platzieren von Beschriftungen auf Karten. In Kapitel 9 geht Strijk speziell auf die Beschriftung von Straßen ein. Dabei stellt er verschiedene Regeln auf, die man bei der Straßenbeschriftung beachten sollte. Dazu gehören zum Beispiel die richtige Zuordnung der Beschriftung zur Straße, ästhetische Regeln und gute Sichtbarkeit. Jede Regel hat wiederum verschiedene Qualitätsmerkmale. Beispielsweise ist die Qualität der Sichtbarkeit eines Labels gut, wenn das Label sich nicht mit anderen Labels überschneidet und vollständig im Inneren der Straße platziert ist. Die Regeln und Qualitätsmerkmale werden in einer allgemeinen Qualitätsfunktion zusammengefasst, die die Qualität einer Karte beschreibt.

Die dargestellten Informationen sind hilfreich für die dynamische Beschriftung von Straßen und sollten berücksichtigt werden. Auch die theoretische Vorgehensweise zur Erzeugung von Label-Kandidaten ist hilfreich und könnte möglicherweise bei der Verwendung des 3-Regel-Algorithmus [WWKS01] berücksichtigt werden. Um eine weitere Verwendung beurteilen zu können, müsste man den gesamten Algorithmus im Detail analysieren, vor allem mit welchem Algorithmus die Abbildung 9.5 erzeugt wurde.

Wagner et al. [NW00] zeigen eine Vorgehensweise zur Beschriftung von Straßen auf, die sich speziell für amerikanische Innenstädte eignet. In den Zentren von amerikanischen Städten, gibt es in der Regel ein sehr regelmäßiges Straßenmuster. Dabei handelt es sich um ein rechteckiges Raster in dem vertikal und horizontal mehrere Straßen verlaufen. Ziel ist es, dass sich keine Beschriftungen der verschiedenen Straßen überschneiden und dass jede Straße beschriftet wird. Die Voraussetzung zum Anwenden der vorgestellten Vorgehensweise sind allerdings sehr begrenzt. So muss der Bereich mit den zu beschrifteten Straßen als eine Art Schachbrett mit gleicher Länge und Breite betrachtet werden. Abbildung 1.1 zeigt das angesprochene „Schachbrett“.

In der Arbeit wird auch auf verschiedene Problematiken eingegangen. Beispielsweise wenn alle Straßenlabels vertikal die gleiche Länge haben, oder wenn kein Straßenlabel länger ist als die Hälfte der Straßenlänge.

Im Speziellen wird eine Art von Backtracking-Algorithmus vorgestellt, der drei verschiedene Fälle berücksichtigt. Dabei werden alle möglichen Beschriftungen in einer Baumstruktur mit verschiedenen Ebenen gehalten und durch mathematischen Formeln optimiert.

In der Arbeit werden diverse Experimente durchgeführt, die sich mit dem bereits bekannten NP-Schwerem Problem [SU00] befassen. Der daraus resultierende Algorithmus ist ohne Ausnahmen polynomial.

Einsatzmöglichkeit im eigenen praktischen Ansatz (siehe Abschnitt 3): Die Berücksichtigung dieser Arbeit ist relativ schwierig. Es wäre denkbar am Ende von Abschnitt 3.2.4 zu prüfen ob das Straßennetz einer Kachel ausreichend durch den bisherigen 3-Regel-Algorithmus beschriftet wurde und ob innerhalb des Straßennetz der Kachel eine „Schachbrett“-Struktur erkennbar ist. Letzteres ist eher ein mathematisch aufwendiger Prozess. Sofern

	C_1	C_2								C_m
R_1	Beachwood Dr.					Serrano Av.	Tulip Street	Irving Blvd.		
R_2	Maplewood Av.									
	Ridgewood Blvd.	Rosewood Av.							Elmwood Av.	
		Oakwood Av.			Ardmore Av.	Wilshire Blvd.				Miller Street
	Clinton St.		Wilton Av.			Gramery Blvd.	Beverly Blvd.			
							Kingsley Dr.			
							Santa Monica Blvd.			
R_n	Manhattan Blvd.									Larchmont Blvd.

Abb. 1.1: Amerikanische Innenstadt [NW00].

die Voraussetzungen gegeben sind, könnten die Kandidaten dieser Kachel mit Hilfe der im Paper aufgeführten Vorgehensweise neu Strukturiert werden.

1.3 Überblick über den hier vorgestellten Ansatz

Die praktische Umsetzung basiert auf verschiedenen Schritten, die sich vor allem um den 3-Regel-Algorithmus drehen. Der 3-Regel-Algorithmus ist also ein zentraler Bestandteil der praktischen Umsetzung. Vorab werden die Möglichkeiten der Ermittlung der Kandidaten in Abschnitt 3.1 aufgeführt. Diese Möglichkeiten werden dann auch weitestgehend praktisch umgesetzt. Auf Grund der begrenzten Ressourcen eines mobilen Endgerätes wird die Beschriftung der Straßen immer pro Kachel durchgeführt. Allerdings werden bestimmte Informationen Kachel-übergreifend gehalten bzw. gecached, siehe Abschnitt 3.2.5.

Zu Beginn werden so viele Kandidaten wie möglich pro Straße bzw pro Feature erzeugt. Es werden auch zu Beginn Schnittpunkte mit anderen Straßen berechnet und somit Kandidaten bzw. Features ermittelt, die nicht in dem 3-Regel-Algorithmus berücksichtigt werden müssen. Zudem werden Prioritäten vergeben um ein bestmögliches Ergebnis zu erhalten. Während der praktischen Umsetzung werden verschiedene Optimierungsmöglichkeiten untersucht und deren Auswirkung dargestellt, wie z.B. das Verwenden des Caches. Am Ende werden zusätzlich auch Beschriftungen von POIs (z.B. Gebiets- und Gebäudebeschriftungen) berücksichtigt, da diese heutzutage ein wichtiger Bestandteil von Karten sind.

2 Schriften und Rohdaten

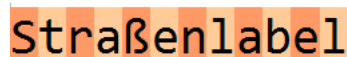
In diesem Kapitel werden Punkte aufgeführt die neben dem eigentlichen Algorithmus auch maßgeblich für die erfolgreiche Umsetzung sind. Zuerst werden Informationen rund um das Thema „Schriften“ und deren optimale Verwendung in dem eigenen Vorgehen aufgeführt. Des weiteren werden die OSM-Rohdaten etwas näher beleuchtet, um die Datenstruktur für den Input besser verdeutlichen zu können.

2.1 Schriftarten, Schriftgrößen und Darstellung von Schrift

Ein wichtiger Bestandteil für die Beschriftung von Straßen sind die zu verwendeten Schriften. Gerade bei kleineren Displaygrößen von mobilen Endgerät steht die Lesbarkeit im Vordergrund. Die Schriftgröße sollte sich an die Höhe der Straße orientieren unter Berücksichtigung von etwas Puffer am oberen und unteren Ende. Betrachtet man die Länge eines Labels so sollte die verwendete Schriftart folgende Eigenschaft mit sich bringen: Die gute Lesbarkeit im Verhältnis zum Platzbedarf sollte ausgewogen sein. Sind die Abstände zwischen den Buchstaben zu groß, so nimmt das Label und damit der Kandidat sehr viel Platz ein. Das hat zur Folge, dass ggf. einige Features nicht oder suboptimal beschriftet werden könnten. Vermieden werden sollten demnach monospaced-Schriften (Nichtproportionale Schriftarten), bei denen jeder Buchstabe eine einheitliche Länge besitzt. So hat der Buchstabe „I“ die gleiche Breite wie der Buchstabe „W“, siehe Abbildung 2.1a.

Optimaler sind Proportionale-Schriftarten bei dem jeder Buchstabe nur die Breite besitzt, die er letztendlich benötigt, siehe Abbildung 2.1b.

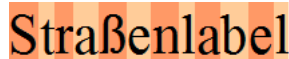
Des weiteren sollte man auf Serife-Schriftarten verzichtet werden, deren Buchstaben durch horizontale Striche geprägt sind und bei kleinerer Schriftgröße durch ihre „verspieltheit“ eher unleserlich sind, siehe Abbildung 2.1c.

The image shows the word 'Straßenlabel' in a monospaced font. Each letter has the same width, regardless of its natural shape. For example, the 'I' and 'W' are the same width.

(a) Buchstaben einheitlicher Länge

The image shows the word 'Straßenlabel' in a proportional font. The width of each letter is proportional to its natural shape. For example, the 'I' is narrower than the 'W'.

(b) Buchstaben proportionaler Länge

The image shows the word 'Straßenlabel' in a serif font. Each letter has a small horizontal line (a serif) extending from its top or bottom edge.

(c) Serife-Schriftart

Abb. 2.1: Schriftarten.

Straßenlabel

Abb. 2.2: Zur besserem Kontrast wird die Beschriftung durch weiße Umrandung hervorhebt.

Als geeignete Schriften, die den oben genannten Anforderungen entsprechen, haben sich die Schriftarten „Arial“, „Helvetica“ oder auch die freie Schriftart „DejaVu Sans“ gut bewährt.

Neben der Schriftgröße und Schriftart spielt der Kontrast eine wichtige Rolle. D.h. Die Farbe der Schriftart sollte zur beschrifteten Straßenfarbe einen ausreichend hohen Kontrast besitzen, siehe Abbildung 2.2. Da dies nicht immer gewährleistet werden kann, sollte man unter Umständen einen weißen Schatten für die Beschriftung verwenden. Somit ist das Label relativ unabhängig von der Farbe, auf dem das Label platziert wird.

2.2 OpenStreetMap

„OpenStreetMap“ ist ähnliche wie Wikipedia ein freies Projekt. Mit Hilfe von OSM werden frei zugängliche Geodaten gesammelt und anderen Nutzern zur Verfügung gestellt. Diese Geodaten werden verwendet um Kartenmaterial zu erstellen. Gegründet wurde das Projekt von Steve Coast im Jahr 2004. Seit dem wächst die Popularität des Projektes stetig. Aktuell gibt es über 600.000 Personen die Geodaten für fast alle Teile der Erde zusteuern. Somit nimmt „OpenStreetMap“ eine wichtige Rolle im weltweiten Markt für Kartenmaterial ein. Es gibt bereits eine Vielzahl von Softwareprodukten, wie z.B. Navigationssoftware die auf dem OSM Datenmaterial basieren.

Zentraler Bestandteil ist die OSM-Datenbank in der alle Geodaten von Nutzern nach einem bestimmten Prinzip gehalten werden. Die Daten werden im XML-Format gehalten und haben die Endung „.osm“. Das OSM-XML ist sehr einfach gehalten. Es besteht aus drei Hauptelementen:

Node Ein einzelner Punkt, der durch die GPS-Informationen latitude and longitude definiert ist.

Way Ist eine Art Linie, die durch eine Anzahl von aneinander folgenden Nodes definiert wird.

Relation Die Relations bezeichnen eine sortierte Anzahl von Nodes und Ways, die dadurch in Beziehung stehen.

Ways können offen oder geschlossen sein. Offene Ways sind aneinander gereite Nodes bei denen der erste und der letzte Node nicht gleich ist. Dazugehören beispielsweise Straßen. Bei geschlossene Ways ist der erste Node gleich dem letzten Node. Diese Ways werden u.a. Areas bezeichnet und dienen zur Darstellung von Flächen wie z.B. Parks oder Seen. Neben Straßen und Flächen sind auch POIs ein Teil der OSM-Daten geworden.

Ein weiteres wichtiges Element ist „Tag“, das in der Regel dazu dient, die anderen Elemente zu beschreiben. Ein „Tag“ hat wiederum die Attribute „k“ und „v“ (key und value), mit denen man so eine beliebige Anzahl von Metainformationen mitgeben kann.

Straßen wie z.B. Autobahnen, werden in der Regel durch den key „highway“ des Elements „tag“ definiert.

Das folgende Beispiel 2.1 zeigt eine Straße innerhalb eines Ortes mit dem Namen „Semmelstraße“.

Listing 2.1: Beispiel für eine Straße.

```
<node id="116117689" version="2" timestamp="2011-02-06T22:01:50Z"
uid="57645" user="KartoGrapHiti" changeset="7210593"
lat="49.7972401" lon="9.9372145" />
<node id="50487231" version="1" timestamp="2007-09-13T10:29:25Z"
uid="7197" user="user_7197" changeset="422370"
lat="49.7973092" lon="9.9373836" />
<node id="691752832" version="1" timestamp="2010-04-10T17:35:17Z"
uid="57158" user="kay_D" changeset="4384942"
lat="49.7974320" lon="9.9377123" />
<node id="297734634" version="2" timestamp="2008-09-18T15:46:38Z"
uid="26299" user="uboot" changeset="659172"
lat="49.7979484" lon="9.9390939" />
<node id="50487234" version="4" timestamp="2009-11-01T14:13:40Z"
uid="148662" user="steffterra" changeset="3006684"
lat="49.7979687" lon="9.9391558" />

<way id="30321025" version="2" timestamp="2010-04-10T17:35:19Z"
uid="57158" user="kay_D" changeset="4384942">
  <nd ref="116117689" />
  <nd ref="50487231" />
  <nd ref="691752832" />
  <nd ref="297734634" />
  <nd ref="50487234" />
  <tag k="abutters" v="retail" />
  <tag k="highway" v="residential" />
  <tag k="maxspeed" v="30" />
  <tag k="name" v="Semmelstrasse" />
</way>
```

Viele Softwareprojekte haben sich zur Aufgabe gemacht diese Sammlungen grafisch darzustellen. Einer der bekanntesten OSM-Renderer ist Mapnik [Mapa]. Auch libosm-cout [Lib] ist solch ein Softwareprodukt.

Ein wichtiger Bestandteil der grafischen Aufbereitung ist die optimale Darstellung von Straßen, welches Gegenstand dieser Arbeit ist.

3 Algorithmus

Die bisher theoretischen Untersuchungen und Recherchen werden nun praktisch umgesetzt, bzw. die aufgeführten Vorgehensweisen werden in einer praktischen Arbeit auf Tauglichkeit untersucht. Konkret wurden verschiedene Punkte technisch mit Hilfe von Objectiv-C auf dem iPhone und iPad umgesetzt.

3.1 Kandidatengenerierung

Um den Algorithmus [WWKS01] durchführen zu können, benötigen wir sinnvolle Kandidaten für die Beschriftungen der Straßen.

Jede Straße besteht aus Segmenten. Ein Segment ist eine gerade Linie. Kurven werden somit durch eine Vielzahl von Segmenten erstellt. Je weiter man in die Karte hineinzoomt, desto größer werden die Segmente. Siehe Abbildung 3.1.

Die Kandidatensuche kann in mehreren Schritten ablaufen, wobei die Segmente eine wichtige Rolle spielen. Ein weiterer Faktor ist das Zoomlevel in dem die Straßen dargestellt werden sollen. Des weiteren spielt die Schriftgröße eine wichtige Rolle.

Mögliche Abfolge:

1. Entlang einer Straße werden Bereiche zur Beschriftung definiert, im Folgenden „Feature“ genannt. Die Features haben einen definierten Radius X . Ggf. ist es sinnvoll den Radius abhängig von Zoomlevel zu definieren. Um eine gute Übersicht gerade bei kleineren Displaygrößen zu gewährleisten, wird ein fester Abstand zwischen den Features definiert. Dieser Abstand kann von Priorität der Straße und Zoomlevel abhängig sein.
2. Man durchläuft alle Segmente einer Straße innerhalb eines Features und platziert in einem Segment das Label sofern das Label in das Segment passt. Bei der Platzierung innerhalb des Segments wird darauf geachtet, dass das Label mittig gesetzt wird. Sofern das Segment eine Länge X überschreitet, werden weitere Labels in einem Segment mit einem Abstand Y platziert. Siehe Abbildung 3.2.
3. Falls innerhalb eines Features kein Segment groß genug ist um darin ein Label zu platzieren, so wird das Label über mehrere Segmente in einer Kurve gemalt. Dabei wird das Label mittig im Feature platziert. Siehe Abbildung 3.3.



Abb. 3.1: Segment.



Abb. 3.2: Mehrere Features.

4. Um eine bessere Darstellung zu gewährleisten, wird geprüft ob der Winkel zwischen den Segmenten größer 90 Grad ist. Falls dies der Fall ist, so wird im Notfall das Feature übersprungen und nicht beschriftet. Es sei denn man befindet sich in höchster Zoomstufe, um mindestens eine Beschriftung zu gewährleisten. Sie Abbildung 3.4.
5. Hat man alle Kandidaten von allen Features so werden alle restlichen Straßen des Kartenabschnitts mit dem Selben Algorithmus durchlaufen. Anschließend wird der 3-Regel-Algorithmus angewendet. Als Ergebnis sollten wir im Idealfall ein Label pro Feature erhalten welches sich nicht mit anderen Labels von anderen Features überschneidet.



Abb. 3.3: Beschriftung über mehrere Segmente.

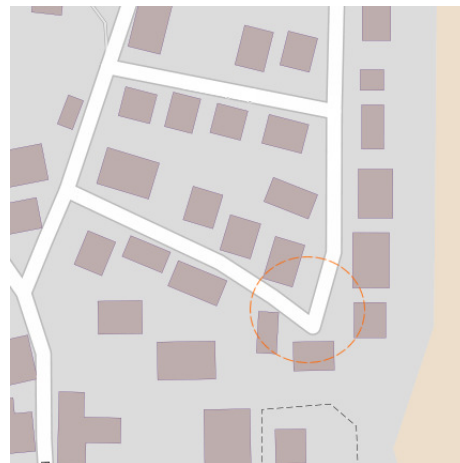


Abb. 3.4: Scharfe Kurve.

Problemstellungen/Erweiterungen:

1. Die Auswahl der Features wird entlang der Straße in Punkt 1 nach fest definierten Abständen durchgeführt. Zudem ist der Radius abhängig vom Zoomlevel fest definiert. Dadurch werden ggf. Segmente einer Straße beschnitten und verkleinert.

Es ist vermutlich sinnvoll die Feature-Größe bzw die Grenzen eines Features an Kanten von Segmenten der Straße dynamisch zu definieren.

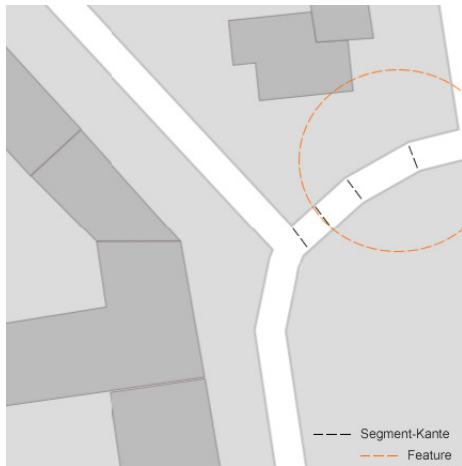


Abb. 3.5: Verkleinerung von Segmenten.

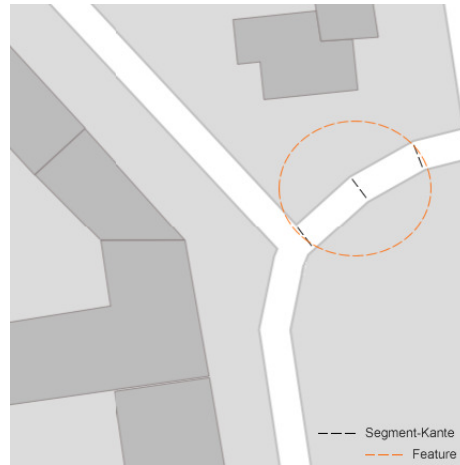


Abb. 3.6: Featuregröße und Position abhängig von Segment-Kanten.

- Eine Weitere Alternative wäre es, bei jeder Segment-Kante ein neues Feature mit einer fest definierten Größe zu setzen.
2. Allerdings muss auch berücksichtigt werden, dass es vorallem in hohen Zoomlevels einzelne Segmente geben kann die sehr lang sind und die der Übersichtshalber durch mehrere Features unterteilt werden müssen.
 3. Wie soll mit Features umgegangen werden die keine Labels enthalten, da diese durch den 3-Regel-Algorithmus entfernt wurden? Ein Lösungsansatz wäre es zu prüfen, ob die Straße, die das Feature enthält, in einem bestimmten Abstand ein anderes erfolgreich beschriftetes Feature enthält. Sofern dies der Fall sein sollte, kann ggf. auf das nicht-beschriftete Feature verzichtet werden. Das gleiche könnte für den Fall gelten, wenn die Straße an der nicht-beschrifteten Stelle aber in einer genaueren Zoomstufe erfolgreich beschriftet ist.
 4. Optimierung der Rohdaten: Ggf. ist es sinnvoll die Anzahl der Segmente pro Feature zu erhöhen um mögliche Überschneidungen mit Segmenten von anderen Straßen bzw. Features zu vermeiden. Beispielsweise können Schnittpunkte mit anderen Straßen dafür verwendet werden um weitere Segmente zu erstellen und somit die Anzahl der Features/Kandidaten zu erhöhen.

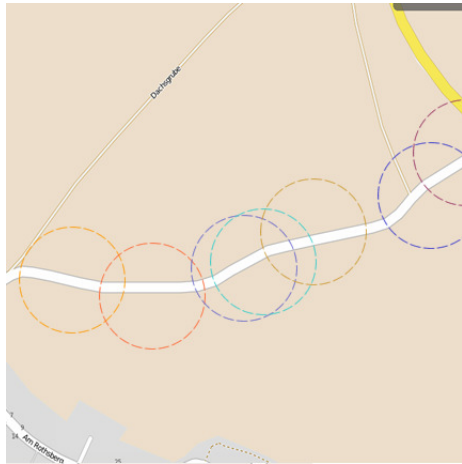


Abb. 3.7: Jeweils ein Segment pro Kante von links nach rechts.

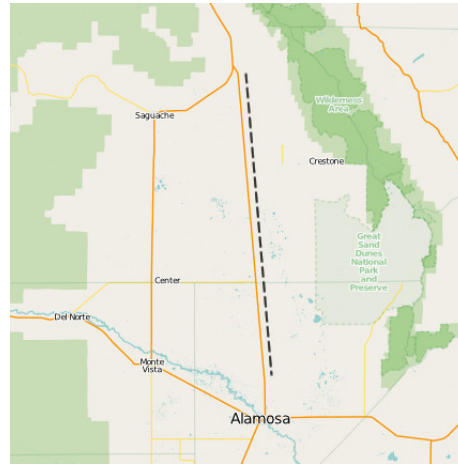


Abb. 3.8: Sehr langes Segment, welches in hoher Zoomstufe zu unterteilen ist.

5. Performance Punkt 3 und Punkt 4: Die Ermittlung der maximalen Anzahl von Kandidaten in Kurven und dies für alle Features von allen Straßen, ist vermutlich sehr rechenaufwendig. Dieser Prozess sollte einmalig pro Zoomstufe erfolgen. Ggf. lässt sich diese Berechnung bereits Serverseitig durchführen, sofern bekannt ist welche Schriftarten/Schriftgrößen am mobilen Endgeräte verwendet werden.
6. Bei Punkt 4 wird das Label mittig platziert. Sofern ein 90 Grad Winkel existiert, so wäre es theoretisch möglich, das Label entlang der Straße innerhalb eines Features zu verschieben, bis das Label nicht mehr über einen 90 Grad Winkel gemalt werden muss. Hier sind weitere Problemfälle zu untersuchen, beispielsweise wenn man das Ende eines Features erreicht.

Optimierung der Rohdaten

Zur Performance-Optimierung wäre es sinnvoll Features zu ermitteln, die keine Schnittpunkte mit anderen Straßen enthalten. Diese Features könnten einen Flag erhalten und müssten bei dem 3-Regel-Algorithmus nicht berücksichtigt werden. Das die blauen Features in Abbildung 3.10. überschneiden sich nicht mit anderen Strassen und könnten gleich beschriftet werden bzw. könnten speziell für den 3-Regel-Algorithmus markiert werden, d.h. Der 3-Regel-Algorithmus könnte diese Features bevorzugen.

Prioritäten setzen bei Kandidaten

Bei der Auswahl der Kandidaten sollten auch Prioritäten gesetzt werden. Dabei sollten Kandidaten bevorzugt werden, bei der die optische Darstellung am besten ist und bei denen die Gefahr von Überschneidungen nicht vorhanden ist. Diese Prioritäten müssen dann im 3-Regel-Algorithmus Aberücksichtigt werden.

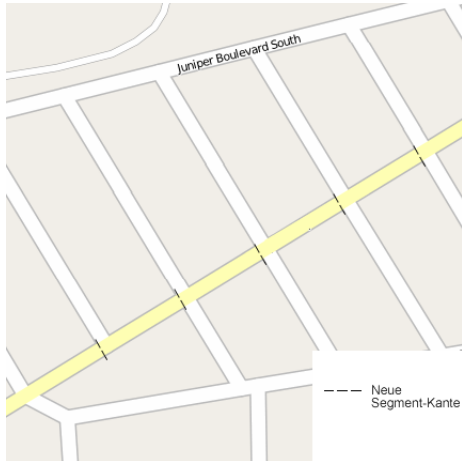


Abb. 3.9: Jeweils ein Segment pro Kante von links nach rechts.

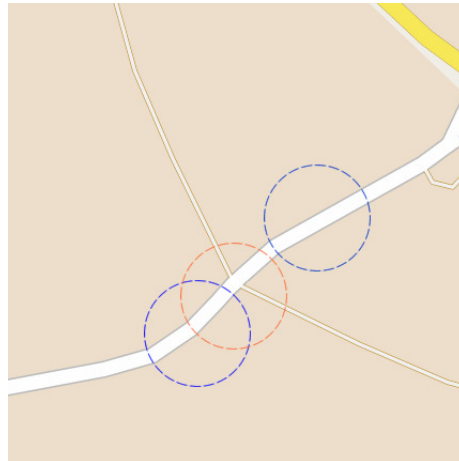


Abb. 3.10: Rohdaten optimieren.

Beispielsweise sollte vermieden werden, dass Labels zu dicht nebeneinander liegen, obwohl theoretisch weitere Labels eines Features möglich wären. Abbildung 3.11. Zeigt einen ungünstigen Fall in dem alle Labels in rot die letztendlich gemalt werden, zu dicht aneinander liegen.

In diesem Fall sollte zumindest das Label in blau eine höhere Priorität bekommen. Siehe Abbildung 3.12.

Bevorzugte Labels könnten sein:

1. Das Label liegt mittig im Feature
2. Das Label passt vollständig in ein Segment und muss nicht als Kurve gemalt werden

Die gesetzte Priorität eines Labels wird anschließend im 3-Regel-Algorithmus berücksichtigt. D.h. : Nachdem alle Labels die sich überschneiden entfernt wurde, so wird nicht einfach nur das erste mögliche Label eines Features gemalt, sondern es wird das Label aus den übrig gebliebenen Kandidaten gewählt, welches die höchste Priorität hat. Diese Optimierung des Vorgehens sollte keine nennenswerte Verschlechterung der Performance bedeuten.

3.2 Kandidatenauswahl

Die Folgenden Schritte beziehen sich pro Kachel. Die Größe der Kachel kann frei definiert werden. Abbildung 3.13 zeigt eine Kachelgröße von 256×256 Pixel. Abbildung 3.14 zeigt eine Kachelgröße von 512×512 Pixel.

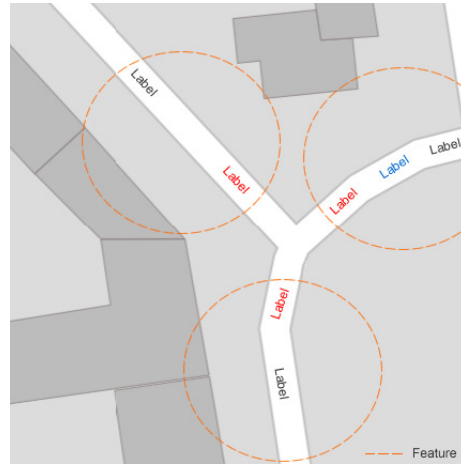
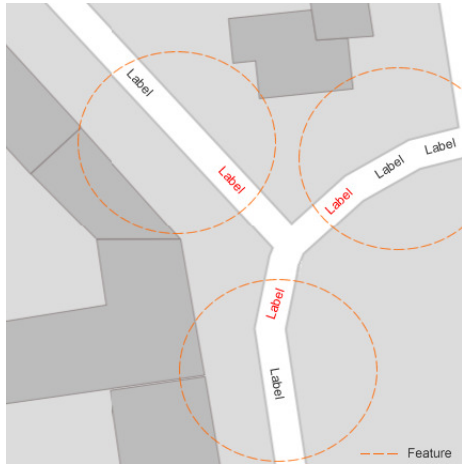


Abb. 3.11: Ausgewählte Labels in rot liegen dicht zu einander. **Abb. 3.12:** Der Label in blau sollte eine höhere Priorität haben.

3.2.1 Feature Definition

Ähnlich wie im Abschnitt 1 beschrieben, werden die Features folgendermaßen gesetzt: Man beginnt an einer Segment-Kante und addiert die Segmente so lange, bis eine bestimmte Länge erreicht oder überschritten ist. Danach wird das Feature „geschlossen“ und man beginnt mit der Erstellung des nächsten Features. Jede Straße besitzt somit 1 bis n Features.

3.2.2 Schnittpunktermittlung

In diesem Schritt werden die Schnittpunkte der Straßen errechnet. Die Schnittpunkte benötigt man für folgende Punkte:

1. Die Schnittpunkte helfen zur Vergabe von Prioritäten bei den Kandidaten. Beispielsweise bekommt ein Kandidat ohne Schnittpunkt eine höhere Priorität.
2. Reduzierung von unnötigen Kandidaten, siehe Schritt 3.2.3
3. Des weiteren benötigt man diese Schnittpunkte für den eigentlichen Algorithmus. Siehe Schritt 3.2.4.

Das Errechnen der Schnittpunkte ist auf dem Gerät relativ aufwändig. Auf Grund der begrenzten Ressourcen kann dafür ein vereinfachter Algorithmus verwendet werden. Anstatt alle Segmente zu durchlaufen und alle mit einander zu vergleichen mit Laufzeit $O(n^2)$, kann beispielsweise der Algorithmus von Bentley und Ottmann [BO79] verwendet werden. Dieses Vorgehen würde die Berechnungszeit logarithmisch verbessern.

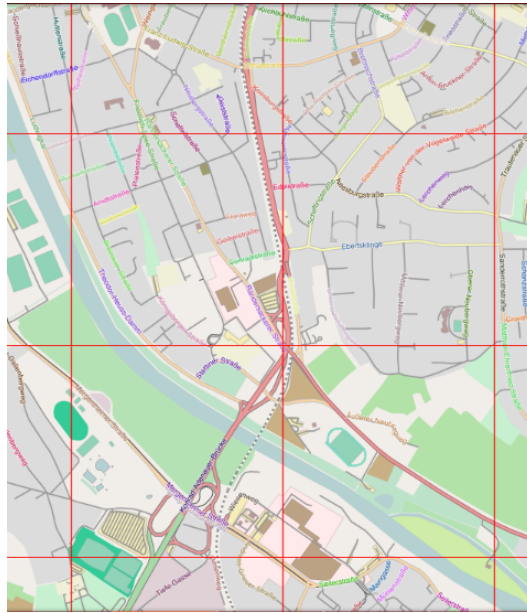


Abb. 3.13: Kachelgröße 256×256 Pixel.

3.2.3 Kandidatengenerierung

Es werden nun alle Kandidaten für jedes Features generiert. Ein Kandidat beginnt an einer Segment-Kante. Abbildung 3.15 zeigt alle Kandidaten von allen Features eines bestimmten Bereiches (mehrere Kacheln).

Da dieser Schritt pro Kachel durchlaufen wird, müssen verschiedene Optimierungen vorgenommen werden. In der Regel ist es so, dass Straßen und somit Features/Kandidaten Kachel-übergreifend verlaufen. Aus diesem Grund werden die bereits von anderen Kacheln definierten Kandidaten zwischengespeichert.

Durch die Zwischenspeicherung erreicht man folgende Optimierungen:

1. Da es zu redundanten Generierungen von Kandidaten kommen kann, werden identische Kandidaten nur ein mal generiert bzw. berechnet.
2. Für bereits beschriftete Features, die beispielsweise über zwei Kachel laufen müssen nicht nochmals die Kandidaten berechnet werden. Der bereits ausgewählte Kandidat wird aus dem Cache geholt.

Abbildung 3.16 zeigt das ein theoretisches Endergebnis ohne Verwendung eines Caches. Anhand des Beispiels der Straße „Werner von Siemens Straße“ erkennt man dass sich der Algorithmus bei der Beschriftung eines Features in der jeweiligen Kachel für unterschiedliche Kandidaten entschieden hat.

Weiterhin werden an dieser Stellen Prioritäten bei den Kandidaten vergeben. Kandidaten die mittig platziert werden können, oder/und Kandidaten die am wenigsten Überschneidungen aufweisen, erhalten eine höhere Priorität. Diese Priorität wird dann vom 3-Regel-Algorithmus verwendet. Siehe Schritt 3.2.4.



Abb. 3.14: Kachelgröße 512×512 Pixel.

Features/Kandidaten, die keine Überschneidungen besitzen bzw. die zwischen Schnittpunkten passen, werden entsprechend Markiert und müssen nicht vom 3-Regel-Algorithmus berücksichtigt werden. D.h. Diese Kandidaten werden als fix angesehen und auf jeden Fall gezeichnet. Dieses Vorgehen ist durch die Ergebnisse aus Schritt 3.2.2 möglich (Schnittpunkte berechnen). Abbildung 3.16 zeigt alle Kandidaten eines bestimmten Bereichs ohne diese Optimierung. Abbildung 3.17 zeigt alle Kandidaten eines bestimmten Bereichs mit dieser Optimierung.

Die Reduzierung der Kandidaten ist durchaus erkennbar. Am Beispiel von der Stadt Würzburg ergibt sich bei aktueller Implementierung der Optimierung eine Reduzierung um ca. 50 Prozent. Diese Optimierung ist theoretisch noch ausbaufähig.

	Ohne Optimierung	mit Optimierung
Anzahl Straßen:	551	551
Anzahl Features:	723	723
Anzahl Kandidaten:	4911	2604

Tab. 3.1: Reduzierung der Kandidaten für den Bereich (lonLeft:9.932 latTop:49.7918 lonRight:9.9656 latBottom:49.7657 zoom: 15) der Stadt Würzburg.

Hinweis

Bei der aktuellen Vorgehensweise zur Generierung von Kandidaten pro Kachel können unter Umständen suboptimale Kandidaten gewählt werden bzw. unbekannte Kandida-

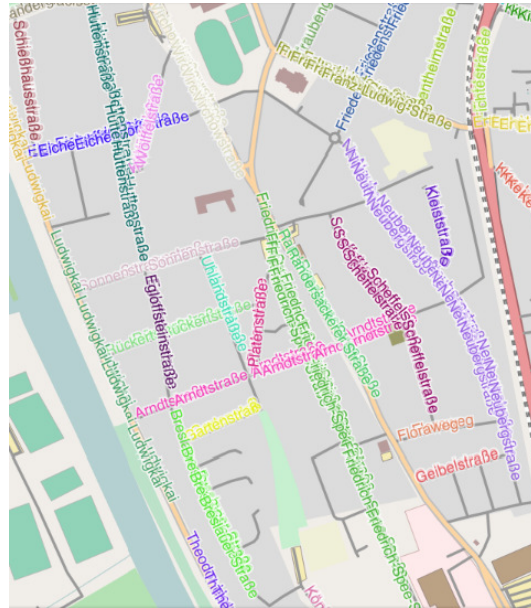


Abb. 3.15: Für jede Feature werden alle mögliche Kandidaten generiert.

ten andere umliegenden Kacheln existieren. Dies geschieht vor allem dann, wenn die Straße über viele Kacheln verläuft. D.h. auch, dass ein Feature der Straße über mindestens 2 Kacheln verläuft. Der Grund ist, dass man nicht genügend Informationen über die umliegenden Kacheln besitzt. Als Lösung könnte man die Maße der Kacheln vergrößern. Allerdings verlängert die Vergrößerung der Kachel auch die Rechenzeit und beeinflusst somit negativ die Usability des Nutzers.

3.2.4 Kandidatenauswahl

In diesem Schritt wird der eigentliche 3-Regel-Algorithmus durchgeführt. Um zwei Beschriftungen/Labels auf Überschneidung zu testen werden unter Anderem folgende Schritte durchgeführt:

1. Für beide Features werden alle Schnittpunkte die aus Schritt 3.2.2 errechnet wurden verwendet. Allerdings werden nur die Schnittpunkte berücksichtigt die auch die Beschriftung betreffen (Schnittpunkte die zwischen Anfang und Endedes Labels liegen).
2. Danach werden die Schnittpunkte bei der Labels verglichen.

Bei der Auswahl eines finalen Labels aus einer Menge übrig gebliebener Kandidaten wird nicht einfach der erste Kandidat ausgewählt, sondern der Kandidat mit der höchsten Priorität. Die Priorität wird in Schritt 3.2.3 vergeben. Die Berücksichtigung von Prioritäten verbessert das Erscheinungsbild stark.

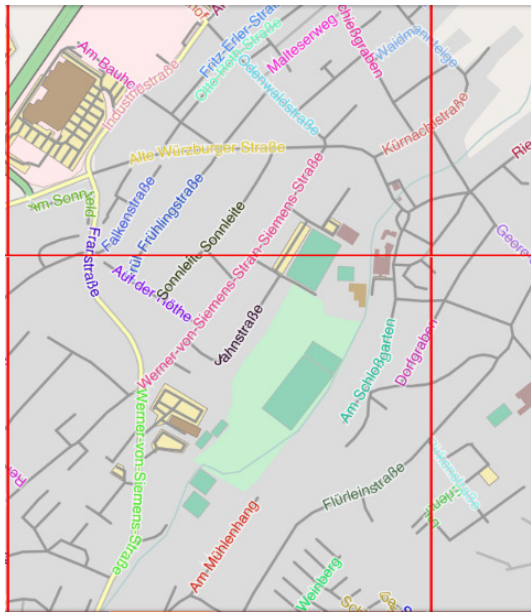


Abb. 3.16: Features werden ohne Verwendung eines Caches beschriftet.

Abbildung 3.18 zeigt das Ergebnis des Algorithmus ohne Berücksichtigung von Prioritäten und Abbildung 3.16 zeigt das Ergebnis mit Berücksichtigung von Prioritäten.

3.2.5 Zeichen

Die durch Schritt 3.2.4 definierten Beschriftungen werden gezeichnet und in den Cache/Zwischenspeicher gelegt. Diese Zwischenspeicher ist für die nächste Kachel wichtig. Siehe Schritt 3.2.3.

3.3 Berücksichtigung von POI-Beschriftungen

Neben den Beschriftungen von Straßen sind bei Karten die Beschriftungen von POIs (Point-Of-Interest) ein wichtiger Bestandteil. Dazu gehören beispielsweise wichtige Gebiets- und Gebäudebeschriftungen wie z.B. Krankenhäuser oder Parkplätze. Aus diesem Grund sollen auch diese Beschriftungen berücksichtigt werden, d.h. Es sollten Überschneidungen von Straßennamen und POI-Beschriftungen vermieden werden. Abbildung 3.20 zeigt eine typische Überschneidung von Straßennamen und POI-Beschriftungen.

Ein einfacher, aber effektiver Ansatz ist es, die Beschriftung eines POI als Kandidat bzw. Feature mit den den bisherigen Ablauf zu berücksichtigen. Konkret betrachtet man die POI-Beschriftung quasi als horizontale gerade Straße mit der Länge X . Diese Beschriftung erhalten dann die höchstmögliche Priorität, analog zu den Straßenbeschriftungen die bereits im Cache fix gesetzt sind und auf jeden Fall gezeichnet werden. D.h.



Abb. 3.17: Für jede Feature werden möglichst nur optimale Kandidaten generiert.

dass diese POI-Beschriftungen immer gezeichnet werden und dass sich andere Straßenbeschriftungen mit Hilfe des 3-Regel-Algorithmus entsprechend anpassen.

Im Detail bedeutet dieser Ansatz, dass man mehr Überschneidungen im Abschnitt 3.2.2 findet, dass mehr Kandidaten bei dem 3-Regel-Algorithmus als feste Beschriftungen betrachtet werden. Der eigentliche Algorithmus muss aber nicht geändert bzw. erweitert werden.

Abbildung 3.21 zeigt, wie bei der Beschriftung der Straße „Stresemannstraße“ ein anderer Kandidat durch den 3-Regel-Algorithmus gewählt wurde. Somit wurde die Überschneidung mit dem POI (Bundesministerium für wirtschaftliche Zusammenarbeit und Entwicklung) vermieden.

Bei den POI-Beschriftungen unter sich existieren keine Überschneidungen, bzw. POI-Beschriftungen die sich überlappen werden nicht gezeichnet. Diese Beschriftungen können dann aber in der Regel bei höheren Zoomstufen platziert werden und sind damit sichtbar.

Optimierungsbedarf bei dieser Vorgehensweise

Aktuell existiert das Problem, dass es zu Überschneidungen kommen kann, sofern ein Feature über zwei Kacheln beschriftet werden muss. In diesem Fall kann die Vorgehensweise in folgendes Problem laufen: Bei der Beschriftung einer Kachel A entscheidet sich der Algorithmus für den Kandidat X eines Features. Dieser Kandidat läuft über die Grenzen der Kachel A in Kachel B hinaus. In Kacheln B existiert aber eine POI-Beschriftung Z die diesen Kandidat X aus Kachel A schneidet. Zum Zeitpunkt des Renderns der Kachel A existieren keine Informationen über die POI-Beschriftung Z, d.h. Kachel A „weiß“



Abb. 3.18: Features werden ohne Berücksichtigung von Prioritäten beschriftet.

nichts über POI-Beschriftung Z aus Kachel B. Somit entscheidet sich der Algorithmus für den Kandidat X. Da der Kandidat X in der Kachel A gezeichnet wird, wird er auch im Cache gespeichert und somit automatisch auch in der Kachel B gezeichnet. Eine Überschneidung mit der POI-Beschriftung Z ist die Folge.

Eine Lösung wäre es, alle POI-Beschriftungen vor dem Rendern der ersten Kachel in den Cache zu laden. Diese POI-Beschriftungen würden somit bei der Kandidatenauswahl pro Kachel berücksichtigt, d.h. dass diese POI-Beschriftungen als fest anzusehen sind und alle Straßenbeschriftungen sich daran „orientieren“ müssen. Problematisch bei diesem Vorgehen wird vermutlich die Performance auf den mobilen Endgeräten sein, da ein längerer Initialisierungsprozess zu erwarten ist.

Des Weiteren wäre es vermutlich sinnvoll auch für die POI-Beschriftungen selbst eine weitere Stufe von Prioritäten einzuführen bzw. zu berücksichtigen. Beispielsweise könnte ein „Krankenhaus“ eine höhere Priorität erhalten als ein „Stadtspark“. In einer weiteren Ausbaustufe könnten man auch die Prioritäten zwischen POI-Beschriftungen und Straßenbeschriftungen vergleichen und entsprechend reagieren. Beispielsweise könnte die Beschriftung einer Hauptstraße in einem kleineren Ort eine höhere Priorität erhalten als beispielsweise ein „Stadtspark“.



Abb. 3.19: Features werden mit Berücksichtigung von Prioritäten beschriftet.

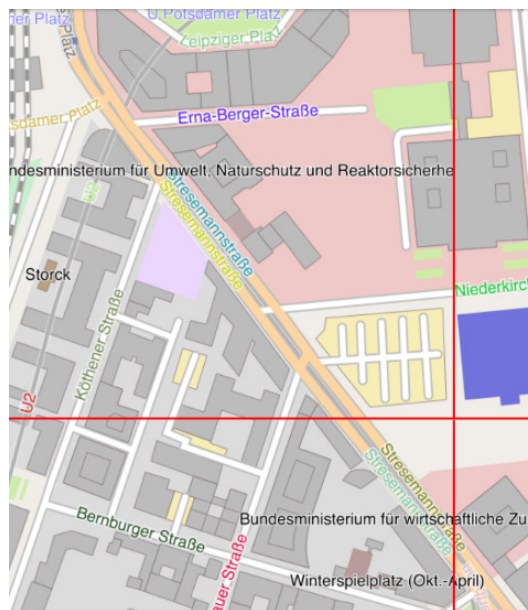


Abb. 3.20: Überschneidung von Straßennamen und POI-Beschriftungen.

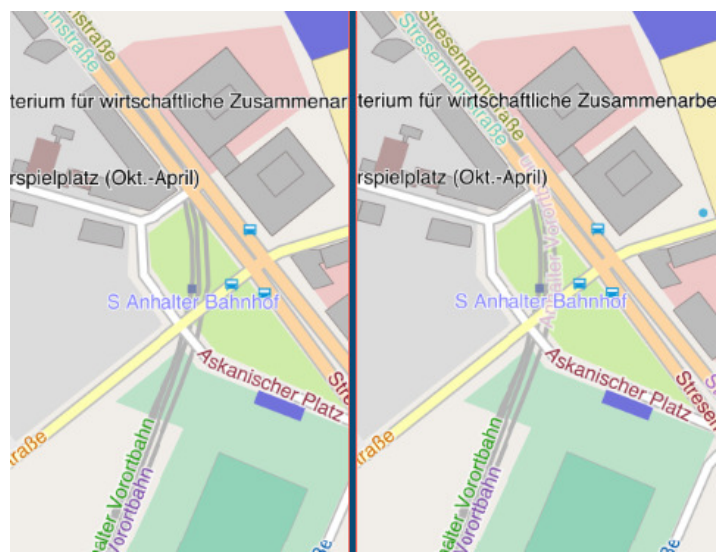


Abb. 3.21: Überschneidung von Straßennamen und POI-Beschriftungen.

4 Experimente

Die folgenden Experimente wurden mit Hilfe von libosmscout unter dem Betriebssystem iOS durchgeführt. Beim dem Testgerät handelt es sich um ein iPhone4 mit dem Betriebssystem iOS Version 6.0.

Zu Beginn werden verschiedene Experimente für vier benachbarte Kacheln durchgeführt. Sie werden unter verschiedenen Einstellungen gemessen, wie z.B. mit Cache und ohne Cache, so wie für alle und einer reduzierten Anzahl von Kandidaten. Gerendert werden die Kacheln in der Reihenfolge 1 bis 4. Die Kacheln enthalten einen bestimmte geografische Bereiche die folgendermaßen definiert sind 4.1.

	longitude left	latitude top	longitude right	latitude bottom	zoom level
Kachel 1	-73.992920	40.763901	-73.981934	40.755580	15
Kachel 2	-73.992920	40.755580	-73.981934	40.747257	15
Kachel 3	-73.981934	40.763901	-73.970947	40.755580	15
Kachel 4	-73.981934	40.755580	-73.970947	40.747257	15

Tab. 4.1: Definitionsbereiche von vier Kacheln aus New York.

Aus der Tabelle 4.2 und Tabelle 4.3 folgt, dass der Algorithmus mit Verwendung des Caches schneller ist, als ohne Cache. Kachel 1 besitzt aber immer die gleiche Laufzeit. Das liegt daran, dass bei der Kandidatenauswahl mit Cache für die benachbarte Kacheln 2 - 4 weniger Features existieren die der Algorithmus berücksichtigen muss. Da manche Features schon durch Kachel 1 beschriftet wurden.

	Dauer in Sek	Strassen beschriftet
Kachel 1	1.14	41 aus 43
Kachel 2	1.17	55 von 57
Kachel 3	1.45	59 von 60
Kachel 4	1.15	55 von 57

Tab. 4.2: Ergebnisse des 3-Regel-Algorithmus ohne Cache für eine reduzierte Anzahl von Kandidaten.

Aus der Tabelle 4.4 und Tabelle 4.5 folgt, dass sich die Geschwindigkeit des Algorithmus verringert, je mehr Kandidaten der Algorithmus berücksichtigen muss.

Folgende Experimente werden für ein relativ großes Gebiet 4.6 mit und ohne Berücksichtigung von POI durchgeführt. Der Cache spielt dabei keine Rolle, da lediglich eine große Kachel erzeugt wird.

	Dauer in Sek	Strassen beschriftet
Kachel 1	1.14	41 aus 43
Kachel 2	0.82	55 von 57
Kachel 3	1.15	59 von 60
Kachel 4	0.71	55 von 57

Tab. 4.3: Ergebnisse des 3-Regel-Algorithmus mit Cache für eine reduzierte Anzahl von Kandidaten.

	Dauer in Sek	Strassen beschriftet
Kachel 1	1.32	41 aus 43
Kachel 2	1.47	55 von 57
Kachel 3	1.89	59 von 60
Kachel 4	1.23	55 von 57

Tab. 4.4: Ergebnisse des 3-Regel-Algorithmus ohne Cache für alle Kandidaten.

Die Ergebnisse in den Tabellen 4.7 und 4.8 ergeben folgende Rückschlüsse. Ohne Berücksichtigung ist der Ablauf zwar etwas schneller aber nur sehr gering. Werden nicht alle Kandidaten berücksichtigt (optimierte Kandidaten), so ist der Algorithmus spürbar schneller.

	Dauer in Sek	Strassen beschriftet
Kachel 1	1.32	41 aus 43
Kachel 2	0.99	55 von 57
Kachel 3	1.41	59 von 60
Kachel 4	0.85	55 von 57

Tab. 4.5: Ergebnisse des 3-Regel-Algorithmus mit Cache für alle Kandidaten.

	longitude left	latitude top	longitude right	latitude bottom	zoom level
Kachel	-74.010940	40.741180	-73.974640	40.714180	14

Tab. 4.6: Definitionsbereiche der Kachel aus New York.

	Dauer in Sek	Strassen beschriftet
nicht alle Kandidaten	32.32	231 aus 247
alle Kandidaten	44.280	231 aus 247

Tab. 4.7: Ergebnisse des 3-Regel-Algorithmus ohne POI-Berücksichtigung

	Dauer in Sek	Strassen beschriftet
nicht alle Kandidaten	32.92	228 aus 247
alle Kandidaten	46.280	228 aus 247

Tab. 4.8: Ergebnisse des 3-Regel-Algorithmus mit POI-Berücksichtigung

5 Glossar

Cache: Eine Art Zwischenspeicher. In speziellen Fall dieser Arbeit handelt es sich um das halten der bisher erfolgreich platzierten Kandidaten.

Feature: Ein bestimmter Bereich entlang einer zu beschrifteten Straße, der ein Mal beschriftet werden soll. Ein Feature kann 1-n Kandidaten enthalten.

Kachel: Eine Kachel ist ein quadratischer Abschnitt einer Karte. Dieser Abschnitt kann verschiedene Größen haben. Beispielsweise 512×512 Pixel

Kandidat: Ein Kandidat ist eine potentielle Beschriftung einer Straße innerhalb eines Features.

POI: Steht als Abkürzung für „Point-of-Interest“ und bezeichnet ein Punkt oder Bereich innerhalb einer Karte, der für den Betrachter bzw. Benutzer der Karte von Interesse sein kann. Dazu gehören z.B. „Parkhäuser“ oder auch „Restaurants“. Straßenbeschriftungen und POI-Beschriftungen sind die zentralen Bestandteile einer Karte.

Segment: Als Segment wird ein gerader Abschnitt einer Straße bezeichnet. Eine Straße besteht ausschließlich aus geraden Segmenten, d.h. Kurven werden somit durch eine Vielzahl von Segmenten erstellt.

Label: Das Label ist die eigentliche grafische Beschriftung. Das Erscheinungsbild des Labels ist Abhängig von Schriftart, Schriftgröße und dem Verlauf der Label (geradlinig oder kurvig).

Literaturverzeichnis

- [BO79] Jon Bentley und Thomas Ottmann: Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, 1979.
- [FW91] Michael Formann und Frank Wagner: A Packing Problem with Applications to Lettering of Maps. In: *Proc. 7th Annu. ACM Sympos. Comput. Geom. (SoCG'91)*, Seiten 281–288, 1991.
- [Lib] Libosmscout. <http://sourceforge.net/projects/libosmscout/>.
- [Mapa] Mapnik. <https://github.com/mapnik/mapnik/wiki>.
- [Mapb] Mapsforge. <http://code.google.com/p/mapsforge/>.
- [NW00] Gabriele Neyer und Frank Wagner: Labeling Downtown. In: *Proc. Italian Conf. on Algorithms and Complexity (CIAC'00)*, Band 1767 der Reihe *Lecture Notes Comput. Sci.*, Seiten 113–125. Springer-Verlag, 2000. http://dx.doi.org/10.1007/3-540-46521-9_10.
- [Ope] OpenStreetMap Project: Liste der Renderer. <http://wiki.openstreetmap.org/wiki/DE:Rendering>.
- [Str01] Tycho Strijk: *Geometric Algorithms for Cartographic Label Placement*. Dissertation, Utrecht University, Department of Computer Science, Januar 2001.
- [SU00] Sebastian Seibert und Walter Unger: The Hardness of Placing Street Names in a Manhattan Type Map. In: *4th Italian Conference, CIAC 2000 Rome, Italy, March 1–3, 2000 Proceedings*, Band 1767 der Reihe *Lecture Notes Comput. Sci.*, Seiten 102–112. Springer-Verlag, 2000. http://dx.doi.org/10.1007/3-540-46521-9_10.
- [WWKS01] Frank Wagner, Alexander Wolff, Vikas Kapoor und Tycho Strijk: Three Rules Suffice for Good Label Placement. *Algorithmica*, 30(2):334–349, 2001. <http://dx.doi.org/10.1007/s00453-001-0009-7>.