Universität Passau
Fakultät für Informatik und Mathematik

Lehrstuhl für Mathematik
Schwerpunkt Digitale Bildverarbeitung
Prof. Dr. Tomas Sauer

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

OSMAR
Augmented Reality Framework
für OpenStreetMap

Benedikt Lang

Studienfach	Internet Computing
Prüfer	Prof. Dr. Tomas Sauer
Matrikelnummer	55857
Datum	16.03.2015

Zusammenfassung

OSMAR verbindet das Kartenmaterial des OpenStreetMap Projekts mit den Sensordaten moderner Android-Smartphones um eine Augmented Reality zu schaffen. Als Framework aufgebaut, ermöglicht es beliebige Daten des OpenStreetMap Projekts grafisch einzubinden. Die vorliegende Arbeit vermittelt neben den theoretischen Grundlagen, der Entstehung und Aufbau dieses Frameworks auch die Umsetzung anhand eines Anwendungsbeispiel in Form einer Darstellung von Geschwindigkeitsbegrenzungen. Im Weiteren werden Schwierigkeiten der Umsetzung und ein Ausblick auf aufbauende Arbeiten gegeben.

Inhaltsverzeichnis

1 Einleitung **8**
- 1.1 Motivation . . . 8
- 1.2 Ziel der Arbeit . . . 9

2 Anforderungen und Funktionsumfang **9**
- 2.1 Für Benutzer . . . 9
- 2.2 Für Anwendungsentwickler . . . 10

3 Grundlagen **10**
- 3.1 Augmented Reality . . . 10
- 3.2 Android . . . 11
  - 3.2.1 Sensoren . . . 12
  - 3.2.2 GPS . . . 12
  - 3.2.3 Kamera . . . 13
- 3.3 OpenGL für Smartphones . . . 13
  - 3.3.1 Shader . . . 13
  - 3.3.2 Zeichnung von Objekten . . . 14
- 3.4 OpenStreetMap . . . 15
  - 3.4.1 Aufbau der Daten . . . 15
  - 3.4.2 GPX Dateiformat . . . 16
- 3.5 OpenCV . . . 16

4 Darstellung der virtuellen Welt **16**
- 4.1 Reale Welt . . . 17
- 4.2 Projektion in die virtuelle Welt . . . 17
  - 4.2.1 Koordinatensystem der virtuellen Welt . . . 17
  - 4.2.2 Projektion geographischer Koordinaten . . . 18
- 4.3 Perspektivische Projektion . . . 18
  - 4.3.1 Kameramatrix . . . 18
  - 4.3.2 Lochkameramodell . . . 19
  - 4.3.3 OpenGL Kameramodell . . . 21
  - 4.3.4 Projektion . . . 24
- 4.4 Perspektivische Division . . . 25
- 4.5 Darstellung auf dem Bildschirm . . . 26
- 4.6 Extrinsische Kameraparameter - Lagesensoren . . . 26

4.7 Problematik: Genauigkeit der Sensoren . . . 27
4.7.1 Lagesensoren . . . 27
4.7.2 Kompass . . . 27
4.7.3 Gravitation . . . 27
4.7.4 GPS . . . 30

5 Implementierung **30**
5.1 Bedienoberfläche . . . 31
5.2 Einlesen von OSM-Dateien . . . 32
5.2.1 Einlesen von Straßen . . . 33
5.2.2 Einlesen von Geschwindigkeitsbegrenzungen . . . 33
5.3 Positionsbestimmung mittels GPS . . . 34
5.4 Datenhaltung . . . 35
5.5 Rendering mit OpenGL . . . 35
5.5.1 Kameranachbildung . . . 36
5.5.2 Veränderung der Kameraposition . . . 36
5.5.3 Buffer und Lifecyle Management . . . 37
5.5.4 Darstellung von Straßen . . . 37
5.5.5 Darstellung von Geschwindigkeitsbegrenzungen . . . 37
5.6 Aufnahme von Wegpunkten . . . 38
5.7 Erweiterbarkeit . . . 39
5.7.1 Parser . . . 40
5.7.2 Zeichenbares Objekt . . . 40

6 Fazit **41**

Abbildungsverzeichnis

3.1 Reality-Virtuality (RV) Continuum von Milgram [15, S. 283, Abb. 1] . . . 10
3.2 Markerbasierte Augmented Reality . . . 11
3.3 OpenGL ES 2.0 Pipeline [18] . . . 14

4.1 Konzept der Transformation von Objekten der realen Welt zur Darstellung auf dem Smartphone Bildschirm . . . 17
4.2 Beispielhafte Kamerakalibrierung mittels OpenCV auf einem Android-Smartphone . . . 19
4.3 Lochkamera nach Hartley-Zisserman [11, S. 154, Fig. 6.1.] . . . 20
4.4 Relation zwischen Bild- und Kamera Koordinatensystem [11, S. 155, Fig. 6.2.][7] . . . 21

4.5 Lochkamera in OpenGL [7] . . . 22
4.6 OpenGL Frustum [2, 7] . . . 23
4.7 Relation zwischen Focal plane und near plane [7] . . . 24
4.8 Clipping eines Dreiecks durch ein Frustum [2] . . . 25
4.9 Relative Lageveränderung der Achsen in Ruhelage . . . 28
4.10 Trägheit der Achsen nach einer vollständigen Drehung . . . 29
4.11 Verschiebung des künstlichen Horizonts bei längerem Einsatz im Fahrzeug 30
4.12 GPS Abweichung bei Aufzeichnung in Schrittgeschwindigkeit auf Straße (GPS in Grün, Straßenverlauf in Rot) . . . 31
5.1 OSMAR: Benutzerinterface mit aufgeklapptem Menü . . . 32
5.2 OSM Kartenausschnitt in JOSM: 1) geteilte Kreuzung, 2) Unterbrechung des Straßenverlaufs, 3) Straßenaufbau in Segmente . . . 33
5.3 OSMAR Screenshot Einblendung auf mehrspuriger Straße . . . 38
5.4 OSMAR Screenshot mit eingeblendetem Schild vor Geschwindigkeitsbegrenzung . . . 39
5.5 Wegpunkte mit Audionotiz in JOSM . . . 40

Quellcode Verzeichnis

1 Parser Interface . . . 41
2 DrawableWorldObject Interface . . . 41

1 Einleitung

Augmented Reality ist spätestens seit der Google Glass Brille in aller Munde und findet immer häufiger auch Anwendung in gewerblichen Bereichen, wie etwa der Raumplanung [14]. Unter Augmented Reality, im Folgenden auch durch AR abgekürzt, versteht man eine um Informationen angereicherte Realität. Hierbei wird beispielsweise das Kamerabild eines Smartphones in Echtzeit mit 3D-Modellen von Möbelstücken überlagert und somit die Möglichkeit geschaffen, Möbelstücke bereits vor dem Kauf in ihrer gewünschten Umgebung auszuprobieren. Moderne Smartphones bieten alle nötigen Sensoren und Rechen- bzw Grafikleistung um AR Anwendungen auch tragbar zur Verfügung zu stellen [6]. Eine besondere Herausforderung ist die Umsetzung einer Augmented Reality Anwendung anhand geographischer Daten, da keine Marker zur Positions- und Lagebestimmung der Kamera zum Einsatz kommen können und dieses Feld besonders viele Einsatzmöglichkeiten bietet. Eine solche Augmented Reality Anwendung benötigt reichhaltige Daten, welche direkt an geographische Koordinaten geknüpft sind, einen solchen Datenbestand bietet das OpenStreetMap Projekt. Hierbei handelt es sich um ein internationales Projekt mit der Zielsetzung eine freie Weltkarte zu erschaffen. Um diese Weltkarte um Informationen zu bereichern, auszubauen und aktuell zu halten sammeln freiwillige Mitarbeiter, sogenannte Mapper, kontinuierlich Geo- und Metadaten und pflegen diese in das Projekt ein [20]. So sind neben den Straßenverläufe im OSM Datenbestand viele weitere Informationen wie beispielsweise Buslinien oder die Standorte von Briefkästen zu finden [23, S. 65 ff.]. Eine Augmented Reality Anwendung basierend auf den Daten von OpenStreetMap kann somit sowohl zur Unterstützung der Arbeit von Mappern dienen, als auch für den Endbenutzer von Wert sein.

Diese Arbeit verbindet das Konzept von Augmented Reality mit den Daten des OpenStreetMap Projekts.

1.1 Motivation

Mapper verwenden neben Satellitenbildern häufig herkömmliche Smartphones um Wegstrecken aufzuzeichen um diese später mittels eines Karteneditors (beispielsweise JOSM) aufzuarbeiten. Neben einem aktuellen und fortwährend verbesserten Straßennetz werden auch Metadaten wie Geschwindigkeitsbegrenzungen, Positionen von Ampeln oder Briefkästen in das Kartenmaterial übernommen. Die Erfassung und Korrektur dieser Metadaten ist jedoch ein langwieriges und aufwändiges Verfahren, da Positionen erfasst, Informationen notiert und diese später manuell eingepflegt werden müssen [23, S. 41-43]. Eine große Herausforderung ist hierbei fehlende oder geänderte Daten zu erkennen und diese möglichst einfach verarbeiten zu können. Diese Arbeit entstand aus der Idee,

bestimmte Arbeitsabläufe für die Mapper zu vereinfachen, indem eine Umgebung geschaffen wird, in welcher bestehende Informationen des Kartenmaterials dem Mapper bereits Vorort vorliegen, um fehlende Informationen und Änderungen einfach zu erkennen. Hieraus entstand eine Augmented Reality Smartphone Anwendung für Android, welche dem Mapper wichtige Informationen einblendet und diesem die Möglichkeit bietet, seine aktuelle Position und damit verbundene Änderungsnotizen schnell zu erfassen.

1.2 Ziel der Arbeit

Da die Verbindung von OSM Daten und Smartphone Sensoren zu einer Vielzahl von Anwendungsfällen führt, welche über das ursprüngliche Ziel der Unterstützung von Mappern hinausgeht und auch für Endbenutzer von Wert ist, wurde das Projekt als Anwendungsframework ausgelegt und beinhaltet eine Beispielimplementierung zur Unterstützung von Mappern anhand eingeblendeter Geschwindigkeitsbegrenzungen. Ziel dieser Arbeit ist somit die Erstellung eines Frameworks für Augmented Reality Anwendungen für Android Smartphones, mithilfe dessen Anwendungen implementiert werden können, welche Informationen aus dem OpenStreetMap Datenbestand in einer Augmented Reality darstellen.

2 Anforderungen und Funktionsumfang

Im Folgenden sollen die Anforderungen und der Funktionsumfang des Frameworks und der daraus entstehenden Anwendungen (Beispielsanwendung) erläutert werden. Hierbei wird zwischen dem Funktionsumfang des Endbenutzers und des Anwendungsentwicklers unterschieden.

2.1 Für Benutzer

Zunächst soll es dem Nutzer der Anwendung möglich sein einen Ausschnitt aus dem OSM Datenbestand in der Anwendung zu laden. Nach einem erfolgreichen Einlesevorgang soll die Position des Smartphones ermittelt werden. Fortan soll die Position und Lage des Smartphones ermittelt werden und das eingelesene Straßennetz auf dem Bildschirm perspektivisch korrekt dargestellt werden. Hierbei soll der Bildschirm als „Fenster “ in die virtuelle Realität betrachtet werden. Eine Änderung der Lage oder Position des Smartphones soll eine möglichst realitätsgetreue Reaktion auf dem Bildschirm nach sich ziehen. Der Benutzer soll jederzeit in der Lage sein die aktuelle Position mit einem Vermerk zu speichern und diese später außerhalb seines Smartphones verarbeiten können. In Abhängigkeit der mit dem Framework entwickelten Anwendung sollen weitere Informationen in die 3D-Welt eingefügt und dem Benutzer angezeigt werden.

2.2 Für Anwendungsentwickler

Dem Anwendungsentwickler und somit Nutzer des Frameworks soll ermöglicht werden weitere Daten aus der vom Benutzer eingelesenen OSM Datei zu verarbeiten und diese in der 3D-Welt zu positionieren. Hierbei soll der Entwickler möglichst viele Freiheiten bezüglich der Darstellung der zu zeichnenden Objekte haben. Ebenso sollen einfache Schnittstellen geschaffen werden, durch die Einfluss auf die Sensorverarbeitung und Positionsbestimmung genommen werden kann.

3 Grundlagen

Die Umsetzung der hier beschriebenen Anwendung umfasst mehrere Teilbereiche, von den Sensordaten über die Darstellung der Objekte bis hin zur Verarbeitung des Kartenmaterials. Im Folgenden werden die Grundlagen zur späteren Umsetzung behandelt.

3.1 Augmented Reality

Augmented Reality, auch als AR abgekürzt, wird häufig in das Realität-Virtualität-Kontinuum (Abbildung 3.1) von Milgram [15] eingeordnet. Hierbei unterscheidet sich die Augmented Reality, also die erweiterte Realität, von der Augmented Virtuality dadurch, dass sie näheren Bezug zur echten Umgebung als zur Virtuellen hat. Eine Augmented Reality erweitert also die echte Umgebung um virtuelle Informationen, beispielsweise durch Einblendung von virtuellen Grafiken in ein Abbild der Umgebung, wie im Zusammenhang dieser Arbeit in das Bild einer Smartphone Kamera.

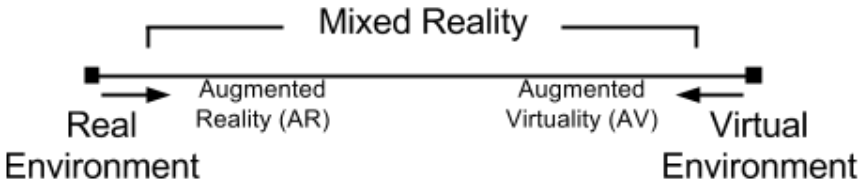

Abbildung 3.1: Reality-Virtuality (RV) Continuum von Milgram [15, S. 283, Abb. 1]

Neben vielen verschiedenen Ansätzen zur Umsetzung einer Augmented Reality fällt die Verwendung eines Smartphones unter die bildschirmbasierte Augmented Reality und unterscheidet sich dadurch von Umsetzungen mittels Brillen oder anderen Anzeigevarianten, bei denen der Benutzer direkt auf eine Augmented Reality blickt. Diese Art der Darstellung wird auch als 'Fenster zur Welt' bezeichnet. [15, S. 284]

Um eine Augmented Reality zu schaffen, bedarf es der Möglichkeit zur Bestimmung der Position der Kamera im umgebenden Raum. Hierbei unterscheidet man zwischen der Position und der Lage bzw. Orientierung der Kamera. Der Anwendungsfall dieser Arbeit setzt eine absolute Positionsbestimmung auf der Erde voraus, sowie eine Lagebestimmung der Kamera. In den meisten AR Anwendungen oder Spielen kommen markerbasierte Positions- und Lagebestimmung zum Einsatz: Hierbei wird ein oder mehrere sogenannte Marker im Sichtfeld der Kamera positioniert durch diese die Position im Raum mittels Bilderkennung bestimmt werden kann (siehe Abbildung 3.2). Dies ist möglich, da keine absolute Positionierung benötigt wird, sondern lediglich eine relative Positionierung zu einem definierten Objekt. Mit steigender Rechenleistung wird zunehmend auch merkmalsbasierte Positions- und Lagebestimmung vorgenommen, welche die Position durch Verarbeitung des Kamerabildes und Abgleich mit gespeicherten Merkmalen bestimmt [6]. Im Fall dieser Arbeit können keine Marker zur Positions- oder Lagebestimmung eingesetzt werden, da diese fest definierte Objekte (meist Schwarz-Weiss-Marken) verwenden, welche im Anwendungsfall nicht vorhanden sind. Eine merkmalsbasierte Positionsbestimmung geht über den Umfang dieser Arbeit hinaus, jedoch wird eine Positionsbestimmung unterstützt durch eine Merkmalerkennung im Ausblick (siehe Abschnitt 6) an einem Beispiel dargestellt. In dieser Arbeit wird deshalb auf die geobasierte Positionsbestimmung zurückgegriffen wird. Hierbei werden GPS Daten (Global Positioning System) ausgewertet, welche die absolute Position des Smartphones auf der Erde bestimmen. Die Lage der Kamera wird durch die Auswertung der Sensoren des Smartphones bestimmt.

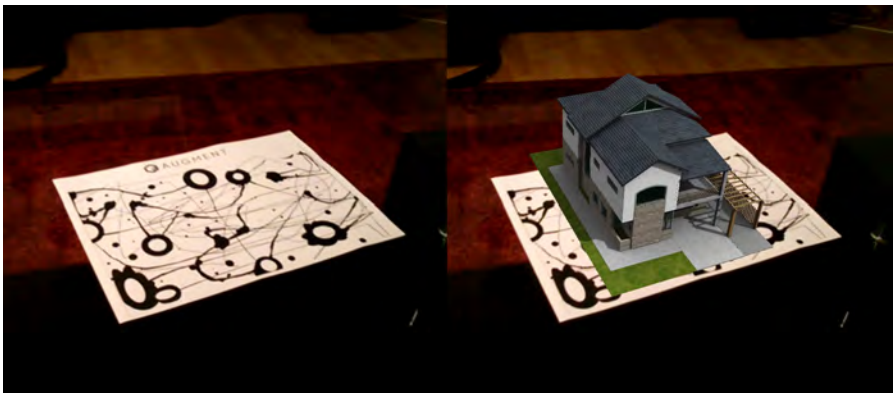

Abbildung 3.2: Markerbasierte Augmented Reality

3.2 Android

Als Plattform zur Umsetzung der Software wurde Android gewählt. Android ist ein weit verbreitetes Betriebssystem für Smartphones und Tablets und bietet ausgereifte Programmierschnittstellen. Entwickelt wurde das Framework und die Beispielsanwendung für ein Google Nexus 4 mit Android Version 4.4, wodurch eine größtmögliche Kompati-

bilität zu modernen Android-Smartphones gewährleistet wird und eine Umsetzung auf neueren Android Versionen nur minimale Änderungen nach sich zieht. Zur Entwicklung wurde das von Google bereitgestellte Android Software Development Kit (Android SDK) in Verbindung mit der IDE (Integrated Development Environment) Android Studio verwendet.

3.2.1 Sensoren

Moderne Smartphones bieten eine Vielzahl von Sensoren, welche dem Entwickler unter der Android API zur Verfügung stehen. Android unterteilt diese in 3 Kategorien[5]:

Bewegungssensoren Sensoren zur Messung von Beschleunigung und Rotation. Hierunter fallen der Beschleunigungssensor, der softwarebasierte Gravitationssensor und das Gyroskop.

Umweltsensoren Sensoren zur Bestimmung von Umweltparametern wie Luftdruck, Umgebungshelligkeit etc.

Positionssensoren Sensoren um die physikalische Position des Geräts zu messen. Hierunter fällt der geomagnetische Feld Sensor (Kompass).

Interessant für den Anwendungsfall der Lagebestimmung sind folgende von Android zur Verfügung gestellte Hardware-Sensoren:

ACCELEROMETER Misst die Beschleunigungskräfte auf allen drei Achsen inklusive der Gravitation in m/s^2.

GYROSCOPE Misst die Rotation auf allen drei Achsen in rad/s.

MAGNETIC_FIELD Misst die Stärke des magnetischen Felds auf allen drei Achsen (Kompass) in μT.

Aus diesen drei Sensoren ergeben sich softwarebasierte virtuelle Sensoren, welche in Android wie hardwarebasierte Sensoren angesteuert werden können. Zu diesen zählt ein 3-Axen Gravitationssensor, welcher aus einer Kombination von Gyroskop und Beschleunigungssensor errechnet wird und der sogenannte Rotationsvektor, welcher die Lage des Geräts auf Basis des Gravitationssensors und des magnetischen Feldsensors als Vektor zur Verfügung stellt [22, S. 13].

3.2.2 GPS

Neben der Lagebestimmung ist eine Positionsbestimmung des Smartphones notwendig. Diese kann durch den integrierten GPS (Global Positioning System) Empfänger eines modernen Smartphones durchgeführt werden. In dieser Arbeit werden Positionsangaben in der Form Längen- und Breitengrad, welche beispielsweise durch das GPS ermittelt wurden, auch als Geopositionen (geographische Positionen/Koordinaten) bezeichnet um diese von anderen Koordinatensystemen zu unterscheiden.

3.2.3 Kamera

Moderne Smartphones besitzen meist zwei integrierte Kameras. Für den Einsatz in dieser Arbeit ist die rückwärtige Kamera von Interesse, da diese die Umwelt hinter dem Bildschirm erfasst und das Kamerabild damit als 'Fenster zur virtuellen Realität' angesehen werden kann.

3.3 OpenGL für Smartphones

Um eine grafische Überlagerung des Kamerabildes zu erreichen, bedarf es der Möglichkeit im dreidimensionalen Raum performant zu zeichnen. Moderne Smartphones verfügen heutzutage neben diverser Sensoren auch über eine leistungsstarke Grafikeinheit für 2D- und 3D-Anwendungen. Um Grafiken auf dem Bildschirm zu zeichnen bietet Android neben einem Canvas Objekt für 2D-Zeichnungen eine OpenGL ES Schnittstelle für 2D- und 3D-Anwendungen [4]. OpenGL ES ist eine für eingebettete Systeme optimierte Variante von OpenGL, eine Schnittstelle zur Entwicklung von 2D- und 3D-Anwendungen. Die Verwendung von OpenGL ES bietet hierbei viele Vorteile, da für die korrekte Überlagerung des Kamerabildes mit Informationen eine Möglichkeit zur Zeichnung in einer 3D-Welt benötigt wird und damit eine aufwendige Implementierung einer eigenen Unterstützung unnötig würde. Zudem verfügt OpenGL ES auf modernen Smartphones über Hardwareunterstützung und bietet damit einen Leistungsvorteil gegenüber anderer Implementierungen. Android unterstützt neben OpenGL ES 1.0/1.1 ab Android 2.2 auch OpenGL ES 2.0. Diese neuere Version arbeitet nicht mehr mit festen API Befehlen, beispielsweise zur Manipulation der Kameraperspektive, sondern gibt dem Nutzer mehr Freiheit durch die Verwendung von Vertex- und Fragment Shadern und arbeitet auf modernen Grafikeinheiten effizienter [18].

OpenGL verarbeitet Daten schrittweise in der Form einer Pipeline. Hierbei werden eingehende Daten durch verschiedene Schritte immer weiter verarbeitet, bis sie später im Frame Buffer zur Zeichnung auf dem Bildschirm bereit liegen (siehe Abbildung 3.3).

3.3.1 Shader

OpenGL Shader sind Programme, entwickelt in einer Shadersprache GLSL, welche an C angelehnt ist, die zur Laufzeit in Maschinencode übersetzt werden und dort in den Shadereinheiten der Grafikkarte (oder auch auf der CPU) ausgeführt werden. Zur Umsetzung der virtuellen 3D-Welt werden Vertex- und Fragmentshader verwendet. Hierbei arbeitet der Vertexshader auf den Eckpunkten (Vertex) von zu zeichnenden Objekten und kann diese beispielsweise mit einer Matrix multiplizieren und somit ihre endgültige Position manipulieren. Der Fragmentshader kann auf die in der Grafikeinheit integrierten Textureinheiten zurückgreifen und wird zur Manipulation von Farbe oder Textur der zu zeichnenden Flächen verwendet. Textureinheiten sind hierbei Zwischenspeicher in der Grafikeinheit, welche sich mit Texturen befüllen lassen. Um mit den kompilierten Shadern zu arbeiten, können darin definierte Variablen über den OpenGL Kontext angesteuert werden.

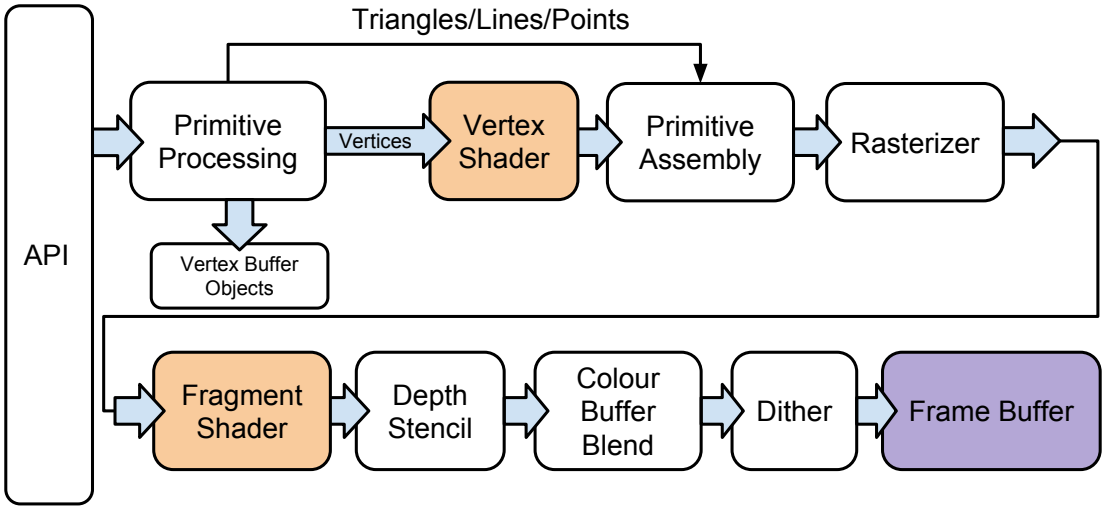

Abbildung 3.3: OpenGL ES 2.0 Pipeline [18]

3.3.2 Zeichnung von Objekten

In OpenGL müssen alle zu zeichnenden Objekte aus den Grundformen Linie, Punkt und Dreieck zusammengesetzt werden. Um beispielsweise ein Rechteck zu zeichnen, müssen mindestens zwei Dreiecke an OpenGL übergeben werden. Die Eckpunkte dieser Dreiecke, auch vertices (vertex) genannt, werden in Java als Fließkomma-Array hinterlegt und mittels eines 'ByteBuffers' in den nativen Speicher geschrieben. Nun können die Vertexdaten an OpenGL übergeben werden.

Vertex Buffer Object

Um Vertexdaten nicht bei jedem Zeichendurchgang erneut an OpenGL übergeben zu müssen, können Vertex Buffer Objects verwendet werden. Hierzu wird ein Buffer in OpenGL generiert und die Daten aus dem nativen Speicher darin eingelesen. Später kann bei jedem Zeichendurchgang der entsprechende Buffer gebunden werden und die darin enthaltenen Vertexdaten verwendet werden, ohne erneut auf den Java-Kontext oder den nativen Speicher zugreifen zu müssen.

Zeichendurchgang

Bei jedem Zeichendurchgang wird nun das Vertex Buffer Object gebunden und eine Refernz zur entsprechenden Variable im Vertexshader aktiviert, in welche die Vertexdaten übertragen werden sollen. Nun werden die Vertices aus dem Buffer an den Vertexshader mittels eines Zeichenbefehls übergeben. Hierbei wird angegeben ob es sich um Vertexdaten von Punkten, Linien oder Dreiecken handelt.

Um das Rechteck in einer bestimmten Farbe zu zeichnen, muss vor dem Zeichenbefehl

eine Referenz zu einer Variable im Fragmentshader aktiviert werden und entsprechende Farbdaten in diese geladen werden.

Wie in Abbildung 3.3 dargestellt werden die Vertexdaten vom Vertexshader verarbeitet und OpenGL setzt die Eckpunkte entsprechend der ausgewählten Form, wie beispielsweise Dreiecke, zusammen, rastert das Ergebnis und führt den Fragmentshader auf dieser Form aus. Hierbei bestimmt der Fragmentshader beispielsweise über Farbe oder Textur der verarbeiteten Form. Nach weiteren graphischen Verarbeitungen erreicht die gezeichnete Form den Frame Buffer und kann auf dem Bildschirm gezeichnet werden. Für den detaillierten Ablauf der Darstellung auf dem Bildschirm siehe Abschnitt 4.

3.4 OpenStreetMap

Das OpenStreetMap Projekt, auch OSM abgekürzt, entstand 2004 mit dem Ziel eine freie Weltkarte zu schaffen [23, S. 3]. Durch die Arbeit freiwilliger Unterstützer, sogenannte Mapper, entsteht im Rahmen des Projekts eine Sammlung geographischer Daten, welche über das reine Kartographieren von Straßenverläufen hinausgeht. So werden von den Unterstützern eine Vielzahl von Metadaten und geographischen Daten gesammelt, wie etwa die Positionen von Briefkästen, Daten über Buslinien, Grundstücksgrenzen oder Positionen von Ampeln [23, S. 65 ff.]. Der entscheidende Unterschied zu Alternativen wie Google Maps ist die Möglichkeit Daten des OpenStreetMap Projekts frei von urheberrechtlichen Ansprüchen nutzen und verarbeiten zu können.

Aufgrund der Vielfalt der verfügbaren Daten und der Verknüpfung mit geographischen Koordinaten wurde die Datenbasis von OSM für diese Arbeit ausgewählt.

Standardmäßig liegen OpenStreetMap Daten im XML-Format vor, jedoch wurde für diese Arbeit das effizientere PBF-Format gewählt. Dieses Format basiert auf Google Protobuf und erreicht neben einer 5-6fach besseren Lese/Schreibgeschwindigkeit auch eine Speicherersparnis von 50% gegenüber einer gzip-komprimierten Datei im XML-Format [21, 9]. Dadurch ist es später möglich ganze Stadtgebiete in kompakter Form auf das Smartphone zu übertragen und mit OSMAR einzulesen.

3.4.1 Aufbau der Daten

Die Daten des OpenStreetMap Projekts bestehen aus den Grundobjekttypen Node, Way und Relation [23, S. 55 ff.]. Um das jeweilige Objekt zu beschreiben werden Metadaten, in Form sogenannter Tags, zugeordnet. Hierbei sind Tags als Key-Value-Paare aufgebaut.

Nodes

Nodes sind Knoten bzw. Punkte und bilden die Grundlage für die übrigen Grundobjekttypen. So repräsentiert eine Node eine geographische Koordinate der Form geographischer Länge und Breite mit optionalen Metadaten als Tags [23, S. 56 f.].

Ways

Ways sind Wege und bestehen aus einer geordneten Liste von mindestens zwei referenzierten Nodes und optionalen Metadaten. Mittels Ways werden linienförmige Objekte wie beispielsweise Straßenverläufe oder Bahnlinien abgebildet [23, S. 57 f.].

Relations

Relations dienen der Modellierung von Beziehungen zwischen Objekten und bestehen aus einer geordneten Liste von sogenannten Membern und optionalen Metadaten. Member sind in diesem Zusammenhang Referenzen auf weitere Relations, Nodes oder Ways. Durch Relations werden beispielsweise Ländergrenzen [23, S. 81] als eine Liste von Ways repräsentiert [23, S. 57 f.].

3.4.2 GPX Dateiformat

Das GPX Dateiformat (GPS Exchange Format) [26] wird von GPS-fähigen Geräten verwendet um Wegpunkte und Strecken in einem einheitlichen Format speichern und auszutauschen zu können [23, S.35]. In JOSM [13], ein Editor für OpenStreetMap, können GPX Dateien eingelesen werden und die importierten Punkte und Wegstrecken direkt in den OSM Datenbestand aufgenommen werden (für weitere Informationen siehe [23, S. 107 ff.]). Zusätzlich unterstützt JOSM eine Erweiterung des GPX Formats, mit Hilfe dessen Audiodateien im WAVE Format mit Kartenpunkten verbunden werden können und direkt im Programm abgespielt werden können. Das GPX Dateiformat wird in dieser Arbeit sowohl für den Export von gespeicherten Wegpunkten, als auch zur Simulation von GPS Positionen verwendet.

3.5 OpenCV

OpenCV, kurz für Open Source Computer Vision, ist eine freie Programmbibliothek für Bildverarbeitung und maschinelles Sehen. In OSMAR kommt OpenCV sowohl zur Kamerakalibrierung (siehe Abschnitt 4.3.1), als auch als Kameraoberfläche (siehe Abschnitt 5.1) zum Einsatz. Hierfür wurde die Java Implementierung von OpenCV für Android übersetzt. Mittels der Integration von OpenCV in das Framework ist, wie im Ausblick Abschnitt 6 beschrieben, eine Verarbeitung des Kamerabildes und damit beispielsweise die Erkennung von Objekten möglich.

4 Darstellung der virtuellen Welt

Die Darstellung der virtuellen Welt ist die größte Herausforderung bei der Erzeugung einer Augmented Reality. In dieser Arbeit soll der Smartphonebildschirm als Fenster zur

virtuellen Realität betrachtet werden. Dies gelingt nur, wenn das Kamerabild korrekt von der Darstellung der virtuellen Welt überlagert wird. Hierzu werden Positionen der echten Welt, im Falle dieser Arbeit Geopositionen der Form Längen- und Breitengrad, in mehreren Schritten transformiert bis sie auf dem Bildschirm des Smartphones angezeigt werden können (Abbildung 4.1).

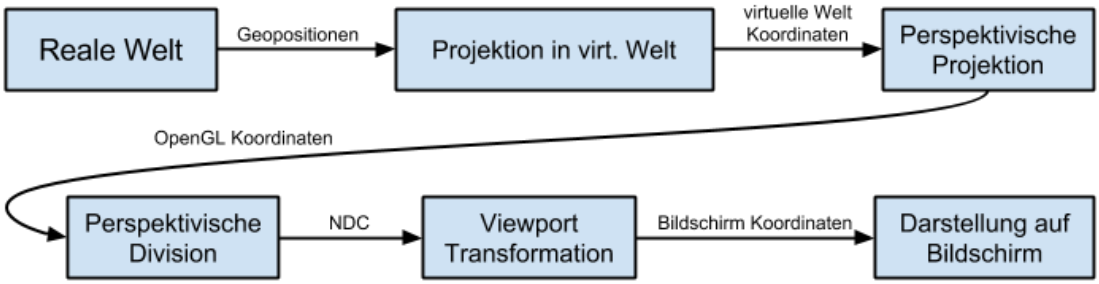

Abbildung 4.1: Konzept der Transformation von Objekten der realen Welt zur Darstellung auf dem Smartphone Bildschirm

4.1 Reale Welt

Punkte der realen Welt werden als Geopositionen im System abgelegt. Als Geopositionen werden in dieser Arbeit geographische Koordinaten der Form Längen- und Breitengrad bezeichnet. Diese Koordinaten stammen beispielsweise aus dem OpenStreetMap Bestand und bieten keine Information über die Höhe, sind somit zwei-dimensional.

4.2 Projektion in die virtuelle Welt

Die Projektion der Geopositionen in die virtuelle Welt umfasst mehrere Teilbereiche die im Folgenden erläutert werden:

4.2.1 Koordinatensystem der virtuellen Welt

Zur Repräsentation der virtuellen Welt muss ein dreidimensionales Koordinatensystem gewählt werden um später Objekte wie Straßenschilder oberhalb der zweidimensionalen Ebene darstellen zu können. Zur Umsetzung wurde ein kartesisches Koordinatensystem gewählt. Die Verwendung eines Koordinatensystems auf Basis von Längen- und Breitengrad wurde aufgrund der krummlinigen Beschaffenheit ausgeschlossen.

Die Skalierung des Koordinatensystems ist durch die darauf folgende perspektivische Projektion frei wählbar. Um die spätere Darstellung der virtuellen Welt möglichst einfach zu gestalten, wurde ein metrisches System gewählt. Hierbei entspricht eine Einheit auf jeder der 3 Achsen jeweils einem Meter. Für die Achsen des orthogonalen Koordinatensystems gilt:

- X-Achse zeigt Richtung Osten.
- Y-Achse zeigt Richtung Norden.
- Z-Achse steht rechtwinklig zur Erdoberfläche und zeigt Richtung Himmel.

4.2.2 Projektion geographischer Koordinaten

Um geographische Koordinaten in das gewählte metrische Koordinatensystem zu überführen bedarf es einer passenden Projektion. Das UTM Koordinatensystem erfüllt diese Aufgabe, da es mittels transversaler Mercator-Projektion die Erdoberfläche ebnet und mit einem kartesischen Koordinatensystem überzieht, welches dem oben gewählten Koordinatensystem entspricht. Ebenso stimmt die metrische Skalierung überein. Für genauere Informationen über das UTM Koordinatensystem siehe Quelle [1]. Somit ist eine Projektion von geographischen Koordinaten in Längen- und Breitengrad in das UTM Koordinatensystem möglich und damit eine einfache Übertragung in das Koordinatensystem der virtuellen Welt mit der Höhe $z = 0$. Dank dieser Umsetzung kann später eine geographische Koordinate als Referenz angesehen werden und beispielsweise auf Basis dieser ein Straßenschild in der Höhe von 2 Metern erstellt werden.

4.3 Perspektivische Projektion

Um Objekte im Koordinatensystem der virtuellen Welt auf dem Bildschirm darstellen zu können ist eine perspektivische Projektion dieser Koordinaten notwendig. Dieser Teilbereich umfasst sowohl die Positionierung und Orientierung der virtuellen Kamera, als auch die eigentliche perspektivische Projektion.

4.3.1 Kameramatrix

Die perspektivische Projektion bildet die Sicht der Kamera auf die virtuelle Welt ab, daher ist es notwendig die verwendete Kamera bestmöglich als virtuelle Kamera umzusetzen, um eine realistische Überlagerung des Kamerabildes zu erreichen. Diese wichtigen Eigenschaften der Kamera lassen sich als intrinsische Parameter abbilden. Hierfür wurde eine Kamerakalibrierung anhand von Kreismustern (siehe Abbildung 4.2), mittels OpenCV durchgeführt [16] und damit die Kameramatrix ermittelt. Um auf Android eine Kamerakalibrierung durchzuführen bedarf es neben der Installation der OpenCV Bibliothek noch der Kalibrierungsanwendung, welche aus den Beispielimplementierungen von OpenCV [17] entnommen werden kann.

Hieraus resultiert die Kameramatrix mit den intrinsischen Parameter der Kamera:

$$\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Hierbei steht f_x bzw. f_y für die Brennweite (focal length) und c_x bzw. c_y für den Hauptpunkt (principle point, überlicherweise die Bildmitte) in Pixel. s wird als Parameter

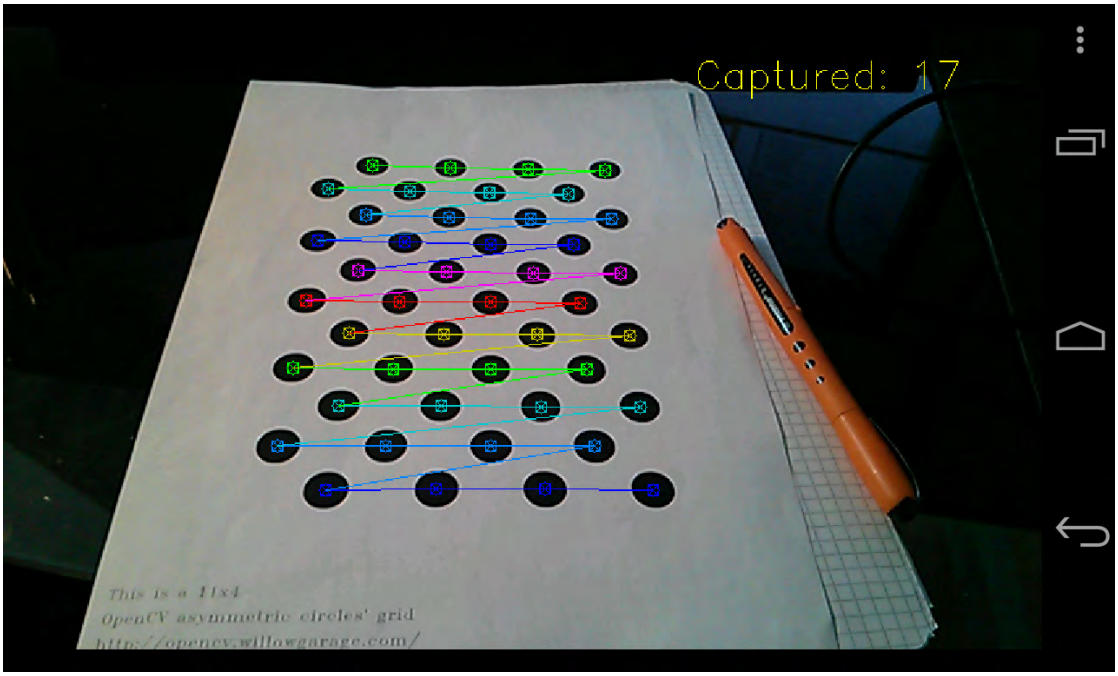

Abbildung 4.2: Beispielhafte Kamerakalibrierung mittels OpenCV auf einem Android-Smartphone

für die Schiefe/Schräge (skew) der Bildkoordinaten bezeichnet und tritt auf wenn die Bildkoordinaten-Axen nicht orthogonal zueinander stehen, hierbei gilt bei modernen Kameras $s = 0$ [25, S. 2][11, S. 157]. Durch die Kamerakalibrierung können zusätzlich Verzerrungsparameter ausgewertet werden um das Kamerabild zu perfektionieren. Im Rahmen dieser Arbeit wurde aufgrund von Unbekannten, wie beispielsweise der endgültigen Kamerahöhe des Benutzers, auf diese Feinjustierung nicht näher eingegangen. Mit diesen intrinsischen Daten kann die gewünschte Perspektivmatrix erstellt werden, welche zur perspektivisch korrekten Projektion der 3D-Welt benötigt wird.

4.3.2 Lochkameramodell

Vor Erstellung einer geeigneten Perspektivmatrix für den Einsatz in OpenGL bedarf es einer Sicht auf das allgemeine Lochkamera-Modell nach Hartley-Zisserman. Das folgende Verfahren zur Ermittlung eines OpenGL Frustums anhand intrinsischer Kamera-Parameter basiert auf Beschreibungen von Koshy George (siehe Quelle [7]). Zur Veranschaulichung des Lochkamera-Modells nach Hartley-Zisserman dient die Abbildung 4.3. Hierbei werden Punkte rechts der Focal plane (auch Kamera- oder Fokus-Ebene) auf die Focal plane abgebildet. Die Z-Achse zeigt Richtung Focal plane und schneidet diese in ihrem Mittelpunkt. p steht hierbei für die 'focal length', also die Brennweite. Die vierte Koordinate w wird auch homogene Vertex Koordinate genannt und dient später zur Abbildung der Tiefe (siehe Abschnitt 4.4). Die Transformation eines drei dimensionalen

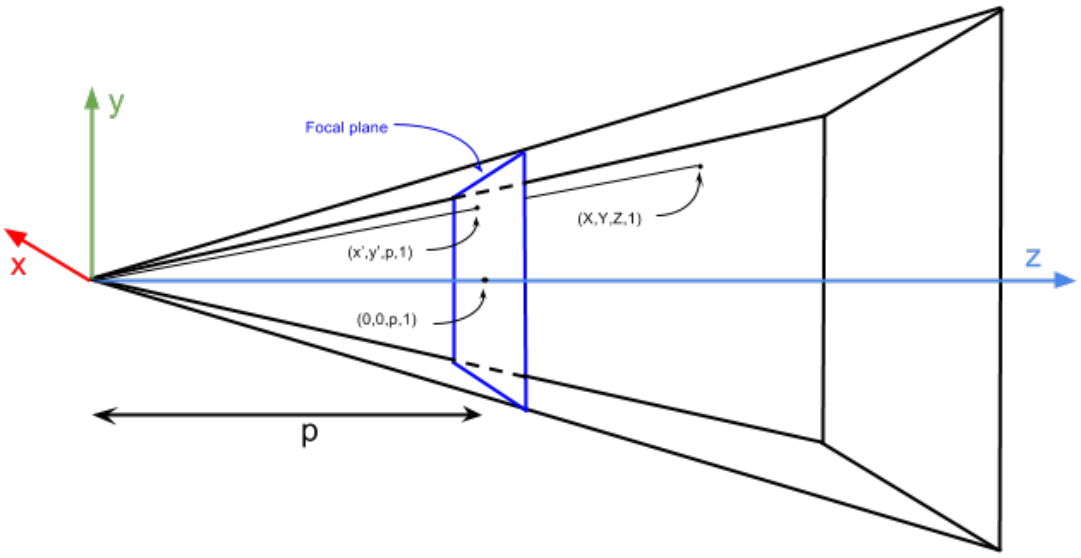

Abbildung 4.3: Lochkamera nach Hartley-Zisserman [11, S. 154, Fig. 6.1.]

Punktes zur Rechten der Focal plane auf die Focal plane kann durch eine Matrixmultiplikation mit anschließender perspektivischen Division (Division durch w, gekennzeichnet durch $\approx$) abgebildet werden.

$$\begin{pmatrix} x' \\ y' \\ p \\ 1 \end{pmatrix} \approx \begin{pmatrix} p & 0 & 0 & 0 \\ 0 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

In diesem Modell bildet die Koordinate $(0, 0, p)^T$ den Mittelpunkt der Focal plane (Kameraebene). Um nun die durch die Kameramatrix repräsentierte Kamera in diesem Modell abzubilden muss der Unterschied der Koordinatensysteme beachtet werden. Die Relation zwischen Kamera- (focal plane) und Bildebene (image plane) wird in Abbildung 4.4 veranschaulicht. Durch die Kameramatrix ist die Koordinate dieses Mittelpunktes auf der Bildebene als (c_x, c_y, p) bekannt.

Die Transformation eines Punktes der Focal plane auf die Bildebene kann wie folgt abgebildet werden:

$$\begin{pmatrix} x \\ y \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} k_x & 0 & \frac{c_x}{p} & 0 \\ 0 & k_y & \frac{c_y}{p} & 0 \\ 0 & 0 & \frac{1}{p} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ p \\ 1 \end{pmatrix}$$

Hierbei steht k_x bzw. k_y für den Skalierungsfaktor zwischen Pixel-Koordinate und Kamera-Raum entlang der X- bzw. Y-Achse und c_x bzw. c_y sind Punkte der Bildebene und bilden den Mittelpunkt der Kameraebene. Beide Transformationen können zusam-

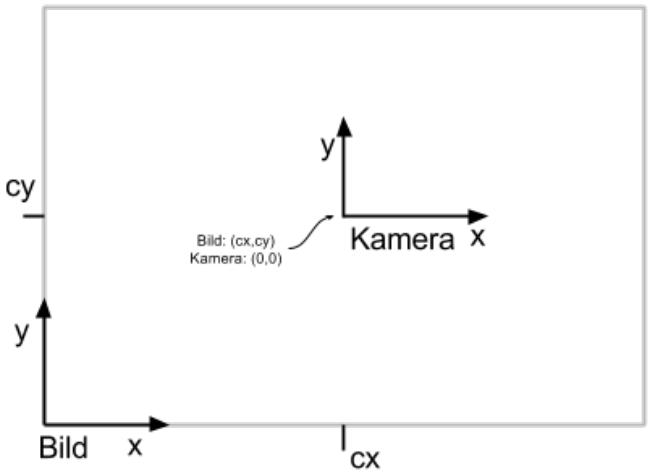

Abbildung 4.4: Relation zwischen Bild- und Kamera Koordinatensystem [11, S. 155, Fig. 6.2.][7]

mengefasst werden:

$$\begin{pmatrix} x \\ y \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} p \cdot k_x & 0 & c_x & 0 \\ 0 & p \cdot k_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

4.3.3 OpenGL Kameramodell

In OpenGL werden Kameras entlang der negativen Z-Achse ausgerichtet (siehe Abbildung 4.5).

Um diesen Umstand auf die vorherige Transformation zu übertragen bedarf es einer Negation der Z-Achse:

$$\begin{pmatrix} x \\ y \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} p \cdot k_x & 0 & -c_x & 0 \\ 0 & p \cdot k_y & -c_y & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Die Perspektivmatrix projiziert später die 3D-Welt-Koordinaten auf eine 2D-Ebene und verwendet die Z-Achse um die Tiefe abzubilden. In OpenGL wird hierbei ein Frustum (Pyramidenstumpf) (siehe Abbildung 4.6) aufgebaut, welches aus einer 'near plane' und einer 'far plane' besteht und damit den sichtbaren Bereich eingrenzt (auch Clipping planes genannt). Hierdurch wird ein sogenanntes Clipping ermöglicht, also ein Abschneiden von Objekten am Rand des sichtbaren Bereichs. Die Ermittlung von Punkten außerhalb des sichtbaren Bereichs welche nicht gezeichnet werden müssen wird auch Frustum Culling genannt.

Aus der OpenGL Literatur kann die notwendige Matrix zur Konstruktion eines Frustums in OpenGL entnommen werden, für eine mathematische Herleitung siehe [2].

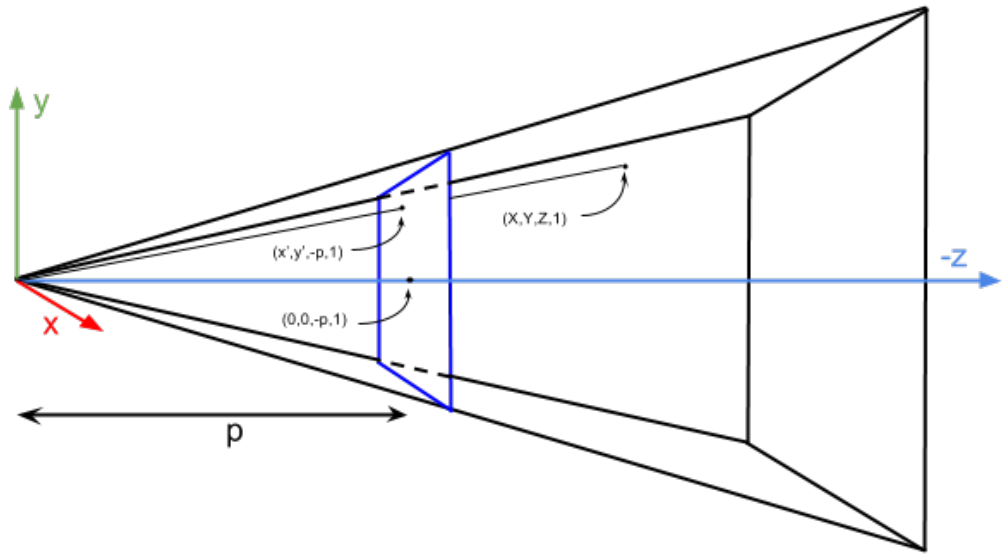

Abbildung 4.5: Lochkamera in OpenGL [7]

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Hierbei gilt: $l = left, r = right$ Koordinaten der Ecken der near plane auf der X-Achse. $t = top, b = bottom$ = Koordinaten der Ecken der near plane auf der Y-Achse. $n = near, f = far$ = Koordinaten zur near- und far plane auf der Z-Achse (siehe Abbildung 4.6).

Da das Frustum der definierten Kamera aus symmetrischen Ebenen besteht, kann diese Projektionsmatrix noch vereinfacht werden (siehe [2]), da $r = -l$ und $t = -b$ resultiert $r + l = 0$, $r - l = 2r$, $t + b = 0$ und $t - b = 2t$:

$$\begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Die Abstände zur Near- und Farplane sind bekannt bzw. können später der Implementierung angepasst werden. Die Unbekannten sind r und t, in welche nun die intrinsischen Parameter der echten Kamera und damit vorherige Überlegungen der Lochkamera einfließen.

Hierfür muss der Zusammenhang zwischen Focal plane des Lochkamera Modells und der Near plane im OpenGL Frustum betrachtet werden (siehe Abbildung 4.7). Die Near plane im OpenGL Frustum entspricht der Bildebene im Lochkamera Modell und muss

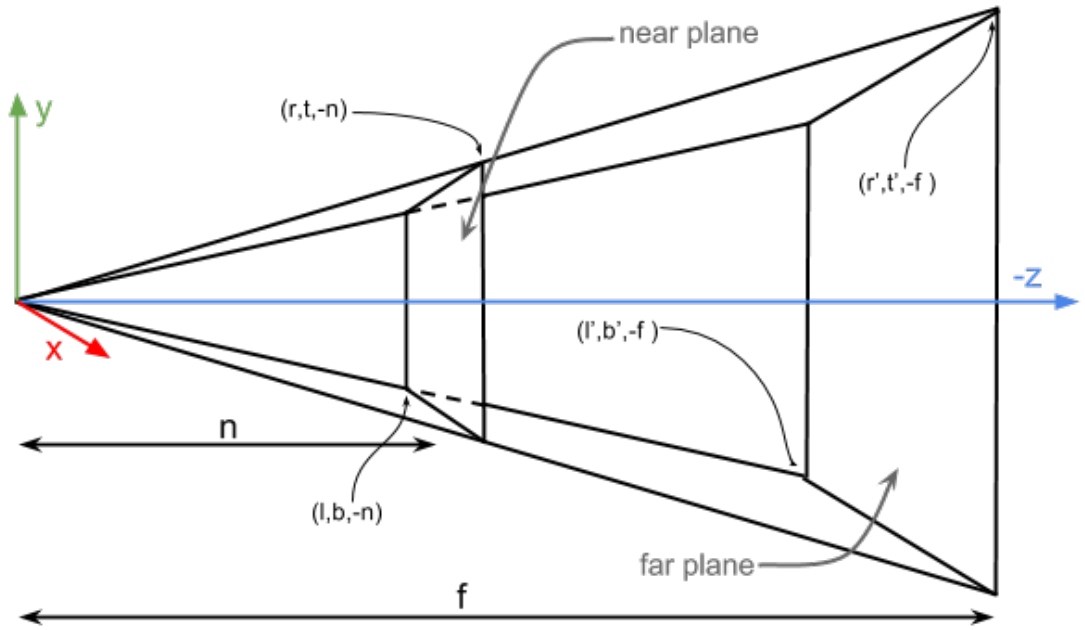

Abbildung 4.6: OpenGL Frustum [2, 7]

nun entsprechend übertragen werden. Hierbei gilt auf der x-Achse für die Verhältnisse zwischen Ausmaß der Near-plane zum Abstand zur Near-Plane und Ausmaß der Focal-plane zum Abstand zur Focalplane:

$$\frac{r}{n} = \frac{s}{p}$$

$$r = \frac{s \cdot n}{p}$$

s entspricht auf der Bildebene c_x. Auf der Kameraebene ergibt sich durch Division mit dem Skalierungsfaktor zwischen Pixel-Koordinatensystem/Kameraraum $s = \frac{c_x}{k_x}$ und somit:

$$r = \frac{c_x \cdot n}{k_x \cdot p}$$

Mit $f_x = k_x \cdot p$ gilt

$$r = \frac{c_x \cdot n}{f_x}$$

und analog für t auf der y-Achse

$$t = \frac{c_y \cdot n}{f_y}$$

.

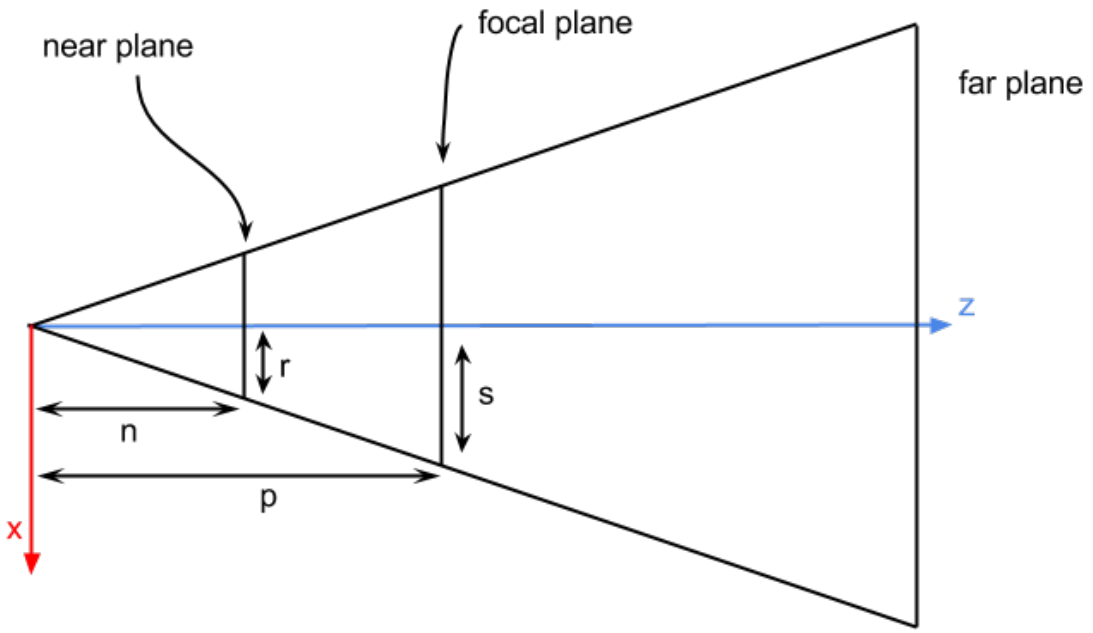

Abbildung 4.7: Relation zwischen Focal plane und near plane [7]

Durch Einsetzen in die ermittelte Frustum-Matrix erhalten wir die endgültige Perspektivmatrix zur Verwendung in OpenGL:

$$\begin{pmatrix} \frac{f_x}{c_x} & 0 & 0 & 0 \\ 0 & \frac{f_y}{c_y} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Diese Matrix dient im Folgenden der perspektiv korrekten Darstellung der 3D-Welt in OpenGL. Im weiteren Prozess werden alle zu zeichnenden Punkte der 3D-Welt mit dieser multipliziert um sie perspektivisch korrekt abzubilden.

4.3.4 Projektion

Die eigentliche Matrixmultiplikation mit dem entsprechenden Vektor findet im Shaderprogramm statt und wird direkt von der Grafikeinheit berechnet. Hierbei wird der dreidimensionale Vektor einer Koordinate in der virtuellen Welt $(x_{world}, y_{world}, z_{world})^T$ in OpenGL als vierdimensionaler Vektor repräsentiert und implizit $w = 1$ gesetzt. w wird auch homogene Vertex Koordinate genannt und dient später zur Normalisierung des

entsprechenden Vektors [19].

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = \begin{pmatrix} \frac{f_x}{c_x} & 0 & 0 & 0 \\ 0 & \frac{f_y}{c_y} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot T \cdot \begin{pmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{pmatrix}$$

Hierbei repräsentiert T die Transformationsmatrix zur korrekten Anordnung der Weltkoordinaten. Diese besteht aus einer Rotation basierend auf den Lagesensoren des Smartphones (siehe Abschnitt 4.6) und einer Translation zur Bewegung der Kamera im Raum (siehe Abschnitt 5.5.2). Es wird also die 'Welt' statt der 'Kamera' transformiert. Dies erleichtert die spätere Verarbeitung, da die Perspektivmatrix nicht ohne weiteres auf diese Weise transformiert werden könnte. Nachdem das Ergebnis dieser Berechnung in der Shadereinheit vorliegt, führt OpenGL ein sogenanntes Clipping durch. Hierbei wird geprüft, ob x_{clip}, y_{clip} und z_{clip} im Wertebereich $\pm w_{clip}$ liegen und werden ansonsten verworfen. Dadurch werden nur sichtbare Vertices gezeichnet [3]. Wenn diese Punkte zu einem Objekt gehören, fügt OpenGL entsprechende Kanten ein, sollten durch das Clipping Eckpunkte verworfen worden sein [2] (siehe Abbildung 4.8).

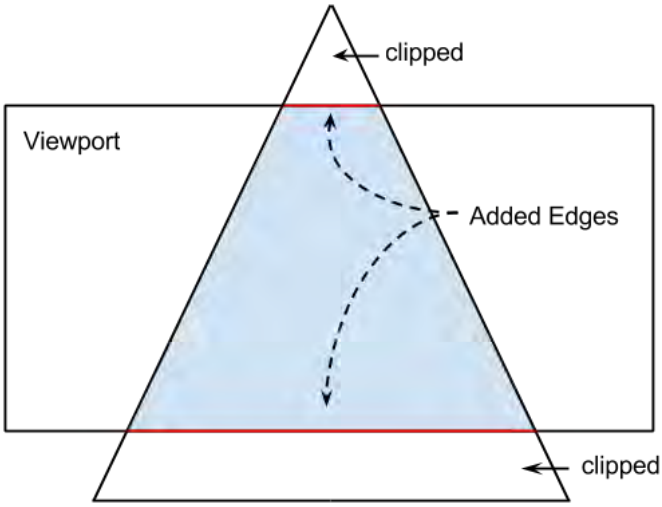

Abbildung 4.8: Clipping eines Dreiecks durch ein Frustum [2]

4.4 Perspektivische Division

Auf den entstandenen Koordinaten der Form $(x_{clip}, y_{clip}, z_{clip}, w_{clip})^T$ führt OpenGL nun eine perspektivische Division (auch perspektivische Teilung) durch. Hierdurch entstehen normalisierte Gerätekoordinaten (NDC, normalized device coordinates) im Wertebereich

von $[-1; +1]$ auf allen 3 Achsen [3].

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

4.5 Darstellung auf dem Bildschirm

Die Darstellung auf dem Bildschirm und damit die Transformation der normalisierten Geräte Koordinaten auf die Zeichenoberfläche übernimmt OpenGL automatisch unter Angabe der Größe der Zeichenoberfläche. Zu diesem Zeitpunkt gibt die z-Achse die Tiefe des zu zeichnenden Punktes an und so existieren unter Umständen auf der selben x- und y-Koordinate mehrere zu zeichnende Pnukte. Ohne weitere Arbeit würde als stets derjenige Punkt gezeichnet, welcher als letztes in einem Zeichendurchgang die OpenGL Pipeline durchläuft. Durch die Aktivierung des Tiefentests und der Angabe einer Tiefenfunktion wird dieser Umstand korrigiert und der Punkt mit der geringsten Tiefe wird gezeichnet [10].

4.6 Extrinsische Kameraparameter - Lagesensoren

Die extrinsischen Kameraparameter, also die Beschreibung von Position und Orientierung, werden aus den Sensordaten des Smartphones gewonnen. Hierbei wird die Position der Kamera später von GPS-Koordinaten bestimmt. Um die Lage des Smartphones bestimmen zu können sind mehrere Kombinationen aus Sensoren möglich und wurden auf ihre Nutzbarkeit getestet: Eine Kombination aus Beschleunigungssensor und Kompass lieferte erste Ergebnisse, ist aber trotz der Verwendung einiger Filter zu fehleranfällig. Eine Kombination aus dem virtuellen Gravitationssensor in Kombination mit dem Kompass liefert bedeutend bessere Ergebnisse. Die besten Ergebnisse lieferte der virtuelle Sensor Rotationsvektor ('ROTATION_VECTOR'). Dieser wird seit Android 2.3 (API Level 9) von Android bereit gestellt und liefert die absolute Lage des Geräts durch eine Fusion von Daten aus dem Gyroskop, dem Beschleunigungssensor und dem Kompass [5] [22, S. 13]. Eine Veranschaulichung der verschiedenen Sensorkombinationen zur Lagebestimmung liefert die Android App 'Sensor fusion' von Alexander Pacha [8].

Durch diesen Sensor erhält man die 3 Elemente des Rotationsvektors des Smartphones:

- Die X-Achse verläuft tangential zur Erdoberfläche und zeigt Richtung Osten.
- Die Y-Achse verläuft ebenfalls tangential zur Erdoberfläche und zeigt Richtung magnetischen Nord Pol.
- Die Z-Achse zeigt Richtung Himmel und steht rechtwinklig zur Erdoberfläche.

Anhand dieser Daten werden in OpenGL die Koordinaten der virtuellen Welt rotiert. Hierzu wird mittels einer android-internen Funktion eine Rotationsmatrix aus den Daten des Rotationsvektors berechnet. Da statt der 'Kamera' die 'Welt' rotiert wird, wird X- und Y-Achse der Rotation vertauscht und die X-Achse negiert.

4.7 Problematik: Genauigkeit der Sensoren

Um eine realistische Augmented Reality zu erzeugen, bedarf es sehr genauer Sensordaten über Lage und Position des Smartphones. Diese Anforderung kann ein aktuelles Android Smartphone nur zum Teil erbringen. Im Folgenden werden die Sensordaten einzeln betrachtet und auf ihren Nutzen geprüft. Alle Messungen der Lage basieren hierbei auf dem in Abschnitt 4.6 genannten Rotationsvektor. Hierfür wurde die Rotation um alle 3 Achsen in Grad ($[-180, +180]$) berechnet.

4.7.1 Lagesensoren

Um genaue Sensorergebnisse zu erhalten muss das Smartphone vor Verwendung der Sensoren auf allen Axen gedreht werden, wodurch eine automatische Kalibrierung stattfindet. Ein Test der Sensoren in Ruhelage (siehe Abbildung 4.9) zeigt eine geringe Abweichung, jedoch ist eine stärkere Abweichung auf der Rotation um die Z-Achse (der magnetische Feld Sensor, Kompass) zu beobachten. Hierbei weicht die Rotation um die X- und Y-Achse in Ruhelage nur weniger als 0,05° ab. Bei der Rotation um die Z-Achse, bestimmt durch den Kompass, sind vergleichsweise hohe Abweichungen um 0,1° zu beobachten.

4.7.2 Kompass

Der Kompass bzw. der magnetische Feld Sensor ist bei der Erzeugung einer Augmented Reality von besonderer Bedeutung, da dieser die grundlegende Kamerarichtung vorgibt. Wie bereits in Abschnitt 3.2.1 angesprochen entspricht der Kompass der Drehung um die Z-Achse innerhalb der Rotationsmatrix. Hierbei konnte eine gute Sensorgenauigkeit in Ruhelage (siehe Abbildung 4.9) bestätigt werden. Der Anwender einer Augmented Reality Anwendung ist jedoch auch auf eine geringe Trägheit der auszuwertenden Sensoren angewiesen, so dass eine Bewegung zur einer realistischen Kamerabewegung führt. Hierzu wurde eine Messung der Trägheit der Rotation um alle 3 Achsen durchgeführt. Die Ergebnisse einer vollständigen Drehung mit anschließender Ruhelage (x=0) sind in Abbildung 4.10 dargestellt. Hierbei ist eine vergleichsweise hohe Trägheit des Kompass zu erkennen, welche sich später auf die Trägheit der virtuellen Kamera auswirkt.

4.7.3 Gravitation

Obwohl die Daten des Gravitationssensors eine annehmbare Trägheit aufweist, so wurden bei längerem Einsatz Probleme beobachtet, welche zur Folge hatten, dass sich der künstliche Horizont verschiebt. Es wird angenommen, dass es sich hierbei um ein Problem der internen Sensorkalibrierung durch Android handelt. Ein weiteres Problem ist der Einsatz in einem Fahrzeug, da hierbei die Lageermittlung durch die Beschleunigungskräfte verfälscht wird und sich die Lage erst nach mehreren Sekunden normalisiert. Abbildung 4.11 zeigt eine besonders starke Abweichung des künstlichen Horizonts.

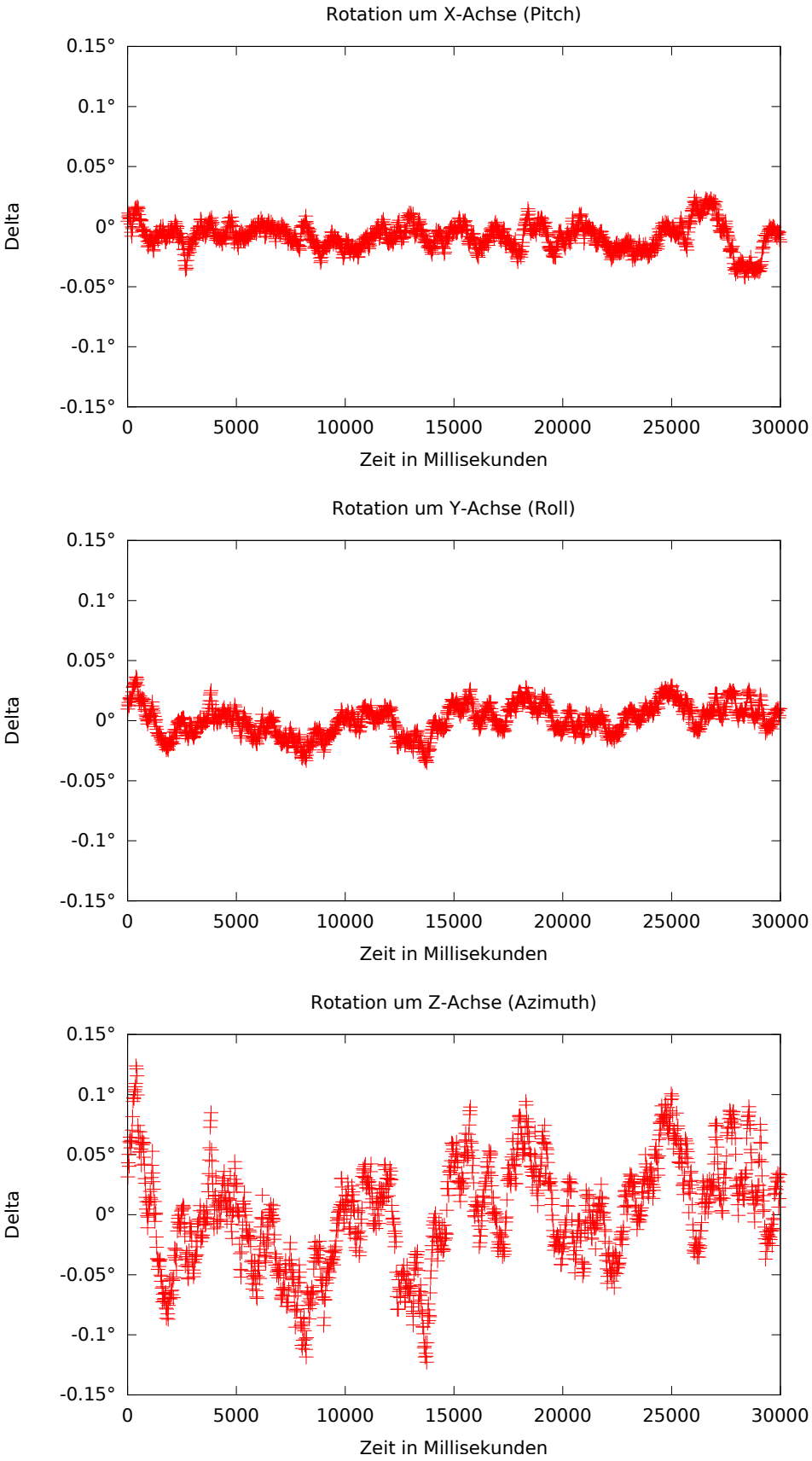

Abbildung 4.9: Relative Lageveränderung der Achsen in Ruhelage

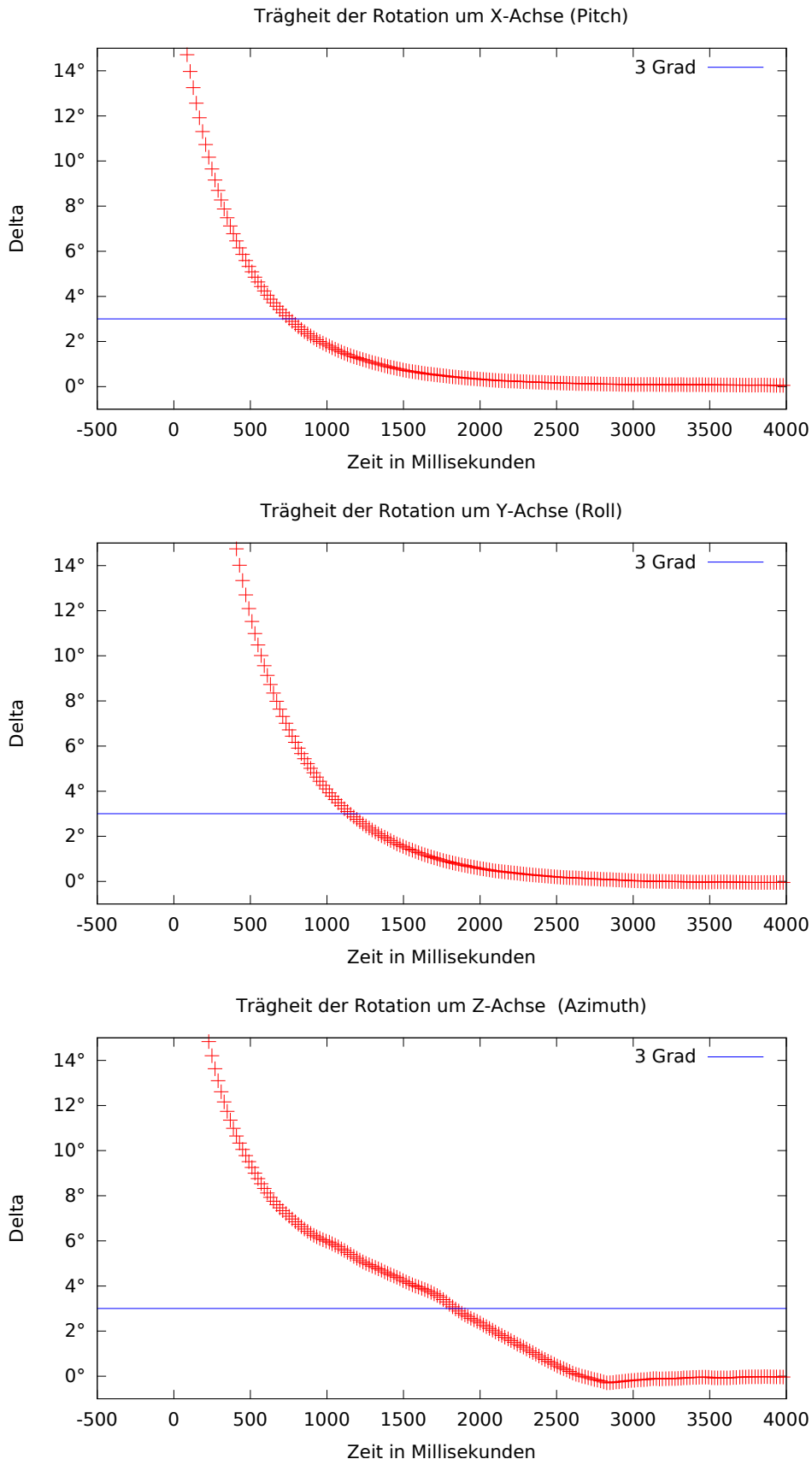

Abbildung 4.10: Trägheit der Achsen nach einer vollständigen Drehung

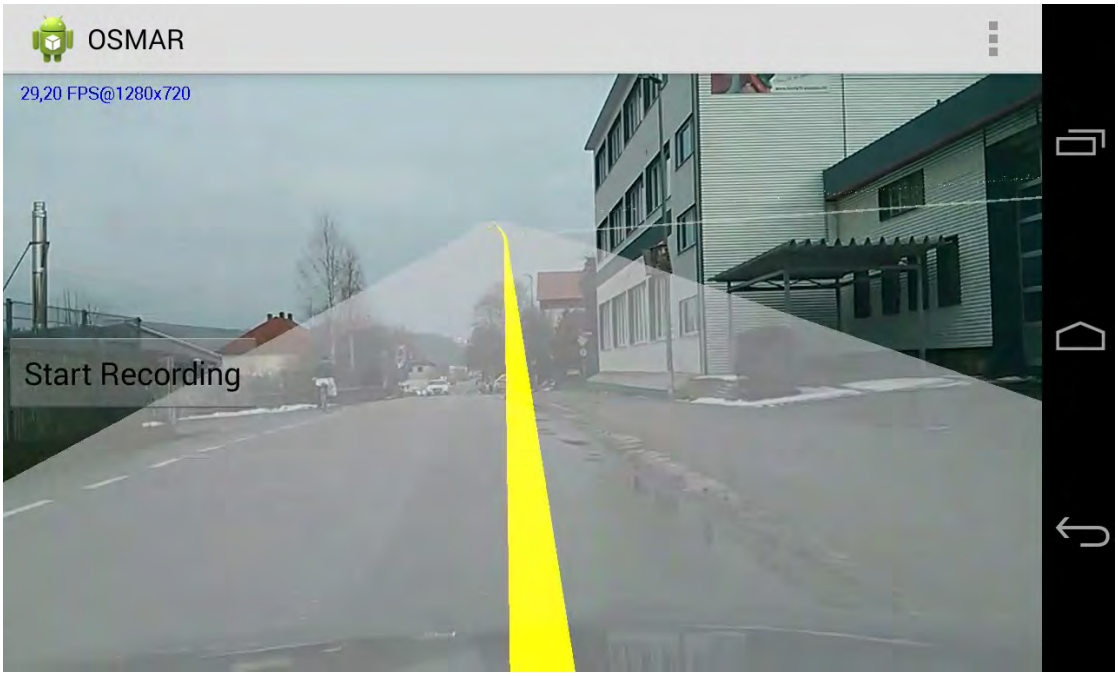

Abbildung 4.11: Verschiebung des künstlichen Horizonts bei längerem Einsatz im Fahrzeug

4.7.4 GPS

Zur Bestimmung der Position wird der integrierte GPS-Empfänger verwendet. Dieser erbringt in Abhängigkeit der Anzahl verfügbarer Satelliten und Störungen durch die Umgebung im Alltagseinsatz häufig nur eine geringe Genauigkeit und ist Schwankungen unterworfen (siehe Abbildung 4.12). OSMAR stellt Schnittstellen zur Verfügung um eingehende GPS Positionsangaben filtern und verarbeiten zu können um die Genauigkeit der Positionsangabe zu verbessern (siehe Abschnitt 5.3). Hierbei kann, wie bereits beispielhaft als GPS Optimierung implementiert (siehe Abschnitt 5.3), eine Optimierung auf Basis der vorliegenden Kartendaten erfolgen.

5 Implementierung

Diese Arbeit wurde für die Android Smartphone API entwickelt und ist kompatibel mit Android Version 4.4.

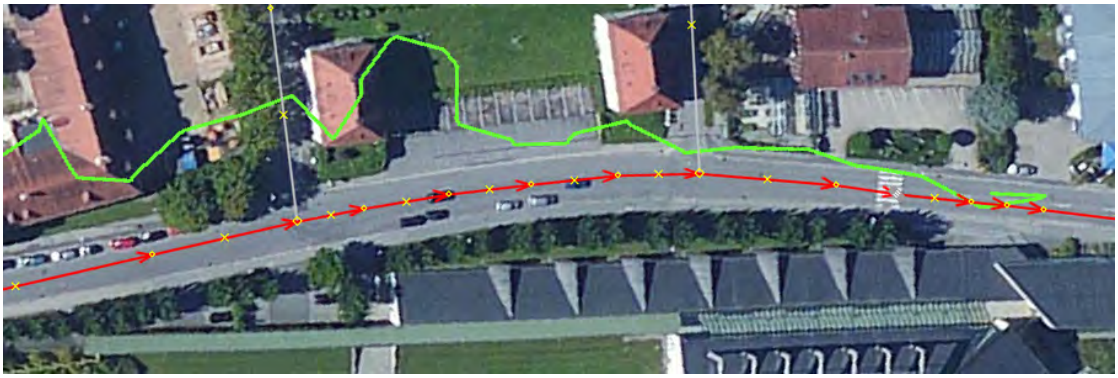

Abbildung 4.12: GPS Abweichung bei Aufzeichnung in Schrittgeschwindigkeit auf Straße (GPS in Grün, Straßenverlauf in Rot)

5.1 Bedienoberfläche

Um den Bildschirm möglichst vollständig zur Darstellung des überlagerten Kamerabildes zu nutzen wurde eine schlichte Oberfläche gewählt, in welcher der Nutzer alle Funktionen über die sogenannte Actionbar im oberen Teil des Bildschirms erreichen kann (siehe Abbildung 5.1). Zu diesen zählt:

- Open PBF File: Einlesen einer OSM Weltdatei
- GPS Settings: Auswahl des GPS Subsystems
- GPS Optimization: Wahl einer GPS Optimierung
- New Session: Starten einer neuen Sitzung (Aufnahme von Wegpunkten)
- Save Session: Speichern der aktuellen Sitzung

Um einen schnellen Zugriff zur Aufnahme von Wegpunkten zu ermöglichen wurde ein Button zum Starten und Beenden einer Wegpunktaufnahme transparent auf der linken Seite des Bildschirms eingesetzt.

Kamerabild

Das Hauptcharakteristika einer Augmented Reality Anwendung ist die Überlagerung eines Live-Kamerabildes mit Informationen. Als Kamera dient die eingebaute rückwärtige Kamera des Smartphones. Hierbei gibt es mehrere Möglichkeiten eine solche Überlagerung herbeizuführen. Eine Möglichkeit ist das Auslesen des Kamerabildes und die Einbettung als Hintergrund in OpenGL. Diese Umsetzung hätte den Vorteil, dass die Framerate des OpenGL Renderings, also der 3D-Welt, und der Kamera synchron verlaufen würden, da ausschließlich OpenGL für die Darstellung verantwortlich wäre. Diese Möglichkeit wurde nicht gewählt, da das Kamerabild eingelesen, als Textur in die Speicherbereiche von OpenGL übertragen und die Position dieser Hintergrundtextur mit

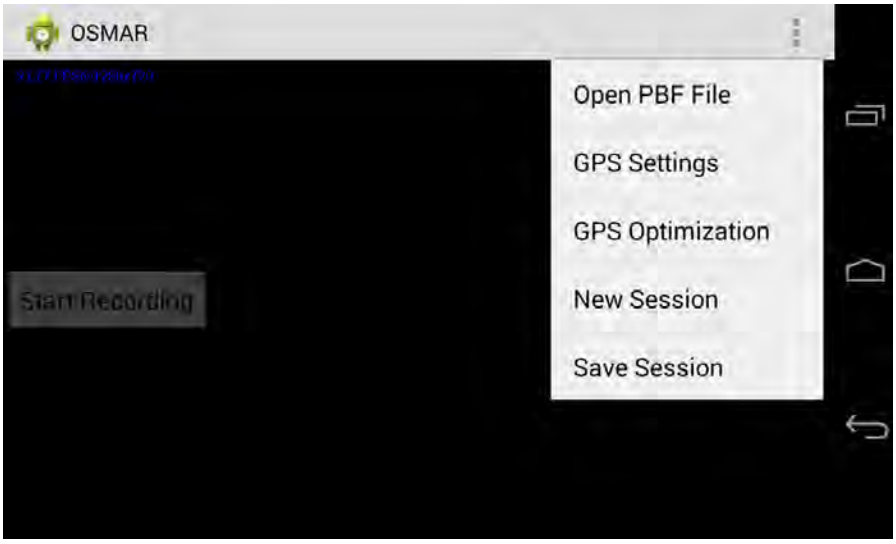

Abbildung 5.1: OSMAR: Benutzerinterface mit aufgeklapptem Menü

der Änderung der Lage des Smartphone verändert werden müsste und somit eine vergleichsweise hohe Latenz erzeugen würde. Eine weitere Möglichkeit ist die Überlagerung einer Android-eigenen Kameraoberfläche, welche selbstständig das Kamerabild anzeigt, mit einer teil-transparenten OpenGL Oberfläche. Hierbei liegt die OpenGL Oberfläche über der Kameraoberfläche und blendet nicht gezeichnete Bereiche transparent aus. So arbeiten beide Oberflächen unanhängig und können ihre jeweils maximalen Frameraten erreichen. Das Überblenden der beiden Oberflächen übernimmt die Android API. Da eine einfache Erweiterbarkeit zur Zielsetzung des Frameworks gehört, wurde statt der Android-eigenen Kameraoberfläche eine ähnlich arbeitende OpenCV Kamera Oberfläche verwendet. Dadurch ist es dem Anwendungsentwickler möglich mittels OpenCV direkt auf der Kamera zu arbeiten und beispielsweise eine Merkmalserkennung zu integrieren. Durch die Entkopplung von Kameradarstellung und OpenGL Rendering kann es dazu kommen, dass die Framerate von OpenGL unter die der Kamera fällt und somit eine Inkonsistenz zwischen Kamerabild und dargestellter 3D-Welt entsteht. Dieser Umstand konnte durch Optimierungen weitestgehend eliminiert werden, so dass diese Art der Darstellung benutzerfreundlicher als eine korrekte Darstellung mit stockenden Kamerabildern ist.

5.2 Einlesen von OSM-Dateien

Das Einlesen von OSM-Dateien im PBF-Format erleichtert die Bibiliothek OSM-Binary [24], welche bereits einen vordefinierten Parser für die verwendeten Protobuf Dateien liefert. Um das Framework möglichst erweiterbar zu gestalten wurde von dieser Bibliothek abstrahiert und die Möglichkeit eingeführt mehrere Parser zu registrieren. Dadurch kann für jede darzustellende Information ein eigenen Parser eingesetzt werden. In OS-

MAR wird ein sogenannter 'WayParser' registriert, welcher alle verfügbaren Straßen verarbeitet. In der Beispielsanwendung zur Darstellung von Geschwindigkeitsbegrenzungen wird ein weiterer Parser eingehängt, welcher ausschließlich zur Verarbeitung von Straßen-Metadaten verwendet wird und daraus die notwendigen Informationen über Geschwindigkeitsbegrenzungen extrahiert.

5.2.1 Einlesen von Straßen

Durch den 'WayParser' werden alle verfügbaren OSM Datenstrukturen der Form 'Way' eingelesen und in der Datenbank gespeichert. Durch die Struktur der OSM Daten, welche eine Straße als Liste von geradlinigen Segmenten aus jeweils 2 Geopositionen (siehe Abbildung 5.2, Punkt 3) definiert, ist es möglich diese Segmente in die Datenbank zu übernehmen und später als gerade Strecke zu zeichnen.

Abbildung 5.2: OSM Kartenausschnitt in JOSM: 1) geteilte Kreuzung, 2) Unterbrechung des Straßenverlaufs, 3) Straßenaufbau in Segmente

5.2.2 Einlesen von Geschwindigkeitsbegrenzungen

Die durch die Beispielanwendung verwendeten Geschwindigkeitsbegrenzungen werden über einen speziellen Parser eingelesen. Hierbei werden alle Datenstrukturen der Form

'Way' ausgewertet und die zulässige Geschwindigkeit über die entsprechenden Metadaten ermittelt. Hierauf wird der Anfangs- und Endpunkt (siehe Abbildung 5.2, Punkt 1 und 2) des jeweiligen Straßenabschnitts mit dem Straßennamen und erlaubter Geschwindigkeit gespeichert. Aufgrund der Organisation der OSM Daten ist es möglich, dass die selbe Straße mit der selben Geschwindigkeitsbegrenzung in mehrere Teilabschnitte aufgeteilt wurde (siehe Abbildung 5.2, Punkt 2, Straßenverlauf unterbrochen). Dies ist notwendig, wenn die einzelnen Abschnitte beispielsweise verschiedene Metadaten, wie etwa Anzahl der Fahrspuren, enthalten müssen. Um zu verhindern, dass zwischen solchen Teilabschnitten eine neue Geschwindigkeitsbegrenzung angezeigt wird, werden Wege anhand ihrer gemeinsamen Anfangs- bzw. Endknoten auf Übereinstimmung des Straßennamens und erlaubter Geschwindigkeit überprüft. Gleiches gilt für eine Kreuzung, an welcher beispielsweise eine Straße der Geschwindigkeitsbegrenzung 30 auf eine Hauptstraße trifft (siehe Abbildung 5.2, Punkt 1). Durch die Struktur der OSM Daten treffen dort 3 Straßen aufeinander, da die Hauptstraße an der Kreuzung geteilt wird. Diesen Fall berücksichtigt der Parser ebenfalls und erkennt den Fortlauf der Hauptstraße.

Problematik: Realitätsgetreue Geschwindigkeitsbegrenzungen

Die realitätsgetreue Abbildung von Geschwindigkeitsbegrenzungen anhand von OSM Daten birgt einige Probleme: Leider fehlt auf einigen Straßen die Angabe der zulässigen Geschwindigkeit, dies kann auf eine fehlende Information im OSM Datenbestand oder auf eine implizierte Geschwindigkeitsbegrenzung zurückgeführt werden.

Um eine implizierte Geschwindigkeitsbegrenzung zu ermitteln müssten Straßen anhand ihres Straßentyps und Position (beispielsweise in einem Stadtgebiet in Deutschland) ausgewertet werden. Durch die Anzeige einer implizierten Geschwindigkeitsbegrenzung wären fehlende Daten zur Geschwindigkeitsbegrenzung schwieriger zu erkennen.

Eine weitere Problematik sind Kreuzungen mit mehr als 3 anliegenden Straßensegmenten: Um eine korrekte Darstellung der Geschwindigkeitsbegrenzungen zu erreichen müsste hier das Vorliegen einer Hauptstraße erkannt werden. Aufgrund dieser Problematik und dem Fokus, Mapper bei ihrer Arbeit zu unterstützen, wurde von einer realitätsgetreuen Abbildung abgewichen und eine einfachere Systematik eingeführt. Hierbei werden Straßen mit fehlender Information zur Geschwindigkeitsbegrenzung mit einem Fragezeichen markiert. Hierdurch kann ein Fehlen von Information leicht erkannt werden und eine Prüfung auf Korrektheit obliegt dem Betrachter. Ebenso werden beginnende Geschwindigkeitsbegrenzungen stets am rechten Fahrbahnrand platziert und eine Auflösung dieser auf die Rückseite gezeichnet. Hierdurch werden Kollisionen mit anderen Schildern verhindert und eine eindeutige Zuordnung zur betreffenden Straße verbessert.

5.3 Positionsbestimmung mittels GPS

Android bietet neben einer Abfrage der aktuellen GPS Position auch die Möglichkeit über eine Google Service API eine energieeffizientere Bestimmung der Position. Unter dem Menüpunkt 'GPS Settings' kann zwischen diesen Varianten gewählt werden. Zusätzlich

ist es möglich mittels der Wegpunkte einer GPX Datei das GPS Signal zu simulieren. Diese Funktionalität bietet die Möglichkeit vor dem Einsatz der Anwendung das verwendete Kartenmaterial auf Vollständigkeit und Performance zu überprüfen. Durch einen modularen Aufbau ermöglicht es OSMAR beliebige sogenannte 'LocationProvider' im System zu registrieren. Hier sind beispielsweise Unterstützungen für externe GPS Empfänger vorstellbar.

GPS Optimierung

Da das GPS Signal häufig ungenau ist und Schwankungen unterworfen ist, wurde das Konzept einer 'GPSStrategy' in OSMAR eingeführt. Hierbei handelt es sich einen programmierbaren Filter, nach dem eigentlichen 'LocationProvider', und ermöglicht die Nachbearbeitung oder Filterung von Positionsdaten. OSMAR liefert einen einfachen Filter für Fahrbahn zentrierte Positionen. Hierbei werden die Positionsangaben des 'LocationProviders' mit den eingelesenen Straßendaten abgeglichen und mittels Lotfußpunkt die Position auf der naheliegensten Straße gesucht.

5.4 Datenhaltung

Früh in der Entwicklung des Frameworks wurde die Notwendigkeit einer effizienten Datenhaltung in Form von Leistungsproblemen offensichtlich. Hierbei mussten Wegstrecken ebenso wie weitere Objekte mit Geopositionen gespeichert und schnell abrufbar gemacht werden. Die Hauptanforderung zur Datenhaltung lieferte der Renderingprozess, da dieser besonders auf die Geschwindigkeit des Abrufens von Daten anhand von Geopositionen angewiesen war. Zur Lösung dieses Problems fiel das Augenmerk auf Geodatenbanken, welche zur Speicherung von Objekten im geometrischen Raum optimiert sind.

Eine Lösung zur Umkreissuche liefert die Bibliothek 'jeospatial' [12] welche basierend auf einem 'Vantage Point Baum' [27] eine Umkreissuche auf Geopositionen ermöglicht. Hierbei können Suchanfragen gegen die Datenbank in $O(log(n))$ ausgeführt werden.

Um eine effiziente Umkreissuche mit Hilfe einer Datenbank basierend auf einem VP-Baum zu implementieren müssen alle darin abgelegten Objekte auf Punkte im Geoformat reduziert werden. Hierbei werden Straßensegmente abhängig von ihrer Länge erneut geteilt und diese Punkte zusammen mit den Eckpunkten mit Referenzen zu den zu zeichnenden Objekten in der Datenbank abgelegt. Bei einer Suche werden die gefunden Punkte wieder zu Objekten aufgelöst. Dieser Ansatz ermöglicht beliebige Objekte, mit mindestens einer Geoposition, zu speichern und mittels einer Umkreissuche aus der Datenbank abzurufen.

5.5 Rendering mit OpenGL

Für die Darstellung der 3D-Welt wird die Schnittstelle OpenGL verwendet. Hierfür wird eine oberhalb des Kamerabildes angeordnete transparente OpenGL Oberfläche initialisiert und mit einem entsprechenden 'Renderer' versehen, welcher von der OpenGL

Schnittstelle zum Zeichnen jedes Frames verwendet wird. Um Objekte in OpenGL zu zeichnen, müssen diese zunächst in Punkten (Vertex) zu Dreiecken umstrukturiert werden und diese in den OpenGL Kontext übertragen werden. Die Übertragung von Daten aus der Androidumgebung in den OpenGL Kontext ist hierbei teuer, daher besteht die Möglichkeit die Daten in Puffer (Vertex Buffer Objects) zu schreiben um diese für mehrere Zeichendurchgänge zu verwenden. Hierauf können die übertragenen Daten als einzelne Punkte in eine Variable des Shaderprogramms geladen und gezeichnet werden.

5.5.1 Kameranachbildung

Um die reale Kamera in OpenGL nachzubilden werden alle Punkte (Vertices) mittels Matrixmultiplikation perspektivisch manipuliert. Hierzu wird die bereits ermittelte Perspektivmatrix mit der fortlaufend berechneten Rotationsmatrix multipliziert. Die daraus entstandene Matrix wird während des Renderings mit allen zu zeichnenden Punkten multipliziert um diese perspektivisch korrekt abzubilden.

Pro Frame Zu Beginn jedes neuen Frames wird die asynchron gespeicherte Rotationsmatrix, welche die spätere Lage der 'Welt' vor gibt, ausgelesen und mit der bereits vorliegenden Projektionsmatrix (siehe Abschnitt 4.3.3) multipliziert. Die daraus entstandene Matrix wird nun um 1,5 Einheiten (Meter) auf der negativen Z-Achse verschoben. Dadurch wird die Kamera auf eine Höhe von 1,5 Metern in der 3D-Welt platziert. Da eine Bestimmung der Höhe des Smartphones nicht möglich ist, wird dieser Wert exemplarisch angenommen. Nun wird die entstandene Matrix in den OpenGL Kontext übertragen und in einer Variable des Vertex Shader gespeichert. Das Shaderprogramm sorgt später dafür, dass jeder zu zeichnende Punkt mit dieser Matrix multipliziert wird und damit perspektivisch korrekt in der virtuellen Welt angeordnet wird. Hierauf beginnt der Zeichendurchgang aller zu zeichnenden Objekte.

5.5.2 Veränderung der Kameraposition

Zu Beginn des Renderings definiert die Position der virtuellen Kamera, und damit die ermittelte GPS Position, den Ursprung des Koordinatensystems (0,0,0). Alle zu zeichnenden Objekte errechnen ihre relative Position basierend auf dieser Ursprungkoordinate. Da die Berechnung und der Aufbau der Puffer der zu zeichnenden 'Vertices' teuer ist, sollten die vorhandenen Puffer möglichst häufig wiederverwendet werden. Bei einer Positionsänderung durch das GPS würde sich, bei einem naiven Ansatz, die Position der Kamera und damit der Ursprung des Koordinatensystems ändern, wodurch eine Neuberechnung aller zu zeichnenden Objekte in Relation zum neuen Ursprung stattfinden müsste. Ein besserer Ansatz ist die Ermittlung der Distanz der neuen Kameraposition zum vorherigen Ursprung und die Verschiebung der virtuellen Kamera ohne Veränderung des Koordinatensystems. Hierdurch ist es möglich alle bestehenden Puffer beizubehalten.

5.5.3 Buffer und Lifecyle Management

Um den latenz-empfindlichen Zeichenprozess nicht zu unterbrechen wurde die Aktualisierung der Menge der zu zeichnenden Objekte in einen seperaten Thread ausgelagert, welcher intervall-basiert die in der Datenbank mittels Umkreissuche ermittelten Objekte mit der aktuellen Zeichenwarteschlage abgleicht. Hierbei wurde darauf geachtet, dass die durch ein Update dieser Menge entstehende Wartezeit für den Zeichenprozess möglichst gering ausfällt. Sollte die definierte Schwelle zur Verschiebung der Kamera erreicht und damit eine Neupositionierung des Koordinatensystems notwendig geworden sein, übernimmt dieser Thread die Systematik des Auf- und Abbaus der entsprechenden Puffer. Da OpenGL Operationen ausschließlich im Zeichen-Thread vorgenommen werden können, werden bei jedem Zeichendurchgang neu hinzugekommene Objekte erkannt und deren Puffer aufgebaut, bzw. die Puffer der zu entfernenden Objekte geleert. Ebenso werden zu zeichnende Objekte über eine Änderung der Zeichenoberfläche, beispielsweise bei einer Pausierung der Anwendung durch Android, informiert, um auf ungültig gewordene Puffer reagieren und diese neu aufbauen zu können.

5.5.4 Darstellung von Straßen

Straßen werden als eine Menge von geradlinigen Strecken aus den OSM Daten importiert und auch als diese gezeichnet. Hierbei werden alle Straßenabschnitte mit einer Fahrbahnbreite von 6,5Metern gezeichnet. Eine Unterscheidung der Fahrspuranzahl sowie der Fahrspurbreite wurde in dieser Arbeit nicht berücksichtigt (siehe Ausblick Abschnitt 6). Durch den Parser werden die eingelesenen Straßenabschnitte mit Anfangs- und Endkoordinate als Straßensegmentobjekte der Datenbank zugeführt. Die einzelnen Straßensegmente werden dann mit grauer Fahrbahn und gelber Mittellinie zwischen den beiden Koordinaten gezeichnet. Hierbei wurde durch die Farbgebung ein Kontrast zur Umgebung und damit eine gute Erkennbarkeit erreicht. Abbildung 5.3 zeigt hierbei die Einblendung auf einer mehrspurigen Straße.

5.5.5 Darstellung von Geschwindigkeitsbegrenzungen

Durch den entsprechenden Parser wurden bereits Objekte zur Geschwindigkeitsbegrenzung der einzelnen Straßenabschnitte der Datenbank zugeführt. Diese Objekte enthalten den Beginn der jeweiligen Geschwindigkeitsbegrenzung und Informationen über die Ausrichtung und Geschwindigkeit. Die Position des zu zeichnenden Schildes wird auf dieser Grundlage an den Beginn des Straßensegments an die rechte Fahrbahnseite gesetzt. Da an Kreuzungen die Startposition eines Straßensegments auf der Mitte der Kreuzung liegt wird die Position dem Straßenverlauf nach um eine Fahrspurbreite verschoben, so dass Kollisionen mehrerer Schilder vermieden werden. Das Schild wird hierauf als Rechteckige Fläche in Form von zwei Dreiecken OpenGL übergeben. Dank eines angepassten Textur-Fragment-Shaders ist es nun möglich anhand der gespeicherten Geschwindigkeit eine entsprechende Textur für Vorder- und Rückseite des Schildes aus einem vorher eingelesenen Bestand auszuwählen und auf dem entstandenen Rechteck zu zeichnen. Hierbei

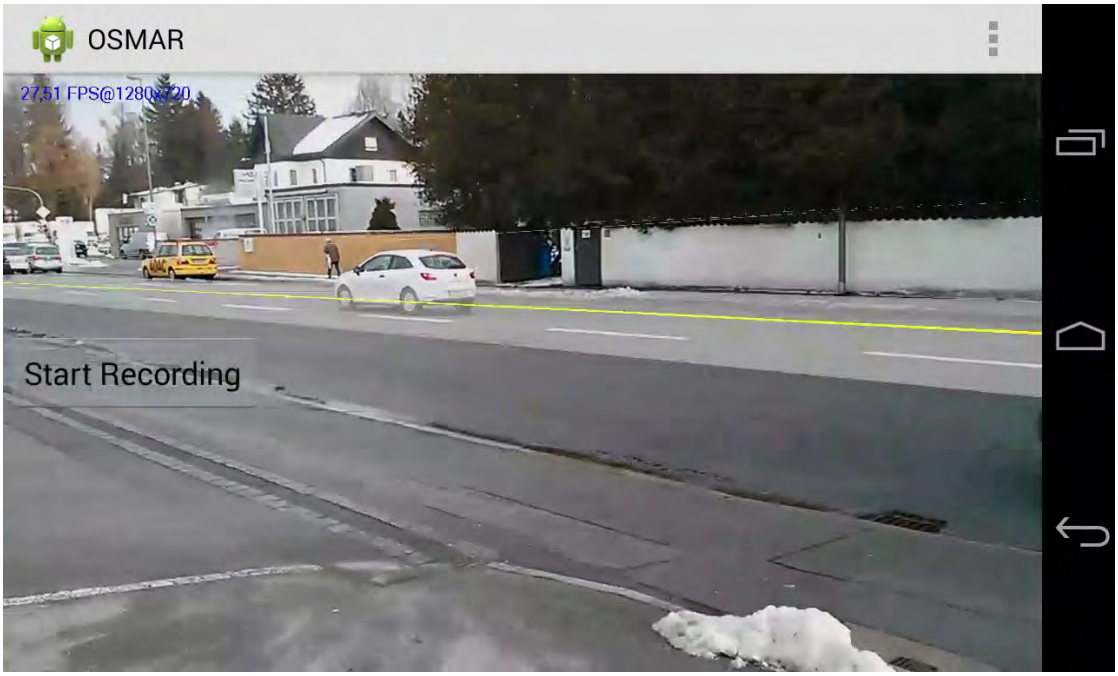

Abbildung 5.3: OSMAR Screenshot Einblendung auf mehrspuriger Straße

werden gewöhnliche deutsche Begrenzungsschilder auf die Vorderseite, die dem Straßenabschnitt zugewandte Seite, und Aufhebungsschilder auf die Rückseite gezeichnet. Abbildung 5.4 zeigt die Anzeige einer solchen Geschwindigkeitsbegrenzung in OSMAR.

5.6 Aufnahme von Wegpunkten

Neben der korrekten Darstellung ist die Aufnahme von Wegpunkten zur späteren Verarbeitung durch den Mapper für die Arbeit von Bedeutung. Um die Speicherung der aktuellen Position mit notwendigen Informationen für eine spätere Bearbeitung zu vereinfachen wurde eine Aufnahmefunktionalität implementiert. In einer sogenannten Session (Sitzung) kann der Nutzer jederzeit über einen zentral gelegenen Button auf der Bedienoberfläche seine aktuelle Position speichern. Hierauf startet eine Aufnahme des eingebauten Mikrofons um weitere Informationen auch während einer Autofahrt erfassen zu können. Ein weiterer Druck auf den Button beendet die Aufnahme. Die Sitzung kann über den entsprechenden Punkt in der Aktionsleiste gespeichert werden.

Beim Start der Aufnahme wird die aktuelle Position zwischengespeichert und die Audioaufnahme in einem temporären Verzeichnis gespeichert. Um eine weitere Verarbeitung mit dem Programm JOSM zu vereinfachen, wurde das WAVE Format für Audioaufnahmen gewählt. Da Android über keine native Unterstützung des WAVE Formats verfügt, wird der interne MediaRecorder mit PCM Output verwendet. Bei Beendigung der Aufnahme wird nun die Audioaufnahme manuell mit WAVE Headern finalisiert und

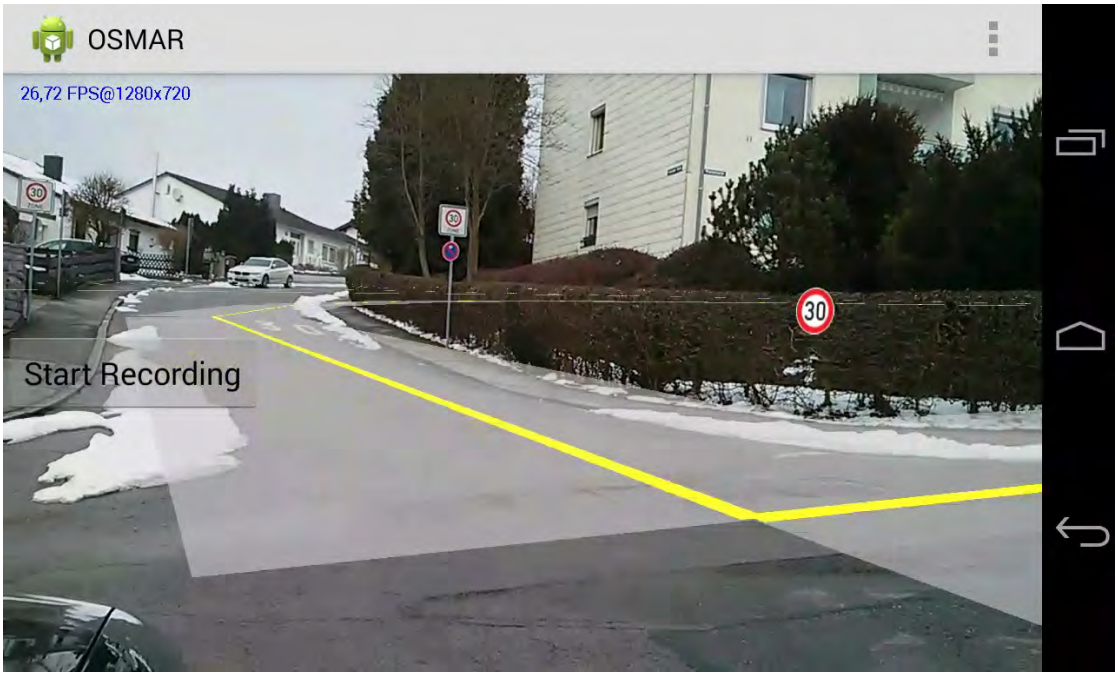

Abbildung 5.4: OSMAR Screenshot mit eingeblendetem Schild vor Geschwindigkeitsbegrenzung

in Verbindung mit der gespeicherten Position in der Sitzung abgelegt. Der Nutzer kann jederzeit die vorhandene Sitzung über den Punkt 'Save Session' in der Aktionsleiste speichern. Hierbei wird eine GPX Datei generiert, welche Wegpunkte zu allen gespeicherten Aufnahmepositionen mit Verknüpfung zu den entsprechenden WAVE Dateien enthält. Diese wird zusammen mit den entsprechenden WAVE Dateien in einer ZIP Datei im Speicher des Smartphones, zugänglich für den Benutzer, abgelegt.

Kompatibilität mit JOSM

JOSM ist ein Programm zur Verarbeitung von OpenStreetMap Daten mit einer grafischen Oberfläche und stellt das Kartenmaterial in 2D für den Benutzer dar. Ebenso beherrscht es das von OSMAR exportierte GPX Format und kann mit Wegpunkten verbundene WAVE Dateien direkt in der Oberfläche abspielen (siehe Abbildung 5.5). Durch dieses Zusammenspiel ist es möglich komfortabel an die gespeicherten Informationen der Wegpunkte zu gelangen und Änderungen am Kartenmaterial vorzunehmen.

5.7 Erweiterbarkeit

Ziel dieser Arbeit ist neben der Umsetzung einer Beispielanwendung auch die einfache Erweiterbarkeit um weitere Funktionalität. Trotz des Umfangs der Implementierung von

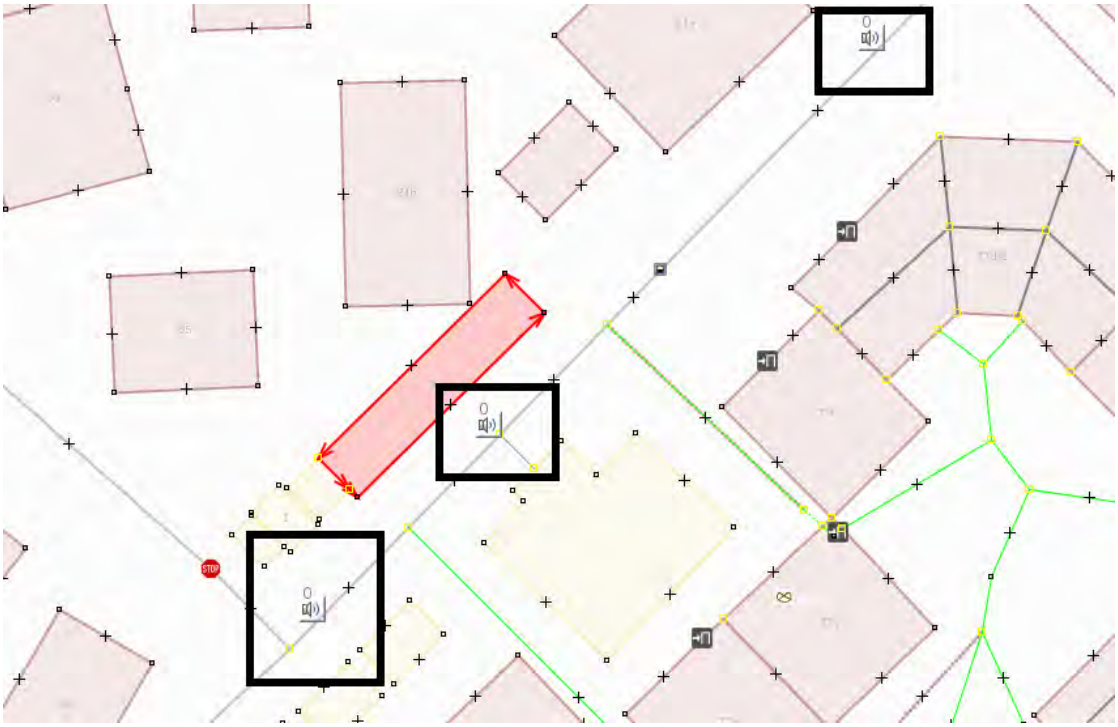

Abbildung 5.5: Wegpunkte mit Audionotiz in JOSM

rund 6000 Zeilen Quellcode können die wichtigen Teilbereiche sehr einfach erweitert werden.

5.7.1 Parser

Um einen Parser zu erstellen und damit weitere Informationen aus den eingelesenen OSM Daten zu extrahieren muss lediglich das Parser Interface (siehe Quellcode 1) implementiert werden. Hierbei werden innerhalb der Methoden der jeweilige OSM Datentyp abgearbeitet, innerhalb 'Ways' beispielsweise Straßen im 'Way' Format.

5.7.2 Zeichenbares Objekt

Nachdem der Parser notwendige Informationen eingelesen hat, müssen Objekte der Datenbank hinzugefügt werden, welche später in der 3D-Welt gezeichnet werden können. Um sowohl die Datenhaltung als auch den Zeichenprozess zu ermöglichen ist die Implementierung nur eines Interfaces notwendig (siehe Quellcode 2). Um die Speicherung des Objekts in die Datenbank zu ermöglichen muss das Objekt mittels 'toPoints' in einzelne Punkte zerlegbar sein, dadurch ist eine spätere Umkreissuche möglich und der Zeichenprozess kann Objekte außerhalb des sichtbaren Bereichs ignorieren. Damit das Objekt gezeichnet werden kann muss es, wie in Abschnitt 5.5.2 beschrieben, auf eine Verschiebung des Koordinatenursprungs in der Methode 'onOriginChanged' reagieren.

```java
public interface Parser {

    void parseRelations(List<Osmformat.Relation> relations, OSMParser parser);

    void parseDense(Osmformat.DenseNodes denseNodes, OSMParser parser);

    void parseNodes(List<Osmformat.Node> nodes, OSMParser parser);

    void parseWays(List<Osmformat.Way> ways, OSMParser parser);

    void parse(Osmformat.HeaderBlock headerBlock, OSMParser parser);

    void complete(OSMParser parser);
}
```

Quellcode 1: Parser Interface

Ebenso muss es Aufräumarbeiten verrichten, sollte das Objekt aus den zu zeichnenden Objekten entfernt werden (Methode 'onRemove') oder die Zeichenoberfläche und damit alle Puffer neu aufgebaut worden sein (Methode 'onDrawContextInvalidated'). Der eigentliche Zeichenprozess findet in der Methode 'onDrawFrame' statt. Hierbei ist es dem Objekt möglich vollständig auf alle OpenGL Methoden zurückzugreifen und besitzt damit größtmögliche Freiheit.

```java
public interface DrawableWorldObject extends SegmentableWorldObject {

    void onDrawFrame(DrawContext drawContext);

    void onOriginChanged(GeoLocation origin);

    void onDrawContextInvalidated();

    void onRemove();

    // Extends SegmentableWorldObject
    Collection<WorldPoint> toPoints(double radius);
}
```

Quellcode 2: DrawableWorldObject Interface

6 Fazit

Trotz aller beschriebenen Einschränkungen konnte ein solides Framework und eine funktionierende Beispielanwendung in Form der Darstellung von Geschwindigkeitsbeschränkungen umgesetzt werden. Mappern kann die Darstellung der 3D-Welt als Hilfe dienen und mittels dem verbauten 'GPX Location Providers' ist eine simulierte Fahrt durch das Kartenmaterial möglich. Wenn auch ein Einsatz durch die Positionsungenauigkeit erschwert wird, so kann OSMAR, durch den Einsatz der implementierten GPS Optimierung, doch eine Hilfe bei der Erkennung von fehlenden Informationen im Kartenmaterial bieten. Durch die Entwicklung konnten tiefe Einblicke in die Sensorauswertung, Darstellung und Umsetzung einer virtuellen Welt erlangt werden. Durch die Umsetzung als Framework und dem modularem Aufbau bietet OSMAR Entwicklern die Möglichkeit direkt mit 3D-Darstellungen oder Sensoroptimierungen zu experimentieren ohne selbst einen entsprechenden Unterbau umsetzen zu müssen. Durch die Vielzahl an Anwendungsmöglichkeiten besteht ebenso für Endbenutzer ein großes Potential.

Ausblick

Mit OSMAR wurde ein Grundstein gelegt, auf dem viele Erweiterungen aufbauen und Anwendungen umgesetzt werden können. Grundsätzlich sind alle Informationen aus dem Kartenmaterial von OSM denkbar, aber auch andere Datenquellen können umgesetzt werden. So können neben Ampeln beispielsweise auch Briefkästen grafisch dargestellt werden. Hier treten allerdings wie bereits beschrieben Probleme mit der Genauigkeit des Kartenmaterials und der Sensoren des Smartphones auf.

OSMAR ist als Framework zu verstehen und bietet nicht genügend Bedienkomfort um es Endverbrauchern ohne OSM Kenntnisse zur Verfügung zu stellen. Dies beginnt bei der manuellen Selektion und Verarbeitung des Kartenmaterials, welches später auf das Smartphone übertragen werden muss, um von OSMAR geladen zu werden. Dennoch ist eine Implementierung zum selbstständigen Nachladen von Kartendaten möglich. Denkbar ist auch eine lokal begrenzte Anwendung, da durch frühe Selektion der Daten ein Stadtgebiet vollständig auf dem Smartphone vorinstalliert werden kann. Das größte Potential zur Verbesserung birgt der Bereich der Positionsbestimmung: Die aktuell implementierte Positionsbestimmung durch GPS ist häufig zu ungenau. Hier ist der Einsatz komplexer Algorithmen vorstellbar, welche die Filterung der Rohdaten aus den Positionssystemen übernehmen. Ebenso sind auch vollständig neue Arten der Positionsbestimmung wie etwa durch Signalstärken von Bluetooth Sendern oder WLan-Stationen um Indoor Navigation zu ermöglichen denkbar.

Neben der gewählten geobasierten Positionsbestimmung des AR Systems ist eine Un-

terstützung durch eine merkmalsbasierte Positionsbestimmung denkbar: Hierbei könnte die Kontur einer Straße im Sichtfeld der Kamera durch Bildverarbeitung erkannt werden und mit den vorhandenen OSM Daten abgeglichen werden, wodurch die vorher bestimmte Positions verfeinert oder korrigiert werden könnte. Um diesen Einsatzfall zu erleichtern wurde OSMAR bereits mit einem OpenCV Subsystem zur Bildverarbeitung ausgestattet.

Aufgrund des Umfangs einer Bachelorarbeit konnten nicht alle Funktionalitäten für ein endgültiges Produkt umgesetzt werden. Der folgende Teil beschreibt einige dieser Einschränkungen und mögliche Lösungen sowie aufbauende Arbeiten.

Kamerakalibrierung

Um eine Portabilität der Anwendung zwischen verschiedenen Geräten zu gewährleisten ist es nötig die intrinsischen Werte aus einer Kamerakalibrierung dem OpenGL Rendering zur Verfügung zu stellen. Hierbei wäre entweder eine Datenbank aus vorkalibrierten Kameras der entsprechenden Geräte nötig oder die Integration einer Kamerakalibrierungsanwendung in die entstandene Anwendung, mithilfe dessen der Benutzer das Rendering an die Eigenschaften seiner Kamera angleichen kann um eine realistische Konstruktion der 3D-Welt zu gewährleisten. Diese Integration konnte im Umfang dieser Arbeit nicht umgesetzt werden und somit sind die intrinsischen Werte der Kamera des Testsmartphones (Google Nexus 4) fest verbaut. Eine zur Laufzeit veränderbare Kameramatrix ist jedoch einfach umzusetzen.

GPS Optimierung

Wie bereits erwähnt, wurde eine einfache GPS Optimierung entwickelt, welche aus den gegebenen GPS Positionen mittels einfacher Berechnungen die naheliegenste Straße ermittelt und die Position entsprechend ändert. Hierbei wurde auf die Erweiterbarkeit durch weitere GPS Optimierungen geachtet, so dass weitere GPS Filter einfach entwickelt werden können. Hier könnte beispielsweise eine Ermittlung des Bewegungsvektors vorgenommen werden, um ein auf der rechten Straßenseite fahrendes Fahrzeug zu simulieren. Ebenso sind diverse Filter möglich um beispielsweise auf dem Profil eines Fußgängers GPS Abweichungen herauszufiltern.

Kartenmaterial

Das Framework und die Beispielsanwendung benötigen einen Ausschnitt des OSM Kartenmaterials in einem speziellen Format, welches nicht automatisch heruntergeladen wird. Hier ist eine Integration mit einem Server Backend vorstellbar, wodurch kein manueller Transfer zum Smartphone nötig werden würde, sondern der Nutzer seinen Kartenbereich am Smartphone auswählt und dieses auf sein Smartphone heruntergeladen wird.

Sensorauswertung

Die Verwendung des Rotationsvektors, welche die Android API zur Verfügung stellt, liefert in Verbindung mit den vergleichsweise ungenauen absoluten Positionsdaten des GPS und der unklaren Höhe des Smartphones akzeptable Ergebnisse. Allerdings reicht die Sensorgenauigkeit nicht um eine vollständig realitätsgetreue Augmented Reality zu schaffen. Jedoch sind diverse Optimierungen vorstellbar um die Lage des Smartphones noch genauer zu ermitteln. Eine Möglichkeit ist die Verwendung eines Kalman Filters in Verbindung mit den Rohdaten der einzelnen Sensoren [22].

Zeichnung von Fahrbahn

Durch die Aufteilung von Straßen in gerade Streckenabschnitte durch OSM tritt bei der Zeichnung der Streckenabschnitte ein Versatz auf (siehe Abbildung 5.4). Zur korrekten Darstellung der Fahrbahn könnte ein Algorithmus zum Verbinden der einzelnen Streckenabschnitte ohne Versatz eingeführt werden. Ebenso berücksichtigt OSMAR bei der Zeichnung der Fahrbahn weder Anzahl, noch Breite der verfügbaren Fahrspuren. Aus den OSM Daten ist eine Ermittlung der Anzahl der Fahrspuren möglich, jedoch nicht der Fahrspurbreite. Da kein direkter Fahrspurverlauf mehrerer Fahrspuren über OSM ermittelt werden kann, würde eine Zeichnung der verschiedenen Fahrspuren auch eine Berechnung des Zusammen- beziehungsweise Auseinanderführen der Fahrspuren nach sich ziehen.

Aufbauende Arbeiten

Neben den oben genannten Verbesserungen bietet diese Arbeit die Möglichkeit für aufbauende Arbeiten. Denkbar sind alle Augmented Reality Anwendungen welche auf geographischen Daten basieren. So könnte beispielsweise der OSM Unterbau entfernt werden und eine ortspezifische Datenbank mit Bildern oder 3D-Objekten für eine multimediale Darstellung eines Gebäudes implementiert werden. Denkbar ist auch die Entwicklung eines Spiels oder die Bereicherung einer Stadtführung um Bilder der Umgebung aus verschiedenen Zeitaltern.

Glossar

Clipping Der Prozess des Beschneidens von grafischen Objekten am Rand des sichtbaren Bereichs. 21, 25

Fragmentshader Art eines Shaderprogramms, arbeitet auf Flächen von Objekten. 13, 15

Framerate Bildwechselfrequenz, Anzahl der Einzelbilder pro Zeiteinheit. 31, 32

Frustum Geometrische Form, Pyramidenstumpf. 19, 21, 22

GPS Global Positioning System, System zur Bestimmung der geographischen Position. 11, 12, 30, 31, 34–36, 42, 43

GPX GPS Exchange Format, Format zur Speicherung von geographischen Positionen und Wegen. 16, 35, 39

JOSM Ein erweiterbarer freier Editor zur Bearbeitung des Datenbestands des OpenStreetMap Projekts. 8

Mapper Bezeichnung von Mitwirkenden des OpenStreetMap Projekts. 8, 9, 34, 38

OpenCV Open Source Computer Vision - Bibliothek zur Bildverarbeitung und maschinelles Sehen, unter anderen 3D Rekonstruktion. 32, 43

OpenGL Open Graphics Library, Schnittstelle zur Entwicklung von 2D- und 3D Anwendungen. 13, 14, 19, 21, 22, 24, 45

OpenGL ES Open Graphics Library for Embedded Systems, für eingebettete Systeme optimierte Version von OpenGL. 13

OSM OpenStreetMap Projekt, quelloffene Weltkarte. 39, 40, 42–44

Shadereinheit Recheneinheit in aktuellen Grafikchips, dient der hardware-unterstützen Verarbeitung von Grafikdaten. 25

Shaderprogramm Auf einer Shadereinheit der Grafikeinheit ausgeführtes Programm. 24, 36

Vertexshader Art eines Shaderprogramms, arbeitet auf den Eckpunkten von Objekten. 13–15

Literatur

[1] Defense Mapping Agency. *The Universal Grids - Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)*. USA: DMA Technical Manual, DMATM 8358.2, 1989. URL: http://earth-info.nga.mil/GandG/publications/tm8358.2/TM8358_2.pdf.

[2] Song Ho Ahn. *OpenGL Projection Matrix*. Website. 2013. URL: http://www.songho.ca/opengl/gl_projectionmatrix.html (besucht am 20. 01. 2015).

[3] Song Ho Ahn. *OpenGL Transformation*. Website. 2013. URL: http://www.songho.ca/opengl/gl_transform.html (besucht am 20. 01. 2015).

[4] *Android Developers - OpenGL ES*. Website. 2015. URL: https://developer.android.com/guide/topics/graphics/opengl.html (besucht am 11. 01. 2015).

[5] *Android Developers - Sensors Overview*. Website. 2015. URL: https://developer.android.com/guide/topics/sensors/sensors_overview.html (besucht am 10. 01. 2015).

[6] Ralf Dörner u. a. „Virtual Reality und Augmented Reality (VR/AR)“. German. In: *Informatik-Spektrum* (2014), S. 1–8. ISSN: 0170-6012. DOI: 10.1007/s00287-014-0838-9. URL: http://dx.doi.org/10.1007/s00287-014-0838-9.

[7] Koshy George. *Koshy George - Calculating OpenGL perspective matrix from OpenCV intrinsic matrix*. Website. 2014. URL: https://kgeorge.github.io/2014/03/08/calculating-opengl-perspective-matrix-from-opencv-intrinsic-matrix/ (besucht am 20. 01. 2015).

[8] *Google Playstore - Sensorfusion (Alexander Pacha)*. Website. 2015. URL: https://play.google.com/store/apps/details?id=org.hitlabnz.sensor_fusion_demo (besucht am 05. 01. 2015).

[9] *Google Protobuf - Protocol Buffers - Overview*. Website. 2015. URL: https://developers.google.com/protocol-buffers/docs/overview (besucht am 05. 01. 2015).

[10] Khronos Group. *OpenGL Docs - glDepthFunc*. Website. 2006. URL: https://www.opengl.org/sdk/docs/man/html/glDepthFunc.xhtml (besucht am 20. 01. 2015).

[11] R. I. Hartley und A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[12] *jchambers/jeospatial - Github*. Website. 2015. URL: https://github.com/jchambers/jeospatial (besucht am 05. 01. 2015).

[13] *JOSM - Extensible Editor for OpenStreetMap*. Website. 2015. URL: https://josm.openstreetmap.de/ (besucht am 08. 01. 2015).

[14] Thomas Jüngling. *Welt - Ikea Augmented Reality*. Website. 2013. URL: http://www.welt.de/wirtschaft/webwelt/article119525750/Ikea-App-projiziert-Moebel-in-die-eigene-Wohnung.html (besucht am 01. 02. 2015).

[15] Paul Milgram u. a. „Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum". In: 1994, S. 282–292. URL: http://web.cs.wpi.edu/~gogo/hive/papers/Milgram_Takemura_SPIE_1994.pdf.

[16] *OpenCV - Camera Calibration and 3D Reconstruction*. Website. 2015. URL: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (besucht am 05. 01. 2015).

[17] *OpenCV Samples - Android camera calibration*. Website. 2015. URL: https://github.com/Itseez/opencv/tree/master/samples/android/camera-calibration (besucht am 05. 01. 2015).

[18] *OpenGL ES 2.X*. Website. 2015. URL: https://www.khronos.org/opengles/2_X/ (besucht am 05. 01. 2015).

[19] *OpenGL Programming Guide - Appendix F - Homogeneous Coordinates and Transformation Matrices (Redbook)*. Website. 2014. URL: http://glprogramming.com/red/appendixf.html (besucht am 22. 01. 2015).

[20] *OpenStreetMap Deutschland*. Website. 2015. URL: http://www.openstreetmap.de/faq.html (besucht am 27. 01. 2015).

[21] *OpenStreetMap Wiki - PBF Format*. Website. 2015. URL: http://wiki.openstreetmap.org/wiki/PBF_Format (besucht am 05. 01. 2015).

[22] Alexander Pacha. „Sensor fusion for robust outdoor Augmented Reality tracking on mobile devices". In: 2013. URL: http://my-it.at/media/MasterThesis-Pacha.pdf.

[23] F. Ramm und J. Topf. *OpenStreetMap: die freie Weltkarte nutzen und mitgestalten*. Lehmanns Media, 2010. ISBN: 9783865413758. URL: http://books.google.de/books?id=J0V6cgAACAAJ.

[24] *scrosby/OSM-binary - Github*. Website. 2015. URL: https://github.com/scrosby/OSM-binary (besucht am 08. 01. 2015).

[25] *The Pinhole Camera - Camera calibration*. Website. 2015. URL: https://www.ics.uci.edu/~majumder/vispercep/cameracalib.pdf (besucht am 20. 01. 2015).

[26] *Topografix GPX: GPS Exchange Format*. Website. 2015. URL: http://www.topografix.com/gpx.asp (besucht am 08. 01. 2015).

[27] Peter N. Yianilos. „Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces". In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1993.

Eidesstattliche Erklärung

Hiermit versichere ich, Benedikt Lang, an Eides statt, dass ich diese Bachelorarbeit selbstständig und ohne Benutzung anderer als der ausgewiesenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind.

Ich erkläre weiterhin, dass ich diese Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, den 16.03.2015

Benedikt Lang