UNIVERSITÄT OSNABRÜCK

Fachbereich Mathematik/Informatik

Studiengang Master of Science Geoinformatik

Studienprojekt Geoinformatik

FART 4 Mobile

Fast Augmented Reality Thin-Client for Mobile environments

Leitung: Dr. rer. nat. Bernhard Höfle

Osnabrück, 30. September 2010

Die Dokumentation des Projektes wurde unter Mitwirkung folgender Studenten des Studiengangs Master of Science Geoinformatik an der Universität Osnabrück durchgeführt:

Petra Burmester

Robert Daniels

Florian Hillen

Hannes Holm

Simon Schröder

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ziele eines Studienprojeks

Das Hauptziel eines Studienprojektes ist die gemeinsame Bearbeitung eines Projektes in einer Gruppe. Dabei kommt es darauf an, dass die Gruppe sich zunächst gemeinsam einen Überblick verschafft und dann die Aufgaben sinnvoll unter sich aufteilt. Eine vernünftige Projektplanung ist dabei unerlässlich. Die Teilnehmer sollen dabei lernen, in einer Gruppe zu arbeiten, größere Problemstellungen zu bewältigen und die Ergebnisse zu dokumentieren und zu präsentieren. Die Dokumentation sollte wissenschaftlichen Kriterien entsprechen, d.h. argumentative Begründungen der Vorgehensweise sowie wissenschaftliche korrekte Zitierweise.
Das Studienprojekt verbindet dabei Theorie und Praxis. Die Teilnehmer entwickeln ihr Projekt eigenständig, lediglich das (grobe) Projektziel ist vom Dozenten vorgegeben. Dieser ist an den Diskussionen zur Projektplanung beteiligt und kann ggf. lenkend eingreifen. Auch notwendige Hintergrundinformationen kann der Dozent im Rahmen eines Studienprojektes liefern. Die Teilnehmer arbeiten sich dann aber selbständig in die Thematik hinein. Dafür werden möglicherweise vergleichbare Projekte und sinnvoll erscheinende Systemkomponenten auf deren Einsatzmöglichkeiten untersucht.
Regelmäßige Projekttreffen werden abgehalten, bei denen jeder Teilnehmer über seinen aktuellen Stand berichtet. So können z.B. Schwierigkeiten bei der Bearbeitung besprochen werden. Außerdem sind so alle Teilnehmer immer auf einem aktuellen Stand, das Gesamtprojekt betreffend. Änderungsvorschläge zum Aufbau des Projekts können diskutiert und ggf. umgesetzt werden. Das Projekt sieht sich i. d. R. während der Umsetzung ständigen Veränderungen ausgesetzt. Die Teilnehmer erlangen neue Erkenntnisse oder haben Verbesserungsvorschläge, die direkt in die Umsetzung einfließen können. Auch kann sich der Stand der Technik im Laufe eines Projekts weiterentwickeln, so dass das Projekt sich möglicherweise daran anpassen muss.

Ziele des diesjährigen Studienprojekts

Den Teilnehmern wurde ein Forschungsauftrag erteilt: „Erstellung eines Prototypen für die Zusammenführung von geokodierten Bildern eines mobilen Klienten (Geotagging) und 3D-GIS-Daten in einem Thin-Client Augmented Reality System“ (Forschungsauftrag siehe Anhang). Dabei sollen ausschließlich Open Source bzw. freie Komponenten eingesetzt werden. Die Wahl des Clients soll offen gestaltet werden, aber auf jeden Fall für moderne mobile Endgeräte geeignet sein. Außerdem soll die Umsetzung auf aktuelle Geodatenstandards und Services aufbauen.

2 Beschreibung der Daten

2.1 Testgebiet

Das Testgebiet liegt im städtischen Bereich von Osnabrück in Niedersachsen. Die Abgrenzungen des Gebietes in Gauß-Krüger-Koordinaten lauten:

Nord: 5795658 m

Süd: 5792058 m

West: 3432000 m

Ost: 3437274 m

Für diesen Bereich liegen sowohl Sachdaten als auch Höhendaten vor. Die Sachdaten konnten aus der freien Weltkarte, OpenStreetMap, exportiert werden und liegen als Shape-Dateien vor.

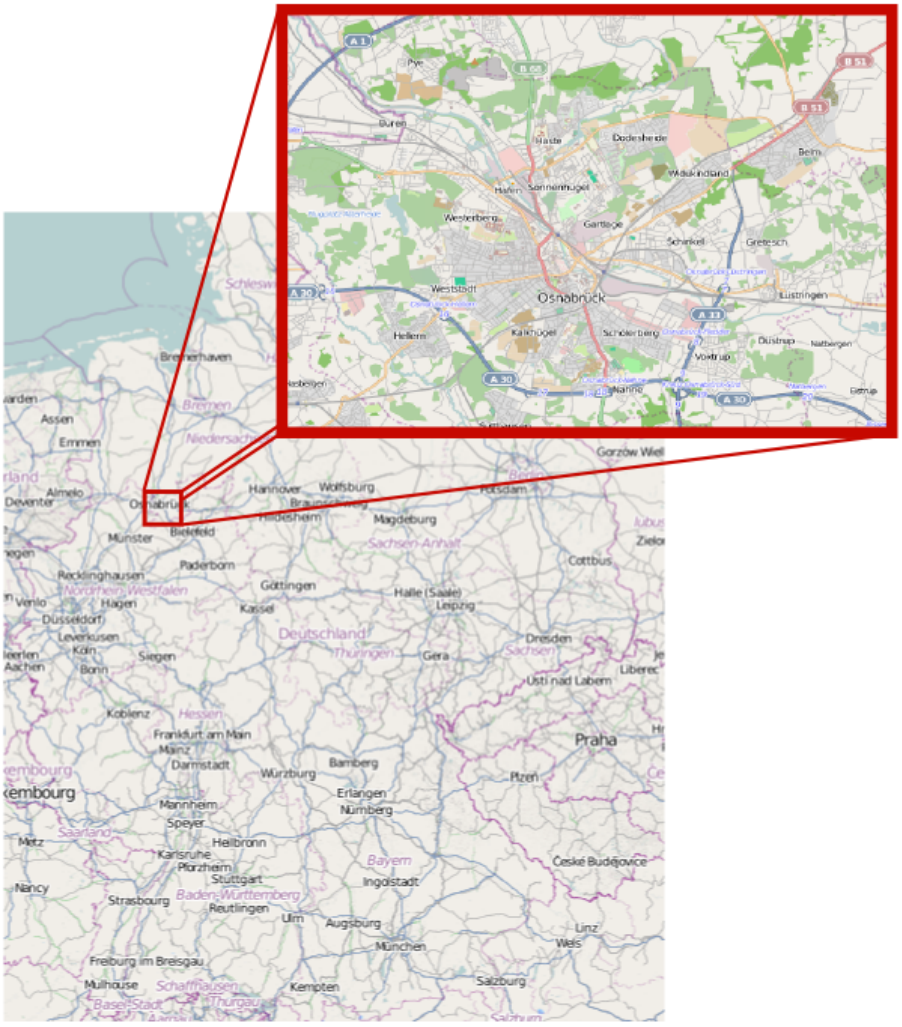


Abbildung 1: *räumliche Einordnung des Untersuchungsgebietes*

Quelle: OpenStreetMap

2.2 Sachdaten

Als Datengrundlage für die Sachdaten dient das Projekt Open Street Map[Fou10d], welches Geodaten für jedermann frei zur Verfügung stellt. Durch den Export dieser Daten erhält man folgende Datenschichten (s. Abb. 2):

- Buildings (Gebäude)
- Natural (Naturflächen)
- Places (Orte)
- Points (Punkte)
- Railways (Gleisverbindungen)
- Roads (Straßen)
- Waterways (Wasserwege)

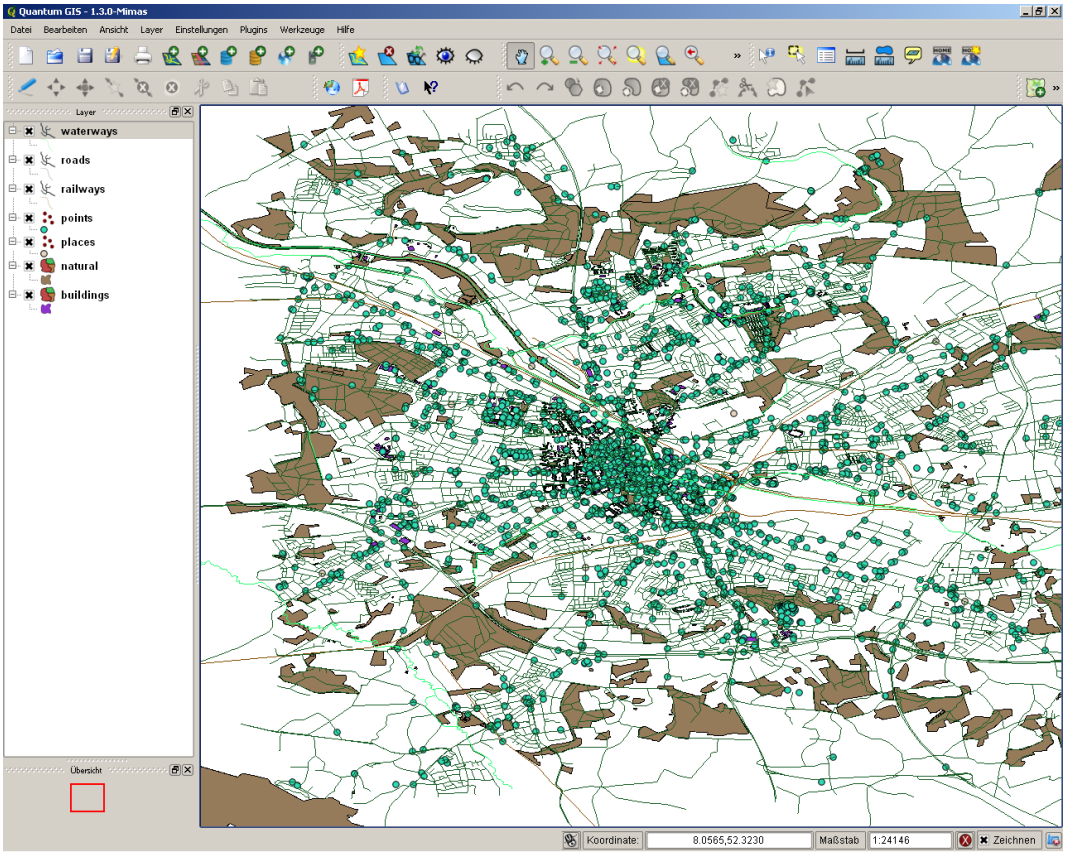


Abbildung 2: *Exportierte Open Street Map Daten dargestellt in Quantum GIS 1.3.0*
Quelle: eigene Darstellung

Für das Projekt ist jedoch lediglich die Datenschicht der Punkte relevant, sodass alle anderen Daten vernachlässigt werden können.

Die Daten liegen originär in geographischen Koordinaten (Einheit: Grad) im Referenzsystem WGS84 vor. Da die Höhendaten bereits in Gauß-Krüger-Koordinaten vorliegen, müssen die Sachdaten zur weiteren Verarbeitung ebenfalls in Gauß-Krüger-Koordinaten überführt werden (s. Abb. 3).

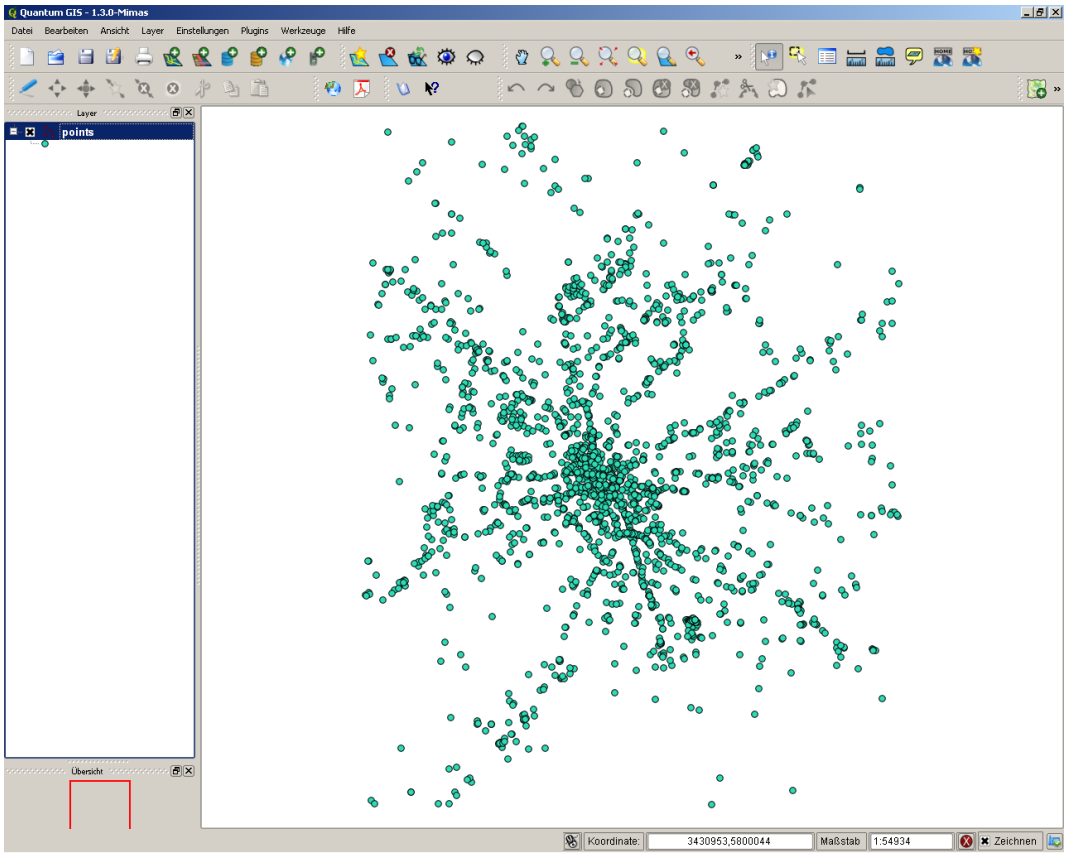


Abbildung 3: *Punktdaten in Gauß-Krüger-Koordinaten*
Quelle: eigene Darstellung

2.3 Höhendaten

Es liegen zwei verschiedene Modelle von Höhendaten aus Airborne Laserscanning Messungen vor. Zum einen ein Digitales Gelände-Modell (DGM, englisch: Digital Terrain Model (DTM)), welches die Höhen auf der Geländeoberfläche beschreibt und zum anderen ein Digitales Oberflächen-Modell (DOM, englisch: Digital Surface Model (DSM)), welches die absoluten Höhen eines Ortes, also z.B. auch von Gebäuden oder Bäumen, beschreibt.
Beide Modelle liegen im ASCII-Format vor, in welchem die Höhendaten nacheinander für jede einzelne Rasterzelle gespeichert werden (s. Abb. 4). Ebenfalls in dieser Textdatei enthalten sind die Informationen für die spätere Rasterdatei, welche beim Einlesen von besonderer Wichtigkeit sind:

- Anzahl der Spalten (*ncols*)
- Anzahl der Zeilen (*nrows*)
- X-Koordinate der unteren linken Ecke (*xllcorner*)
- Y-Koordinate der unteren linken Ecke (*yllcorner*)
- Zellengröße (*cellsize*)
- Wert für eine Zelle ohne Inhalt (*NODATA_value*)

```
ncols         5269
nrows         3593
xllcorner     3432001.500000000000
yllcorner     5792063.500000000000
cellsize      1.000000000000
NODATA_value  -9999
 86.830001831054688 84.980003356933594 82.129997253417969
74.540000915527344 71.699996948242188 70.980003356933594
72.680000305175781 74.300003051757813 74.650001525878906
74.139999389648437 72.669998168945313 70.540000915527344
70.55999755859375 74.080001831054687 72.180000305175781
70.629997253417969 71.019996643066406 70.569999694824219
69.989997863769531 69.930000305175781 70.599998474121094
70.160003662109375 70.529998779296875 69.699996948242188
69.330001831054688 69.419998168945313 69.349998474121094
69.599998474121094 69.30999755859375 69.410003662109375
69.459999084472656 69.269996643066406 69.269996643066406
69.279998779296875 70.639999389648437 72.709999084472656
74.709999084472656 70.589996337890625 76.400001525878906
75.430000305175781 74.620002746582031 73.279998779296875
71.040000915527344 69.239997863769531 69.239997863769531
69.230003356933594 69.30999755859375 69.80999755859375  69.25
69.699996948242188 69.370002746582031  69.25 69.540000915527344
69.599998474121094 69.400001525878906 69.30999755859375
69.209999084472656 70.69000244140625 72.599998474121094
72.639999389648438 72.230003356933594 70.629997253417969
70.230003356933594 70.900001525878906 70.389999389648438
69.680000305175781 69.910003662109375 71.779998779296875
```

Abbildung 4: *Das Digitale Oberflächen-Modell im ASCII-Format*
Quelle: eigene Darstellung

In einer extra Datei werden Definitionen über die Projektion der Höhendaten gespeichert. Es handelt sich hierbei um Gauß-Krüger-Koordinaten im Referenzsystem DHDN. Nach dem Import der ASCII-Dateien beträgt die Auflösung der Rasterdateien 1 Meter.
Für das Projekt werden lediglich die Höhendaten des Digitalen Oberflächen-Modells benötigt, da nur in diesem eine sinnvolle Sichtbarkeitsanalyse durchgeführt werden kann. In einer farblichen Darstellung sind die relevanten Höhendaten in Abbildung 5 dargestellt. Rot beschreibt hier höhere und grün tiefere Werte.

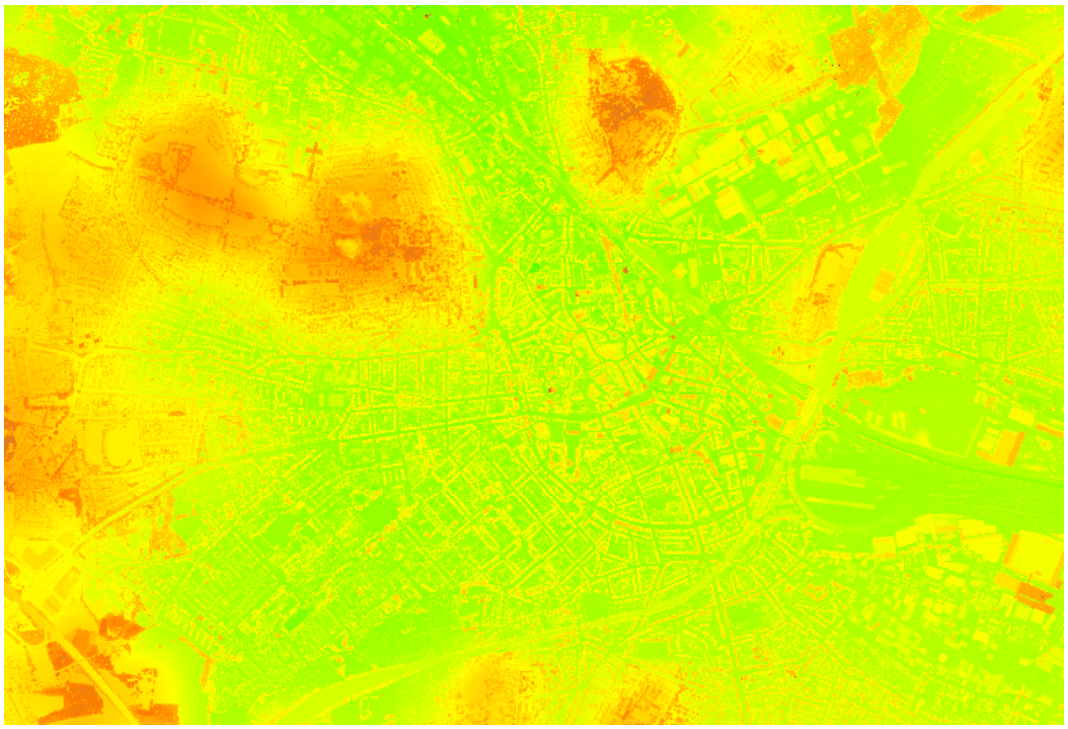


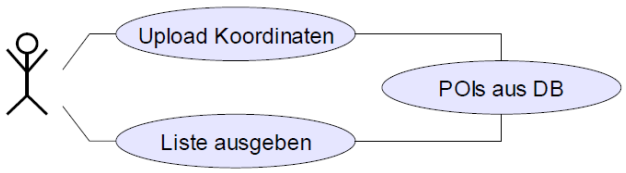
Abbildung 5: *Farbliche Darstellung des Digitalen Oberflächen-Modells*
Quelle: eigene Darstellung

3 Detailbeschreibung der Systemkomponenten

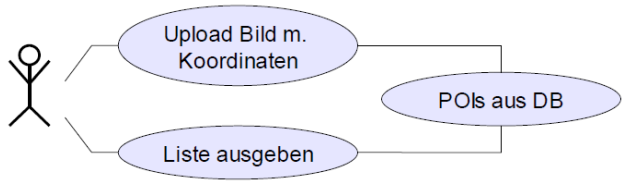
3.1 Use-Case-Diagramme

Im Arbeitspaket 2 wurden von den Projektteilnehmern Use-Cases aus den Ergebnissen von Arbeitspaket 1 abgeleitet, welche Prozesse und Abläufe aufzeigen, die das System unterstützen soll. Dazu wurden mehrere Entwicklungsphasen definiert:

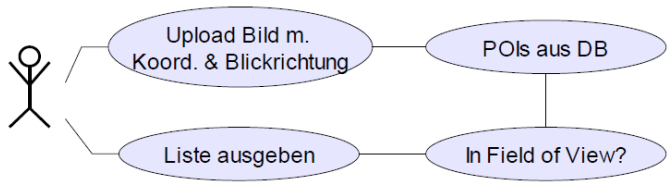
1. Dem User wird die Möglichkeit gegeben Koordinaten händisch über eine Schnittstelle zum System zu übermitteln. Das System sucht dazu alle POIs aus der Datenbank und gibt eine Liste an den User zurück.



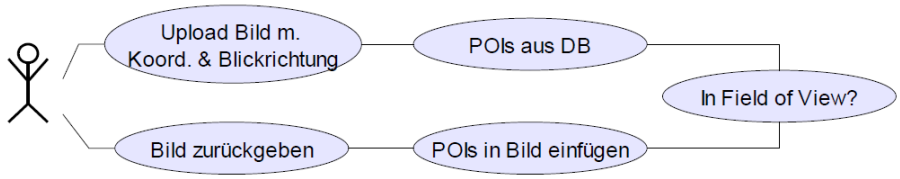
2. Im Weiteren kann der Anwender ein Bild, welches Koordinaten enthält an die Anwendung übergeben. Zurückgegeben wird wie in 1 eine Liste mit POIs aus der Datenbank.



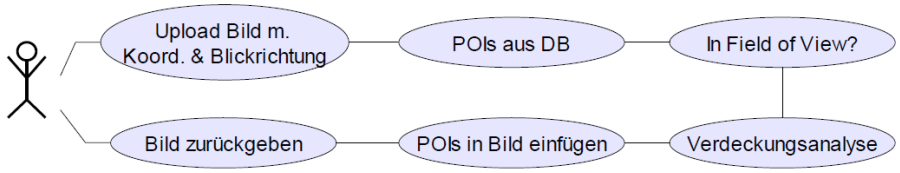
3. Die Anwendung wird erweitert um die Möglichkeit, dass das Bild sowohl Koordinaten als auch die Blickrichtung des Anwenders enthält. Dazu werden ebenfalls zunächst die entsprechenden POIs aus der Datenbank abgefragt, welcher jedoch eine Abfrage folgt ob die POIs in einem definierten Field of View liegen und somit zur Auszugebenden Liste hinzugefügt werden.



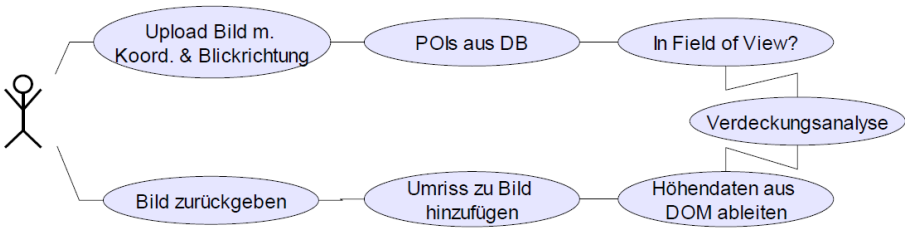
4. Im nächsten Schritt werden die Koordinaten nicht in einer Liste, sondern im Hochgeladenen Bild eingefügt an den Benutzer zurückgegeben.



5. In eine weiteren Entwicklungsstufe ermöglicht im Anschluss an die Field of View Abfrage, die Analyse ob die POIs durch Objekte verdeckt werden und diese nicht in das Bild eingefügt werden.



6. Die Letzte Stufe erweitert die Anwendung, um die Integration eines Digitalen Oberflächen Modells zur Ableitung von Höhendaten um im Weiteren Umrisse zum Bild hinzufügen zu können.



3.2 Systementwurf

Abbildung 6 zeigt den Systementwurf für FART4Mobile. Da Client-Software oft plattformabhängig und daher nur auf bestimmten mobilen Endgeräten lauffähig ist, wird FART4Mobile als Webanwendung aufgesetzt, die potentiell von jedem internetfähigen Endgerät unterstützt wird.

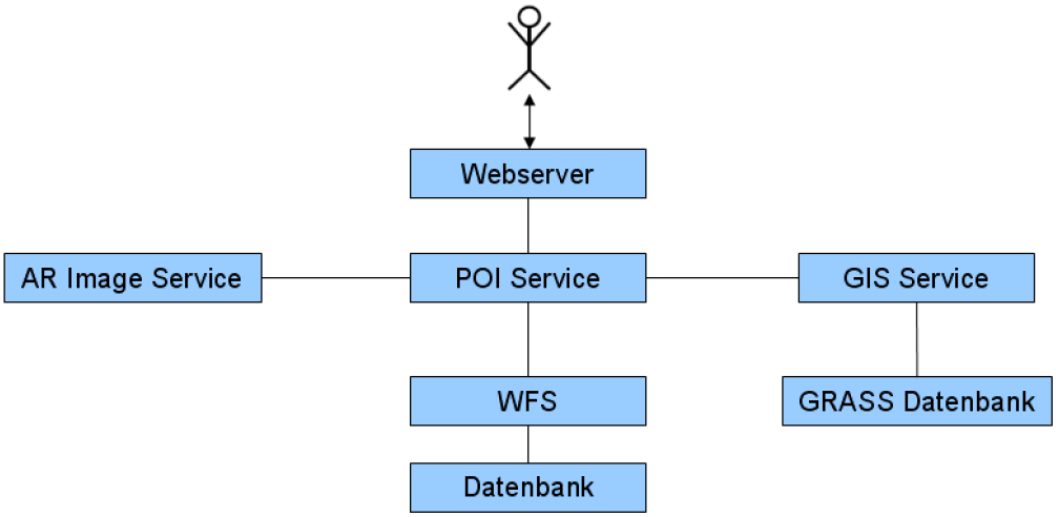


Abbildung 6: *Systementwurf FART4Mobile*

Der Benutzer interagiert dabei mit einer Internetseite die auf einem Webserver läuft. Hier müssen die notwendigen Daten, wie z.B. das zu bearbeitende Foto und die Kamerabrennweite, für die Weiterverarbeitung angegeben werden. Der POI Service nimmt diese Daten entgegen und startet mit diesen die Prozesskette. Zunächst werden dazu die verfügbaren POIs in einem bestimmten Umkreis um die Betrachterposition aus einem Web Feature Service [Ope10] ermittelt. Dieser Service des OGC bietet eine einheitliche Schnittstelle an und wurde daher vor die Datenbank geschaltet. So bleibt der Datenbestand einfach austauschbar und unabhängig von etwaigen datenbankspezifischen Unterschieden. Ein weiterer Vorteil ist, dass der WFS die Daten direkt im Standardaustauschformat GML ausliefert, das alle Services interpretieren können.
Die vom WFS übergebenen Daten werden nun vom POI Service an den GIS Service weitergeleitet. Dieser greift auf eine interne GRASS GIS Datenbank zu, in welcher die Höhendaten in geeigneter Form gespeichert werden und führt auf diesen Daten eine Sichtbarkeitsanalyse durch, um zu ermitteln, welche POIs vom Betrachter aus wirklich sichtbar sind. Ebenso ergänzt der GIS Service die fehlenden Höheninformationen der POIs, da diese für den AR Image Service benötigt werden. Der AR Image Service erhält nun die vollständigen POI Informationen und zeichnet die sichtbaren POIs in das Foto des Benutzers ein. Der POI Service erstellt die Ergebnisausgabe und liefert diese über den Webserver an den Benutzer zurück.

3.3 Webserver

Der Webserver stellt die Verbindung zwischen Anwender und dem POI Service her. Aufgabe des Webservers ist es, dem User, je nach Entwicklungsstufe die Möglichkeit zu geben, Koordinaten bzw. ein Bild mit Koordinaten hochzuladen. Der Webserver übermittelt diese Daten an den POI Service, welcher die weitere Verarbeitung vornimmt. Der Webserver wartet im Folgenden auf Rückmeldung und gibt anschließend die Liste mit POIs bzw. das Bild mit eingefügten POIs an den User zurück.
Für die Anwendung wurde der Apache HTTP Server gewählt, da dieser frei verfügbar ist. Apache ist ein Webserver, der auf dem HTTP-Protokoll basiert. Um ein Dokument aus dem Internet zu beziehen, baut der Client eine TCP/IP Verbindung zum Server auf und sendet einen HTTP-Request. Der Server verarbeitet den Request und gibt einen entsprechenden Response mit der angeforderten Datei aus [Heu].
Zur Installation wurde zunächst der Apache Webserver in der Version 2.2.14 heruntergeladen. Das Entpacken des Quellcodes aus dem Apache HTTPD Tarball besteht aus einem Dekomprimieren und danach „Ent-tarren“:

```
$ gzip -d httpd-2.2.14.tar.gz
$ tar xvf httpd-2.2.14.tar
```

Dies erstellt unterhalb des aktuellen ein neues Verzeichnis, das den Quellcode für die Distribution enthält.
Der nächste Schritt besteht aus der Konfiguration des Apache-Codebaumes für die spezielle Plattform und persönlichen Bedürfnisse. Dies wird mit dem Skript `configure` durchgeführt, welches im Wurzelverzeichnis der Distribution enthalten ist. An dieser Stelle wurde festgelegt, welche Funktionalitäten in den Apache aufgenommen werden sollen, indem Module aktiviert oder deaktiviert werden. Der Apache bindet standardmäßig einen Satz von Basismodulen ein. Andere Module werden mit Hilfe der Option `--enable-module` aktiviert, wobei `module` den Namen des Moduls ohne das Präfix `mod_` darstellt. Es können auch Module als „Shared Objects (DSOs)“ kompiliert werden, welche zur Laufzeit ge- und entladen werden. Dazu wird die Option `{enable-module=shared` verwendet. Entsprechend können Basismodule mit der Option `--disable-module` deaktiviert werden. Der Webserver wurde dazu wie folgt konfiguriert:

In diesem Fall wird LDAP dazu verwendet, die Webseite auf Uni-User zu beschränken. Dazu wurde im Verzeichnis /usr/local/apache2/htdocs eine .htaccess Datei mit folgendem Inhalt angelegt:

```
AuthName "Study project GI 2009 { Login"
AuthType Basic
AuthBasicProvider ldap
```

`./configure`	
`--with-ldap --enable-ldap`	
`--enable-authnz-ldap`	Nutzerbeschränkung einfügen
`--enable-headers`	HTTP-Request-und Response-Header kontrollieren und ändern.
`--enable-ssl`	Verschlüsselung ermöglichen
`--enable-rewrite`	regelbasierte URL-Manipulation
`--enable-static-htpasswd`	Erstellen und Aktualisieren der Flat-Dateien, um Benutzername und Passwort für die Authentifizierung von HTTP-Basic-Benutzer zu speichern
`--enable-http`	
`--enable-cgi --enable-cgid`	CGI-Skripte unterstützen
`-{enable-mods-shared=all`	Alle Module werden Geladen

```
AuthzLDAPAuthoritative off
AuthLDAPURL "ldaps://ldap-01.uos.de:636/ou
=people,dc=uni-osnabrueck,dc=de?uid"
require valid-user
```

Mit `$make` werden die verschiedenen Teile, die das Apache-Paket bilden, erstellt. Der Aufruf `$make install` installiert das Package unter dem konfigurierten Installations-Pfad: `/usr/local/apache2/`
Der Apache HTTP Server kann nun mit `$/usr/local/apache2/bin/apachectl start` gestartet werden. `$PREFIX/bin/apachectl stop` beendet den Server [Fou09b].
Um den Zugriff von außen zu ermöglichen, wurden die Einstellungen der Firewall (iptables) des Servers durch folgende Aufrufe verändert:

```
/sbin/iptables -I INPUT -p tcp --dport 80 -j ACCEPT
cp /etc/sysconfig/iptables /etc/sysconfig/iptables.old
/sbin/iptables-save > /etc/sysconfig/iptables
/sbin/service iptables restart
```

Dieser Aufruf schaltet das TCP/IP Protokoll auf Port 80 für den Server frei.
Damit der HTTP Server auf alle Domains und somit sowohl von localhost als auch über die IP Adresse oder den Domainnamen angesprochen werden kann, wurden folgende Änderungen vorgenommen:

```
#Edit configuration file
/usr/local/apache2/conf/httpd.conf
Change line 40 to:
Listen *:80
```

3.4 Datenbank

Diese Datenbank dient der Speicherung und Verwaltung von Sach- und Geodaten bestimmter Orte mit hoher Bedeutung. Die Anfragen des POI-Services werden verarbeitet und die entsprechenden Daten zurückgeliefert. Die Umsetzung der Datenbank findet mittels der räumlichen Erweiterung PostGIS[Res10] der Datenbank PostgreSQL[Gro10] statt. Die POIs liegen als Shapefiles (ESRI) vor und können über den Befehl shp2pgsql in die Datenbank importiert werden.

3.4.1 Installation

Im Folgenden soll eine Übersicht der benötigten Softwarepakete gegeben werden, die installiert werden müssen, um eine solche Datenbank realisieren zu können. Die jeweiligen Versionen entsprechen dem aktuellen Stand zu Projektbeginn.

PostgreSQL

Die wichtigste Komponente wird zuerst installiert. Dazu wird der PostgreSQL Source Code der Version 8.4.2 heruntergeladen und entpackt:

```
wget http://wwwmaster.postgresql.org/
     redir/56/f/source/v8.4.2/postgresql-8.4.2.tar.bz2
tar xfvj postgresql-8.4.2.tar.bz2
```

Nachfolgend wird das Configure-Script ausgeführt und zusätzlich der Python- und XML-Support hinzugefügt:

```
cd postgresql-8.4.2
./configure --with-python --with-libxml
```

Anschließend wird der Build angelegt und die Installation ausgeführt.

```
gmake
sudo gmake install
```

Die Installation von PostgreSQL ist hiermit abgeschlossen. Bevor die räumliche Erweiterung PostGIS installiert werden kann, muss zunächst die Cartographic Projections Library (PROJ.4[War10b]) und die Geometry Engine Open Source (GEOS[War10a]) installiert werden. Es handelt sich hierbei um Bibliotheken, die von PostGIS benötigt werden um dessen vollen Funktionsumfang ausschöpfen zu können.

PROJ.4

Die PROJ.4-Bibliothek ist eine Sammlung kartographischer Projektionen und ermöglicht beispielsweise die Transformation von Koordinaten. Auch hier muss zunächst der PROJ.4 Source Code der Version 4.7.0 heruntergeladen und entpackt werden:

```
wget http://download.osgeo.org/proj/proj-4.7.0.tar.gz
tar xfvz proj-4.7.0.tar.gz
```

Im Anschluss daran wird das Configure-Script ausgeführt:

```
cd proj-4.7.0
./configure
```

Zuletzt wird der Build angelegt und die Installation ausgeführt:

```
gmake
sudo gmake install
```

GEOS

Die GEOS-Bibliothek ist eine Kollektion geometrischer Funktionen und Operationen. PostGIS wird dadurch um Funktion zur räumlichen Analyse erweitert. Der Source Code der GEOS-Bibliothek (Version 3.2.0) wird ebenfalls heruntergeladen und entpackt:

```
wget http://download.osgeo.org/geos/geos-3.2.0.tar.bz2
tar xfvj geos-3.2.0.tar.bz2
```

Daraufhin wird das Configure-Script ausgeführt:

```
cd geos-3.2.0
./configure
```

Abschließend wird der Build angelegt und die Installation ausgeführt:

```
gmake
sudo gmake install
```

Die Bibliotheken PROJ.4 und GEOS sind nun installiert.

PostGIS

Mit Hilfe der Erweiterung PostGIS, können den Datenbankobjekten räumliche Informationen zugeordnet werden. Der Source Code der Version 1.4.1 wird heruntergeladen und entpackt:

```
wget http://postgis.refractions.net/download/postgis-1.4.1.tar.gz
tar xfvz postgis-1.4.1.tar.gz
```

Danach wird das Configure-Script ausgeführt:

```
cd postgis-1.4.1
./configure
```

Abschließend wird der Build angelegt und die Installation aufgerufen:

```
gmake
gmake install
```

Die Installation der POI-Datenbank ist hiermit vollständig abgeschlossen. Nichtsdestotrotz ist die nachträgliche Installation von PostgreSQL-Erweiterungen oder PostGIS-Bibliotheken möglich.

3.4.2 Daten importieren

Die Datenbank ist erfolgreich installiert und soll nun mit Daten versorgt werden. Diese entstammen dem Projekt OpenStreetMap[Fou10d] und liegen als Shapefile vor (points.shp). Nachfolgend soll erläutert werden, wie diese Daten in die Datenbank importiert werden können und worauf dabei geachtet werden sollte.

Vorbereitung der Daten

Die optische Sichtung der Daten bzw. ihrer Attributtabelle verschafft relativ schnell einen Eindruck über die Qualität der Daten. Dazu kann eine Open-Source-Software wie Quantum GIS 1.4.0 [Fou10c] benutzt werden. Zum Beispiel weist die Attributtabelle unseres Shapefiles diverse Einträge in der Spalte „name“ ohne Wert (NULL) auf. Diese Einträge sollen nicht importiert werden. Es besteht nun die Möglichkeit, die Einträge mit NULL-Wert manuell aus der Attributtabelle zu entfernen und das bereinigte Shapefile abzuspeichern. Eine alternative Methode bietet PostgreSQL in Form der Constraints an.
Damit ein fehlerfreies Importieren des Shapefiles in die Datenbank erfolgen kann, muss außerdem festgestellt werden, in welchem Koordinatensystem und mit welcher Zeichencodierung die Daten vorliegen.
Das Koordinatenreferenzsystem lässt sich in Quantum GIS 1.4.0 über 'Eigenschaften' ⇒ 'Metainfo' des jeweiligen Shapefiles ermitteln. In diesem Fall liegt die Datei points.shp im EPSG-Code 4326 (WGS84) vor.
Die Zeichencodierung des Shapefiles ist u.a. für die korrekte Darstellung von Umlauten notwendig. Damit der Import der Daten mit dem richtigen Zeichensatz erfolgen kann, muss dieser zunächst ermittelt werden. Dazu wird ein dBase-Import in OpenOffice.org Calc 3.2 durchgeführt. Zu diesem Zweck wird über 'Einfügen' ⇒ 'Tabelle aus Datei', die zum Shapefile zugehörige Datenbankdatei (points.dbf) eingefügt. Im Folgenden kann aus der Liste der Zeichensätze 'Unicode' (UTF-8) ausgewählt und abschließend mit 'OK' bestätigt werden. Bei korrekter Darstellung der Umlaute in der erzeugten Tabelle, liegt die Datenbankdatei und somit auch das Shapefile, im UTF-8 Zeichensatz vor. Bei verkehrter Darstellung der Umlaute, müssen die verschiedenen Zeichencodierungen während des dBase-Imports manuell ausprobiert werden.
Das geänderte Shapefile wird anschließend in das Datenverzeichnis von PostgreSQL kopiert. Dazu stellen wir eine Verbindung mit dem Datenbankserver her und melden uns als Benutzer

postgres an. Dieses wird mit dem Befehl `sudo su postgres` durchgeführt. Alle nachfolgenden Schritte finden als dieser Benutzer statt.
Zum Kopieren des Shapefiles sowie der notwendigen Dateien werden im Verzeichnis `/usr/local/pgsql/data/` folgende Befehle ausgeführt:

```
wget http://www.meinserver.de/points.shp
wget http://www.meinserver.de/points.shx
wget http://www.meinserver.de/points.dbf
```

Bevor das Shapefile importiert werden kann, muss zunächst noch die Arbeitsumgebung eingerichtet werden.

Einrichten der Arbeitsumgebung und Anlegen der Datenbank

Damit PostgreSQL auf die gemeinsamen Bibliotheken (Shared Libraries) zugreifen kann, muss folgender Suchpfad in der Umgebungsvariable gesetzt werden:

```
LD_LIBRARY_PATH=/usr/local/pgsql/lib
export LD_LIBRARY_PATH
```

Zusätzlich muss die Systemumgebung (Environment) angepasst werden. Diese Zeilen lassen sich auch in benutzerdefinierte Startskripte einfügen:

```
PATH=/usr/local/pgsql/bin:$PATH
export PATH
```

Auch für die Hilfeseiten sollte die Umgebungsvariable angepasst werden:

```
MANPATH=/usr/local/pgsql/man:
$MANPATH
export MANPATH
```

Damit sind die notwendigen Vorbereitungen abgeschlossen und die Datenbank kann angelegt werden. Zuvor muss der Datenbankserver mit dem folgenden Befehl gestartet werden:

```
pg_ctl -D /usr/local/pgsql/data -l /usr/local/pgsql/data/logfile start
```

Anschließend kann eine neue Datenbank angelegt werden (hier: poi):

```
createdb poi
```

Folgender Befehl öffnet die Datenbank poi:

```
psql poi
```

Shapefile importieren und transformieren

Zum Importieren des Shapefiles wird zunächst die Datenbank verlassen und anschließend mit dem folgenden Befehl das Shapefile in eine SQL-Syntax umgewandelt:

```
shp2pgsql -c -I -s 4326 -W UTF-8 points.shp public.points > points.sql
```

Nachfolgend sollen kurz die jeweiligen Auswirkungen der verschiedenen Optionen erläutert werden. Der Parameter `-c` sorgt dafür, dass eine neue Tabelle (hier: points) angelegt wird. Diese hat die selben Attribute wie die Attributtabelle des Shapefiles. Mit `-I` wird der sogenannte GiST-Index erzeugt, worauf später ausführlicher eingegangen werden soll. Das Koordinatenreferenzsystem wird mit Hilfe des Parameters `-s` und dem jeweiligen EPSG-Code angegeben. Die Zeichencodierung wird schließlich mit dem Parameter -W und dem jeweiligen Kürzel des Zeichensatzes bestimmt.
Die SQL-Datei kann nun in die Datenbank eingelesen werden:

```
psql -d poi -f points.sql
```

In der Datenbank `poi` wurde nun die Tabelle `points` angelegt, die gemäß den Einträgen des Shapefiles, die gewünschten Informationen und Koordinaten der POIs im Osnabrücker Stadtgebiet enthält. Das Koordinatenbezugssystem ist WGS84. Die Höhendaten liegen jedoch als Gauß-Krüger-Koordinaten vor. Zur Vereinheitlichung des Koordinatenreferenzsytems innerhalb des Projektes wird eine Koordinatentransformation der Tabelle points durchgeführt. Zu diesem Zweck wird die neue Tabelle `points_gk` erzeugt, welche das Koordinatenreferenzsystem DHDN3 (EPSG-Code 31467) hat.

```
create table points_gk as select gid, osm_id, name, type,
ST_Transform(the_geom,31467) as the_geom from points;
```

Die POIs liegen nun zusätzlich in der Tabelle points_gk im Gauß-Krüger-Koordinatensystem Zone 3 vor.

GiST Index

Der GiST (Generalized Search Trees) Index bietet die Möglichkeit Suchabfragen zu beschleunigen. Dabei werden insbesondere räumliche Abfragen unregelmäßiger Datenstrukturen unterstützt. Inwiefern eine Verbesserung bei der praktischen Anwendung unserer Datenbank entsteht, soll anhand von Laufzeitmessungen untersucht werden. Zu diesem Zweck wurde ein Datensatz inklusive GiST Index mit einem Datensatz ohne GiST Index verglichen.
Mit Hilfe dieser folgenden räumlichen Suchabfrage sollen alle POIs in einem Umkreis von etwa 100 Meter der Hausbrauerei Rampendahl ausgegeben werden:

```
SELECT asewkt(the_geom), name FROM points WHERE the_geom
&& SetSRID('BOX3D(8.0425331 52.2774499, 8.0441931 52.2791099)'::box3d,4326)
AND ST_Distance (the_geom,
ST_GeomFromText('POINT(8.0433631 52.2782799)', 4326)) < 0.00083;
```

Die Abfrage ergibt 10 POIs innerhalb der vorgegebenen Entfernung. Die gemittelten Laufzeiten betragen 15 ms, sowohl mit als auch ohne GiST Index. Der GiST Index führt hier zu keiner Leistungsverbesserung, was auf die geringe Größe des Datensatzes zurückzuführen ist [Eis03].

Tabellen ändern und Constraint anlegen

Abschließend soll die Tabelle um zusätzliche Spalten erweitert werden, um weitere Daten für den GIS Service und den AR Image Service bereitzustellen.
Die Spalte 'visible' wird für die Sichtbarkeitsanalyse benötigt und auf den zuvor festgelegten Defaultwert 1 gesetzt. Dieses kann durch folgende Befehle innerhalb der gestarteten POI-Datenbank erfolgen:

```
ALTER TABLE points_gk ADD COLUMN visible double precision;
ALTER TABLE points_gk ALTER COLUMN visible SET DEFAULT 1;
UPDATE points_gk SET visible = DEFAULT;
```

Außerdem wird die Spalte 'height' benötigt, um die Positionierung der Daten im Bild zu ermöglichen. Für diese Spalte wurde der Defaultwert -9999 festgelegt:

```
ALTER TABLE points_gk ADD COLUMN height double precision;
ALTER TABLE points_gk ALTER COLUMN height SET DEFAULT -9999;
UPDATE points_gk SET height = DEFAULT;
```

Schließlich wird noch die Spalte 'distance' zur Distanzberechnung gebraucht. Auch hier wurde der Defaultwert -9999 festgelegt:
Spalte distance:

```
ALTER TABLE points_gk ADD COLUMN distance double precision;
ALTER TABLE points_gk ALTER COLUMN distance SET DEFAULT -9999;
UPDATE points_gk SET distance = DEFAULT;
```

Wie bereits erwähnt besteht die Möglichkeit Einträge mit NULL-Wert automatisch zu entfernen bzw. nicht in der Tabelle zu speichern. Das kann mit Hilfe eines Constraints (eine vordefinierte Beschränkung) geschehen. In unserem Beispiel kann der NOT-NULL-Constraint angewendet werden:

```
ALTER TABLE points_gk ALTER COLUMN name SET NOT NULL;
```

Bei der nächsten Aktualisierung der Datenbank werden nur noch Einträge vorhanden sein, die in der Spalte 'name' einen Wert aufweisen, der nicht NULL ist [Eis03].

3.5 MapServer

MapServer ist eine in C geschriebene Open Source Geo-Rendering-Engine. Sie ermöglicht die Erstellung von Webkarten aus Geodaten, welche direkt an den User übertragen werden können. MapServer wurde ursprünglich von der University of Minnesota (UMN) in Zusammenarbeit mit der NASA und dem Minnesota Department of Natural Resources (MNDNR) für das ForNet Projekt entwickelt. Später wurde es vom TerraSIP Projekt, einem von der NASA unterstützten Projekt zwischen der University of Minnesota und einem Konsortium aus Landbewirtschaftungs- Interessentsverbänden, abgelöst. MapServer ist heute ein Projekt der Open Source Geospatial Foundation (OSGeo) und wird von einer wachsenden Zahl von Entwicklern (ca. 20) aus der ganzen Welt betreut. Diese werden von verschiedenen Organisationen im Bereich Weiterentwicklung und Wartung, unter Federführung des OSGeo MapServer Project Steering Committee, unterstützt.[Min10] Für diese Anwendung wurde der Web Feature Service verwendet.

3.5.1 Installation

Zur Installation des MapServer wurde zunächst von der Seite `http://download.osgeo.org/` die Installationsdatei mit dem Aufruf:
`wget http://download.osgeo.org/mapserver/mapserver-5.6.3.tar.gz`
heruntergeladen und im Verzeichnis `/mnt/fs4/studgi09/src/` gespeichert.
Im Weiteren wird die Datei mit `tar xzf mapserver-5.6.3.tar.gz` entpackt. Im Verzeichnis mapserver-5.6.3 wurden dann die GDAL Bibliotheken zum Verarbeiten von Rasterdaten, und OGR zum Verarbeiten von Vektordaten mit `yum install libgdal1-1.3.2` und `yum install libgdal1-1.3.2-dev` installiert.
Für die Konfiguration werden verschiedene Parameter mit `./configure` definiert. Um dem MapServer die Funktionalitäten des Web Feature Service zuzuweisen wird dieser durch die Aufrufe `--with-wfs` und `--with-wfsclient` hinzugefügt. Da die Daten in einer Postgis Datenbank vorgehalten werden, ist die Funktion `--with-postgis` einzubinden. Anschließend kann mit `make` der MapServer kompiliert werden. Das mapserv-binary wird nun nach `/usr/local/apache2/cgi-bin` kopiert. Dieser Schritt schließt die Installation der MapServers ab, sodass dieser einsatzbereit ist.

3.5.2 Web Feature Service

Die Web Feature Service-Spezifikation wurde durch das Open Geospatial Consortium (OGC) entwickelt und ermöglicht den Zugriff auf Geodaten nach räumlichen und nicht räumlichen (alphanumerischen) Gesichtspunkten. [Mit08, S. 253]. Die Kommunikation zwischen Client und WFS erfolgt dabei über HTTP-Requests und Response (s. Abb. 7).

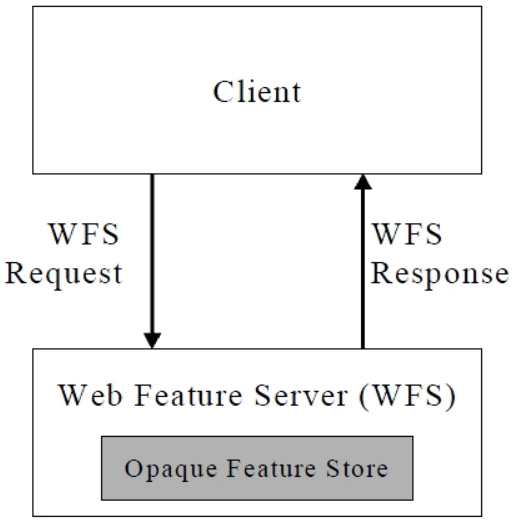


Abbildung 7: *WFS Client - Server Kommunikation*
Quelle: Vretanos (2002)

Umsetzung

Der Web Feature Service (WFS) dient als Schnittstelle zwischen den Punktobjekten aus der Datenbank und dem POI-Service. Für die Erstellung des WFS wird ein Mapfile benötigt, welches in die im Weiteren erläuterten Abschnitte gegliedert ist.

1. Der Map-Block initialisiert den WFS Server und definiert dabei einen Namen für diesen. Er beinhaltet alle anderen Blöcke des WFS.

   ```
   MAP
   NAME pointsWFS
   [...]
   END
   ```

2. Der Server kann verschiedene Projektionen verarbeiten und diese on the fly umprojizieren. Der Aufruf

   ```
   PROJECTION
   'init=epsg:31467'
   END
   ```

   definiert das an den POI-Service zurückgegebene Koordinatensystem. Der Service erwartet Koordinaten in Gauß-Krüger Zone 3.

3. In der Sektion `METADATA` sind Metadaten zu den übermittelten Geodaten enthalten. Diese dienen der Erschließung des Informationsgehalts der Daten. [Lan05] Sie erleichtern

bei einer Vielzahl von Servern die Zuordnung von Inhalten. Wichtig sind dabei der Titel `WFS_TITLE`, eine kurze Beschreibung der Inhalte `WFS_ABSTRACT`, das Verzeichnis `WFS_ONLINESOURCE` und die inbegriffenen Projektionen `WFS_SRS`.

4. Das Feld `LAYER` füllt nun den Server mit Inhalt. Jedem Layer wird dazu ein eindeutiger Name `NAME poi` zugeordnet und die Geometrie wird als Punktobjekt mit `TYPE POINT` definiert.
   Im Weiteren wird die Datenquelle definiert. Der Server greift mit dem Aufruf: `CONNECTIONTYPE postgis` auf die Datenbank aus Kapitel 3.4 zu. Dazu wird eine Verbindung mit `CONNECTION "user=poiservice password=******* dbname=poi host= http://vm116.rz.uos.de port=5432"` hergestellt. Aufgenommen werden die Daten mit `DATA 'the_geom from points'`. An dieser Stelle wird die `PROJECTION` der Eingangsdaten in WGS84 Koordinaten `'init=epsg:4326'` definiert. Jedem Layer werden wiederum Metadaten aus den oben genannten Gründen zugewiesen.

Die Attribute der Geodaten aus der Datenbank können vom WFS mit einem getFeature Request abgerufen werden.

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/mapserv?
  map=/usr/local/apache2/htdocs/wfs/wfs_server.map
    &service=wfs
    &version=1.0.0
    &request=getfeature
    &typename=pois
```

Zur Abfrage der Geodaten durch den POI - Service (s. Kap. 3.9) eines ausgewählten Bereichs wird der Server durch einen Request angesprochen, welcher den `getFeature` - Request um einen `filter` erweitert. Auf diese Weise kann der Standpunkt des Users als auch der Radius in dem Punkte angefordert werden sollen vom Server abgefragt werden.

```
&Filter=
  <Filter>
    <DWithin>
      <PropertyName>Geometry</PropertyName>
      <gml:Point>
        <gml:coordinates>3435729.371258,5796702.860358</gml:coordinates>
      </gml:Point>
      <Distance units='m'>10</Distance>
    </DWithin>
  </Filter>
```

Als Antwort wird eine xml - Datei mit dem nachfolgend dargestellten Inhalt an den POI - Service zurückgegeben.

```
<gml:featureMember>
  <ms:pois fid="pois.26">
    [...]
    <ms:msGeometry>
      <gml:Point srsName="epsg:31467">
        <gml:coordinates>
          3435729.371258,5796702.860358
        </gml:coordinates>
      </gml:Point>
    </ms:msGeometry>
    <ms:gid>26</ms:gid>
    <ms:osm_id>56978358</ms:osm_id>
    <ms:name>Erlöserkirche</ms:name>
    <ms:type>place_of_worship</ms:type>
    <ms:visible>1</ms:visible>
    <ms:height>-9999</ms:height>
    <ms:distance>-9999</ms:distance>
  </ms:pois>
</gml:featureMember>
```

3.6 GIS Service

Der GIS Service stellt die Schnittstelle zwischen dem POI Service und der GIS Datenbank her und besitzt dabei zwei zentrale Aufgaben. Die erste Aufgabe des Services ist es, die übermittelten POIs vom POI Service auf Sichtbarkeit zu überprüfen. Dazu wird die Betrachterposition sowie die Höhendaten aus der GIS Datenbank verwendet. Nach der Sichtbarkeitsanalyse wird ein entsprechendes Feld für die Sichtbarkeit in der GML-Datei aktualisiert und diese zurück an den POI Service gesandt. Die zweite Aufgabe des Services besteht darin, die fehlenden Höhenwerte der jeweiligen POIs in der GML-Datei zu aktualisieren. Diese werden vom AR-Service benötigt und müssen daher bereits im Voraus aus der GIS Datenbank gelesen werden.
Die vom Open Geospatial Consortium (OGC) vorgeschlagene Spezifikation für den Web Processing Service (WPS)[Ope07] eignet sich für die oben beschrieben Aufgaben in besonderer Weise. Über die URL des Internetbrowsers kann der WPS aufgefordert werden einen definierten Prozess mit bestimmten Parametern auszuführen. Diese Parameter werden durch den jeweiligen Prozess definiert und können sowohl aus numerischen Werten oder Zeichenketten, sowie Dateien (z.B. GML-Dateien) bestehen. Der POI Service erhält durch diese Spezifikation die Möglichkeit sowohl die GML Datei als auch optionale Angaben wie z.B. die maximale Entfernung für die Sichtbarkeitsanalyse zu übergeben. Als Antwort erhält der POI Service einen vorher definierten Dateityp, in diesem Fall ebenfalls GML.
Eine bereits Implementierte Version der OGC Spezifikation bietet der PyWPS[Cep10] auf Basis der Programmiersprache Python[Fou]. Es handelt sich um ein Open Source Projekt und kann daher frei genutzt werden. Ein großer Vorteil von PyWPS ist die native Unterstützung von Befehlen des Geoinformationssystems GRASS GIS[Tea10], welches auch als Verwaltung der GIS Datenbank (s. Kap. 3.7) dient. Somit lassen sich Befehle auf der GIS Datenbank direkt über den PyWPS ausführen. Der PyWPS bietet zudem die Möglichkeit, mittels Python, eigene Prozesse zu schreiben und kann somit leicht an die eigenen Bedürfnisse angepasst werden.
Damit der PyWPS auf dem Projektserver korrekt funktioniert, mussten einige Vorarbeiten geleistet werden. Python in der Version 2.4, GRASS GIS und Python-htmltmpl mussten zunächst in folgenden Schritten installiert werden:

```
wget http://www.python.org/ftp/python/2.4.6/Python-2.4.6.tgz
tar -xvf Python-2.4.6.tgz
cd Python-2.4.6
./configure
sudo make && make install

wget http://grass.itc.it/grass64/source/grass-6.4.0RC5.tar.gz
tar -xvf  grass-6.4.0RC5.tar.gz
```

```
cd grass-6.4.0RC5
./configure  --with-opengl=no --without-fftw
sudo make && make install

wget http://downloads.sourceforge.net/project/htmltmpl/htmltmpl/
     1.22/htmltmpl-1.22.tar.gz?use_mirror=kent
tar -xvf htmltmpl-1.22.tar.gz
cd htmltmpl-1.22
sudo python setup.py install
```

Nun kann auch der PyWPS installiert werden:

```
wget http://wald.intevation.org/frs/download.php/589/pywps-3.1.0.tar.gz
tar -xvf pywps-3.1.0.tar.gz
cd pywps-3.1.0
sudo python setup.py install
```

Um den WPS auch über den Webserver erreichbar zu machen, muss nun die Konfigurationsdatei des PyWPS angepasst und im cgi-bin Ordner des Apache Webservers folgendes Skript unter dem Namen „wps“ erstellt werden:

```
#!/bin/sh
export PYWPS_CFG=/usr/lib/python2.4/site-packages/pywps/etc/pywps.cfg
export PYWPS_PROCESSES=/usr/lib/python2.4/site-packages/pywps/processes/
/usr/bin/wps.py
```

Bei Aufruf dieses Skriptes über den Webserver werden die benötigten Pfade zur Konfigurationsdatei und den PyWPS-Prozessen gesetzt, sodass diese korrekt angesteuert werden können und der WPS-Prozess wird gestartet. Hierzu werden jedoch die entsprechenden Rechte benötigt, die mit nachfolgendem Befehl gesetzt werden können:
`sudo chmod 755 wps`
Über den Internetbrowser können nun die Daten des WPS abgerufen werden:

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/wps?service=wps&version=1.0.0
&request=getcapabilities
```

In der *GetCapabilities*-Abfrage werden sowohl allgemeine Metadaten des WPS, als auch die verfügbaren Prozesse und eine zugehörige Kurzbeschreibung zurückgeliefert. Diese Prozesse können über den Parameter Execute in der URL ausgeführt werden:

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/wps?service=wps&version=1.0.0
&request=execute&identifier=exampleBufferNoInputsProcess
```

Eine detaillierte Beschreibung der Prozesse, mit allen notwendigen Input-Parametern und deren Formate, erhält man über den Parameter *DescribeFeature*:

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/wps?service=wps&version=1.0.0
& request=describeprocess&identifier=exampleBufferNoInputsProcess
```

Die erste zentrale Aufgabe, die Sichtbarkeitsanalyse, wurde im Prozess *los* implementiert:

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/wps?service=wps&version=1.0.0
&request=execute&identifier=los_temp&datainputs=
east=3434828.338583;north=5793602.881890;height=1.80
```

Als Parameter werden die Nord- und Ostkoordinate, die Höhe der aktuellen Position (1.80m = Annahme für die durchschnittliche Personengröße), der Pfad zur GML-Datei, mit den zu überprüfenden Points-of-Interest, sowie die maximale Distanz, bis zu der die Sichtbarkeit überprüft werden soll, übergeben.
Der Prozess führt zunächst die eigentliche Sichtbarkeitsanalyse auf den Höhendaten der GIS Datenbank mit Hilfe des folgenden GRASS GIS-Befehls durch:
`r.los input=DSM output=los coordinate=%f,%f obs_elev=%f max_dist=1000`
Die Platzhalter *%f* stehen in diesem Fall für die Eingabe-Parameter Nord- und Ostkoordinate, Höhe und maximale Distanz.
Nach der Berechnung des Sichtbarkeitslayers wird nun die GML-Datei im UTF-8-Format gelesen und mit Hilfe der Python-Bibliothek *minidom*in einer lokalen Variable gespeichert:

```
fileObj = codecs.open( self.gml.value, "r", "utf-8" )
xmlData = fileObj.read()
baum = dom.parseString(xmlData.encode( "utf-8" ))
fileObj.close()
```

Die erstellte Variable besitzt eine Baumstruktur und kann mit Hilfe einer for-Schleife durchlaufen werden. Die Elemente werden dabei so überprüft, dass nur solche vom Typ *gml:feature Member* (dies entspricht einem POI) weiter bearbeitet werden:

```
for eintrag in baum.firstChild.childNodes:
if eintrag.nodeName == "gml:featureMember":
coords = xcoord = ycoord = visible = None
```

Wurde ein POI identifiziert durchläuft eine weitere for-Schleife den Inhalt dieses POIs und ermittelt auf diese Weise zunächst dessen Koordinaten:

```
for knoten in eintrag.childNodes:
if knoten.nodeName == "gml:Point":
for inner in knoten.childNodes:
if inner.nodeName == "gml:coordinates":
coords = inner.firstChild.data.strip()
xcoord = float(coords.split(",")[0])
ycoord = float(coords.split(",")[1])
```

Im weiteren Verlauf wird das GML-Feld *visible* aktualisiert. Dazu wird eine GRASS GIS-Abfrage gestartet, die den Wert, an der zuvor ermittelten Position der POIs im anfänglich berechneten Sichtbarkeitslayer, ermittelt. Ist dieser Wert -1 so ist der POI von der Betrachterposition aus nicht zu sehen und das Feld *visible* erhält den Wert *no*, andernfalls bleibt der Standardwert *yes* erhalten:

```
elif knoten.nodeName == "visible":
visible = self.cmd("r.what -f input=los east_north=%f,%f null=-1"
          % (xcoord,ycoord))
visible = visible.split("|")[3]
if visible == "-1":
knoten.firstChild.data = "no"
```

Das Ausgabe-GML wird zunächst temporär auf den Server geschrieben und anschließend an den Browser bzw. den POI-Service zurückgeliefert. Die POIs sind somit auf Sichtbarkeit überprüft worden und die GML-Felder *visible* der jeweiligen POIs besitzen zur Weiterverarbeitung nun den korrekten Wert.
Die zweite zentrale Aufgabe, die Höhenbestimmung, wurde im Prozess *height* implementiert:

```
http://vm116.rz.uni-osnabrueck.de/cgi-bin/wps?service=wps&version=1.0.0
&request=execute&identifier=height&datainputs=gml=...
```

Als Parameter benötigt dieser Prozess lediglich die zu aktualisierende GML-Datei. Das Auslesen dieser Datei und die Ermittlung der POI-Koordinaten verlaufen analog zum Sichtbarkeitsprozess. Nachdem die Koordinaten eines POIs bekannt sind, wird nun eine GRASS GIS-Abfrage direkt auf das digitale Oberflächen Modell gestellt:

```
elif knoten.nodeName == "height":
height = self.cmd("r.what -f input=DSM east_north=%f,%f null=-9999"
         % (xcoord,ycoord))
height = height.split("|")[3]
knoten.firstChild.data = height
```

Als Antwort wird neben den Koordinaten als dritter Wert auch die Höhe dieser Position zurück geliefert. Diese wird analog zum Sichtbarkeitsprozess aus der Antwort extrahiert und in das entsprechende GML-Feld geschrieben.
Das Ausgabe-GML wird ebenfalls temporär auf den Server geschrieben und anschließend an den Browser bzw. den POI-Service zurückgeliefert.

3.7 GIS Datenbank

Die GIS Datenbank hat die Aufgabe die in Kapitel 2.3 beschriebenen Höhendaten in geeigneter Form zu verwalten und Abfragen auf eben diese möglich zu machen. Hierfür wurde das Geoinformationsystem GRASS GIS verwendet, welches viele Befehle, speziell für die Verarbeitung von Raster- bzw. Höhendaten, zur Verfügung stellt. GRASS GIS verwaltet die Daten eigenen Datenbanken, den sog. GRASS Datenbanken. Externe Programme oder Skripte können auf diese Datenbanken zugreifen und mit der GRASS spezifischen Kommandosprache verschiedene Befehle ausführen. Die Installation von GRASS GIS auf dem Server wurde bereits im voran gegangenen Kapitel genauer beschrieben.
Für die Daten wurde zunächst eine neue Datenbank namens „als“ angelegt. In diese müssen nun die Höhendaten (siehe Kapitel 2.3) importiert werden. Datei im ASCII-Format können über den folgenden Befehl eingelesen werden:

```
r.in.ascii input=PFAD output=DTM
```

Die Daten liegen nun in der Datenbank vor und können verarbeitet werden. Zum Test der Daten wird mit folgendem Befehl eine Sichtbarkeitsanalyse durchgeführt:

```
r.los input=DSM@PERMANENT output=test
coordinate=3434828.338583,5793602.881890 obs_elev=1.80 max_dist=1000
```

Als Ausgabe wird ein Bild (s. Abb. 8) geliefert, dessen Werte den vertikalen Winkel zum Betrachter in Grad angeben. Der Wert 0 liegt also direkt unter dem Betrachter (rot), der Wert 90 liegt zum Betrachter horizontal (hellblau) und der Wert 180 liegt genau über dem Betrachter (dunkel blau). Die weißen Flächen im nachfolgenden Bild sind hingegen vom Betrachter aus nicht sichtbar.
Damit andere Programme oder Skripte auf diese Datenbank zugreifen können, ist es unerlässlich für diesen Ordner die entsprechenden Rechte zu setzen:

```
sudo chmod 777 /data/studgi09/grassdb/als
```

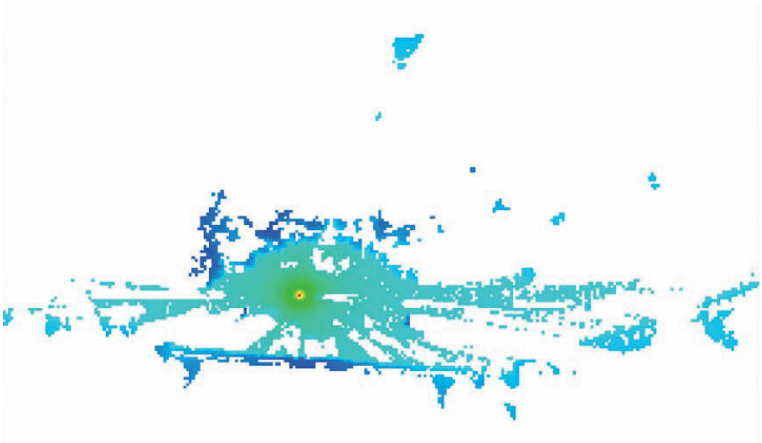


Abbildung 8: *Testergebnis einer Sichtbarkeitsanalyse in GRASS GIS*

3.7.1 Performanceprobleme

Es fiel auf, dass die Sichtbarkeitsanalysen eine enorme Berechnungszeit in Anspruch nehmen und somit die gesamte Prozessierung stark verlangsamen. Daher wurde im Folgenden die Performance dieser Analysen genauer untersucht und Überlegungen angestellt, wie diese verbessert werden kann.
Bei der Geschwindigkeit der Sichtbarkeitsanalyse spielen zwei Faktoren eine signifikante Rolle. Zum Einen die maximale Distanz bis zu welcher die Sichtbarkeit berechnet werden soll (in GRASS: max_dist) und zum Anderen die Rasterauflösung des digitalen Oberflächen Modells (engl: Digital Surface Model, nachfolgend kurz: DSM), welche in den Originaldaten einen Meter beträgt. Durch Anpassungen dieser Werte wird sowohl die Geschwindigkeit als auch die Qualität der Analyse beeinflusst. Eine angestrebte Geschwindigkeitsverbesserung darf also nicht mit einem zu hohen Qualitätsverlust einhergehen. Ideal wäre eine gleichbleibende Qualität des Ergebnisses.
Im Folgenden soll zunächst die Veränderung der Berechnungsgeschwindigkeit in Abhängigkeit der zuvor genannten Faktoren analysiert werden. Dazu werden aus dem originalen DSM, mit einer Rasterauflösung von einem Meter, weitere Modelle abgeleitet, welche eine Rasterauflösung von 2, 5 und 10 Metern besitzen (nachfolgend auch: DSM 1, DSM 2, DSM 5, DSM 10). Ebenso werden Schrittweiten für die maximale Distanz (max_dist) der Analyse festgelegt. Diese liegen bei 1000, 500, 300, 200 sowie 100 Metern. Für jede mögliche Kombination dieser zwei Faktoren wurde daraufhin eine Sichtbarkeitsanalyse durchgeführt und die Dauer der Berechnung gemessen (siehe Tabelle 1).
Die Messungen wurden in Kategorien eingeordnet und dementsprechend farblich markiert.

		Rasterauflösung digitales Oberflächen Modell [m]			
		1	**2**	**5**	**10**
max_dist [m]	**1000**	> 4 Min.	27,6 Sek.	2,8 Sek.	1,4 Sek.
	500	33,3 Sek.	10,4 Sek.	1,9 Sek.	1,0 Sek.
	300	15,2 Sek.	8,4 Sek.	1,7 Sek.	0,9 Sek.
	200	11,3 Sek.	7,6 Sek.	1,7 Sek.	0,9 Sek.
	100	9,3 Sek	1,7 Sek.	1,7 Sek.	0,9 Sek.

Tabelle 1: *Berechnungszeit der Sichtbarkeitsanalyse abhängig von den jeweiligen Faktoren*

In die beste Kategorie (in Tabelle 1 in Grün markiert) fallen die Werte die im Bereich unter 10 Sekunden liegen. Hierfür wird angenommen, dass diese Wartezeit dem Benutzer nicht negativ auffällt und Wartezeiten bis zu 10 Sekunden bei aufwändigen Berechnungen auch auf anderen Seiten im Internet durchaus vorkommen können. In die zweite Kategorie (in Tabelle 1 in Gelb markiert) wurden Werte zwischen 10 und 20 Sekunden eingeordnet. Diese werden dem Benutzer vermutlich bereits negativ auffallen und sind daher eher kritisch zu betrachten. Die schlechteste Kategorie (in Tabelle 1 in Rot markiert) umfasst alle Werte größer als 20 Sekunden. Diese Wartezeit ist in der Regel nicht mehr vertretbar und speziell Wartezeiten

im Minutenbereich führen bei manchen Webservern und Internet-Browsern unter Umständen zum Abbruch auf Grund eines internen Timeouts für zu lange Prozesse. Für die Verwendung des Services für eine breite Masse sollten daher Wartezeiten in der besten Kategorie angestrebt werden. Unmittelbar verbunden mit der Verkürzung der Wartezeit, durch die unterschiedlichen Einstellungen der oben genannten Faktoren, ist eine eventuelle Verminderung der Ergebnisqualität. Durch die Vergrößerung der Rasterauflösung des digitalen Oberflächen Modells werden die Basisdaten und die damit verbundenen Informationen, die zur Berechnung der Sichtbarkeit verwendet werden, teils stark verringert. Die nachfolgende Abbildung 9 zeigt den Unterschied der verschiedenen Auflösungen für einen Teilbereich von Osnabrück.

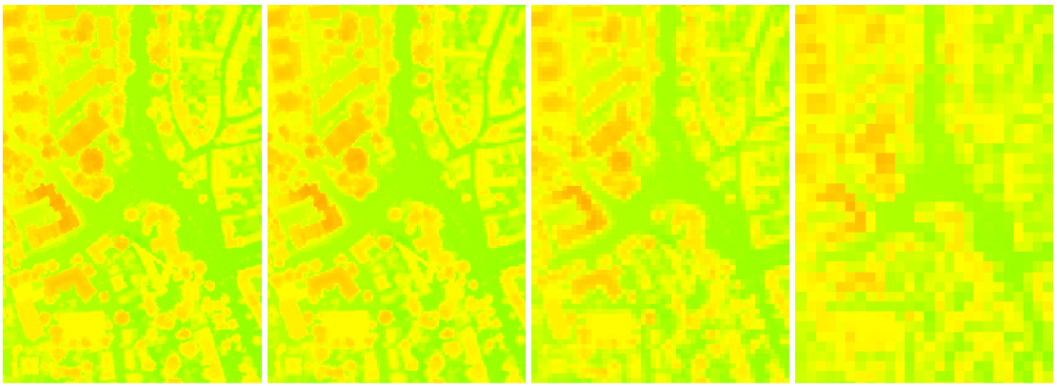


Abbildung 9: *Unterschiedliche Rasterauflösung des DSM (von links: 1m, 2m, 5m, 10m)*

Es lässt sich erkennen, dass der Unterschied zwischen dem 1 und 2 Meter Modell noch weitgehend gering ist, dieser jedoch im 5 Meter Modell schon deutlich zunimmt. Im 1 Meter Modell sehr deutliche, räumliche Strukturen lassen sich im 10 Meter Modell hingegen nur noch schwer erkennen.
Abbildung 10 verdeutlicht die Unterschiede von Sichtbarkeitsanalysen mit gleicher maximaler Distanz auf unterschiedlichen Rasterauflösungen. Analog zu den zugrunde liegenden digitalen Oberflächen Modellen verlieren auch die Ergebnisse der Sichtbarkeitsanalysen bei geringerer Rasterauflösung stark an Genauigkeit. Die Rasterauflösung ist daher ein sehr entscheidender Faktor und sollte entsprechend der Anforderungen der Anwendung gewählt werden.
Die Relevanz der maximalen Distanz soll anhand von Abbildung GGG veranschaulicht werden. Wie bereits zuvor erwähnt spielt diese hinsichtlich der Berechnungsgeschwindigkeit für höhere Rasterauflösungen eine sehr große Rolle (vgl. Tabelle 1). Es sollte jedoch speziell in städtischen Gebieten beachtet werden, dass eine größere Distanz nicht unbedingt einen Mehrwert an Informationen liefert. Abbildung 11 zeigt zwei Sichtbarkeitsanalysen auf dem 2 Meter Oberflächen Modell mit unterschiedlichen maximalen Distanzen. Nach Tabelle 1 dauert die Berechnung mit 1000 Metern Distanz 27,6 Sekunden, die Berechnung mit 200 Meter hingegen lediglich 7,6 Sekunden. Ein Unterschied in den Ergebnissen lässt sich lediglich in den Randbereichen erkennen. Dies lässt sich darauf zurückführen, dass in städtischen Bereichen nur

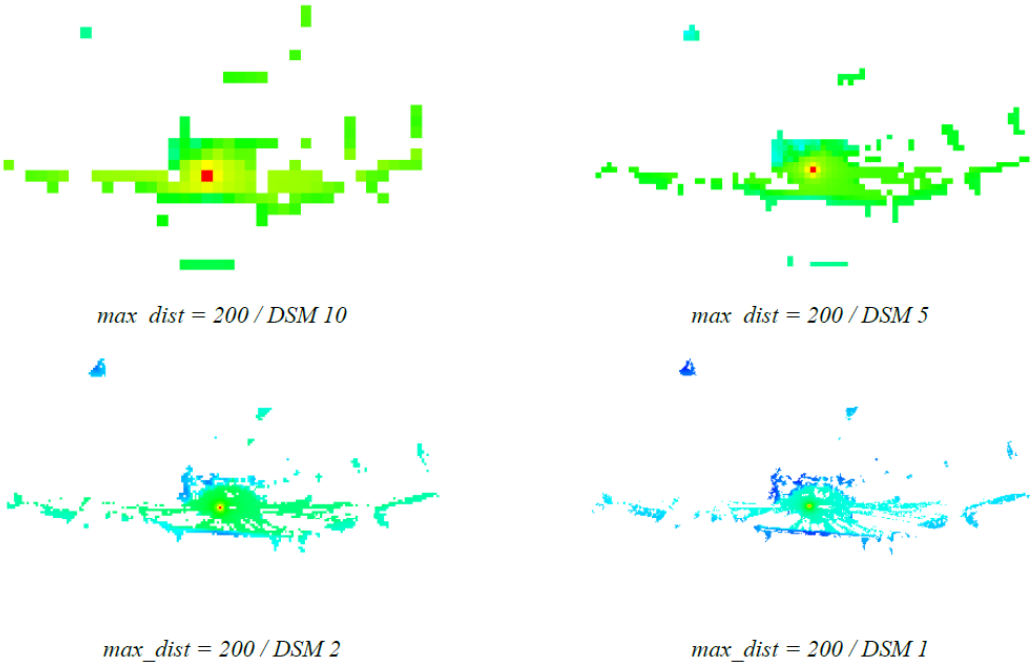


Abbildung 10: *Vergleich von Sichtbarkeitsanalysen auf unterschiedlichen digitalen Oberflächen Modellen mit der gleichen maximalen Distanz (200 Meter)*

selten Sichtweiten von 1000 Metern vorhanden sind. Die höhere Berechnungsdauer steht somit in keinem Verhältnis zum daraus resultierenden Mehrwert. Für Anwendungen in ländlichen Bereichen oder zum Beispiel zur Erkennung von Berggipfeln wird eine maximale Distanz von 200 Metern bei weitem nicht ausreichen um die gewünschten Ergebnisse zu erzielen. Diese Variable ist also analog zur Rasterauflösung stark vom jeweiligen Anwendungszweck abhängig.

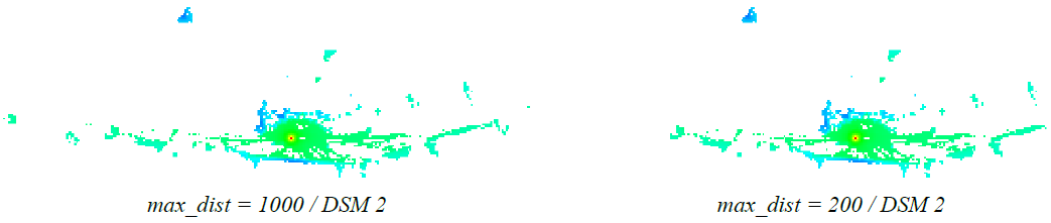


Abbildung 11: *Vergleich von Sichtbarkeitsanalysen auf dem DSM 2 und unterschiedlichen maximalen Distanzen.*

Zusammenfassend lässt sich daher sagen, dass sowohl die maximale Distanz als auch die Rasterauflösung auf den jeweiligen Anwendungsfall angepasst werden sollten. Wählt man einen der Parameter nicht angemessen, so wirkt sich dieses im schlimmsten Fall stark auf Berechnungszeit und/oder die Qualität der Ergebnisse aus.

3.7.2 Ausblick

Die Sichtbarkeitsanalysen liefern durchaus brauchbare Ergebnisse und lassen sich für die meisten Anwendungen mit wenigen Einschränkungen nutzen. Probleme gibt es in diesem Fall lediglich bei der Erkennung von Gebäuden. Points-of-Interest werden häufig ins Zentrum von Gebäuden gesetzt. Diese Flächen sind für den Betrachter jedoch nur selten sichtbar und fallen daher auch bei der Sichtbarkeitsanalyse in den nicht sichtbaren Bereich. Für die spätere Auswertung ist dies ein großes Problem, da so auch deutlich sichtbare Objekte oft nicht als sichtbar ausgegeben werden. Man könnte dieses Problem umgehen, indem man Gebäudeinformationen in die Sichtbarkeitsanalyse mit einfließen lässt. Diese können sowohl aus den Höhendaten abgeleitet werden oder aus amtlichen Beständen stammen. Gebäudeflächen die direkt an den sichtbaren Bereich grenzen könnten dann zusätzlich als sichtbar markiert werden. Eine andere Möglichkeit wäre es, die Betrachterhöhe deutlich anzuheben (z.B. auf 10 Meter), um so eine Vogelperspektive zu simulieren. Aus dieser Position wären die naheliegenden Gebäudedächer ebenfalls sichtbar.
Ein weiterer wichtiger Punkt bleibt die Geschwindigkeit der Analysen. Gerade bei größeren Auswertegebieten oder bei steigender Benutzeranzahl spielt die Performance eine große Rolle. Bei der Durchführung des Projektes sind drei mögliche Ansätze zur Verbesserung der Performance der Sichtbarkeitsanalysen entstanden.
Eine erste Idee war es für jeden Point-of-Interest eine Sichtbarkeitsanalyse durchzuführen um somit alle Punkte zu erhalten, die man vom Point-of-Interest aus sehen kann. Man müsste dann nur noch jedes Ergebnisbild überprüfen, ob die Betrachterposition im sichtbaren Bereich liegt oder nicht. Der Vorteil liegt klar auf der Hand: Es muss keine Sichtbarkeitsberechnung on-the-fly mehr durchgeführt werden was die Geschwindigkeit vermutlich deutlich erhöht. Nachteile sind sicherlich der hohe Vorberechnungsaufwand und das daraus resultierende hohe Datenaufkommen. Weiterhin von Nachteil sind die mangelnde Aktualität, da neue POIs wieder aufwendig eingefügt werden müssen, sowie die fehlende Variation in der maximalen Distanz, da keine on-the-fly Berechnung mehr durchgeführt wird. Es bleiben ebenso die offenen Fragen wie schnell der Zugriff auf einen Rasterlayer bei ca. 1000 verschiedenen Layern sein wird und ob dieses Verfahren bei vielen POIs wirklich performanter ist.
Eine weitere Überlegung baut auf dem ersten Ansatz auf und basiert darauf, die zuvor berechneten Sichtbarkeitsergebnisse in einer Vektorkarte zu vereinen. Diese Vektorkarte soll aus Polygonen bestehen, dessen Attribute die sichtbaren POIs enthalten. Somit würde eine Sichtbarkeitsabfrage reduziert werden auf eine simple Point-in-Polygon-Abfrage. Mutmaßliche Vorteile dieser Methode sind ein schneller Zugriff auf die Daten auf Grund des zugrunde liegenden Vektormodells und die Einsparung aufwendiger Rasterabfragen sowie die geringere Speichergröße. Ebenso lassen sich, bei einem vorhandenen System, neue POIs vermutlich deutlich schneller in die Speicherstruktur integrieren was die Aktualität enorm steigert. Von Nachteil sind hingegen der enorm hohe Vorberechnungsaufwand und die ebenfalls nicht vor-

handene Variation in der maximalen Distanz. Wie performant ein Vektorlayer mit vielen, kleinen Polygonflächen in der Realität wirklich ist bedarf ebenfalls weiterer Untersuchungen. Der letzte Ansatz zur Performanceverbesserung der Sichtbarkeitsanalyse verwirft den Ansatz der rasterbasierten Sichtbarkeitsberechnung zugunsten der Berechnung auf Triangulationsbasis. Dazu muss aus den Höhendaten einmalig ein trianguliertes Netzwerk (z.B. ein Triangulated Irregular Network, kurz: TIN) erstellt werden. Die Berechnung der Sichtbarkeit erfolgt dann vektorbasiert durch eine einfache Verbindung zwischen zwei Punkten. Wird diese Verbindung durch die Oberflächenstruktur geschnitten, so können sich die beiden Punkte nicht sehen. Diese einfache Berechnung der Sichtbarkeit und die damit verbundene potentiell hohe Aktualität stellen neben dem geringen Speicheraufwand die größten Vorteile dieser Methode dar. Ein Nachteil könnten mögliche Generalisierungsfehler bei der Erstellung der TINs darstellen. Häufigkeit und Auswirkungen dieser Fehler sowie die Genauigkeit der Sichtbarkeitsanalysen auf TINs müssten genauer untersucht werden. Ebenso muss geklärt werden bis zu welcher Rasterauflösung ein extrahiertes TIN überhaupt performant verwendet werden kann.

3.8 AR Image Service

3.8.1 Funktion

Der AR Image Service bekommt vom POI-Server das Bild übergeben, welches vom Nutzer hochgeladen wurde. Zusätzlich bekommt er die Metadaten des Bildes übergeben und die POIs in Form einer GML Datei. Der Service wertet aus, welche POIs auf dem Bild sichtbar sind und transformiert die geographischen Koordinaten in Bildkoordinaten, um die POIs an den passenden Stellen ins Bild einzutragen. Danach speichert er dieses und gibt den Pfad zurück an den POI-Server.

3.8.2 Technische Anforderungen und Probleme

Grundvorrausetzung ist ein Bild, dass in den Metadaten die Position des Fotografen und die Blickrichtung gespeichert sind, da hieraus errechnet wird, was auf dem Bild sichtbar ist. Deshalb ist es auch wichtig, dass die Kamera bzw. das Handy mit Foto-Funktion diese Daten auch möglichst genau ermittelt und abspeichert. In der Praxis ist dies aber häufig nicht der Fall. Erst nach längerer Laufzeit werden die Daten genauer. Auch muss sich der Fotograf bewegen, damit die Blickrichtung richtig erkannt werden kann. Selbst im Idealfall sind Abweichungen von 10-20 Metern durchaus normal. Zu dem wird die Position nur in Grad, Minuten und Sekunden ohne Nachkommastellen gespeichert, was ebenfalls zu mehreren Metern Abweichung führt. Dies wirkt sich vor allem auf Objekte in der Nähe aus, da hier 10 Meter Abweichung bereits darüber entscheiden können, ob ein Objekt sichtbar ist oder nicht. Je weiter ein Objekt entfernt ist, umso höher ist die Toleranz. Zu dem wird für die Berechnung des Sichtfeldes (Field of View) entweder die Größe des CCD Chips und die Brennweite benötigt, oder ein Wert für die Brennweite umgerechnet auf ein 35mm Negativ. Diese Daten sind aber leider nicht in den Metadaten enthalten und müssen entweder vom Nutzer übergeben werden oder aus einer Datenbank geholt werden.

3.8.3 Umsetzung

Der AR Image Service besteht hauptsächlich aus einem Java-Programm, welches die notwendigen Berechnungen vornimmt und das Bild bearbeitet. Hierfür benutzt er nur Java-Bibliotheken, die bereits mitgeliefert werden.
Der erste Schritt der Umsetzung war ein einfaches Testprogramm, welches das Einfügen von Text in ein bestehendes Bild vornimmt. Hierzu wird zunächst das Bild eingelesen und die Auflösung des Bildes ausgelesen, also die Anzahl der Pixel.

```
Image img = new ImageIcon(ImageIO.read(new File("Dateipfad"))).getImage();
BufferedImage Img = new BufferedImage(img.getWidth(null),
img.getHeight(null), BufferedImage.TYPE_INT_RGB);
```

Zur Bearbeitung werden hauptsächlich die Java-Bibliotheken Graphics [Sun10a] und Graphics2D [Sun10b] verwendet. Letztere ist eine Erweiterung von Graphics. Das Bild wird in ein Graphics Objekt überführt und dann in ein Graphics2D Objekt, damit alle Funktionen verfügbar sind.

```
Graphics g = Img.getGraphics();
g.drawImage(img, 0, 0, null);
Graphics2D g2d=(Graphics2D)g;
```

In der Testklasse wird eine Schriftart und Farbe bestimmt. Dann werden die Pixelkoordinaten angegeben, in die der Text eingefügt werden soll.

```
Font myFont=new Font("Arial", Font.BOLD|Font.PLAIN, 24);
g2d.setFont(myFont);
g.setColor(Color.red);
g.drawString("Studienprojekt", 20, 130);
g.drawString("Geoinformatik", 23, 155);
g.dispose();
```

Zuletzt wird das Bild als Datei geschrieben und Pfad und Dateiname ausgegeben.

```
ImageIO.write(Img, "jpeg", new File("Dateipfad"));
```

Abbildung 12: *Testprogramm für jpg-Bearbeitung*

Danach wurde die Implementierung des eigentlichen Programms begonnen. Im Folgenden werden die einzelnen Schritte beschrieben, die das Programm durchführt.

Als erstes werden die übergebenen Daten verarbeitet. Die x und y Koordinaten sowie die Blickrichtung und die Brennweite für 35mm werden als double-Wert übergeben und können direkt verwendet werden. Die übrigen Daten werden als String übergeben.
Als nächstes wird das Field of View berechnet. Dieser Wert gibt in Grad an, wie viel horizontal auf dem Bild zu sehen ist. Hierzu bekommt das Programm den Wert für die Brennweite umgerechnet auf ein 35mm Negativ übergeben. Bei bekannten Werten kann der Nutzer seine Kamera aus einer Liste wählen. Bei unbekannten Kameras muss er diesen Wert selbst angeben. Ein solches Negativ hat eine Größe von 36 x 24 mm. Mit der Formel:

```
FOV = 2 * Arctan( (Negativbreite/2) / (2*35mmBrennwert) )
```

wird das Field of View berechnet. Das Ergebnis wird dann noch in Grad umgesetzt.
Der nächste Schritt ist das auslesen der GML Datei. Diese enthält sämtliche POIs. Ein POI in dieser Datei sieht z. B. so aus:

```
<gml:featureMember>
      <ms:pois fid="pois.58">
        <gml:boundedBy>
          <gml:Box srsName="epsg:31467">
            <gml:coordinates>3435916.742709,5792329.502065
              3435916.742709,5792329.502065
            </gml:coordinates>
          </gml:Box>
        </gml:boundedBy>
        <ms:msGeometry>
        <gml:Point srsName="epsg:31467">
          <gml:coordinates>3435916.742709,5792329.502065</
              gml:coordinates>
        </gml:Point>
        </ms:msGeometry>
        <ms:gid>58</ms:gid>
        <ms:osm_id>56978731</ms:osm_id>
        <ms:name>St. Joseph</ms:name>
        <ms:type>place_of_worship</ms:type>
        <ms:visible>0</ms:visible>
        <ms:height>93.75</ms:height>
        <ms:distance>-9999</ms:distance>
      </ms:pois>
    </gml:featureMember>
```

Zum Auslesen werden die Java Bibliotheken DocumentBuilder und NodeList verwendet. Der DocumentBuilder ermöglicht es durch den Befehl

```
getElementsByTagName
```

direkt nach bestimmten Tags zu suchen. In diesem Fall wird nach ms:poi gesucht, da in dessen Unter-Tags sämtliche benötigten Daten stehen. Diese Elemente werden zunächst in einer NodeList gespeichert.
Falls das GML leer ist, weil im Bereich des Fotografen keine POIs vorhanden sind, ist diese Liste leer. In diesem Fall wird unter das Bild lediglich der Hinweis gesetzt, dass keine POIs vorhanden sind. Dieser Hinweis kommt auch dann, wenn das Bild des Nutzers gar keine GPS Daten enthält.
Ein weiterer übergebener Wert ist „Orientation“. Dieser gibt an, ob das Bild gedreht ist. Ein Wert von 1 bedeutet, dass das Bild nicht gedreht ist, die übrigen Werte geben eine Drehung z. B. um 90 oder 180 Grad an. Im Rahmen des Projekts wurde beschlossen, nur Bilder mit Orientation 1 zu nutzen, da ein Drehen des Bildes in Java viele mathematische Vorgänge benötigt. Sollte das übergebene Bild nicht den Wert 1 haben, wird unter das Bild ein Hinweis gesetzt, dass man die Kamera nicht drehen soll.
Der übergebene Wert „ZoomRatio“ gibt den Zommfaktor an. Falls nicht gezoomt wurde, ist dieser Wert leer und wird vom Programm auf 1 gesetzt. Ansonsten wird der Zoomfaktor berechnet, da dieser Wert in Form eines Strings mit 2 Werten übergeben wird, die durcheinander geteilt werden müssen.
Sollte die GML Datei zu viele POIs enthalten, weil vom Benutzer ein Fehler gemacht wurde oder der Wert für die maximale Entfernung zu hoch gesetzt wurde, werden nur die ersten 100 POIs überprüft und gegenfalls eingetragen.
Nun wird die NodeList durchlaufen. Hierzu wird wieder der Befehl getElementsByTagName genutzt. Nacheinander wird so nach den Tags für Name, Koordinaten, Höhe und Sichtbarkeit gesucht. Sollte kein Name vorhanden sein, wird dieser auf unbekannt gesetzt.

```
// Element Name auslesen
Element fstElmnt = (Element) fstNode;
NodeList NmElmntLst = fstElmnt.getElementsByTagName("ms:name");
Element NmElmnt = (Element) NmElmntLst.item(0);
NodeList Nm = NmElmnt.getChildNodes();

// Falls kein Name angegeben, Namen auf unbekannt setzen
// sonst Name auslesen und in die Liste einfuegen
if (Nm.getLength()!=1) {
daten.add("unbekannt");
} else {
daten.add(Nm.item(0).getNodeValue());
System.out.println(Nm.item(0).getNodeValue());
}
```

Die Koordinaten müssen erst getrennt werden, da beide Werte in einem Tag gespeichert sind. Alle ausgelesenen Werte werden dann zur weiteren Verarbeitung in eine String-Liste gepackt. Der POI Server liefert alle POIs, die in einem bestimmten Umkreis (vom Benutzer gewählt) um den Fotografen vorhanden sind. Deshalb muss ausgewertet werden, welche POIs überhaupt im Bereich des Bildes sind. Hierfür wird über ein Dreieck bestimmt, wo sich die Objekte befinden. Dies ist möglich, da ja sowohl die Koordinaten des Fotografen als ich die Koordinaten des POIs vorhanden sind:

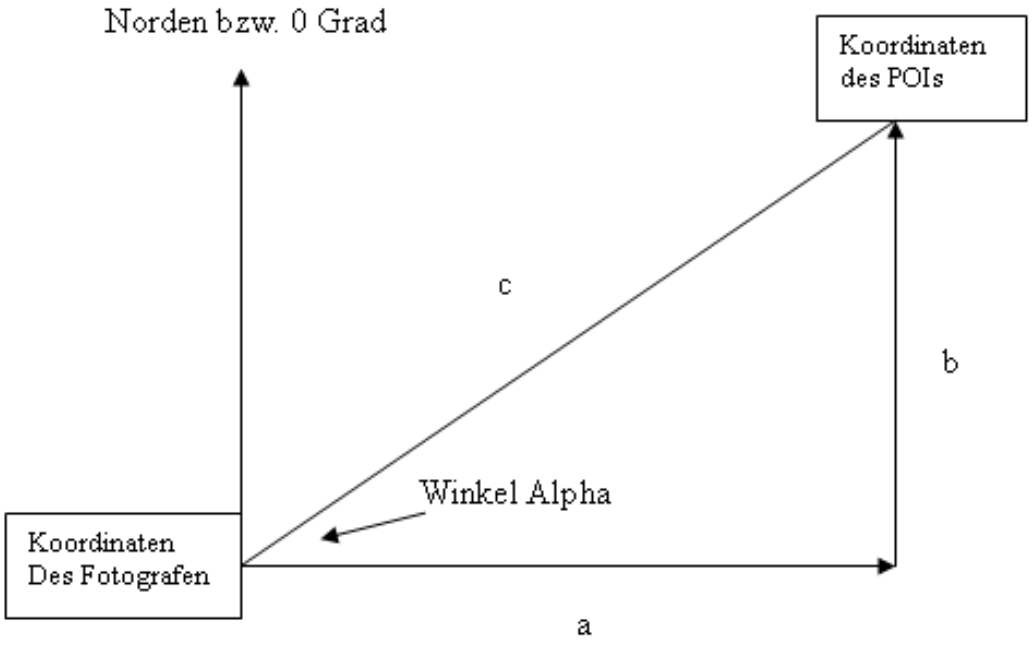


Abbildung 13: *Entfernungs- und Winkelberechnung*

Die Strecken a und b können berechnet werden, in dem Rechtswert und Hochwert (x und y) voneinander abgezogen werden. Strecke c wird dann über den Satz des Pytagoras errechnet. Strecke c ist somit auch gleich die Entfernung des Objektes. Nun können mit den Winkelfunktionen die Innenwinkel des Dreiecks berechnet werden und durch einen Vergleich der x und y Werte wird bestimmt, welcher Winkel genutzt werden muss. In diesem Fall muss also 90°-Alpha berechnet werden um zu erfahren, bei wie viel Grad der POI liegt.
Die Blickrichtung gibt in Grad den Mittelpunkt des Bildes an. Wenn man das Field of View halbiert und diesen Wert einmal vom Mittelpunkt abzieht und einmal addiert, hat man die rechte und linke Kante des Bildes in Grad. Für jeden POI wird also ermittelt, ob sein Winkel zwischen diesen beiden Kanten liegt.
Nun wird der Wert für die Breite des Bildes in Pixeln genommen und durch das Field of View geteilt. Das Ergebnis ist dann der Wert, wie viele Pixel der Text pro Grad vom linken Rand entfernt sein muss. Mit diesem Wert werden dann alle Winkel der einzutragenden POIs

multipliziert und als Ergebnis erhält man den X-Wert für die Position.

```
double pixel_per_degree = width/fov;
double left_border = exif_richtung-(fov/2);
double right_border = exif_richtung+(fov/2);
boolean nichtsichtbar = false;

// Falls das Objekt nicht sichtbar ist
if((winkel<left_border)||(winkel>right_border)){
nichtsichtbar=true;
} else {
// Unter das Bild:
g.drawString(name+", Entfernung: "+(int)c+"m, Hoehe: "
+(int)hoehe+"m", 20, zeilen);
}
pixel_x = ((winkel-left_border)*(pixel_per_degree));
```

Um den Pixelwert in Y-Richtung zu ermitteln, wird zunächst aus allen übergebenen POIs der höchste und der niedrigste POI ermittelt. Der höchste Wert +20 Meter und der niedrigste Wert -20 Meter werden als Rand des Bildes festgelegt. So ist etwas Abstand für Himmel und Boden vorhanden. Der Höchstwert minus dem niedrigsten Wert ergibt, wie viele Meter das Bild in der Höhe darstellt. Die Höhe des Bildes in Pixeln wird dann durch diesen Wert geteilt. Man weiß nun, wie viele Pixel in Y-Richtung einem Meter entsprechen. Von dem jeweiligen Höhenwert des POIs wird nun der niedrigste Höhenwert abgezogen. Dieser stellt den Rand des Bildes da und muss berücksichtigt werden, weil der Rand nicht 0 Metern entspricht. Das Ergebnis wird mit dem Pixelwert in Y Richtung multipliziert und man erhält die Y Koordinate im Bild.

```
double min_hoehe = 0;
double max_hoehe = 0;
for(int j=0;j<daten.size();j=j+5){

if(j==0){
min_hoehe = hoehe2;
max_hoehe = hoehe2;
}

if((hoehe2<min_hoehe)&&(hoehe2!=0.0)){
min_hoehe = hoehe2;
}
```

```
if(hoehe2>max_hoehe){
max_hoehe = hoehe2;
}
}

double top = max_hoehe+20;
double buttom = min_hoehe-20;
double area = top-buttom;
double pixel_per_meter = height/area;
pixel_y = height - (pixel_per_meter * (hoehe - buttom));
```

Nun wird der POI Name an der ermittelten X und Y Koordinate ins Bild eingefügt. Gleichzeitig wird der POI noch mit Namen, Entfernung und Höhe unter das Bild geschrieben.

```
g.drawString(name,(int)pixel_x,(int)pixel_y);
```

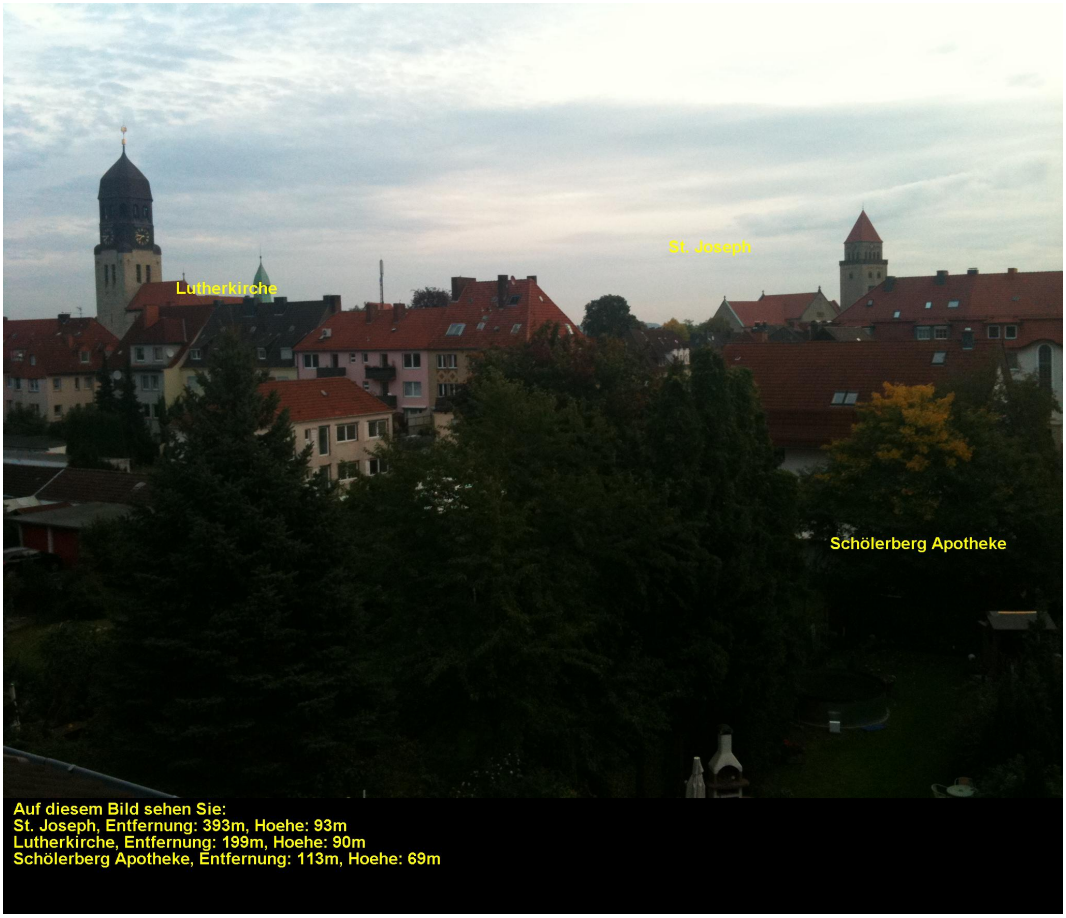


Abbildung 14: *Beispiel mit Blickrichtung*

Da viele Kameras zwar GPS Koordinaten speichern, aber keine Blickrichtung, können diese Bilder ebenfalls verarbeitet werden. Hier kann natürlich nicht die Position im Bild bestimmt

werden. Stattdessen werden alle POIs in einem vom Nutzer definierten Umkreis unter das Bild geschrieben, wieder mit Namen, Entfernung und Höhe. So haben Nutzer älterer Kameras wenigstens einen Anhaltspunkt darüber, was auf dem Bild oder in ihrer Nähe ist.

3.8.4 Probleme und Lösungsansätze

Die Ergebnisse des AR-Services sind schon relativ genau, könnten aber weiter verbessert werden. Das größte Problem ist die bereits erwähnte Ungenauigkeit der Kameras. Eine Berechnung der X und Y Pixelwerte vom dreidimensionalen ist zweidimensionale über Blickrichtung, Drehung um Achsen und Multiplikation mit Matrizen ist zwar möglich in diesem Fall aber nicht erforderlich, da die gegebenen Daten eine genaue Berechnung meist eh nicht zulassen.
Genauso ist die Berechnung in Y-Richtung teilweise eher eine Schätzung. Hier ist das größte Problem, dass viele Kameras nicht zuverlässig den Höhenwert des Fotografen liefern. Dieser ist manchmal gar nicht vorhanden, obwohl die Kamera dies unterstützt oder er ist so weit daneben, das bei einer Berechnung kein sinnvolles Ergebnis erscheint. Zu dem hält der Fotograf unter Umständen seine Kamera nicht senkrecht sondern schräg, wenn z. B. ein höheres Gebäude fotografiert wird. Die Auswertung aller POIs im Umkreis gibt zumindest einen Anhaltspunkt über die Höhe der Fotografenposition.
Ein zusätzliches Problem ist, dass die Werte für die Sichtbarkeit der Objekte in der GML-Datei nicht brauchbar sind (siehe Kapitel Florian). Der AR-Service kann diesen Wert bereits in der aktuellen Version berücksichtigen, aber leider sind dort viele Objekte als nicht sichtbar gekennzeichnet, die deutlich zu sehen sind. Deshalb werden alle POIs im Bildbereich eingetragen, auch wenn diese von anderen Objekten verdeckt sind.
Bei einer Weiterentwicklung des Services müssten also zunächst die Kameras genauer werden und die Sichtbarkeitsanalyse der Datenbank funktionieren. Auch die Angaben in den Metadaten der Bilder sollten erweitert werden, damit ohne Kenntnisse des Benutzers das Field of View berechnet werden kann. Dem Nutzer die Angabe über die Brennweite umgerechnet auf ein 35 mm Negativ zu überlassen wurde deshalb als Lösung gewählt, da fast keine Kamera Angaben über die Größe des CCD Chips in den Metadaten speichert. Mit dieser Angabe bräuchte der Nutzer den Wert nicht angeben.
Auch das Drehen des Bildes könnte noch hinzugefügt werden. Hierzu könnte man die java Bibliothek AffineTransformation verwenden. Die Drehung des Bildes müsste dann natürlich auch bei der Positionierung der POIs berücksichtigt werden.

Abbildung 15: *Beispiel ohne Blickrichtung*

3.9 POI-Service

Der POI-Service koordiniert die Abfrage, die der Nutzer gestellt hat. Er verarbeitet die über das Formular eingegebenen Daten und formuliert daraus Anfragen an die verschiedenen Services. Auch die Ausgabe der Ergebnisse über den Webserver wird durch diesen Service realisiert.

3.9.1 Umsetzung

Um den Inhalt der Webseite dynamisch ändern zu können, ist eine entsprechende Schnittstelle zwischen Webserver und einer entsprechenden Anwendung erforderlich. Die verbreitetsten Technologien dafür sind CGI (Common Gateway Interface), JSP (Java Server Pages) bzw. Servlets, ASP (Active Server Pages) und PHP (Hypertext Preprocessor). Da es für den Einsatz von Servlets bereits diverse Java-Bibliotheken gibt, die für die Anforderungen in diesem Projekt in Frage kommen, wurde sich für den Einsatz dieser Technologie entschieden. Für die Umsetzung von Servlets ist ein sogenannter Servlet-Container erforderlich. In diesem Fall wurde der Apache Tomcat[Fou10b] eingesetzt, da dieser zum Einen der derzeit verbreitetste Servlet-Container ist und auch direkt mit dem als HTTP-Server eingesetzten Apache Webserver verbunden werden kann.

Apache Tomcat

Der Apache Tomcat besteht aus einem Servlet-Container und einem Webserver. Als Servlet-Container stellt er eine Umgebung zur Ausführung von Java-Code auf Webservern bereit. Tomcat hat den Ruf, eine hohe Performance und Stabilität zu besitzen.[Wik10] In diesem Projekt wird Tomcat in der Version 6.0.20 eingesetzt. Da der integrierte Webserver i. d. R. nur für die Entwicklung eingesetzt wird, wurde Tomcat mit dem Apache Webserver verbunden.

Installation Für die Installation wurde auf dem Server unter /usr/local/ ein Verzeichnis tomcat angelegt. In dieses Verzeichnis wurde der Tomcat heruntergeladen und entpackt:

```
wget http://.../v6.0.20/bin/apache-tomcat-6.0.20.tar.gz
tar -xvf apache-tomcat-6.0.20.tar.gz
```

Der Tomcat kann anschließend direkt über:

```
./apache-tomcat-6.0.20/bin/startup.sh
```

gestartet werden. Eine Testseite bekommt man über den Aufruf:
`http://localhost:8080` bzw. von extern
`http://vm116.rz.uni-osnabrueck.de:8080`.

Der Port 8080 ist für den Tomcat Webserver reserviert. Um Tomcat mit dem Apache Webserver zu verbinden, ist ein sog. Konnektor erforderlich. Die entsprechenden Binaries werden heruntergeladen, in mod_jk.so umbenannt und im Apache2-Verzeichnis bei den Modulen abgelegt:

```
wget http://apache.linux-mirror.org/tomcat/tomcat-connectors/jk/binaries/
     linux/jk-1.2.28/x86\_64/mod\_jk-1.2.28-httpd-2.2.X.so
mv mod_jk-1.2.28-httpd-2.2.X.so /usr/local/apache2/modules/mod_jk.so
```

Die Datei server.xml im Tomcat-Verzeichnis conf/ wird um den Eintrag

```
<Listener className="org.apache.jk.config.ApacheConfig"
modjk="/usr/local/apache2/modules/mod_jk.so" />
```

erweitert. Durch einen Neustart des Tomcat werden weitere Konfigurationsdateien erzeugt. Im Verzeichnis conf/auto/ befindet sich nun die Datei mod-jk.conf. Diese wird nun in das Apache-Config-Verzeichnis kopiert und dort um einige Zeilen ergänzt:

```
JKWorkersFile "/usr/local/tomcat/apache-tomcat-6.0.20/conf/workers.properties"
JKLogFile "/usr/local/tomcat/apache-tomcat-6.0.20/logs/mod_jk.log"

JKLogLevel emerg

JKMount /POIServer ajp13
JKMount /POIServer/* ajp13
```

Mit JKWorkersFile wird auf die Konfigurationsdatei `workers.properties` verwiesen, die unter diesem Pfad ggf. neu angelegt werden muss. Eine Standarddatei befindet sich im Anhang. Im Wesentlichen werden dort sog. „Worker“ angelegt, die Zugriff auf bestimmte Ports erhalten.
Der Eintrag JKLogFile enthält Pfad und Dateiname für das Logfile, JKLogLevel bestimmt die Art des Logfileinhalts. JKMount weist nun das Verzeichnis POIServer aus dem Tomcat-Webapps-Verzeichnis mit dem Standard-Worker ajp13 dem Apache-Webserver zu, ebenso wie alle Unterverzeichnisse und Dateien (mit `POIServer/*`). Wird nun in der Konfigurationsdatei des Apache (httpd.conf) die Datei mod_jk.conf über
`Include mod_jk.conf`
eingebunden, kann der Webserver neu gestartet werden. Nun können das im Webapps-Verzeichnis des Tomcat angelegte Verzeichnis POIServer und die darin enthaltenen Dateien direkt vom Webserver angesprochen werden:
`http://vm116.rz.uni-osnabrueck.de/POIServer`

Startseite des POI-Services

Die Startseite namens index.html (s. Abb. 16) enthält ein Formular, über welches Informationen an den POI-Service übergeben werden. Dies sind zunächst die geogr. Länge und Breite des Standorts, sofern kein Photo hochgeladen werden soll sind diese Eingaben zwingend erforderlich. Des Weiteren können die Blickrichtung und die maximale Entfernung, die die POIs von diesem Standort haben sollen, eingegeben werden. Wird ein Photo hochgeladen, ist auch die Angabe der Brennweite (35mm) der eingesetzten Kamera notwendig. Diese kann entweder in das entsprechende Textfeld eingetragen werden oder das Kameramodell kann, sofern vorhanden, aus der Liste ausgewählt werden. Bislang ist dort das Modell „iPhone 3G/3GS“ eingetragen, da damit die Testbilder erstellt wurden und dementsprechend die Brennweite bekannt ist. Das letzte Textfeld ist für eine Bilddatei vorgesehen, die über den Button „Durchsuchen...“ oder einen Klick in das Textfeld ausgewählt werden kann. Das Photo muss im Format „JPG“ sein, da in anderen Bildformaten nicht die erforderlichen Metadaten vorhanden sind.

Um Dateien auf einen Server hochladen zu können, wird das HTML-Formular mit dem Zu-

Abbildung 16: *Startseite FART4Mobile*

satz `enctype="multipart/form-data"` versehen. Dieser Parameter beschreibt den über das Formular übergebenen Inhalt.

3.9.2 Implementierung

Die Umsetzung besteht aus 5 Java-Klassen: Dem Servlet für die Kommunikation mit dem Webserver, einer ausführenden Klasse, einem Datenbank-Interface, einer Datenbankklasse für

PostgreSQL und einer Konfigurationsdatei, die die URLs für die verschiedenen Services sowie Pfade und Dateinamen enthält.
Die Servlet-Klasse POIServlet enthält einen Konstruktor, der beim Start des Servlets die Konfigurationsdatei lädt, und die Funktionen `doGET()` und `doPOST()` zur Auswertung der vom Webserver übergebenen Parameter. Da das Formular der Startseite die Parameter per POST übergibt, wurde nur die Funktion `doPOST()` ausformuliert. Um die Inhalte der Formularfelder auslesen zu können, ist die Java-Bibliothek FileUpload, die vom Apache Commons Projekt zur Verfügung gestellt wird [Fou10a], erforderlich. Damit kann die Datei gelesen und unter dem vorgegebenen Pfad mit dem gleichen Dateinamen gespeichert werden. Auch die Textfelder für die Koordinaten und die anderen Informationen können damit ausgelesen werden. Diese werden zunächst gemeinsam in einem String gespeichert. Zur weiteren Verarbeitung müssen sie getrennt und in das Format Double überführt werden. Für die Koordinaten wird dies allerdings nur durchgeführt, sofern in den Metadaten des gespeicherten Fotos keine GPS-Koordinaten gespeichert sind. Um diese auszulesen wurde die Java-Bibliothek Sanselan, ebenfalls vom Apache Commons Projekt[Fou09a], eingesetzt. Diese ermöglicht den Zugriff auf die Exif-Metadaten eines Photos und so können die erforderlichen Daten direkt in entsprechende Variablen gespeichert werden.
Wurde kein Photo übergeben und/oder waren die Textfelder nicht korrekt ausgefüllt, wird eine Fehlermeldung ausgegeben. Ansonsten wird die Funktion `POIMain()` aus der Klasse POISelektion aufgerufen, die die weiteren Funktionsaufrufe vornimmt. Die Funktion liefert einen String bestehend aus Pfad und Dateinamen zum bearbeiteten Bild und die Namen der umliegenden POIs zurück, so dass Bild und Namen dann wieder über den Webserver ausgegeben werden können.
Die Klasse POISelektion enthält neben der Funktion `POIMain()` die Funktionen `setupTrustAll()`, `doTransformation()`, `getWFS()`, `getWPS()` und `deleteFile()`. Die Funktion `setupTrustAll()` wird benötigt, um auf URLs mit einer SSL-Verschlüsselung zuzugreifen, wie es beim Aufruf des Coordinate Transformation Services (CTS) erforderlich ist. Diese Routine greift ein, wenn ein Zertifikat dem Browser nicht bekannt ist. Standardmäßig würden die Browser einen Dialog öffnen, um die Annahme des Zertifikats zu bestätigen. Diese Bestätigung übernimmt diese Funktion.
In der Funktion `doTransformation()` werden die Koordinaten, die bislang im WGS84-Format vorliegen, durch den CTS des Geodatenzentrums des Bundesamtes für Kartographie und Geodäsie [Geo] in das Gauß-Krüger-Koordinatensystem transformiert. Dies ist erforderlich, da die Daten, auf die der Web Processing Service (WPS) zugreift, im Gauß-Krüger-Format vorliegen. Auch in der Datenbank wurden die Koordinaten schon dafür umgerechnet (s. Kapitel 3.4), so dass auch für die Abfrage des Web Feature Services (WFS) bereits Gauß-Krüger-Koordinaten benötigt werden. Die Funktion liefert am Ende die Koordinaten zurück an die Funktion `POIMain()`.
Die Koordinaten werden im Anschluss an die Funktion `getWFS()` übergeben. Diese stellt eine

GetFeature-Anfrage an den WFS, um die umliegenden POIs um den übergebenen Standort als GML-File zu erhalten. Dieses GML wird in einer Datei abgespeichert. Pfad und Dateiname dieser GML-Datei werden als Ergebnis zurückgeliefert.
Diese GML-Datei wird für die Anfrage an den WPS (s. Kapitel 3.6) benötigt. An den WPS werden 2 Anfragen gestellt, zunächst wird eine Sichtbarkeitsanalyse durchgeführt und anschließend die Höhe der POIs ermittelt. Da die Aufrufe sehr ähnlich sind, werden beide über die Funktion `getWPS()` aufgerufen, die Sichtbarkeitsanalyse durch den zusätzlichen Parameter „los“, die Höhenermittlung durch den Parameter „height“. In der Funktion wird jeweils eine Parameter-Datei im XML-Format erzeugt, die erforderlich ist, um die GML-Datei an den WPS per POST zu übergeben. Das Ergebnis der Sichtbarkeitsanalyse wird in einer GML-Datei gespeichert und diese wiederum an die Funktion zur Ermittlung der Höhe übergeben. Am Ende wird wieder ein String mit Pfad und Dateiname der GML-Datei an die Funktion `POIMain()` zurückgegeben.
Aus dieser Angabe wird nun ein File-Objekt erzeugt, welches an die Klasse AR_Service (s. Kapitel 3.8.1) zusätzlich zum Photo, den Koordinaten und den ermittelten Metadaten des Photos übergeben wird. Die Funktion liefert den Dateinamen des bearbeiteten Bildes zurück. Aus dem GML des WFS werden nun noch die Namen der umliegenden POIs ausgelesen. Dafür werden u. a. die Java-Bibliotheken DocumentBuilder und DOM verwendet. Dadurch kann auf die einzelnen Elemente des GML zugegriffen werden. So können alle Namen aus dem GML extrahiert und in einem String gespeichert werden. Diesen String gibt die Funktion `POIMain()` zusammen mit der Rückgabe des AR-Services an das Servlet zurück.
Für die Ausgabe des Photos wurde der Content-Type der Ausgabe auf „text/html“ gesetzt und ein ServletOutputStream erzeugt. So kann über einzelne Ausgaben HTML-Code erzeugt werden, der dann über den Webserver ausgegeben wird. Die Ergebnisse werden auf der Startseite in einem IFrame visualisiert. Ausgegeben werden die Gauß-Krüger-Koordinaten, die maximale Entfernung, das bearbeitete Bild (sofern vorhanden) und zur Information alle POIs, die im angegebenen Umkreis liegen (s. Abb. 17).
Für die Anbindung einer beliebigen Datenbank anstelle des WFS wurde ein Datenbankinterface erzeugt. Darin enthalten ist die Funktion `ConnectDB()`. Ein Interface hat den Vorteil, das für verschiedene Datenbanken ein Client angelegt werden kann, in der Main-Methode des Programms aber nur einmal angegeben werden muss, mit welcher Datenbank sich verbunden werden soll. Der Funktionsaufruf hingegen bleibt gleich. Um auf die vorhandene PostgreSQL-Datenbank zuzugreifen, wurde die Klasse DBClientPostGIS angelegt. Hier wird die Funktion `connectDB()` ausformuliert. Es wird eine Datenbankverbindung hergestellt und eine 'Select'-Abfrage an die Datenbank erstellt. Da letztlich der WFS aufgebaut wurde, um die Anfrage an die Datenbank zu übernehmen, wird diese Datenbankverbindung nicht verwendet.

Abbildung 17: *Ergebnis FART4Mobile ohne Photo*

3.9.3 Probleme und Lösungsansätze

Aufruf des AR-Services Der Aufruf des AR-Services sollte nach dem Systementwurf wie der Aufruf des GIS-Services über eine HTTP-Verbindung durchgeführt werden, um eine Serviceorientierte Architektur herzustellen. Dazu sollte der AR-Service auf den im POI-Service vorhandene Webserver-Funktionalität zugreifen. Tests zeigten jedoch, dass dies nicht so umzusetzen war, da die Servlet-Funktionalität auf eine bestimmte Art der Parameterübergabe eingestellt ist. Ein Anpassung wäre mit größerem Aufwand verbunden gewesen, so dass dies zugunsten der übrigen Funktionalität zurückgestellt wurde. Daher wird direkt auf die Java-Klasse des AR-Services zugegriffen.

Implementierung des Datenbankzugriffs Das Datenbankinterface und der Client für die PostgreSQL-Datenbank sind nicht vollständig implementiert worden. Es müssten min. 3 Funktionen vorhanden sein, eine für die Verbindung zur Datenbank, eine für die Abfrage und eine zum Schließen der Verbindung. Bislang gibt es nur eine Methode, die sowohl die Verbindung als auch die Abfrage enthält. Da es zunächst Probleme mit dem Datenbankzugriff gab, wurde die Anbindung an die PostgreSQL-Datenbank nicht weiter getestet. So wurde

zunächst mit einer Test-GML-Datei der Zugriff auf die anderen Services implementiert und schließlich der WFS eingebunden und der direkte Datenbankzugriff nicht weiter verfolgt.

4 Zusammenfassung

Die für das Projekt erforderlichen Systemkomponenten wurden auf einem zur Verfügung gestellten Linux-Server installiert und konfiguriert. Dazu gehören der Apache Webserver, das PostgreSQL/PostGIS- Datenbanksystem, der Apache Tomcat Servlet-Container, GRASS GIS, PyWPS und der MapServer.
Für die Anwendung wurde eine HTML-Seite aufgebaut, die ein Formular mit verschiedenen Eingabemöglichkeiten für Koordinaten, ein Photo u. ä. enthält. Diese Eingaben werden per POST vom Webserver an das Servlet des POI-Services übertragen. Hier werden die Eingaben ausgewertet, d. h. die Formulareinträge in Variablen bzw. das Photo auf dem Server gespeichert. Enthält das Photo GPS-Koordinaten in seinen Metadaten, werden diese Koordinaten übernommen und evtl. über das Formular übergebene Koordinaten nicht berücksichtigt.
In dem PostgreSQL/PostGIS-Datenbanksystem wurde eine Datenbank zum Speichern der POIs erzeugt und mit Daten gefüllt. Diese Daten wurden dem OSM-Projekt für unser Testgebiet Osnabrück entnommen. Um die Daten in das Gauß-Krüger-Koordinatensystem zu überführen wurde eine Transformation durchgeführt. Außerdem wurden der Datenbanktabelle weitere Spalten hinzugefügt, die dann in den GML-Dateien der verschiedenen Services mit Werten gefüllt werden. Für die Beschleunigung der Datenbankabfragen wurde zudem ein Index erstellt.
Um den WFS aufzubauen wurde ein Mapfile erzeugt, welches den WFS initialisiert. Dort wird u. a. die Projektion festgelegt und ein Punktlayer angelegt. Dieser Punktlayer greift auf die Daten der PostgreSQL-Datenbank zu. Um bestimmte Punkte abzufragen, ermöglicht der WFS den Einsatz von Filtern, so dass nur ein definierter Ausschnitt zurückgegeben wird. Der WFS liefert eine GML-Datei mit den Ergebnissen zurück. Für den Aufruf des WFS werden die über die Webseite übergegebenen Koordinaten, welche im Format WGS84 vorliegen, durch einen CTS im das Gauß-Krüger-Format transformiert.
Der GIS-Service führt zwei Abfragen mit Hilfe des PyWPS, welcher mit dem Apache Webserver verbunden ist, aus, eine Sichtbarkeitsanalyse und eine zur Ergänzung der fehlenden Höheninformationen. Dafür wurden Höhendaten für das Testgebiet in GRASS GIS importiert. Die Performance des gesamten Systems ist stark von der des GIS-Services abhängig. Daher wurde versucht, diese zu verbessern. Als Ergebnis liefert der GIS-Service ebenfalls eine GML-Datei.
Der AR-Service ist für die Darstellung der POIs in dem Photo zuständig. Dafür berechnet er aus den ihm zur Verfügung stehenden Daten, wo welche POIs in dem Photo angezeigt werden. Die Koordinaten der POIs müssen dazu in Bildkoordinaten umgerechnet werden. Dann kann das Photo um die Informationen ergänzt und gespeichert werden. Den neuen Dateinamen dieses Bildes gibt der AR-Service zurück.
Der POI-Service bildet die Schnittstelle zwischen den verschiedenen Services. Er wertet das Formular der Webseite aus und ruft danach nacheinander die Services aus. Diesen übergibt

er jeweils die zuvor erhaltenen Rückgaben in Form von Koordinaten oder GML-Dateien. Am Ende erhält er (falls vorhanden) den Dateinamen des neuen Bildes vom AR-Service und liefert dieses zusammen mit den Namen der umliegenden POIs zurück an die Webseite.

Literatur

[Cep10] Cepicky, Jachym: *PyWPS*. Version: 2010. http://pywps.wald.intevation.org/, Abruf: 18.02.2010. Homepage 3.6

[Eis03] Eisentraut, Peter: *PostgresQL - das offizielle Handbuch*. Bonn : mitp-Verlag, 2003 3.4.2, 3.4.2

[Fou] Foundation, Python S.: *Python*. http://www.python.de/, Abruf: 18.02.2010. Homepage 3.6

[Fou09a] Foundation, Apache S.: *Apache Commons, Sanselan*. Version: 2009. http://commons.apache.org/sanselan/, Abruf: 22.02.2010. Homepage 3.9.2

[Fou09b] Foundation, Apache S.: *Apache HTTP Server Version 2.0, Kompilieren und Installieren*. Version: 2009. http://httpd.apache.org/docs/2.0/install.html, Abruf: 26.08.2010 3.3

[Fou10a] Foundation, Apache S.: *Apache Commons, FileUpload*. Version: 2010. http://commons.apache.org/fileupload/, Abruf: 22.02.2010. Homepage 3.9.2

[Fou10b] Foundation, Apache S.: *Tomcat*. Version: 2010. http://tomcat.apache.org/, Abruf: 30.08.2010. Homepage 3.9.1

[Fou10c] Foundation, Open Source G.: *Quantum GIS*. Version: 2010. http://www.qgis.org/, Abruf: 23.09.2010. Homepage 3.4.2

[Fou10d] Foundation, OpenStreetMap: *OpenStreetMap*. Version: 2010. http://www.openstreetmap.org, Abruf: 30.08.2010 2.2, 3.4.2

[Geo] Geodatenzentrum: *Bundesamt für Kartographie und Geodäsie*. https://upd.geodatenzentrum.de, Abruf: 22.09.2010. Homepage 3.9.2

[Gro10] Group, PostgreSQL Global D.: *PostgreSQL*. Version: 2010. http://www.postgresql.org, Abruf: 23.09.2010. Homepage 3.4

[Heu] Heuschkel, Steffen: *Homepage*. http://www.html-world.de/program/apache_1.php, Abruf: 26.08.2010 3.3

[Lan05] Lange, Norbert de: *Geoinformatik in Theorie und Praxis*. 2. Auflage. Berlin : Springer, 2005 3

[Min10] Minnesota, University of: *About MapServer*. Version: 2010. http://mapserver.org/about.html, Abruf: 27.09.2010. Homepage 3.5

[Mit08] Mitchell, Tyler: *Web Mapping mit Open Source-GIS-Tools*. 1. Auflage. Köln : O'Reilly, 2008 3.5.2

[Ope07] Open Geospatial Consortium (Hrsg.): *OpenGIS - Web Processing Service Standard*. Open Geospatial Consortium, 2007. `http://portal.opengeospatial.org/files/?artifact_id=24151`, Abruf: 19.02.2010 3.6

[Ope10] Open Geospatial Consortium (Hrsg.): *OpenGIS - Web Feature Service*. Open Geospatial Consortium, 2010. `http://www.opengeospatial.org/standards/wfs`, Abruf: 23.09.2010 3.2

[Res10] Research, Refraction: *PostGIS*. Version: 2010. `http://postgis.refractions.net/`, Abruf: 23.09.2010. Homepage 3.4

[Sun10a] Sun: *Java Graphics*. Version: 2010. `http://java.sun.com/j2se/1.3/docs/api/java/awt/Graphics.html`, Abruf: 22.02.2010. Homepage 3.8.3

[Sun10b] Sun: *Java Graphics2D*. Version: 2010. `http://java.sun.com/j2se/1.3/docs/api/java/awt/Graphics2D.html`, Abruf: 22.02.2010. Homepage 3.8.3

[Tea10] Team, GRASS D.: *GRASS GIS*. Version: 2010. `http://grass.itc.it/`, Abruf: 18.02.2010. Homepage 3.6

[War10a] Warmerdam, Frank: *GEOS*. Version: 2010. `http://trac.osgeo.org/geos/`, Abruf: 23.09.2010. Homepage 3.4.1

[War10b] Warmerdam, Frank: *PROJ.4*. Version: 2010. `http://trac.osgeo.org/proj/`, Abruf: 23.09.2010 3.4.1

[Wik10] Wikipedia: *Wikipedia*. Version: 2010. `http://de.wikipedia.org/wiki/Apache_Tomcat`, Abruf: 22.02.2010. Homepage 3.9.1