Ein HTML5-Reverse-Geocoder auf Straßenebene für den Offlinebetrieb

Markus Mayr
Technische Universität, Wien · mmayr@geoinfo.tuwien.ac.at

Short paper

Zusammenfassung

Die Nutzung von HTML5 zur Entwicklung von mobilen Applikationen erfreut sich großer Beliebtheit. Der Vorteil liegt darin, eine Applikation für verschiedene Endgeräte mit unterschiedlichen Betriebssystemen nur ein einziges Mal entwickeln zu müssen. In manchen Situationen kann ein permanenter Zugang zum Internet nicht garantiert werden. Der Zugriff zu oftmals benötigten Reverse-Geocoding-Diensten kann daher nicht vorausgesetzt werden. In diesem Artikel wird die Umsetzung eines Reverse-Geocoders für den HTML5-Offlinebetrieb dargestellt und Erweiterungsmöglichkeiten beschrieben.

1 Ausgangssituation

Die Implementierung einer mobile Applikation auf HTML5-Basis, welche Touristen historische Informationen zu jener Straße anzeigt, in der sie sich gerade aufhalten, hat ein interessantes Problem aufgezeigt: Die für die Bestimmung des Straßennamens notwendigen Reverse-Geocoding-Services setzen eine bestehende Internetverbindung voraus, welche aufgrund der touristischen Natur der Applikation (Benutzung im Ausland) nicht immer als gegeben angenommen werden kann. Es galt, dieses Problem zu lösen.

1.1 Mobilität und Datenverfügbarkeit

Der Nutzen moderner Smartphones hängt entscheidend von der Verfügbarkeit bestimmter Daten ab. Im Regelfall stellt dies durch die angestrebte permanente Verbindung zum Internet kein Problem dar. Nicht nur der Bezug von Informationen nach Bedarf, sondern selbst Datenspeicherung wird bereits in das Netz ausgelagert.

Leider ist die kosteneffiziente Verfügbarkeit eines Internetzuganges nicht in allen Situationen gegeben. Besonders im Ausland muss man auf eine durchgehende Verbindung zum World Wide Web verzichten. Dies ist eine besondere Herausforderung für Applikationen, welche Touristen als Zielgruppe vorsehen. Um eine praktische Anwendbarkeit verschiedener angebotener Services sicherzustellen, muss Bedacht darauf genommen werden, auch im Falle eines Verbindungsausfalles alle Daten bereitzuhalten.

1.2 Hybride HTML5-Lösungen

Wie SALEH (2014) in seinem Vorwort beschreibt, verlangt das Entwickeln von mobilen Applikationen vom Entwickler ein breit gefächertes Spektrum an Fähigkeiten ab. Wenn ein

AGIT – Journal für Angewandte Geoinformatik, 1-2015. © Herbert Wichmann Verlag, VDE VERLAG GMBH, Berlin/Offenbach. ISBN 978-3-87907-557-7, ISSN 2364-9283, doi:10.14627/537557057.

und dieselbe Anwendung einen möglichst großen Markt bedienen soll, muss diese in verschiedenen Programmiersprachen für verschiedene Systeme implementiert werden.

Eine Lösung für dieses Problem stellen sogenannte hybride HTML5-Applikationen (HTML, JavaScript, CSS) dar. Ein vor dem User versteckter und um diese Web Applikation gepackter Browser stellt diese auf dem Endgerät dar. Mit der Einführung des HTML5-Standards (WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP 2015) werden eine Vielzahl an Möglichkeiten geboten, auf Sensoren und Funktionen eines mobilen Endgerätes zuzugreifen. Ein Nachteil dieser Variante ist die im Vergleich zu einer nativen Anwendung schlechtere Performance und Speichereffizienz.

2 Existierende Lösungen

Es ist üblich, Geocoding und Reverse-Geocoding als Dienste über das Internet zu beziehen. SCHMITT & SIMPSON (2011) bieten in ihrem Lehrbuch zu HTML5 auf Seite 154 beispielsweise folgende Lösung an: „Problem: You want to convert latitude and longitude coordinates into a human-friendly address. Solution: Use the Google Maps JavaScript API [...].“. Der Google Geocoder wird trotz seiner kapazitiven (2500 Anfragen pro Tag für private Nutzer) und legalen Einschränkungen (GOOGLE INC. (2013; Abschnitt 'Nutzungsbegrenzungen') schreibt vor: „Das Geocoding API darf nur in Verbindung mit einer Google-Karte verwendet werden, das heißt, das Abrufen von Geocodierungsergebnissen, die nicht auf einer Karte angezeigt werden, ist nicht zulässig.“) oft selbstverständlich angewendet, beispielsweise durch JAKKHUPAN (2014) auf Seite 315. Neben Google existieren bereits eine Vielzahl an proprietären und freien Anbietern mit ähnlichen Angeboten (Gisgraphy[1], OpenCage[2], der auf OpenStreeMap basierende Nominatim Geocoder (HOFFMANN 2013) und CloudMade wie von TESLYA (2014) beschrieben). Diese Dienste bieten zum Großteil eine höhere Granularität in der Adressauflösung (Hausnummern) an als im Einzelfall benötigt wird (Straßennamen). Die Nutzung ist jedoch ausschließlich als Web Service möglich und basiert auf Datenbanken, welche auf einem größeren Server im Hintergrund in Betrieb sein müssen.

Im Bereich der offline verfügbaren Geocoding und Reverse-Geocoding-Lösungen lassen sich aufgrund deren zugänglichen Implementierung exemplarisch drei Varianten einfacher (Reverse)-Geocoding-Lösungen in der Programmiersprache Python nennen[3]. Diesen Implementierungen ist gleich, dass sie einen KD-Tree zur Datenorganisation und Suche verwenden, welcher ausschließlich mit Punktinformationen arbeitet. Dies und die Tatsache, dass gewisse Bedenken, die bei mobilen Anwendungen von Bedeutung sind (On-Demand-Performance, Speicherbedarf, Arbeitsspeicher), bei einer nicht mobilen Umsetzung aber eine kleinere Rolle spielen, gibt diesen Beispielen nur eine eingeschränkte Vorbildfunktion für die in diesem Artikel angestrebte Implementierung.

---

1 http://www.gisgraphy.com/documentation/user-guide.htm#streetservice
2 http://geocoder.opencagedata.com/api.html
3 http://geocoder.readthedocs.org
https://bitbucket.org/richardpenman/reverse_geocode
https://github.com/thampiman/reverse-geocoder

DEIDDA et al. (2013) entwickelten eine mobile Software, welche speziell an Touristen gerichtet ist und ebenfalls eine Geocoding-Engine benötigt. Sie stellen zwar einen eigenen Geocoder zur Verfügung, dieser wird aber als Web Service an die mobile Applikation gebunden. Die Verfügbarkeit ist abermals durch das Existieren einer Internetverbindung eingeschränkt.

3 Eigene Lösungen

Der Quellcode für die hier beschriebene Methode kann unter *https://github.com/scubbx/mobile-reverse-geocoder* eingesehen werden.

Eine entscheidende Erleichterung für das Implementieren einer offline Reverse-Geocoding-Funktionalität stellt die Erkenntnis dar, dass in vielen Fällen der Straßenname als Ergebnis ausreicht. Nachdem zunächst ein rasterbasierter Ansatz angestrebt wurde, hat sich eine vektorbasierte Lösung als praktikabler herausgestellt.

Unabhängig von der gewählten Variante des Reverse-Geocoders ist das Ermitteln der eigenen Position in jedem Fall notwendig. HTML5 bietet dafür ein API an (POPESCU 2014). Die eigene Position wird dabei aus GPS, Netzwerksignalen, IP Adresse, RFID, WIFI, Bluetooth und CellIDs berechnet. Allerdings hat man keine Garantie, dass die erhaltene Position tatsächlich die derzeitige Position des Users widerspiegelt. So beschreibt etwa HOLDENER (2011) auf Seite 38 die „cache“ Funktionalität, welche bei oftmaligem Abfragen der Position einen zwischengespeicherten Positionswert zurückliefert um Ressourcen zu schonen. Bei einer Auflösung auf Straßenlevel ist die Aktualität allerdings ausreichend.

3.1 Rasteransatz

Der erste Ansatz war, das HTML5 „Canvas“ Element (BOULOS et al. 2010) zu benutzen und die Geocodierungsinformation in eine Rastergraphik zu kodieren um sich die für Bilddaten existierenden Kompressionsalgorithmen zu Nutze zu machen (vgl. Abb. 2, links). Dafür wurde eine georeferenzierte Rasterkarte erstellt, welche pro Pixel den Indexwert der jeweils entsprechenden Straße aufweist. Eine Tabelle weist jeder ID einen Straßennamen zu. Für eine hohe Effizienz der Bildkompression ist es vorteilhaft, große und homogene Flächen mit gleichen Werten zu erzeugen.

Ein 8-Bit-Rasterdatensatz kann pro Pixel 255 verschiedene Einzelwerte annehmen. Dies limitiert die Anzahl unterschiedlicher Straßen, die solch ein Datensatz repräsentieren kann. Die Verwendung eines 16-Bit-Rasters ist aufgrund von Einschränkungen des HTML5 „Canvas“ Elements nicht möglich: Dieses kann in jedem Fall nur Werte im 8-Bit-Bereich auslesen, wie im Abschnitt 4.12.4.2.16 von der WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP (2015) beschrieben ist. Man kann dieses Verhalten umgehen, indem man ein Mehrkanalraster verwendet. Ein normales 3-Kanal-Farbenraster kann somit $8\times8\times8$ verschiedene Bits pro Pixel annehmen, unter Einbeziehung des Alphakanals sogar $8\times8\times8\times8$. Ein einfacher ID Wert muss auf die vier 8-Bit-Kanäle aufgeteilt und bei einer Abfrage wieder zusammengesetzt werden.

Trotz der schlussendlich erwiesenen Machbarkeit wurde von dieser Methode abgesehen, da die erhoffte Speicherreduktion durch Kompression nicht ausreichend hoch ausfällt (unkomprimiert: 51,6 MB, PNG Kompression: 12,9 MB). Auch benötigt das notwendige 4-Band-Rasterbild den vierfachen Speicher des ursprünglich erwarteten 1-Band-Graustufenbildes.

3.2 Vektoransatz

Als Ausgangsdatensatz dienen alle Straßen Wiens des OpenStreetMap-Projekts (RAMM & TOPF 2010), welche eine Namensbezeichnung aufweisen.

3.2.1 Reduktion der Datenmenge

Die Reduktion der Datenmenge erfolgte durch zwei Methoden: Wahl des Datenformates und Generalisierung. Für eine schnelle Implementierung wurde zunächst das in JavaScript gebräuchliche GeoJSON Format gewählt. Dieses kann, was für das Generieren eines später benötigten R-trees vorteilhaft ist, vorberechnete Bounding Boxes beinhalten. Durch Entfernen von nicht benötigten Attributen konnte die Dateigröße merklich reduziert werden. Die größte Reduktion an Speicherbedarf konnte durch Umstieg auf das CSV-Format erreicht werden (von ca. 5 MB im JSON-Format auf 3,5 MB im CSV Format). Dieses ist zwar immer noch ASCII basiert, verzichtet aber auf unnötige Steuerzeichen (vgl. Abb. 1).

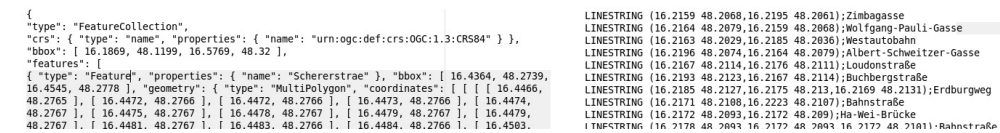

Abb. 1: Repräsentation der Straßen als GeoJSON (links) und als CSV mit WKT (rechts)

In einem weiteren Schritt wurden die verwendeten Objekte mithilfe des Douglas-Peu ker Algorithmus mit einem Minimalabstand von 10 Metern generalisiert (von 3,5 MB auf 2,2 MB). Zu guter Letzt wurden beim Abspeichern die verbleibenden Koordinaten nach 4 Nachkommastellen abgeschnitten. Dies stellt eine ausreichend hohe Genauigkeit dar (von 2,2 MB auf 1,2 MB).

Eine nicht weiter verfolgte Reduktion kann durch Kompression des endgültigen Datensatzes erreicht werden. Das bzip2 Format, für welches auch JavaScript Implementierungen existieren, konnte eine Datei mit einer Größe von 182,3 KB erzeugen.

3.2.2 Methoden

Alle Methoden basieren auf einem R-tree, einem Index für räumliche Abfragen. Es wird die derzeitige Position des Benutzers bestimmt, um daraufhin mit dieser mögliche Überschneidungen mit den Bounding Boxes von Objekten (Straßen) innerhalb des R-trees zu ermitteln. Falls mehrere Objekte gefunden werden, werden diese einer genaueren Überprüfung unterzogen (es wird nicht nur deren Bounding Box überprüft, sondern die tatsächliche Geome-

trie). Für diese Überprüfung und andere notwendige GIS Operationen wird die JavaScript Library TurfJS [4]verwendet.

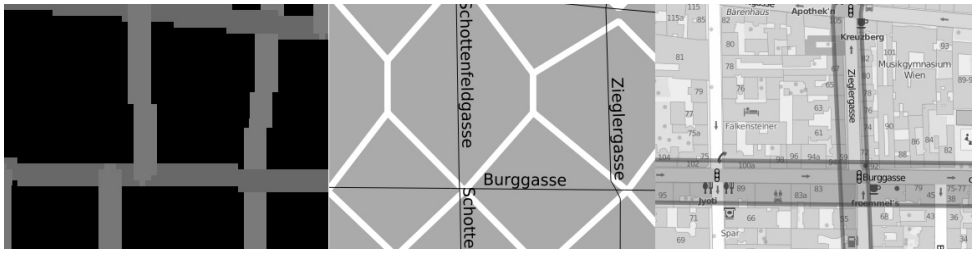

Abb. 2: IDs codiert in Rasterwerte (links), Voronoi-Flächen der Straßen (Mitte), auf Bedarf berechnete generalisierte Puffer-Zonen (rechts)

In einem ersten Versuch wurde um alle Straßenpolygone ein Puffer mit 20 Metern erzeugt. Dieser Puffer stellt jene Zone dar, in welcher eine Zuordnung zur Straße erfolgt. Um eindeutigere Ergebnisse zu erzielen, wurden nicht eindeutig zuordenbare Flächen (beispielsweise an Kreuzungen) entfernt. Das Verwenden von vorberechneten Pufferzonen reduziert den Berechnungsaufwand am Gerät selbst, da nun nur mehr überprüft werden muss, ob die eigene Position innerhalb einer Pufferzone liegt. Dies erzeugt jedoch einen unnötig vergrößerten Datensatz, da pro Straßenzug nicht mehr nur eine Linie, sondern ein ganzes Polygon gespeichert werden muss.

Als Nächstes war die Überlegung, um die Straßenlinien Voronoi-Diagramme zu berechnen (vgl. Abb. 2 Mitte). Die Datenmenge sollte sich auf etwa gleichem Niveau wie bei ausschließlicher Nutzung der Straßen bewegen. Es konnte in den verwendeten Open-Source-Tools jedoch keine zufriedenstellende Implementierung eines Voronoi-Algorithmus für Linien gefunden werden. Zugunsten der dritten Methode wurde darauf verzichtet, diesen Weg weiter zu verfolgen.

Der letzte Versuch bestand darin, die Straßenlinien in generalisierter Form als solche zu belassen und die Pufferzonen für die Abfrage nur bei Bedarf am Gerät selber zu erzeugen (vgl. Abb. 2 rechts). Hierzu kommt die Eigenschaft des R-trees, nach Bounding Boxes abzufragen, zu tragen: Auch ohne Berechnung der Pufferzone können durch die Bounding Box bereits jene Straßen ermittelt werden, welche sich möglicherweise mit der derzeitigen Position schneiden. Wichtig ist, bei der Erstellung des R-trees die Bounding Boxes der Straßen um etwa 20 Meter zu erweitern, da ansonsten an den Endpunkten oder bei nahezu orthogonaler Lage und Form des Objektes keine Erkennung erfolgt. Für die aus dieser Abfrage resultierenden Objekte wird nun eine Bounding Box berechnet und überprüft, welche Straße sich tatsächlich mit der derzeitigen Position schneidet.

Es zeigte sich, dass die dritte Methode, welche den Straßendatensatz im Vorfeld am wenigsten verändert, trotz des erhöhten Prozessierungsaufwands am Endgerät, ohne Probleme funktioniert und auch die kleinste Dateigröße aufweist.

[4] http://turfjs.org

4 Weitere Überlegungen

Ausgehend von der hier beschriebenen Implementierung lassen sich weitere Verbesserungen oder Optimierungen vornehmen.

Datenhaltung: In der derzeitigen Implementierung wird der gesamte Datensatz aus einer Datei geladen und die notwendigen Indizes beim Programmstart im Arbeitsspeicher generiert. Es ist anzustreben, HTML5 Speichertechniken zu benutzen. Nichtsdestotrotz kann durch die geringe Dateigröße beispielsweise leicht eine Aktualisierung des Datensatzes erfolgen.

Doppelte Straßennamen: Falls mehrere unterschiedliche Straßen mit gleichem Namen existieren, ließe sich durch den Einsatz einer Graphendatenbank und dem Mitverfolgen der zuletzt benutzten Straße eine eindeutige Zuweisung ermitteln. Für JavaScript existieren beispielsweise die Bibliotheken HelioJS[5] und LevelGraph[6]. Während die erste einen Graphen im Arbeitsspeicher des Gerätes aufbaut und somit bei eher kleinen Datenmengen infrage kommt, verwendet die zweite einen persistenten Speicher mit wesentlich höherer Kapazität.

Geocoding: Der hier beschriebene Prozess ermöglicht ein effizientes Reverse-Geocoding. Im Falle von Geocoding wäre ein weiterer Index notwendig. Im einfachsten Fall würde dieser aus einer alphabetisch sortierten Liste aller Straßennamen bestehen, auf die ein Suchwort angewendet wird.

Routing: Bei Einsatz einer Graphendatenbank ist die Umsetzung eines einfachen Routenplaners denkbar. Da die Daten als Graph vorliegen, kann mit wenig Umstand eine kürzeste Route zwischen zwei Punkten ermittelt werden. Diese Route kann als Abfolge von Straßennamen, die es zu durchqueren gilt, ausgegeben werden. Dies wäre eine vor allem für Fußgänger denkbare Anwendung.

Höhere Granularität: Die Granularität des zugrunde liegenden Datensatzes ist nicht auf Straßen beschränkt. Jede einzelne Straße kann beliebig zerschnitten und attribuiert werden. Hierbei gilt es, ein Optimum zwischen Nutzen und Leistung zu erreichen.

5 Conclusio

Mit dieser Implementierung konnte gezeigt werden, dass die performante Umsetzung eines HTML5 basierten Reverse-Geocoders für mobile Endgeräte machbar ist. Vor allem im Bereich von Tourismusapplikationen muss unbedingt Bedacht auf fehlende Internetverbindung genommen werden. Das Datenvolumenkonnte ausreichend stark reduziert werden. Auch hat sich gezeigt, dass selbst bei Einsatz von HTML5 heutige Smartphones sehr leistungsfähig sind. Die hier vorgeschlagene Methode kann leicht um weitere Fähigkeiten erweitert werden, der Prozessierungsaufwand steigt dadurch allerdings. Die Schwachstelle der beschriebenen Methode ist der räumliche Index, welcher ausschließlich im Arbeitsspeicher angelegt wird. Obwohl dessen Erstellung pro Programmstart nur etwa 2 Sekunden in

[5] http://entrendipity.github.io/helios.js
[6] https://github.com/mcollina/levelgraph

Anspruch nimmt, bringt dies Smartphones mit weniger als 1 GB an Arbeitsspeicher an ihre Grenzen. Es sollte möglich sein, die benötigten Datenstrukturen auf eine persistente Speicherebene auszulagern. Dies würde die Erstellung weiterer Indizes ermöglichen, um weitere Funktionalitäten zu implementieren.

Literatur

BOULOS, M., WARREN, J., GONG, J. & YUE, P. (2010), Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. International Journal of Health Geographics, 9, 14.

DEIDDA, M., PALA, A., & VACCA, G. (2013), An example of a tourist location-based service (LBS) with open-source software. Applied Geomatics, 5, 73-86.

GOOGLE INC. (2013), Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/?hl=de (31.03.2015).

HOFFMANN, S. (2013), Der OpenStreetMap-Geocoder Nominatim. https://www.youtube.com/watch?v=f-lGGpuUtOw (31.03.2015).

HOLDENER, A. T. (2011), HTML5 Geolocation. O'Reilly Media Inc., Sebastopol.

JAKKHUPAN, W. (2014), A Prototype of Mobile Speed Limits Alert Application Using Enhanced HTML5 Geolocation. Computational Collective Intelligence. Technologies and Applications.

POPESCU, A. (2014), Geolocation API Specification. http://dev.w3.org/geo/api/spec-source.html (31.03.2015).

RAMM, F. & TOPF, J. (2010), OpenStreetMap: die freie Weltkarte nutzen und mitgestalten. Lehmanns Media, Berlin.

SALEH, H. (2014), JavaScript Mobile Application Development. Packt Publishing, Birmingham.

SCHMITT, C. & SIMPSON, K. (2011), HTML5 Cookbook. O'Reilly Media Inc., Sebastopol.

TESLYA, N. (2014), Web mapping service for mobile tourist guide. In: Proceedings of 15th Conference of Open Innovations Association FRUCT, 135-143.

WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP (2015), HTML Living Standard – Pixel Manipulation. https://html.spec.whatwg.org/multipage/scripting.html#pixel-manipulation (31.03.2015).