

Institut für Formale Methoden der Informatik

Universität Stuttgart  
Universitätsstraße 38  
D–70569 Stuttgart

Bachelorarbeit

## **Generierung von virtuellen Welten aus OSM-Daten**

Julian Blumenröther

**Studiengang:** Informatik

**Prüfer/in:** Prof. Dr. Stefan Funke

**Betreuer/in:** Prof. Dr. Stefan Funke

**Beginn am:** 20. Mai 2019

**Beendet am:** 11. November 2019



## **Kurzfassung**

Thema dieser Bachelorarbeit ist die automatisierte Generierung von kleinen, mit dem Browser oder der VR-Brille explorierbaren, virtuellen Welten aus OpenStreetMap-Daten (.osm-Dateien). Diese virtuellen Welten bestehen aus den Gebäuden, Straßen, Bahnlinien, Bäumen und Naturgebieten des ausgewählten Gebietes, welche in drei Dimensionen auf die entsprechende Position der Weltkugel gemappt werden. Das Einlesen und Verarbeiten der OSM-Daten geschieht mittels eines in Java geschriebenen Parsers. Die Implementierung der virtuellen Welt beruht auf Javascript, HTML und A-Frame.

Nachdem verwandte Arbeiten untersucht wurden, wird auf einige Grundlagen, die zum Verständnis der Implementierung wichtig sind, eingegangen. Anschließend werden die Details der Vorgehensweise beim Parsen der OSM-Daten sowie die Erstellung der virtuellen Welt in A-Frame näher betrachtet. Auf eine Anleitung zur Ausführung des Projektes folgt die Evaluation. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick auf Erweiterungs- und Anwendungsmöglichkeiten ab.



# Inhaltsverzeichnis

<b>1 Einleitung und Motivation</b>	<b>13</b>
<b>2 Verwandte Arbeiten</b>	<b>15</b>
2.1 OSMBuildings . . . . .	15
2.2 VR Map . . . . .	16
<b>3 Grundlagen</b>	<b>17</b>
3.1 Geografische Grundlagen . . . . .	17
3.1.1 Längen- und Breitengrad . . . . .	17
3.1.2 Berechnung von kartesischen Koordinaten . . . . .	17
3.2 OpenStreetMap-Daten . . . . .	18
3.2.1 Nodes . . . . .	19
3.2.2 Ways . . . . .	20
3.2.3 Relations und Members . . . . .	21
3.3 Three.js . . . . .	22
3.3.1 Geometry . . . . .	22
3.3.2 Material und Mesh . . . . .	23
3.3.3 Performance Verbesserung . . . . .	24
3.4 A-Frame . . . . .	25
3.4.1 Entität-Komponenten-System . . . . .	25
3.5 JavaScript Object Notation . . . . .	26
<b>4 Parsen und Behandeln der OSM-Daten</b>	<b>27</b>
4.1 Parsen der OSM-Daten . . . . .	27
4.1.1 Behandeln der Nodes . . . . .	27
4.1.2 Behandeln der Ways . . . . .	27
4.1.3 Behandeln der Relations und Members . . . . .	28
4.2 Erstellen der Java-Objekte . . . . .	28
4.2.1 Umwandeln der Nodes in Bäume . . . . .	28
4.2.2 Umwandeln der Ways in Straßen . . . . .	28
4.2.3 Umwandeln der Ways in Naturgebiete . . . . .	30
4.2.4 Umwandeln der Ways in Bahnlinien . . . . .	30
4.2.5 Umwandeln der Ways in Gebäude . . . . .	30
4.2.6 Umwandeln der Relations in Gebäude . . . . .	31
4.3 Bearbeiten der Einstellungen . . . . .	31
4.4 Erstellen der JSON-Datei . . . . .	31
<b>5 Erstellen der virtuellen Welt in A-Frame</b>	<b>33</b>
5.1 Erstellen der Umgebung . . . . .	33

5.2	Erstellen der Karte . . . . .	34
5.2.1	Erstellen der Straßen . . . . .	35
5.2.2	Erstellen der Gebäude . . . . .	36
5.2.3	Erstellen der Bäume . . . . .	37
5.2.4	Erstellen der Naturgebiete . . . . .	38
5.2.5	Erstellen der Bahnlinien . . . . .	39
5.3	Benutzte Kameras . . . . .	39
5.3.1	Fly Controls . . . . .	40
5.3.2	Spherical Controls . . . . .	40
<b>6</b>	<b>Anleitung zur Ausführung des Projektes</b>	<b>41</b>
<b>7</b>	<b>Evaluation</b>	<b>43</b>
7.1	Benchmarking des OSMParsers . . . . .	44
7.2	Benchmarking der Objektgenerierung . . . . .	45
7.3	Benchmarking der Darstellung . . . . .	46
7.4	Beispiele . . . . .	47
7.4.1	Stuttgart Hauptbahnhof . . . . .	47
7.4.2	Tokio . . . . .	48
7.4.3	Manhattan - New York . . . . .	49
<b>8</b>	<b>Zusammenfassung und Ausblick</b>	<b>51</b>
8.1	Zusammenfassung . . . . .	51
8.2	Ausblick . . . . .	51
<b>Literaturverzeichnis</b>		<b>53</b>

# **Abbildungsverzeichnis**

2.1	OSMBuildings Website . . . . .	15
2.2	VR Map Website . . . . .	16
3.1	OpenStreetMap Website . . . . .	18
3.2	Entstandenes Shape . . . . .	24
4.1	Berechnung der Eckpunkte der Straßen . . . . .	29
5.1	Umgebung der virtuellen Welt . . . . .	33
5.2	Exemplarischer Kartenausschnitt . . . . .	34
5.3	Beispiel für erstellte Straßen in A-Frame . . . . .	35
5.4	Beispiel für erstellte Gebäude in A-Frame . . . . .	36
5.5	Beispiel für erstellte Bäume in A-Frame . . . . .	37
5.6	Beispiel für erstellte Naturgebiete in A-Frame . . . . .	38
5.7	Beispiel für erstellte Bahnlinien in A-Frame . . . . .	39
7.1	Benchmarking des Parsers . . . . .	44
7.2	Benchmarking der Objektgenerierung . . . . .	45
7.3	Benchmarking der Darstellung . . . . .	46
7.4	Umgebung Stuttgart Hauptbahnhof . . . . .	47
7.5	Tokio . . . . .	48
7.6	Manhattan - New York . . . . .	49



# **Tabellenverzeichnis**

3.1	Bestandteile der Nodes . . . . .	19
3.2	Bestandteile der Ways . . . . .	20
3.3	Bestandteile der Relations . . . . .	21
3.4	Bestandteile der Members . . . . .	21
4.1	Straßentypen und deren Eigenschaften . . . . .	29
4.2	Naturgebiettypen und deren Farben . . . . .	30
7.1	Systemeigenschaften von System 1 . . . . .	43
7.2	Systemeigenschaften von System 2 . . . . .	43
7.3	Umfang des Kartenausschnittes Stuttgart Hauptbahnhof . . . . .	47
7.4	Umfang des Kartenausschnittes Tokio . . . . .	48
7.5	Umfang des Kartenausschnittes Manhattan - New York . . . . .	49



# **Verzeichnis der Listings**

3.1	Umwandeln von Längen- und Breitengrad in das kartesische Koordinatensystem . . . . .	18
3.2	Beispiel eines Nodes aus den OSM-Daten . . . . .	20
3.3	Beispiel eines Ways aus den OSM-Daten . . . . .	20
3.4	Beispiel einer Relation aus den OSM-Daten . . . . .	21
3.5	Eigenhändige Erstellung einer Geometry . . . . .	22
3.6	Erstellung eines Shapes mit ShapeGeometry . . . . .	23
3.7	Erstellung eines Meshs aus Material und Geometry . . . . .	23
3.8	Zusammenfügen von Geometries . . . . .	24
3.9	Entität der Weltkugel . . . . .	25



# 1 Einleitung und Motivation

Das 21. Jahrhundert ist ein Zeitalter, in dem der Alltag der Menschen immer mehr von digitalen Technologien bestimmt wird. Ein großer Prozentsatz des Lebens der Menschen läuft bereits digital über Computer, Smartphones oder andere Smart-Geräte ab. Vielen Menschen ist ihre Internet-Persönlichkeit auf den großen Social-Media-Plattformen wie Facebook, Twitter oder Instagram genauso wichtig oder sogar wichtiger als ihre echte Persönlichkeit. Es gibt kaum noch einen Haushalt ohne computergestützte Geräte, die den Menschen beim Meistern des Alltags helfen. Der digitale Wandel betrifft aber nicht nur die gesamte Gesellschaft, sondern auch die Unternehmen sind von der Digitalisierung in hohem Maße betroffen. Eine dieser neuen digitalen Technologien ist die virtuelle Realität (VR). Sie kann in sehr vielen Anwendungsfeldern genutzt werden. Zum einen kann sie für alltägliche Dinge wie zum Beispiel Treffen, Spiele, Kurse, Bildung und Ausbildung zum Einsatz gebracht werden. Zum anderen kann sie von der Industrie unter anderem für 360 Grad Werbevideos, Modellansichten, VR-Websites, Meetings, Simulationen oder zur Forschung genutzt werden. Ein Beispiel ist das autonome Fahren. Gäbe es eine perfekte Repräsentation der echten Welt in der virtuellen Welt, könnte das autonome Fahren in nahezu jeder erdenklichen Situation getestet werden, um sicherzugehen, dass beim autonomen Fahren in der realen Welt keine Fehler auftreten. Es wird ersichtlich, dass die Anwendungsgebiete der realen Welt in der virtuellen Welt auch in der Zukunft sehr groß sein werden. Ziel dieser Arbeit ist es, mithilfe von OpenStreetMap-Daten (OSM-Daten) die reale Welt auf die virtuelle Welt in A-Frame abzubilden.

Die OSM-Daten bieten eine gute Möglichkeit, die echte Welt in die virtuelle Welt zu übertragen und dabei einen Sinn dafür zu bekommen, was die virtuelle Realität zukünftig noch erreichen kann. Im Rahmen dieser Arbeit wird behandelt, wie die Erstellung einer derartigen Welt durch OSM-Daten möglich ist. Dabei werden zunächst einige Grundlagen besprochen, die beim Verständnis des Systems helfen. Insbesondere werden die OSM-Daten etwas genauer betrachtet und ein kleiner Einblick in deren Funktionalität sowie Vor- und Nachteile gegeben. Anschließend wird erklärt, wie diese OSM-Daten, mithilfe eines mit Java programmierten Parsers, in ein für Javascript nutzbares Datenformat Javascript Object Notation (JSON) übertragen werden können. Daraufhin wird tiefer auf die Erstellung der virtuellen Welt in A-Frame mit three.js, Javascript und HTML eingegangen und eine Anleitung zur Ausführung des Projektes gegeben. Bei der Evaluation der Arbeit, werden mehrere Benchmarkings und einige Beispiele aufgelistet. Die Arbeit schließt mit einer Zusammenfassung sowie einem Ausblick auf zukünftige Erweiterungsmöglichkeiten und Anwendungsgebiete ab.



## 2 Verwandte Arbeiten

Es bestehen bereits einige Arbeiten<sup>1</sup>, die die OSM-Daten in drei Dimensionen (3D) darstellen. Die meisten sind jedoch nicht auf VR zugeschnitten und stellen deshalb keine *first person perspective* beim Navigieren durch die Welten zur Verfügung. Da der Bereich WEBVR noch sehr jung ist, wurden noch nicht viele virtuelle Umgebungen mittels OpenStreetMap-Daten erstellt. In diesem Kapitel wird kurz auf zwei solcher Applikationen eingegangen.

### 2.1 OSMBuildings

OSMBuildings [Mar18] ist Open Source und daher für alle Nutzer frei verfügbar. In dieser Applikation kann der Benutzer mittels eines begrenzten Zooms die Welt in einer *Top-Down* Ansicht navigieren. Zusätzlich können Orte auf der gesamten Welt mithilfe der Suchfunktion gefunden werden. Die Karte beinhaltet vor allem detaillierte 3D-Modelle von Gebäuden und Straßen. Einige Gebäude wurden extra 3D modelliert, um den Detaillierungsgrad zu verbessern. Kleinere Details wie zum Beispiel Bäume wurden in dieser Applikation nicht implementiert. Abbildung 2.1 zeigt den Stuttgarter Schlossplatz auf der OSMBuildings Website.

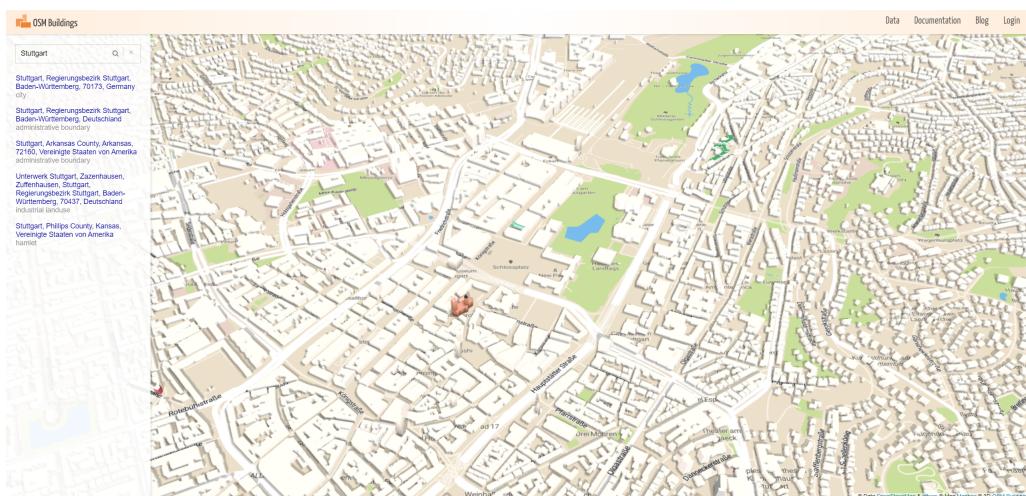


Abbildung 2.1: OSMBuildings Website [Mar18]

Die Applikation benutzt jedoch mehr als nur die OSM-Daten, um die Welt zu generieren, unter anderem auch 3D-Visualisierungstools. Außerdem wird der Kartenausschnitt in einer Ebene erstellt.

<sup>1</sup>[https://wiki.openstreetmap.org/wiki/List\\_of\\_OSM-based\\_Services](https://wiki.openstreetmap.org/wiki/List_of_OSM-based_Services)

## 2.2 VR Map

Ein Beispiel für eine Applikation, die auch mithilfe von A-Frame und OSM-Daten erstellt wurde, ist VR Map [Kai17]. In dieser Applikation kann der Benutzer Längen- und Breitengrad angeben oder eine der vorgegebenen Gebiete auswählen, um eine kleine virtuelle Welt der Umgebung erstellen zu lassen. Genauso wie bei OSMBuildings wird hier der Kartenausschnitt nicht auf eine 3D-Weltkugel gemappt, sondern in einer Ebene erstellt. Abbildung 2.2 zeigt die Startwebsite von VR Map.



Abbildung 2.2: VR Map Website [Kai17]

# 3 Grundlagen

In diesem Kapitel werden wichtige Grundlagen behandelt, die für das Verständnis des Gesamtsystems gebraucht werden. Darunter befinden sich sowohl eine Einführung in die geografischen Grundlagen und OSM-Daten, als auch Einblicke in die benutzten Frameworks A-Frame und three.js. Außerdem wird das für die Datenübertragung zwischen Java und Javascript benutzte Datenformat JSON kurz erklärt.

## 3.1 Geografische Grundlagen

Die folgenden geografischen Grundlagen werden bei vielen Berechnungen im OSMParse gebraucht, um die OSM-Daten in das kartesische Koordinatensystem zu übertragen.

### 3.1.1 Längen- und Breitengrad

Längen- und Breitengrad stellen die geografischen Koordinaten auf der Sphärenoberfläche dar. Die Längengrade verlaufen dabei immer vom Süd- zum Nordpol und liegen zwischen -180 und 180 Grad. Die Breitengrade verlaufen orthogonal zum Längengrad und parallel zum Äquator von -90 bis 90 Grad. Zusammen bilden sie ein Netz, auf dem alle Punkte auf der Sphäre mit Längen- und Breitengrad eindeutig identifizierbar sind.

### 3.1.2 Berechnung von kartesischen Koordinaten

Da die OSM-Daten alle Punkte nur in Längen- und Breitengrad angeben, müssen diese in das kartesische Koordinatensystem der virtuellen Welt in A-Frame umgewandelt werden. Die Umwandlung kann mit der in Listing 3.1 gegebenen Funktion [Rbr08] ausgeführt werden. Sie nimmt Längengrad, Breitengrad, den Radius der Sphäre und die Distanz zur Sphäre als Eingabe, um daraus einen resultierenden Punkt auf der Oberfläche der Sphäre zu berechnen. Da die Längen- und Breitengrade im Gradmaß angegeben sind, müssen sie zunächst wie in Zeile 1 und 2 in das Bogenmaß umgewandelt werden. Anschließend können die Berechnungen für die Koordinaten durchgeführt und der Ergebnisvektor ausgegeben werden.

### 3 Grundlagen

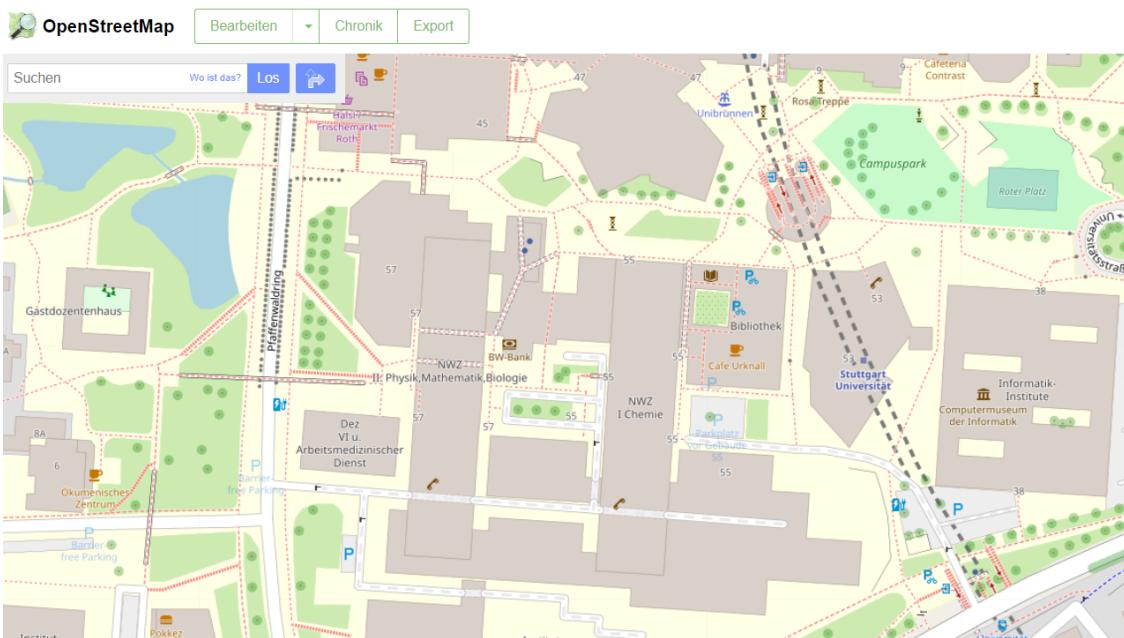
---

```
1 function getCartesianCoordinates(lat,lon,sphereradius,distance){  
2     latitude = (lat * 2 * Math.PI) / 360;  
3     longitude = (lon * 2 * Math.PI) / 360;  
4     radius = sphereradius + distance;  
5     x = radius * Math.cos(latitude) * Math.cos(longitude);  
6     y = radius * Math.cos(latitude) * Math.sin(longitude);  
7     z = radius * Math.sin(latitude);  
8     return [x,y,z];  
9 }
```

**Listing 3.1:** Umwandeln von Längen- und Breitengrad in das kartesische Koordinatensystem

## 3.2 OpenStreetMap-Daten

OpenStreetMap ist eine Ansammlung von Daten, die von jedem frei genutzt werden kann. Diese Daten enthalten geografische Informationen, welche Objekte auf der Erdoberfläche mit Längengrad und Breitengrad auf der richtigen Position in der gesamten Welt darstellen [Ben10]. Die Daten werden von vielen anderen Onlinekarten, Webanwendungen oder auch Softwareprojekten dazu genutzt, um sie auf unterschiedliche Weisen darzustellen oder damit hilfreiche Funktionen zu schaffen. Dabei entstehen die Daten von den Nutzern, die sie bearbeiten, kopieren, weiterverbreiten und auch zu kommerziellen Zwecken benutzen können. Abbildung 3.1 zeigt einen Kartenausschnitt der Universität Stuttgart aus den OSM-Daten auf der Website.



**Abbildung 3.1:** OpenStreetMap Website<sup>1</sup>

---

<sup>1</sup><https://www.openstreetmap.org>

Ziel der OSM-Daten ist es, Karten mit einer hohen Datenqualität zur Verfügung zu stellen, indem die Nutzer iterativ diese Daten um neue beziehungswise fehlende Daten über die Zeit hinweg ergänzen. Dieses Modell der Datengewinnung kann deshalb kleine Abweichungen zur Realität haben. Die OSM-Daten sind für diese Arbeit gut nutzbar, um die virtuelle Welt detailliert darzustellen.

Es existieren einige unterschiedliche Formate<sup>2</sup>, um OSM-Daten zu speichern. Das bekannteste Format ist OSM-XML, welches die Daten in Form von XML-Dateien darstellt. Die wichtigsten Vorteile dieses Formats sind, dass die Daten für Menschen gut lesbar und einfach zu parsen sind. Jedoch kann das Parsen der OSM-XML-Dateien je nach Größe einiges an Zeit und Speicher in Anspruch nehmen. Zwei weitere Formate sind OSM-PBF und o5m. Der größte Vorteil dieser Formate ist, dass die Daten sehr komprimiert binär gespeichert werden. Das führt dazu, dass Lese- und Schreibaktionen sehr schnell durchgeführt werden können. Da sie binär geschrieben sind, können sie von Menschen nicht gelesen werden. Das Format o5m hat dabei dieselbe Struktur wie OSM-XML. Das Format Level0L ist ein weiteres Format, das dem OSM-XML Format sehr ähnelt. Es ist jedoch noch leichter lesbar für Menschen. Das letzte Format ist Overpass JSON. Dieses Format erlaubt es, die OSM-Daten einfach in Javascript einzulesen. Im Rahmen dieser Arbeit wird allerdings das OSM-XML-Format benutzt, da die aufwendige Generierung der virtuellen Welt nur die Benutzung kleinerer Dateien erlaubt. In den folgenden Abschnitten wird daher das OSM-XML-Format näher erklärt.

### 3.2.1 Nodes

Nodes sind die wichtigsten Bestandteile der OSM-Daten. Sie repräsentieren einen Punkt auf der Weltkarte, welcher durch Längen- und Breitengrad angegeben ist. Ein allein existierender Node besitzt meistens einen Tag, der seinen Daseinsgrund beschreibt. Jedoch sind sie meistens Teile anderer Bestandteile der OSM-Daten, den Ways oder den Relations. Tabelle 3.1 beinhaltet die wichtigsten Bestandteile der Nodes.

Attribut	Beschreibung
ID	Eindeutiger Identifikator des Nodes als 64-bit Integer
Lat	Breitengrad des Nodes, angegeben in Grad von -90 bis +90
Lon	Längengrad des Nodes, angegeben in Grad von -180 bis +180
Tags	Menge von Schlüssel-Werte Paaren zur Beschreibung des Nodes

**Tabelle 3.1:** Bestandteile der Nodes

---

<sup>2</sup>[https://wiki.openstreetmap.org/wiki/OSM\\_file\\_formats](https://wiki.openstreetmap.org/wiki/OSM_file_formats)

### 3 Grundlagen

---

Listing 3.2 zeigt ein Beispiel für einen Node aus den OSM-Daten in der Umgebung der Universität Stuttgart. Der Tag deutet darauf hin, dass es sich bei diesem Node um einen Baum handelt.

---

```
<node id="5062278503" visible="true" version="1"
    changeset="51484452" timestamp="2017-08-27T15:09:20Z"
    user="Nanoilite" uid="2554996" lat="48.7457561" lon="9.1028677">
    <tag k="natural" v="tree"/>
</node>
```

---

**Listing 3.2:** Beispiel eines Nodes aus den OSM-Daten

#### 3.2.2 Ways

Die Ways beschreiben eine Abfolge von Nodes, welche je nach Tag unterschiedliche Objekte in der Welt repräsentieren können. So stellen zum Beispiel Ways mit dem Tag „highway“ den Verlauf der Straßen und Ways mit dem Tag „building“ die Umrisse der Gebäude dar. Die Ways enthalten dabei die Referenzen auf die Nodes. In Tabelle 3.2 sind die wichtigsten Bestandteile der Ways aufgelistet.

Attribut	Beschreibung
ID	Eindeutiger Identifikator des Ways als 64-bit Integer
Nds	Menge von Referenzen auf Knoten, die den Verlauf des Weges beschreiben
Tags	Menge von Schlüssel-Werte Paaren zur Beschreibung

**Tabelle 3.2:** Bestandteile der Ways

Listing 3.3 enthält ein Beispiel eines Ways aus den OSM-Daten. Der Tag „highway“ mit dem Wert „footway“ zeigt, dass der Way einen einfachen Fußweg darstellen soll. In diesem Fall besteht der Way nur aus zwei Nodes, d.h. die Verbindungsgeraden zwischen diesen Nodes ist der Verlauf des Weges.

---

```
<way id="23691222" visible="true" version="3" changeset="32144446"
    timestamp="2015-06-22T17:54:32Z" user="Jojo4u" uid="109062">
    <nd ref="256540352"/>
    <nd ref="256540353"/>
    <tag k="highway" v="footway"/>
</way>
```

---

**Listing 3.3:** Beispiel eines Ways aus den OSM-Daten

### 3.2.3 Relations und Members

Eine Relation enthält mehrere Members, die miteinander in Verbindung stehen. Zum Beispiel zählen zur Relation „Universität Stuttgart“ mehrere Gebäude der unterschiedlichen Fakultäten sowohl in Stuttgart Vaihingen als auch in Stuttgart Stadtmitte. Tabelle 3.3 und Tabelle 3.4 enthalten die wichtigsten Bestandteile der Relations und Members.

Attribut	Beschreibung
ID	Eindeutiger Identifikator der Relation als 64-bit Integer
Members	Menge von Bestandelementen der Relation
Tags	Menge von Schlüssel - Werte Paaren zur Beschreibung

**Tabelle 3.3:** Bestandteile der Relations

Attribut	Beschreibung
Type	Typ des Members (Relation/Way/Node)
Ref	Die Referenz-ID zur Bestimmung des referenzierten Objektes
Role	Rolle des Members in der Relation

**Tabelle 3.4:** Bestandteile der Members

Ein Beispiel ist die in Listing 3.4 aufgeführte Relation und ihre Members. In dieser Relation sind mehrere Tags und Members gegeben. Die Members mit dem Type „way“ beschreiben den Grundriss des Gebäudes. Dabei sind die Members mit der Rolle „outer“ die Umrandung des Gebäudes und die Members mit den Rollen „inner“ die Löcher in dem Gebäude (Innenhöfe). Die große Liste an Tags beschreibt viele Eigenschaften der Relation. Insbesondere deutet der Tag „building“ darauf hin, dass es sich bei dieser Relation um ein Gebäude handelt. Der Rest der Tags steht für die detailliertere Beschreibung des Gebäudes zur Verfügung.

```
<relation id="2922269" visible="true" version="6" changeset="50090948"
    timestamp="2017-07-06T18:10:28Z" user="liondog" uid="5060480">
    <member type="way" ref="221018878" role="inner"/>
    <member type="way" ref="221018877" role="inner"/>
    <member type="way" ref="221018880" role="inner"/>
    <member type="way" ref="221018879" role="inner"/>
    <member type="way" ref="22741007" role="outer"/>
    <tag k="addr:city" v="Stuttgart"/>
    <tag k="addr:country" v="DE"/>
    <tag k="addr:housenumber" v="38"/>
    <tag k="addr:street" v="Universitaetsstrasse"/>
    <tag k="building" v="university"/>
    <tag k="building:colour" v="white"/>
    <tag k="building:levels" v="3"/>
    <tag k="name" v="Informatik-Institute"/>
    <tag k="type" v="multipolygon"/>
    <tag k="wheelchair" v="yes"/>
</relation>
```

**Listing 3.4:** Beispiel einer Relation aus den OSM-Daten

### 3.3 Three.js

Three.js<sup>3</sup> ist ein auf der WebGraphics Library (WEBGL) basiertes Framework, dessen Ziel es ist, WEBGL für Entwickler einfach zugänglich zu machen. WEBGL sorgt dafür, dass die Grafik Hardware (Graphics Processing Unit) des Benutzercomputers im Browser zum Einsatz kommt, unter anderem um dreidimensionale Objekte zu erstellen, zu animieren und in einer Szene im Webbrower anzuzeigen. Jedoch ist es sehr aufwändig, direkt mit WEBGL zu arbeiten, da Angelegenheiten wie die Szene, Beleuchtung, Schattierung, Materialien usw. eigenhändig programmiert werden müssen [Dir13]. Three.js übernimmt diese Arbeit für den Entwickler und bietet simple Funktionen, um dreidimensionale Szenen zu entwickeln. In diesem Abschnitt wird kurz auf die wichtigsten Komponenten und die in dieser Arbeit benutzten Funktionen von three.js eingegangen.

#### 3.3.1 Geometry

Um komplexe dreidimensionale Formen zu erstellen, wird die Klasse THREE.Geometry benutzt. Dafür sind zwei Optionen gegeben. Entweder kann eine Form vom Entwickler erstellt werden, indem die Eckpunkte und Flächen eigenhändig erstellt und der Geometry übergeben werden oder es kann eine der vordefinierten Geometry-Kindklassen verwendet werden. Beim eigenhändigen Erstellen werden die Eckpunkte der „vertices“ Liste der Geometry als THREE.Vector3 übergeben. Anschließend können diese über THREE.Face3, welche den Index der Eckpunkte in der „vertices“ Liste übergeben bekommt, verbunden werden. Die THREE.Face3 werden dann noch der „faces“ Liste der Geometry hinzugefügt. Listing 3.5 zeigt den genauen Ablauf der eigenhändigen Erstellung einer Geometry.

---

```
1 var geometry = new THREE.Geometry();
2 geometry.vertices.push(
3   new THREE.Vector3(100, 600),
4   new THREE.Vector3(1100, 600),
5   new THREE.Vector3(1100, 100),
6   new THREE.Vector3(100, 100) );
7 geometry.faces.push(new THREE.Face3(0, 1, 2));
8 geometry.faces.push(new THREE.Face3(0, 2, 3));
```

---

**Listing 3.5:** Eigenhändige Erstellung einer Geometry

Der Benutzer muss also dabei eigenhändig die Triangulierung der dreidimensionalen Form übernehmen. Ein vereinfachtes Modell der Erzeugung bieten die Geometry-Kindklassen. Diese beruhen zwar auf THREE.Geometry, besitzen jedoch einzigartige Übergabeparameter, die das Erstellen der Form für den Entwickler übernehmen. So kann zum Beispiel THREE.ShapeGeometry dazu verwendet werden, eine Form als ein zweidimensionales Shape mit individuellen Kanten zu errichten. Dazu wird so wie in Listing 3.6 dieser Geometry beim Erstellen ein THREE.Shape als Parameter übergeben. Zur Erstellung des THREE.Shapes werden die umrandenden Punkte übergeben.

---

<sup>3</sup><https://threejs.org/docs/>

---

```

1 let shapePoints = [];
2 shapePoints.push(
3   new THREE.Vector3(100, 600),
4   new THREE.Vector3(1100, 600),
5   new THREE.Vector3(1100, 100),
6   new THREE.Vector3(100, 100)
7 );
8 let shape = new THREE.Shape(shapePoints);
9 let shapeGeometry = new THREE.ShapeGeometry(shape);

```

---

**Listing 3.6:** Erstellung eines Shapes mit ShapeGeometry

Ist die Übertragung einer THREE.ShapeGeometry auf die dritte Dimension gewünscht, so kann das erreicht werden, indem jeder zweidimensionale Punkt in der „vertices“ Liste der THREE.ShapeGeometry zu einem dreidimensionalen Punkt umgewandelt wird. Dabei bleibt insbesondere die „faces“ Liste der genannten Geometry erhalten, welche die Kanten genauso wie bei der zweidimensionalen ShapeGeometry trianguliert. Das Resultat ist dann das gleiche Shape, nur auf die dritte Dimension übertragen.

Zusätzlich zur ShapeGeometry werden in dieser Arbeit noch die SphereGeometry und BoxGeometry benutzt. Diese bekommen als Parameter nur die Größe des zu erstellenden Objektes übergeben.

### 3.3.2 Material und Mesh

Ein Material gibt die Oberfläche und damit das generelle Aussehen der Form an. Die Basisklasse THREE.Material definiert alle wichtigen Bestandteile, die von den Material-Kindklassen zur Verarbeitung gebraucht werden. Je nach ausgewähltem Material verändert sich das Verhalten des Objektes mit Lichtquellen und Schatten. Es könnte jedoch ähnlich wie bei den Geometries auch eigenständig ein Material erstellt werden, das den Bedürfnissen des Entwicklers entspricht, indem die Basisklasse verwendet wird. Ein Mesh kombiniert eine Geometry und ein Material, um eine sichtbare Oberfläche des Objektes in die Szene zu zeichnen. Das folgende Listing 3.7 schließt an Listing 3.6 an und zeigt, wie eine Geometry mit einem Material zu einem Mesh kombiniert werden kann.

---

```

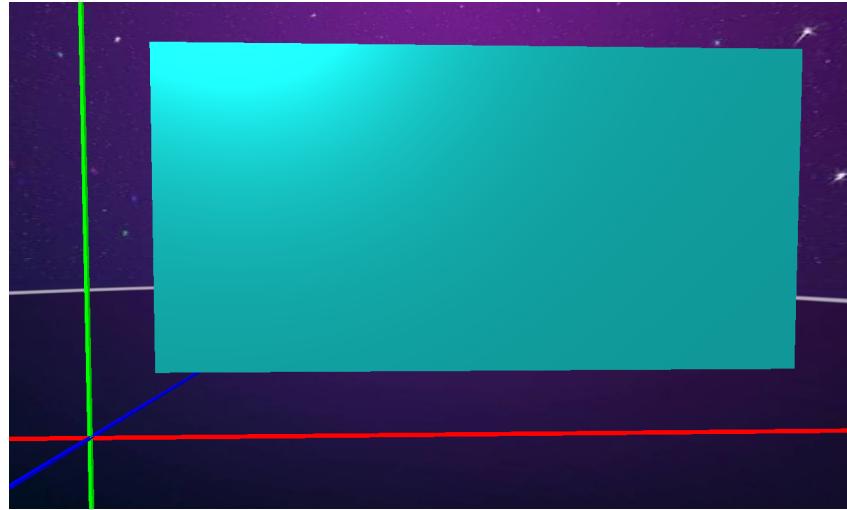
1 let material = new THREE.MeshStandardMaterial({
2   color: 0x1bffff,
3   side: THREE.DoubleSide,
4   wireframe: false
5 });
6 let mesh = new THREE.Mesh(shapeGeometry, material);

```

---

**Listing 3.7:** Erstellung eines Meshs aus Material und Geometry

Das in Abbildung 3.2 resultierende Shape besitzt aufgrund der benutzten ShapeGeometry und der übergebenen Punkte die Form eines Rechtecks. Das benutzte Material sorgt für das Aussehen. Würde statt der in Listing 3.6 benutzten Geometry die in Listing 3.5 erstellte Geometry verwendet werden, wäre das Resultat dasselbe.



**Abbildung 3.2:** Entstandenes Shape

### 3.3.3 Performance Verbesserung

Bei der Entwicklung eines Systems, bei dem nicht viele Objekte notwendig sind, entsteht kaum ein Problem mit der Performance. Jedoch braucht jedes Objekt, das einzeln hinzugefügt wird, einen *Draw Call*. Alle *Draw Calls* werden jeweils beim Rendern der Szene aufgerufen. Das kann bei einer großen Menge an Objekten dazu führen, dass die *frames per second* (FPS) deutlich absinken.

Da die virtuelle Welt aus einer großen Menge an Objekten bestehen kann, können bei einer Szene je nach Anzahl der Nodes die Anzahl der angezeigten Objekte schnell in die Tausende beziehungsweise Zehntausende gehen. Dies führt zu einer entsprechenden Anzahl an *Draw Calls*, wodurch auch die FPS absinken.

Um die Verschlechterung der Performance zu vermeiden, muss die Anzahl der *Draw Calls* reduziert werden. Dies kann erreicht werden, indem die Geometries der einzelnen Objekte zusammengefügt werden. Bedingung für das Zusammenfügen ist, dass die Objekte dasselbe Material besitzen, da das Mesh genau ein Material für die zusammengefügte Geometry benutzt. Listing 3.8 zeigt den Ablauf für das Zusammenfügen von Geometries. Dafür wird eine neue Geometry erstellt, die alle Geometries beinhaltet soll. Die weiteren Geometries, die zusammengefügt werden sollen, werden in einem Mesh erstellt. Zum Zusammenfügen muss die Matrix des Meshs aktualisiert werden. Danach kann die Geometry des Meshs mit der singleGeometry zusammengefügt werden und ein neues Mesh erstellt werden, das die singleGeometry und das gewählte Material übergeben bekommt. Durch diese Methode wurden beispielsweise die *Draw Calls* aller Gebäude (ohne die Löcher), auf eins reduziert, da sie dasselbe Material besitzen.

---

```

1 let singleGeometry = new THREE.Geometry();
2 let geometry = new THREE.Geometry();
3 let mesh = new THREE.Mesh(geometry);
4 mesh.updateMatrix();
5 singleGeometry.merge(mesh.Geometry, mesh.Matrix);
```

---

**Listing 3.8:** Zusammenfügen von Geometries

## 3.4 A-Frame

A-Frame<sup>4</sup> ist ein three.js basiertes Framework, um virtuelle Umgebungen im Webbrowser zu erstellen. Dazu werden HTML und Javascript als Programmiersprachen verwendet. Das Ziel von WEB-VR ist es, den Benutzern einen einfachen Einstieg in die virtuelle Realität zu ermöglichen. Dazu muss nur ein Skript in die HTML-Datei miteinbezogen werden. A-Frame bietet mehrere Primitiven zum einfachen Erstellen von virtuellen Umgebungen. Jedoch reichen diese Primitiven nicht aus, um eine detaillierte Darstellung der Weltoberfläche in drei Dimensionen zu erstellen. Da A-Frame auf Javascript und damit auch auf three.js aufbaut, können three.js-Objekte in das A-Frame Projekt miteinbezogen werden. Diese können dazu benutzt werden, detailliertere Objekte zu modellieren. Für diese Arbeit wurde A-Frame in Kombination mit three.js verwendet, um die virtuelle Welt zu erstellen.

### 3.4.1 Entität-Komponenten-System

Das Framework basiert auf dem Entität-Komponenten-System. Die Entitäten werden durch `<a-entity>` in HTML repräsentiert und gruppieren mehrere Komponenten zusammen. Die Komponenten spezifizieren dabei das Aussehen, das Verhalten und die Funktionalität der Entität. Ein direktes Beispiel aus dieser Arbeit ist die Weltkugel, die als Entität erstellt ist, enthalten in Listing 3.9.

---

```
<a-entity id="sphere" geometry="primitive: sphere;
  segmentsHeight: 100; segmentsWidth: 100;" scale="2000 2000 2000" position="0 0 0"
  rotation="0 0 0" material="color: #050012; repeat: 100000 100000; transparent: true;
  metalness:0.6; roughness: 0.4; sphericalEnvMap: #sky;">
</a-entity>
```

---

**Listing 3.9:** Entität der Weltkugel

Hier besitzt die Entität „sphere“ mehrere Komponenten. Dazu zählen die Geometry, Scale, Position, Rotation und das Material. Das Entität-Komponenten-System bietet einige nützliche Vorteile. Zum einen ist es dadurch sehr leicht, Entitäten auf unterschiedlichste Art und Weise darzustellen, indem unterschiedliche Kombinationen von Komponenten benutzt werden. Zum anderen ist es einfach, Erweiterungen hinzuzufügen. Gute Skalierbarkeit bei sehr großen Anwendungen ist ein weiterer Vorteil. Insgesamt bietet das Entität-Komponenten-System sehr nützliche und übersichtliche Funktionen, um Applikationen für die virtuelle Realität zu entwickeln.

---

<sup>4</sup><https://aframe.io/>

### 3.5 JavaScript Object Notation

JavaScript Object Notation (JSON)<sup>5</sup> dient der kompakten Datenspeicherung und -übertragung zwischen Applikationen. Das Format ist mit der Hintergrundidee erstellt worden, für Menschen lesbar, für Computer einfach zu parsen und unabhängig von Programmiersprachen nutzbar zu sein [Bas15]. Die meisten bekannten Programmiersprachen bieten Bibliotheken an, um JSON-Dateien zu erstellen und diese einzulesen. Speziell für Javascript ist dieses Format sehr einfach einzulesen, da Javascript-Objekte auch mithilfe von JSON erstellt werden können. Die Daten werden in Form von Name-Werte Paaren und geordneten Listen erstellt. In dieser Arbeit wird das Format dazu genutzt, die beim OSM-Parsen entstandenen Java-Objekte in JSON abzuspeichern und in Javascript zur Erstellung der virtuellen Welt einzulesen.

---

<sup>5</sup><https://www.json.org/>

# 4 Parsen und Behandeln der OSM-Daten

Das System kann in vier Komponenten unterteilt werden: die OSM-Daten, die Konsolen-Java-Applikation OSMParse, die JSON-Dateien und die virtuelle Welt in A-Frame. Der generelle Ablauf des Systems besteht darin, dass die OSM-Daten von dem OSMParse eingelesen und verarbeitet werden. Dabei erstellt der OSMParse eine JSON-Datei, um die resultierenden Java-Objekte in ein für Javascript gut lesbares Format zu speichern. Zuletzt wird die entstandene JSON-Datei von dem Javascript-Programm eingelesen und aus dessen Inhalt werden dreidimensionale Objekte erstellt. In diesem Kapitel wird auf die genauere Funktionsweise des OSMParsers und damit auch auf die Erstellung der JSON-Datei eingegangen.

## 4.1 Parsen der OSM-Daten

Zum Einlesen der OSM-Daten wird die externe Java-Bibliothek org.xml.sax<sup>1</sup> verwendet, die Funktionen für das Parsen und Behandeln der OSM-Daten bietet. Der ContentHandler definiert, wie der Inhalt der eingelesenen Datei behandelt werden soll. In diesem Abschnitt wird auf die genaue Re Behandlung der Objekte beim Einlesen, also auf die Implementierung des ContentHandlers, eingegangen.

### 4.1.1 Behandeln der Nodes

Wird ein Element mit Namen „node“ eingelesen, erstellt der ContentHandler ein Objekt vom Typ „Node“, das die Position in Längen- und Breitengrad sowie die ID enthält. Das Objekt wird dann mit seiner ID zur Identifikation als Key zu einer Liste von Schlüssel-Werte Paaren hinzugefügt. Sollte anschließend ein Element mit Namen „tag“ eingelesen werden, so wird dieser zu der Liste von Tags des zuletzt gelesenen Node hinzugefügt.

### 4.1.2 Behandeln der Ways

Wird ein Element mit Namen „way“ eingelesen, dann wird dieser als ein Objekt vom Typ „Way“ mit einer leeren Liste von Nodes erstellt und zu der Liste von Ways hinzugefügt. Wenn ein Element mit dem Namen „nd“ eingelesen wird, so ist es Teil eines Ways mit einem Attribut „ref“ als Referenz auf den zugehörigen Node. Nun wird dieses Attribut dazu genutzt, diesen in der Liste der Nodes ausfindig zu machen und ihn zu der Liste der Nodes des zuletzt gelesenen Ways hinzuzufügen. Sollte anschließend ein Element mit Namen „tag“ eingelesen werden, so wird er, genauso wie bei den Nodes, zu der Liste von Tags des zuletzt gelesenen Ways hinzugefügt.

---

<sup>1</sup><https://docs.oracle.com/javase/7/docs/api/org/xml/sax/package-summary.html>

### 4.1.3 Behandeln der Relations und Members

Für jedes Element mit dem Namen „Relation“, das eingelesen wird, wird ein Objekt vom Typ „Relation“ mit einer leeren Liste von Members erstellt und der Liste von Relations hinzugefügt. Wird ein Member-Element eingelesen, so wird anschließend ein Member-Objekt mit seiner Referenz-ID, seinem Typ und seiner Rolle erstellt und der Liste von Members der zuletzt eingelesenen Relation hinzugefügt. Sollte anschließend ein Element mit Namen „tag“ eingelesen werden, so wird er, genauso wie bei den Nodes und den Ways, zu der Liste von Tags der Relation hinzugefügt.

## 4.2 Erstellen der Java-Objekte

Nachdem das Parsen der OSM-Datei durchgeführt und damit alle OSM-Elemente eingelesen und zu Objekten umgewandelt wurden, können diese dazu benutzt werden, neue Objekte der Bestandteile der virtuellen Welt zu erstellen. Die folgenden Funktionen benutzen die erzeugten OSM-Elemente, um Objekte zu erstellen, deren Attribute später von dem Javascript-Programm genutzt werden können.

### 4.2.1 Umwandeln der Nodes in Bäume

Um herauszufinden, ob ein Node einen Baum darstellt, werden die Nodes auf den Tag „natural“ mit dem Wert „tree“ untersucht. Wurde ein Baum gefunden, so wird für diesen die Position des Stammes und mit gering größerer Distanz zur Sphäre die Position der Baumkrone mit der Funktion in Abschnitt 3.1.2 ausgerechnet. Zusätzlich werden die Tags des Nodes auf den Schlüssel „leaf-type“ untersucht. Ist der Wert davon „broadleaved“, so ist der Baum ein Laubbaum. Ist dieser „needleleaved“, so ist der Baum ein Nadelbaum. Anschließend wird ein neues Tree-Objekt mit den berechneten Parametern erstellt und der Liste von Trees hinzugefügt.

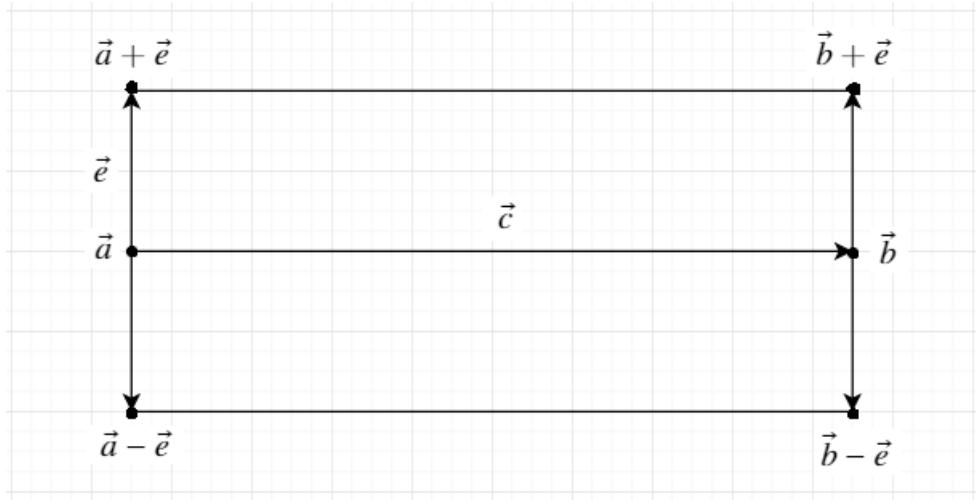
### 4.2.2 Umwandeln der Ways in Straßen

Zuerst muss herausgefunden werden, welche Ways Straßen darstellen, da Ways viele unterschiedliche Objekte darstellen können. Dazu müssen die Ways auf den Tag „highway“ untersucht werden. Ist dieser in dem Way vorhanden, so stellt er eine Straße dar. Die Abfolge der Nodes in dem Way repräsentiert dann den Verlauf der Straße. Wie bereits gezeigt, sind Tags Schlüssel-Werte Paare, also kann die Art der Straße je nach Wert des Keys „highway“ bestimmt werden. Je nach Art der Straße ändert sich auch die Breite und die Farbe. Zum Beispiel haben Straßen mit dem Wert „motorway“ eine rote Farbe und eine relativ große Breite. Außerdem haben die Straßen einen kleinen Höhenunterschied, der mit der Breite der Straße skaliert, damit diese keine Kollisionsprobleme beim Erstellen in A-Frame haben. Die Breite und Farbe für die unterschiedlichen Straßen wurde entsprechend der Repräsentation auf der OpenStreetMap-Website gehalten. Jedoch besitzen nur die wichtigsten Straßentypen eine eigene Farbe und Breite. Dem Rest der Typen wird ein Standardwert für die beiden Attribute zugewiesen. Tabelle 4.1 enthält die behandelten Straßentypen und deren benutzten Farben im Hexadezimalsystem sowie die zugewiesene Breite der Straße.

Typ der Straße	Farbe	Breite
default	#DBDBDB	$b$
motorway	#FF4C4C	$2.2 \cdot b$
trunk	#FFA500	$2.0 \cdot b$
primary	#FFB732	$1.7 \cdot b$
secondary	#FFFF66	$1.4 \cdot b$
tertiary	#E5E5E5	$1.2 \cdot b$

**Tabelle 4.1:** Straßentypen und deren Eigenschaften

Da die Ways nur eine Abfolge von Nodes und damit den Verlauf der Straße beschreiben, müssen zusätzlich die Punkte der Ecken der Teilstraße zum Erstellen eines Shapes in A-Frame berechnet werden. Es sei  $\vec{a}$  der Vektor eines Nodes und  $\vec{b}$  der Vektor des darauf folgenden Nodes, die den Verlauf der Straße angeben und damit gegeben. Der Vektor zwischen dem ersten und dem zweiten Node des Verlaufes der Straße wird von  $\vec{c}$  dargestellt. Dieser kann mit der Vektorsubtraktion von  $\vec{a}$  und  $\vec{b}$  berechnet werden. Die Breite der Straße ist mit  $b$  bezeichnet. Abbildung 4.1 zeigt den gesuchten Vektor  $\vec{e}$  und die gesuchten Eckpunkte der Teilstraße.

**Abbildung 4.1:** Berechnung der Eckpunkte der Straßen

Um den gesuchten Vektor  $\vec{e}$  zu finden, muss gelten:

$$\vec{a} \circ \vec{e} = 0, \quad \vec{c} \circ \vec{e} = 0, \quad |\vec{e}| = \frac{b}{2} \quad (4.1)$$

Die ersten zwei Gleichungen sorgen dafür, dass  $\vec{e}$  im rechten Winkel zu  $\vec{a}$  und auch zu  $\vec{c}$  steht. Zur Berechnung des gesuchten Vektors  $\vec{e}$  wird das Kreuzprodukt der beiden Vektoren  $\vec{a}$  und  $\vec{c}$  berechnet. Die letzte Gleichung setzt die Länge des Vektors  $\vec{e}$  auf die Hälfte der Breite der Straße. Dazu wird, nach der Berechnung des Kreuzproduktes, der Vektor normiert und anschließend mit  $b$  multipliziert. Der resultierende Vektor wird dann sowohl auf  $\vec{a}$  und  $\vec{b}$  addiert als auch von diesen subtrahiert, um die vier gesuchten Eckpunkte zu erhalten. Diese werden für alle zwei aufeinanderfolgende Punktpaare des Ways berechnet. Schließlich werden alle berechneten Parameter dazu benutzt, ein neues Street-Objekt zu erstellen und dieses der Liste von Streets hinzuzufügen.

### 4.2.3 Umwandeln der Ways in Naturgebiete

Naturgebiete können viele unterschiedliche Repräsentationen in den OSM-Daten haben. Um diese zu finden, wird zuerst nach dem Tag „natural“ und „landuse“ gesucht. Wenn diese einen Wert haben, der als Naturgebiet gesehen werden kann, werden zu diesen eine entsprechende Farbe angegeben und die umrandenden Punkte wie bisher von Längen- und Breitengrad in das kartesische Koordinatensystem umgewandelt. Tabelle 4.2 enthält alle behandelten Schlüssel-Werte Paare und deren Farbcodes, angegeben im Hexadezimalsystem. Die Farben wurden entsprechend der Repräsentation auf der OpenStreetMap Website gehalten. Nachdem alle wichtigen Parameter berechnet wurden, wird mit ihnen ein neues NaturalArea-Objekt erzeugt und der Liste von NaturalAreas hinzugefügt.

Typ des Gebiets	Farbe
natural-default	#6FE300
natural-water	#0077BE
natural-barerock/scree/shingle	#BEBEBE
natural-sand/beach	#C2B280
natural-mud	#794C13
natural-wetland	#0077BE
natural-glacier	#ADD8E6
landuse-grass/meadow	#6FE300

**Tabelle 4.2:** Naturgebiettypen und deren Farben

### 4.2.4 Umwandeln der Ways in Bahnlinien

Weiterhin ist es möglich, dass ein Way eine Bahnlinie repräsentiert. Für das Umwandeln werden die Ways auf den Tag „railway“ untersucht, um herauszufinden, ob es sich um eine Bahnlinie handelt. Wenn dieser vorhanden ist, so wird für diesen Way jeder Node in das kartesische Koordinatensystem umgewandelt und die Abfolge der Punkte als Liste gespeichert. Das anschließend erstellte Railway-Objekt wird dann der Liste von Railways hinzugefügt.

### 4.2.5 Umwandeln der Ways in Gebäude

Zu Beginn werden die Ways auf den Tag „building“ untersucht, um herauszufinden, ob der Way ein Gebäude darstellen soll. Sollte dieser vorhanden sein, wird folgendermaßen vorgegangen. Zunächst werden alle Nodes des Ways in Dachpunkte und Bodenpunkte umgewandelt. Um die Dachpunkte zu berechnen, muss die Höhe des Gebäudes bestimmt werden. Diese kann in den Tags spezifiziert sein. Dabei gibt es drei Möglichkeiten. Entweder ist sie direkt angegeben als „height“, dann kann das Attribut direkt dazu genutzt werden, die Höhe des Gebäudes zu bestimmen. Alternativ kann die Anzahl der Stockwerke des Gebäudes als „building:levels“ angegeben sein. In diesem Fall kann zwar die Höhe des Gebäudes nicht exakt bestimmt werden, jedoch kann sie approximiert werden, indem die Standardhöhe der Gebäude mit der Anzahl der Stockwerke multipliziert wird. Sollte keiner der beiden Tags vorhanden sein, wird die Standardhöhe verwendet, die davon ausgeht, dass

das Gebäude nur ein Erdgeschoss besitzt. Nachdem die Höhe berechnet wurde, können die Dach- und Bodenpunkte in das kartesische Koordinatensystem umgewandelt werden. Damit sind alle wichtigen Parameter für die Erstellung der Gebäude in A-Frame berechnet. Anschließend wird mit ihnen ein Building-Objekt mit den berechneten Parametern erstellt und der Liste von Buildings hinzugefügt.

#### 4.2.6 Umwandeln der Relations in Gebäude

Anfänglich müssen die Relations wieder auf den Tag „building“ untersucht werden, um festzustellen, welche Relations ein Gebäude darstellen sollen. Bei der Umwandlung dieser Relations in Gebäude werden die Höhe sowie Dach- und Bodenpunkte analog zur Umwandlung der Ways in Gebäude berechnet. Jedoch werden die Members mit der Rolle „outer“ als umrandende Ways der Gebäude benutzt. Zusätzlich müssen hierbei die Löcher des Dachs berechnet werden. Dazu wird die Relation auf Members mit dem Typ „way“ und der Rolle „inner“ untersucht. Diese Members stellen die Umrandung der Löcher in dem Gebäude dar. Zu allen dieser Members werden die Punkte des Ways in das kartesische Koordinatensystem umgewandelt und dabei „innerRoofPoints“ und „innerFloorPoints“ erzeugt. Anschließend wird ein neues Building-Objekt erstellt und der Liste von Buildings hinzugefügt.

### 4.3 Bearbeiten der Einstellungen

Zusätzlich können vom Benutzer einige Einstellungen geändert werden. Dabei kann er angeben, welche Objektarten er in der virtuellen Welt angezeigt haben möchte. Außerdem kann er bestimmen, ob nur die Triangulierung der Objekte oder die vollständigen Objekte in der virtuellen Welt angezeigt werden sollen. Der Benutzer kann auch bestimmen, welche Kamera er für die Navigation der virtuellen Welt benutzen möchte. Dabei kann er sich zwischen der Spherical-Kamera, die auf der Sphäre angebracht ist oder der Fly-Kamera, welche sich frei bewegen kann, entscheiden. Die Distanz der Kamera zur Sphäre ist auch veränderbar. Zusätzlich kann er sich die Statistiken ausgeben lassen. Diese enthalten alle Elemente und die erstellten Bestandteile der virtuellen Welt sowie deren Häufigkeiten, die durch die eingelesenen OSM-Dateien entstanden sind.

### 4.4 Erstellen der JSON-Datei

Die aus der Benutzung des OSMParsers resultierende JSON-Datei beinhaltet alle Daten, die zur Erstellung der Objekte in A-Frame benötigt werden. Jedoch haben diese Objekte eine unterschiedliche Repräsentation in der JSON-Datei. Die genaue Formatierung der Daten wird in diesem Abschnitt näher erläutert. Nachdem alle Bestandteile der virtuellen Welt berechnet wurden, können nun die JSON-Objekte zum Schreiben der JSON-Datei erstellt werden.

Die Kamera benötigt nur ihren Punkt zur korrekten Positionierung, deshalb enthält sie genau drei Einträge in der JSON-Datei. Der erste Eintrag ist der „cameraPoint“, welcher den Punkt in kartesischen Koordinaten darstellt. Um diesen zu bestimmen, wurden die kartesischen Koordinaten

#### 4 Parsen und Behandeln der OSM-Daten

---

des ersten eingelesenen Nodes benutzt. Die beiden weiteren Einträge sind „cameralat“ und „cameralon“, die die Position des doppelten Nodes in Längen- und Breitengrad wiedergeben.

Ein Gebäude in der JSON-Datei besitzt vier verschiedene Listen. Die Listen „roofPoints“ und „floorPoints“ geben die umrandenden Punkte des Daches und des Bodens in kartesischen Koordinaten an. Zusätzlich können die Gebäude Löcher besitzen. Die Listen „innerRoofPoints“ und „innerFloorPoints“ beinhalten die kartesischen Koordinaten der Umrandungen der Löcher in dem Dach und in dem Boden.

Wie in dem Abschnitt des OSM-Parsers beschrieben, brauchen die Straßen zur Erstellung die Eckpunkte. Diese sind auch in der JSON-Datei zu finden. So liegt für jede Straße in der JSON-Datei eine Liste der Teilstraßen, also die Verbindungen zwischen den einzelnen Nodes, vor. Diese Liste „streetPoints“ enthält mehrere Listen von Punkten im kartesischen Koordinatensystem. Außerdem enthält eine Straße eine Liste von „circlePoints“, die den Verlauf der Straße wiedergeben. Zusätzlich können die Straßen unterschiedliche Farben besitzen. Die Farbe ist als Hexadezimalwert in „color“ angegeben.

Die Bäume brauchen jeweils nur einen Punkt für die Baumkrone, den „leavesPoint“ und einen Punkt für den Stamm, den „trunkPoint“. Auch diese sind im kartesischen Koordinatensystem angegeben.

Die Naturgebiete enthalten je nach Art eine Farbe, die ebenso als Hexadezimalwert in „color“ angegeben ist. Zusätzlich enthalten sie eine Liste von Punkten, den „naturalAreaPoints“ im kartesischen Koordinatensystem, die die Umrandung des Naturgebiets wiedergeben.

Die Bahnlinien bestehen nur aus einer Liste von Abfolgpunkten im kartesischen Koordinatensystem, die den Verlauf der Bahnlinie wiedergeben.

# 5 Erstellen der virtuellen Welt in A-Frame

Wenn die Daten in einem korrekten Format für die Erstellung der Objekte auf der Erdoberfläche vorhanden sind, kann die virtuelle Welt in A-Frame generiert werden. Jeder Objekttyp besitzt dabei seine eigene Erzeugungsmethode. In diesem Kapitel wird die ausführliche Vorgehensweise bei der Erstellung der Umgebung und der Objekte sowie die Handhabung der Kamera behandelt. Zusätzlich wird auf die benutzten Skripte eingegangen. Zum Erstellen der Welt wurde A-Frame in Kombination mit HTML und Javascript benutzt.

## 5.1 Erstellen der Umgebung

Die Umgebung ist statisch und wurde deshalb in der HTML-Datei „index.html“ geschrieben. Die Szene beinhaltet die Weltkugel, den Himmel und die X-, Y- und Z-Achsen, die zur Orientierung dienen.

Die Sphäre, die in diesem Projekt als Weltkugel dient, wurde als A-Frame Primitive A-Sphere erschaffen. Die Position der Sphäre ist das Zentrum des Koordinatensystems. Der Radius der Sphäre beträgt 2000 Längeneinheiten. Das verwendete Material sorgt für die Textur und das Behandeln des Lichteinfalls sowie der Farbe, etc. Auf diese Sphäre wird später der Kartenausschnitt aus den OSM-Daten in drei Dimensionen erstellt. Abbildung 5.1 zeigt die erstellte Umgebung in A-Frame.

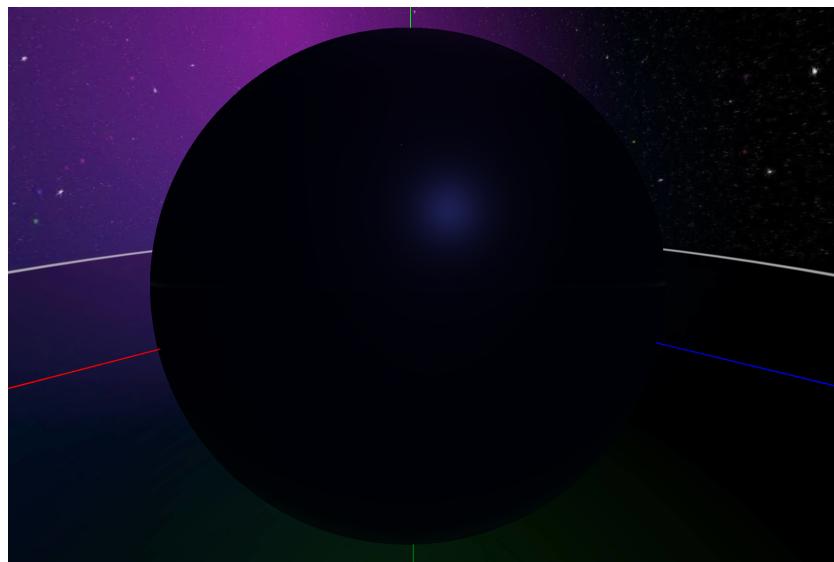


Abbildung 5.1: Umgebung der virtuellen Welt

## 5.2 Erstellen der Karte

Die Erstellung der Objekte auf der Oberfläche der Sphäre in A-Frame ist von Objekt zu Objekt unterschiedlich. Zur Erstellung wurde nur three.js verwendet. Deshalb werden hier die grundlegenden Konzepte aus Abschnitt 3.3 benutzt. Auf die genauere Vorgehensweise bei der Erstellung der Objekte wird in diesem Abschnitt eingegangen. Alle Punkte, die aus der JSON-Datei gelesen werden, werden vorher in Vector3-Objekte zur korrekten Formatierung umgewandelt. Zum Importieren der JSON-Datei wird die Funktion „fetch“ genutzt, die dafür sorgt, dass die JavaScript-Objekte in der Applikation verwendet werden können. Abbildung 5.2 enthält einen exemplarisch generierten Kartenausschnitt, der alle erstellten Objekttypen enthält.



Abbildung 5.2: Exemplarischer Kartenausschnitt

### 5.2.1 Erstellen der Straßen

Um die Straßen anzufertigen, wird für jede Teilstraße aller Straßen in der JSON-Datei, ein Shape-Objekt mithilfe der in Abschnitt 5.2.1 vier berechneten Eckpunkte des Shapes in zwei Dimensionen erstellt. Anschließend wird mithilfe dieses Shapes eine ShapeGeometry erzeugt. Daraufhin werden die Eckpunkte der erstellten ShapeGeometry auf die dritte Dimension übertragen. Zusätzlich wird zur Abrundung der Kanten der Straßen für jeden in der JSON-Datei gegebenen Punkt in der Liste der „circlePoints“ eine Circlegeometry, der die Breite der Straße, also „width“ aus der JSON-Datei übergeben bekommt, erstellt. Dabei werden sowohl die erstellten CircleGeometries als auch die erstellten ShapeGeometries zu einer einzigen Geometry namens „singleGeometry“ zusammengefügt. Da zum Zusammenfügen der Geometries jeweils nur ein Material benutzt werden kann, bekommen die Straßen mit unterschiedlichen Farben unterschiedliche „singleGeometries“. Die Straßen sind nach ihrer Farbe in der JSON-Datei sortiert. Sollte während der Erstellung der Straßen ein Farbwechsel erkannt werden, so wird das Mesh der bisher erstellten „singleGeometry“ der Szene hinzugefügt und anschließend die „singleGeometry“ zurückgesetzt, sodass die Straßen mit der nächsten Farbe auch wieder zusammengefügt werden können. Diese Umstände haben jedoch den Effekt, dass die *Draw Calls* der Straßen gleich der Anzahl unterschiedlicher Farben sind. Es wird dann für jede „singleGeometry“ ein Mesh mit der in dem Material erhaltenen zugehörigen Farbe erstellt und der Szene hinzugefügt. Ein Beispiel für das Resultat wird in Abbildung 5.3 gezeigt.

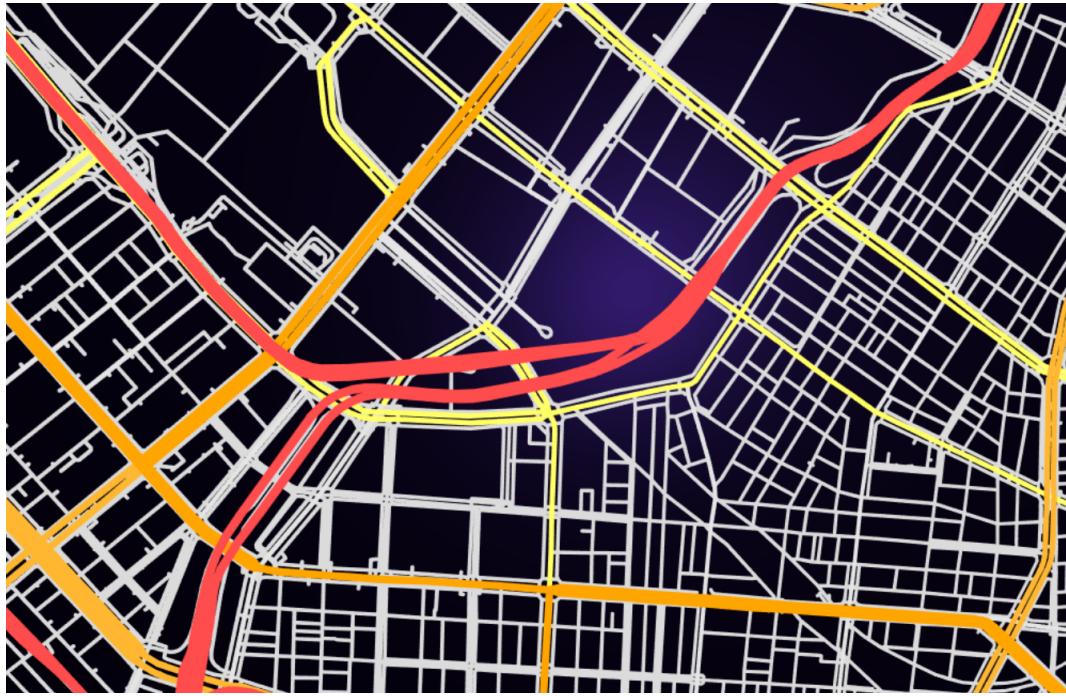
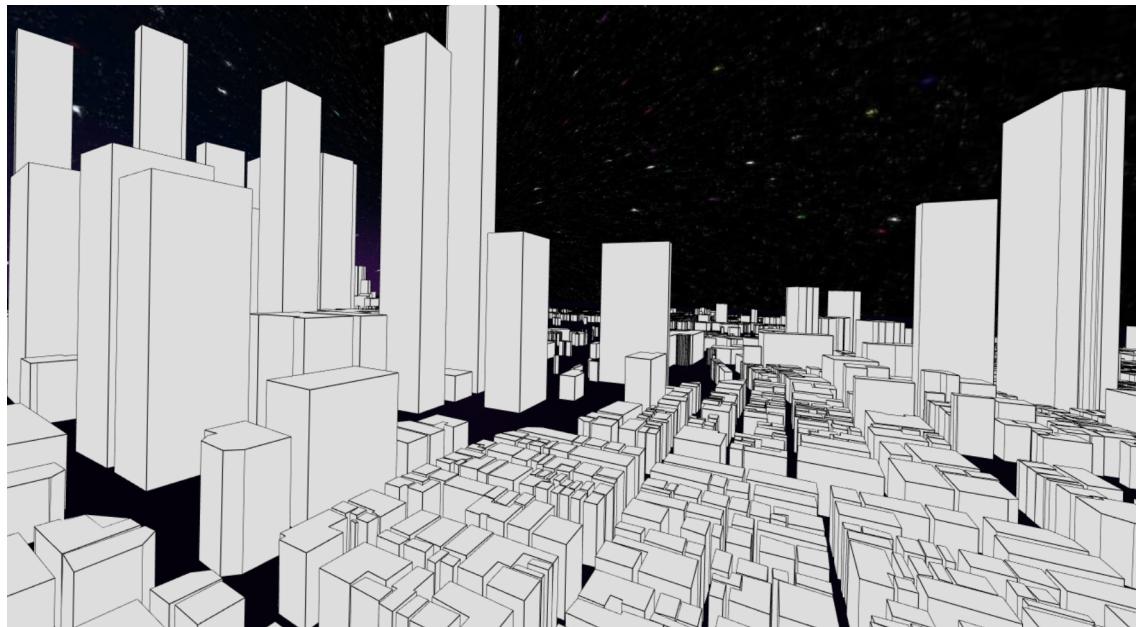


Abbildung 5.3: Beispiel für erstellte Straßen in A-Frame

### 5.2.2 Erstellen der Gebäude

Beim Erstellen der Gebäude wird eine „singleGeometry“ für die Gebäude, eine „singleInnerGeometry“ für die Löcher der Gebäude und eine „singleLineGeometry“ für die schwarzen Kanten der Gebäude als Geometry definiert. Der „vertices“ Liste der singleLineGeometry werden die Punkte von den Linien der Gebäudekanten hinzugefügt. Für jedes Gebäude werden für Dach und Boden zwei Shape-Objekte erstellt. Diese werden einer neu erstellten ShapeGeometry übergeben und zu der „singleGeometry“ zusammengefügt. Sollten Löcher in dem Gebäude vorhanden sein, so wird für jedes Loch ein weiteres Shape mit der Umrandung des Lochs mitsamt der ShapeGeometry erstellt. Diese wird zu der „singleInnerGeometry“ hinzugefügt. Um die Wände der Gebäude zu erstellen, wird ein Shape-Objekt für jedes aufeinanderfolgende Paar von Dach- und Bodenpunkten angefertigt und die mit diesem Shape anschließend erstellte ShapeGeometry der „singleGeometry“ hinzugefügt. Die erstellten Meshs für die „singleGeometries“ werden dann der Szene hinzugefügt. Abbildung 5.4 zeigt Gebäude, die aus der beschriebenen Vorgehensweise resultieren.



**Abbildung 5.4:** Beispiel für erstellte Gebäude in A-Frame

### 5.2.3 Erstellen der Bäume

Bei dem Erstellen der Bäume werden eine „singleTrunkGeometry“ für die Stämme und eine „singleLeavesGeometry“ als Geometry für die Blätter definiert. Für jeden aus der JSON-Datei eingelesenen Baum wird jeweils eine BoxGeometry für den Stamm und eine SphereGeometry für die Baumkrone erstellt. Die Position der erstellten BoxGeometry wird dann auf den „trunkpoint“, der für jeden Baum in der JSON-Datei enthalten ist, gesetzt und mit der Object3D.lookAt Funktion in Richtung des Zentrums der Sphäre rotiert. Ebenso wird die Position der SphereGeometry auf den „leavespoint“, der ebenfalls in der JSON-Datei für jeden Baum vorhanden ist, gesetzt. Um das Material für die beiden „singleGeometries“ zu erstellen, erhält die „singleTrunkGeometry“ eine braune Textur, die die Farbe eines Stammes simuliert. Gleichermaßen erhält die „singleLeavesGeometry“ eine grüne Textur, um die Farbe der Baumkrone darzustellen. Die singleGeometries werden dann zusammen mit dem Material zu einem Mesh kombiniert und der Szene hinzugefügt. Abbildung 5.5 zeigt ein Beispiel für die erstellten Bäume.

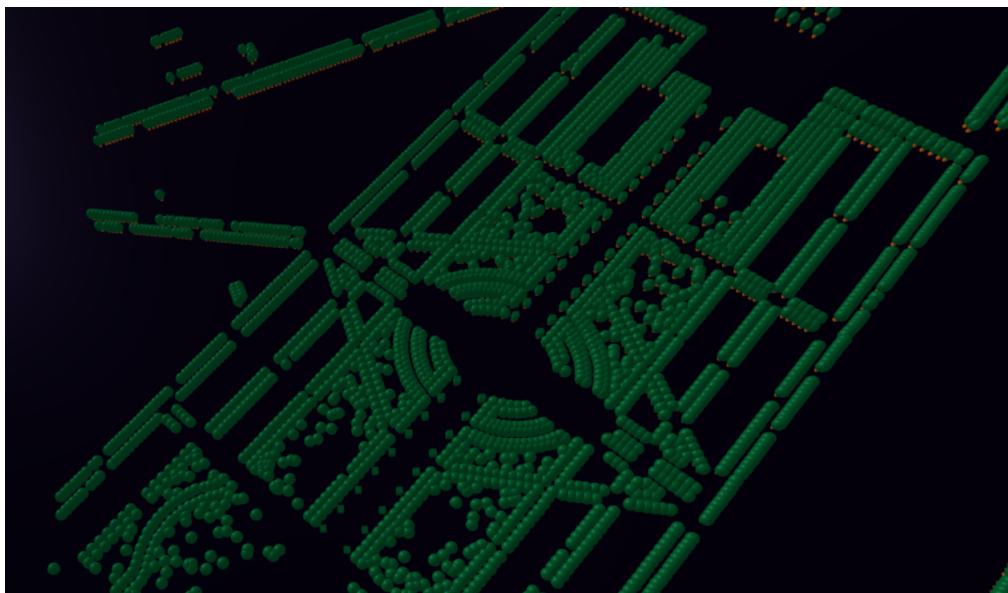


Abbildung 5.5: Beispiel für erstellte Bäume in A-Frame

#### 5.2.4 Erstellen der Naturgebiete

Beim Erstellen der Naturgebiete wird eine „singleGeometry“ als Geometry definiert. Für jedes Naturgebiet in der JSON-Datei wird ein Shape-Objekt erstellt, das die gesamten umrandenden Punkte des Gebietes übergeben bekommt. Es wird eine ShapeGeometry des Shapes erstellt und mit der zugehörigen „singleGeometry“ zusammengefügt. Auch die Naturgebiete sind nach Farbe in der JSON-Datei sortiert. Deshalb wird, wie bei dem Erstellen der Straßen, jeweils eine neue singleGeometry erstellt, falls ein Farbwechsel stattfindet. Dabei wird die alte singleGeometry mit einem Mesh der Szene hinzugefügt und diese zurückgesetzt. Abbildung 5.6 zeigt das Ergebnis von der Erstellung der Naturgebiete.



**Abbildung 5.6:** Beispiel für erstellte Naturgebiete in A-Frame

### 5.2.5 Erstellen der Bahnlinien

Zum Erstellen der Bahnlinien werden für alle Bahnlinien in der JSON-Datei jede zwei aufeinanderfolgenden Punkte der „vertices“ Liste einer erstellten Geometry hinzugefügt. Es wird ein neues Linesegments-Objekt erstellt, das für alle zwei aufeinanderfolgende „vertices“ in der Geometry eine Linie erzeugt. Das für diese Geometry erstellte Mesh wird der Szene hinzugefügt. Abbildung 5.7 zeigt ein Beispiel mehrerer, durch diese Methode erstellter, Bahnlinien.

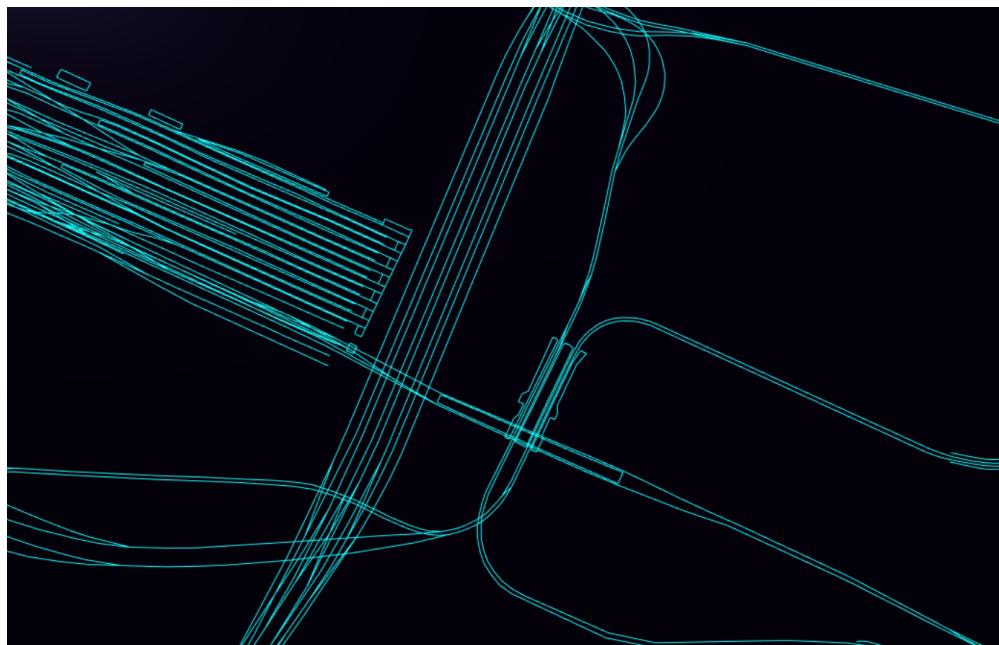


Abbildung 5.7: Beispiel für erstellte Bahnlinien in A-Frame

## 5.3 Benutzte Kameras

Zum Navigieren in der virtuellen Welt gibt es zwei verschiedene Möglichkeiten: die Fly Controls und die Spherical Controls. Um diese zu ändern, kann der Benutzer bei den Einstellungen beim Parsen einer OSM-Datei angeben, ob er die Spherical oder die Fly Controls benutzen möchte. Diese Information wird der JSON-Datei mitgegeben, welche von dem Javascript-Programm ausgewertet und verarbeitet wird. In diesem Abschnitt werden die beiden unterschiedlichen Kamerabewegungs-methoden näher erklärt.

### 5.3.1 Fly Controls

Die Fly Controls sorgen für freie Bewegungen der Kamera in alle Richtungen. Der Benutzer kann also beliebig die Distanz zur Sphäre ändern. Die Fly Controls bestehen aus einer Kombination der „movement-controls“ [McC17] mit dem Parameter „fly = true“, die für die Umpositionierung der Kamera mit den „wasd-Tasten“, sorgen und den „look-controls“, die für die Rotation der Kamera bei Bewegung des Cursors sorgen.

### 5.3.2 Spherical Controls

Für die Spherical Controls wurde eine minimal abgeänderte Version der „Spherical Controls Component“ [Tre18] benutzt. Die Änderung besteht darin, dass die kartesischen Koordinaten bei der „Spherical Controls Component“ anders ausgerechnet wurden, als bei der in dieser Arbeit verwendeten Methode in Abschnitt 3.1.2. Dabei ist es zu Problemen bei der korrekten Kamerapositionierung gekommen. Die Berechnung wurde daher angepasst. Die „Spherical Controls Component“ sorgt dafür, dass die Kamera an die Erdoberfläche angeheftet ist. Dabei ist die Kamera so rotiert, als würde der Benutzer auf der Sphäre stehen.

# 6 Anleitung zur Ausführung des Projektes

Zur Ausführung des Projektes sind einige Installationen sowie Vorbereitungen notwendig.

Um eine OSM-Datei zu erhalten, kann unter der Website  
[www.openstreetmap.org/export](http://www.openstreetmap.org/export)

ein Kartenausschnitt markiert und extrahiert werden, der in der virtuellen Welt angezeigt werden soll. Dabei können jedoch nur Kartenausschnitte von bis zu 50.000 Nodes extrahiert werden. Optional können größere Kartenausschnitte auf der Website

<https://download.geofabrik.de>

heruntergeladen werden. Jedoch kann es sein, dass das Parsen einer großen OSM-XML-Datei den Parser oder zumindest die Generierung der Objekte in der virtuellen Welt überfordert und diese OSM-Datei damit nicht benutzt werden kann.

Die neueste Java-Version sollte installiert sein, um den OSMParse benutzen zu können. Anschließend muss der Benutzer das Terminal aufrufen und den Befehl:

```
java -jar osmparser.jar map.osm
```

eingeben. Dabei ist „osmparser.jar“ die auszuführende Applikation und „map.osm“ die OSM-Datei, die in eine JSON-Datei umgewandelt werden soll. Es können auch mehrere Dateien eingelesen werden, indem dem OSMParse mehr als nur eine Datei übergeben wird. Wenn der Benutzer die Einstellungen editieren will, kann er dem OSMParse als ersten Parameter „-s“ übergeben. Der Benutzer würde dann dazu aufgefordert werden, für jede Objektart „true“ oder „false“ einzugeben, je nachdem, ob er die genannte Objektart in der virtuellen Welt angezeigt haben möchte oder nicht. Außerdem kann der Benutzer anschließend gleichermaßen die anderen Einstellungen bearbeiten. Ist das Parsen der OSM-Daten und das Erstellen der JSON-Datei erfolgreich gewesen, erscheint im selben Ordner, in dem der Parser ausgeführt wurde, eine JSON-Datei.

Ein HTTP-Server muss vorliegen, der die nötigen Skripte dem Browser zur Erstellung der virtuellen Welt zur Verfügung stellt. Es gibt verschiedene Wege einen HTTP-Server zu starten. Im Folgenden werden zwei Möglichkeiten vorgestellt.

## 6 Anleitung zur Ausführung des Projektes

---

Installation via NPM: Dazu muss nodejs installiert sein, welches auf der Website

<https://nodejs.org/de>

heruntergeladen werden kann. Anschließend kann der HTTP-Server installiert werden, indem

```
npm install -g http-server
```

in das Terminal eingegeben wird.

Installation via Python: Wenn Python nicht installiert ist, kann auf der Website

<https://www.python.org/downloads>

die neuste Python-Version heruntergeladen und installiert werden. Dies beinhaltet auch gleichzeitig die Installation des Webservers.

Der HTTP-Server kann dann gestartet werden. Dazu muss das Terminal in dem Ordner, in dem auch die Dateien „index.html“ und „ascript.js“ zu finden sind, aufgerufen werden und

```
bei nodejs: http-server .
```

```
bei python: python -m SimpleHTTPServer 8000
```

eingegeben werden. Nachdem der HTTP-Server gestartet wurde, gibt er die Localhost-Adresse (zum Beispiel „localhost:8000“) an. Sie kann im Browser aufgerufen werden, um die virtuelle Welt zu öffnen.

## 7 Evaluation

In diesem Kapitel wird auf das Ergebnis dieser Arbeit eingegangen. Dabei wird ein Benchmarking der Generierung der JSON-Datei des OSMParser und der Objektgenerierung sowie der Objektdarstellung in A-Frame gezeigt. Anschließend wird auf drei Beispiele der Darstellung der virtuellen Welt eingegangen. Beim Benchmarking werden in allen drei Kategorien zehn unterschiedliche Größen der eingelesenen OSM-Daten skalierend auf der Anzahl der Nodes miteinander verglichen. Dabei erhöht sich die Größe der eingelesenen OSM-Datei immer um ungefähr 50.000 Nodes bis eine Größe von ungefähr 500.000 Nodes erreicht wird. Die Performance ist auch abhängig von den Eigenschaften der benutzten Systeme. Für die Auswertung wurden zwei unterschiedliche Systeme benutzt. Das erste System besitzt dabei wesentlich bessere Systemeigenschaften als das zweite System. Die Systemeigenschaften der Systeme 1 und 2 sind in Tabelle 7.1 und in Tabelle 7.2 aufgelistet.

Komponente	Beschreibung
Betriebssystem	Windows 10 Home, Version 1809
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor
Prozessor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Installierter RAM	16.0 GB
Grafikkarte	NVIDIA Geforce GTX 1070
Maximale Bildfrequenz	120 Hz
Benutzer Browser	Google Chrome

**Tabelle 7.1:** Systemeigenschaften von System 1

Komponente	Beschreibung
Betriebssystem	Ubuntu Linux, Version 18.04
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor
Prozessor	Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
Installierter RAM	8.0 GB
Grafikkarte	Intel HD Graphics 520
Maximale Bildfrequenz	60 Hz
Benutzer Browser	Mozilla Firefox

**Tabelle 7.2:** Systemeigenschaften von System 2

## 7.1 Benchmarking des OSMParsers

Mit der Größe der eingelesenen OSM-Dateien verändert sich die Laufzeit des Parsers. Das Einlesen der OSM-Datei, die Erstellung der Java-Objekte und das Schreiben der JSON-Datei sind die Aspekte, die am meisten Zeit in Anspruch nehmen. Abbildung 7.1 zeigt die Zeit, die der Parser zum Generieren der JSON-Datei benötigt im Verhältnis zur Größe der OSM-Datei. Dabei wird die Zeit von Beginn der Ausführung bis nach dem Erstellen der JSON-Datei gemessen.

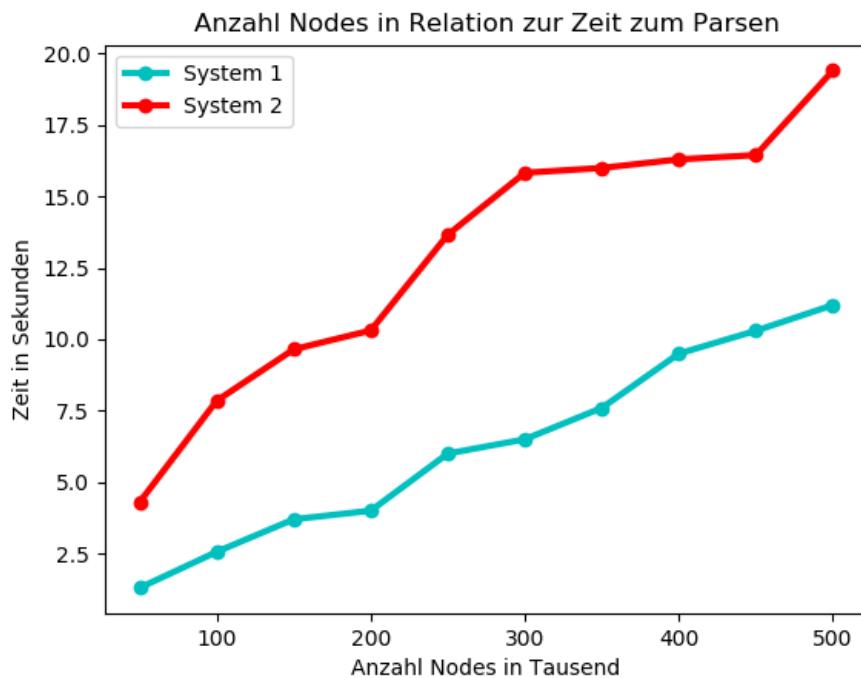


Abbildung 7.1: Benchmarking des Parsers

Es ist deutlich zu erkennen, dass die Zeit bis zur Erstellung der JSON-Datei mit ansteigender Anzahl Nodes bei beiden Systemen steigt. Jedoch ist sie bei beiden gering genug, um kleine OSM-Dateien schnell in JSON-Dateien umzuwandeln.

## 7.2 Benchmarking der Objektgenerierung

Die Objektgenerierung kann einige Zeit in Anspruch nehmen. Wenn sie zu viel Zeit in Anspruch nimmt, reagiert der Internetbrowser nicht mehr, das heißt, dass diese Größe der OSM-Datei für die Generierung nicht möglich ist. Abbildung 7.2 zeigt das Verhältnis zwischen der Größe der OSM-Datei und der benötigten Zeit zum Generieren.

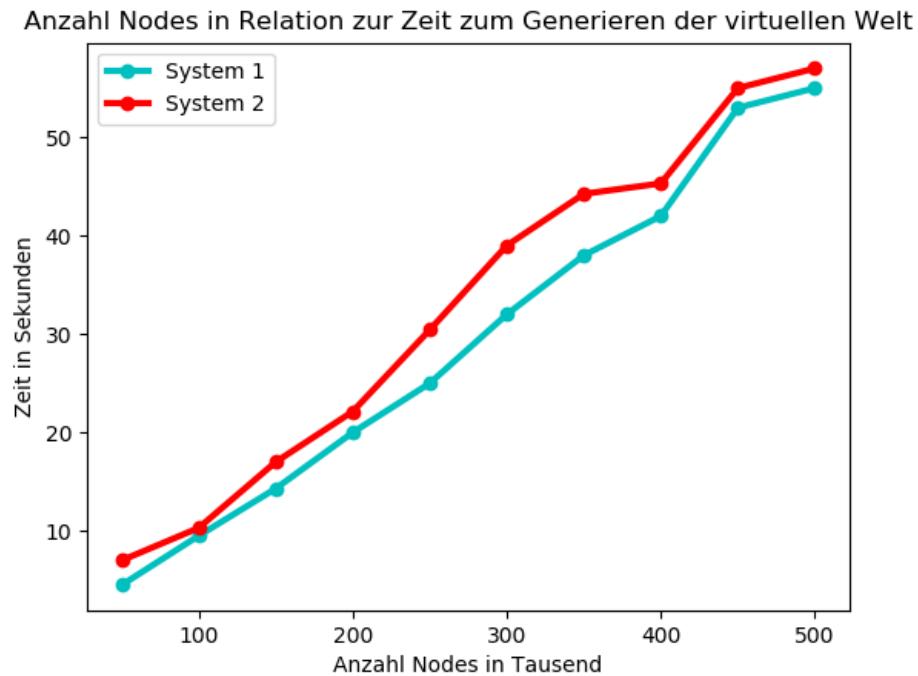
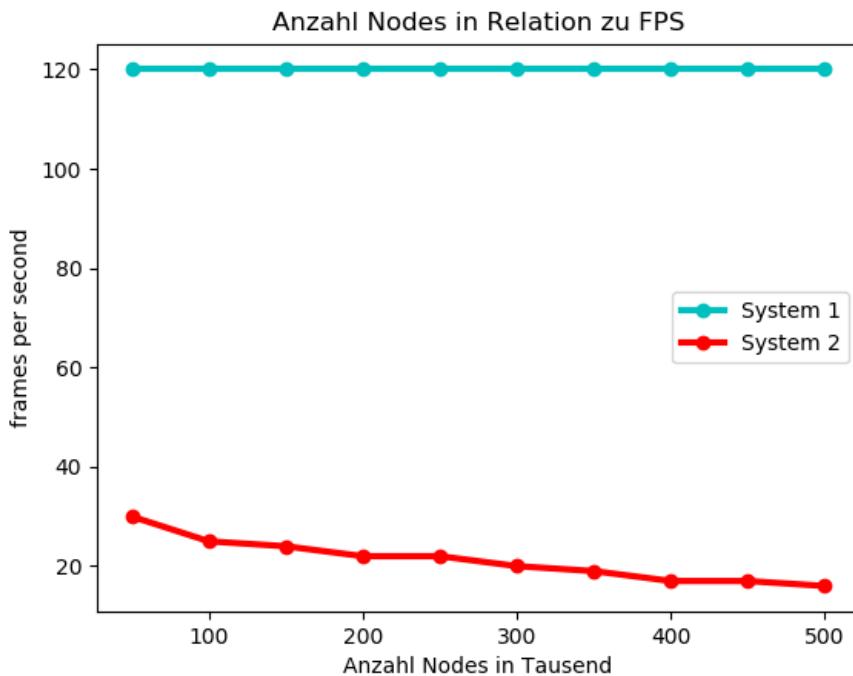


Abbildung 7.2: Benchmarking der Objektgenerierung

Die Grafik zeigt, dass die Generierung der Objekte in der virtuellen Welt deutlich mehr Zeit in Anspruch nimmt, je größer die OSM-Datei wird. Dabei gibt es nur einen kleinen Unterschied in der benötigten Zeit zwischen den beiden Systemen. Das bedeutet, dass die Eigenschaften der Systeme wenig Einfluss auf die Zeit der Generierung haben.

### 7.3 Benchmarking der Darstellung

Auch die FPS in der Darstellung der virtuellen Welt sind abhängig von der Größe der OSM-Datei. Um ein gutes Benutzererlebnis zu gewährleisten, sollte eine Grenze von 30 FPS nicht unterschritten werden. Abbildung 7.3 zeigt das Verhältnis zwischen den FPS und der Anzahl Nodes in der OSM-Datei.



**Abbildung 7.3:** Benchmarking der Darstellung

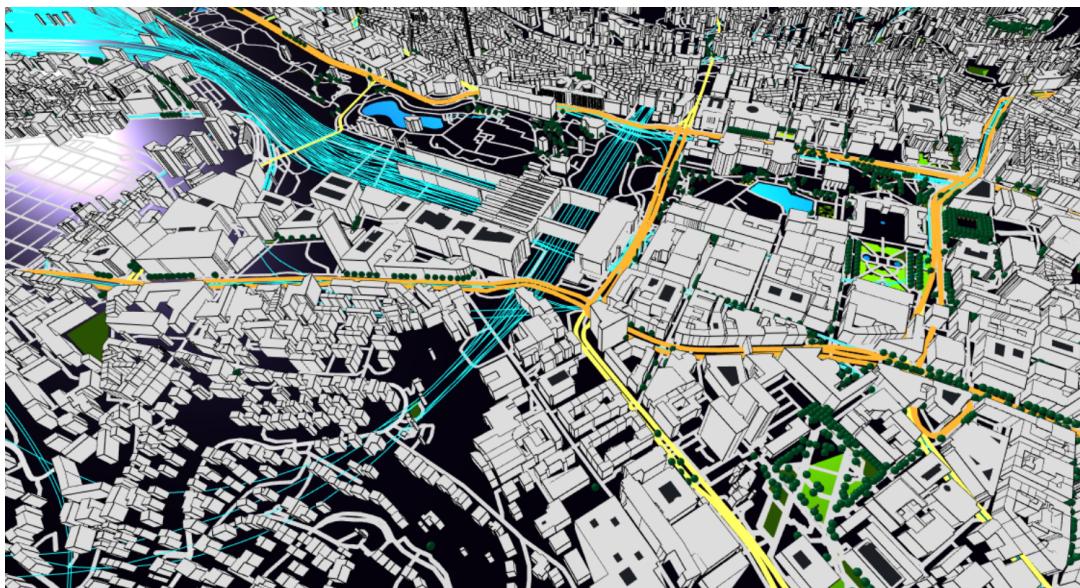
In dieser Grafik ist erkennbar, dass die Performance der Grafikkarten für das Benchmarking ausschlaggebend ist. Die Grafik zeigt, dass mit steigender Anzahl von Nodes die FPS bei System 1 nicht absinken. Das heißt, wenn die Objektgenerierung nicht so viel Zeit in Anspruch nehmen würde, könnten noch viel größere OSM-Dateien benutzt werden, ohne dass die gesetzte Grenze von 30 FPS unterschritten wird. Bei System 2 ist sie jedoch bereits bei Beginn auf 30 FPS, die mit steigender Anzahl Nodes immer weiter abfallen und schließlich bei ungefähr 500.000 Nodes mit 15 FPS enden. Die gesetzte Grenze ist bei System 2 bereits bei ungefähr 50.000 Nodes erreicht. Zusammenfassend kann gesagt werden, dass die Performance der Grafikkarte einen großen Einfluss auf das Benutzererlebnis hat, jedoch sollten die meisten neuen GPUs bei annehmbaren Größen der OSM-Dateien die gesetzte Grenze nicht unterschreiten. Ohne die Verbesserung der Performance durch das Zusammenfügen von Geometries wäre das Benchmarking bei beiden Geräten deutlich schlechter ausgefallen.

## 7.4 Beispiele

Die folgenden drei Beispiele für die Darstellung der virtuellen Welt zeigen das Ergebnis des Projektes. In den gezeigten Kartenausschnitten sind alle Objekttypen vorhanden. Zusätzlich werden auch die Größe der OSM-Datei und die Anzahl der erstellten Objekte aufgelistet.

### 7.4.1 Stuttgart Hauptbahnhof

Das erste Beispiel umfasst die Umgebung von Stuttgart Hauptbahnhof. Abbildung 7.4 zeigt den generierten Kartenausschnitt in A-Frame.



**Abbildung 7.4:** Umgebung Stuttgart Hauptbahnhof

In diesem Kartenausschnitt sind u.a. der Schlossplatz und die Königsstraße von Stuttgart zu sehen. Die meisten Gebäude haben keine Höhendaten in den OSM-Daten angegeben, deshalb wurde für ihre Höhe die Standardhöhe verwendet. Es ist auch deutlich die große Anzahl von über- sowie unterirdischen Bahnlinien, die in dem Hauptbahnhof enden, zu sehen. Zusätzlich lassen sich die beiden Stuttgarter Seen erkennen. Tabelle 7.3 enthält die Elemente des Kartenausschnittes der Umgebung Stuttgart Hauptbahnhof und deren Häufigkeiten.

Element	Anzahl	Element	Anzahl
Nodes	182322	Straßen	12219
Ways	34983	Bäume	4773
Relations	1627	Naturgebiete	683
Gebäude	16542	Bahnlinien	1627

**Tabelle 7.3:** Umfang des Kartenausschnittes Stuttgart Hauptbahnhof

### 7.4.2 Tokio

Im zweiten Beispiel ist ein Kartenausschnitt des Zentrums von Tokio zu sehen. Abbildung 7.5 zeigt den Kartenausschnitt in A-Frame.



**Abbildung 7.5:** Tokio

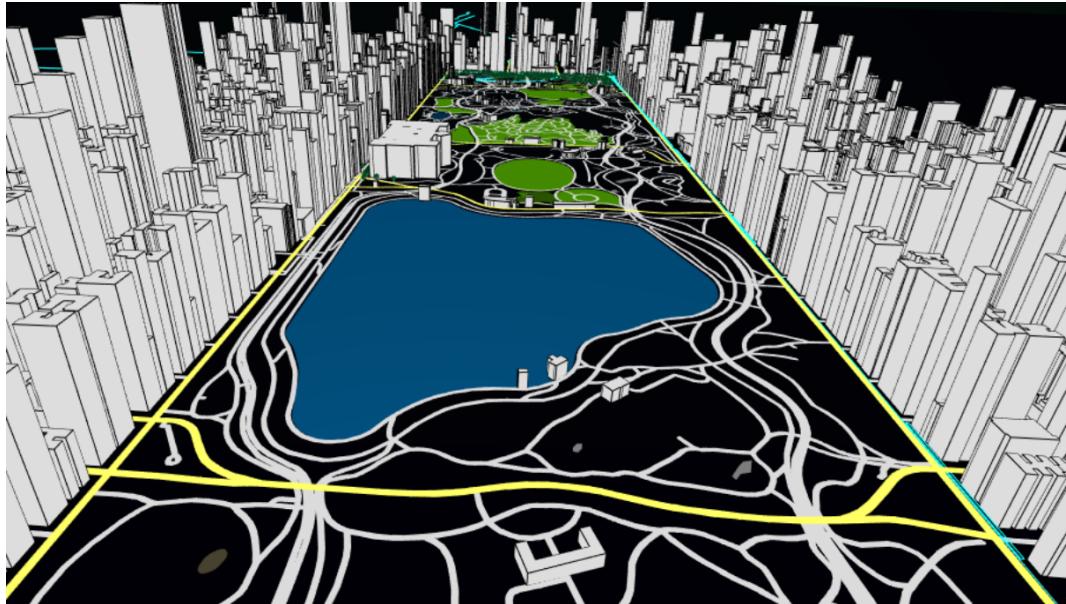
In diesem Kartenausschnitt fallen die Wolkenkratzer auf, die über allen anderen Gebäuden hervorragen. Der anliegende Park im Chiyoda Bezirk hinter den Wolkenkratzern sowie das enge Straßennetz der Stadt sind gut zu erkennen. Darin sind die unterschiedlichen Straßentypen durch ihre Farbe und Breite gekennzeichnet. Zusätzlich ist der Verlauf der Bahnlinien zwischen den Wolkenkratzern in Richtung Tokio Station sichtbar. Tabelle 7.4 enthält die unterschiedlichen Elemente in diesem Kartenausschnitt und deren Häufigkeiten.

Element	Anzahl	Element	Anzahl
Nodes	219093	Straßen	12325
Ways	41885	Bäume	1308
Relations	1903	Naturgebiete	417
Gebäude	23776	Bahnlinien	1133

**Tabelle 7.4:** Umfang des Kartenausschnittes Tokio

### 7.4.3 Manhattan - New York

Das letzte Beispiel zeigt den Central Park in Manhattan - New York. In Abbildung 7.6 ist dieser Kartenausschnitt in A-Frame enthalten.



**Abbildung 7.6:** Manhattan - New York

Bei diesem Kartenausschnitt sind die Unterschiede in den Höhen der Gebäude deutlich sichtbar. Das liegt daran, dass die Benutzer mehr Höhenangaben zu den Gebäuden in den OSM-Daten angegeben haben. Jedoch fehlt bei diesem Kartenausschnitt ein Großteil der Bäume in den OSM-Daten, welche der Szene noch mehr Form verleihen würden. Tabelle 7.5 enthält die unterschiedlichen Elemente in diesem Kartenausschnitt und deren Häufigkeiten.

Element	Anzahl	Element	Anzahl
Nodes	202278	Straßen	5702
Ways	23432	Bäume	857
Relations	292	Naturgebiete	315
Gebäude	13899	Bahnlinien	292

**Tabelle 7.5:** Umfang des Kartenausschnittes Manhattan - New York



# **8 Zusammenfassung und Ausblick**

Dieses Kapitel beinhaltet eine übersichtliche Zusammenfassung des Inhalts der Arbeit, sowie einen Ausblick auf zukünftige Anwendungs-, Erweiterungs- und Verbesserungsmöglichkeiten des Systems.

## **8.1 Zusammenfassung**

Ziel dieser Bachelorarbeit war, automatisiert aus OSM-Daten, die reale Welt in die virtuelle Welt mithilfe von A-Frame zu übertragen. Dabei sollten mindestens die Gebäude, Straßen und die Erdoberfläche sichtbar dargestellt sein und zusätzlich eine ausführliche Beschreibung der Vorgehensweise gegeben werden. Dazu wurden einige andere Systeme betrachtet, die ein ähnliches Ziel besitzen. Diese wurden mit dem Ziel dieser Arbeit verglichen. Es wurde ein Einblick in die Grundlagen des Systems gegeben, insbesondere wurden die OSM-Daten und ihre unterschiedlichen Dateiformate sowie deren Vor- und Nachteile näher betrachtet. Zusätzlich gab es einen Einblick in die zugrundeliegenden Funktionen und Ziele von A-Frame, three.js und JSON. Nach der Behandlung der Grundlagen wurde auf die Implementierung des Parsers eingegangen. Hier wurde insbesondere die Vorgehensweise bei der Extraktion und Behandlung der OSM-Daten und deren Umwandlung in die jeweiligen Objekttypen sowie die Erstellung der JSON-Datei und deren Inhalt ausführlich erklärt. Die Vorgehensweise bei der Generierung der virtuellen Welt mittels A-Frame und three.js wurde anschließend behandelt. Dabei wurde die Erstellung der jeweiligen Objekttypen und die benutzten Kameras gezeigt. Danach wurde eine Anleitung für die Ausführung des Projektes gegeben. Das Benchmarking in der Evaluation hat die effiziente Darstellung der virtuellen Welt sowie die benötigte Zeit des Parsers und der Generierung gezeigt. Die darauf folgenden Beispiele haben das Ergebnis des Projektes veranschaulicht.

## **8.2 Ausblick**

Um die reale Welt noch besser auf die virtuelle Welt abzubilden, gibt es einige Erweiterungsmöglichkeiten. Viele nützliche Informationen der OSM-Daten wurden aus Zeitgründen nicht implementiert. Flüsse, Straßenschilder, Parkbänke, Straßenlaternen und viele weitere Objekte, die in den OSM-Daten verfügbar sind, können der virtuellen Welt noch hinzugefügt werden. Insbesondere kann das System durch viele andere Erdoberflächentexturen ergänzt werden. Außerdem können je nach Gebäudeart auch die Gebäude unterschiedliche Texturen bekommen. Bei manchen sehr komplexen Shapes, die erstellt werden, schlägt die Triangulierungsfunktion von three.js fehl. Das hat zur Folge, dass manche Dach-, Boden- und Naturgebietpolygone eine inkorrekte Form haben. Außerdem wurden die Löcher der Gebäude als schwarze Shapes über den Dächern und unter den Böden erstellt. Eine Verbesserungsmöglichkeit besteht darin, eine andere Herangehensweise an die Erstellung

## 8 Zusammenfassung und Ausblick

der Gebäude zu wählen, um die Triangulierung und die Löcher in allen Gebäuden korrekt zu erstellen.

Eine andere Erweiterungsmöglichkeit besteht darin, größere Karten einzulesen. Dazu muss hauptsächlich die Generierung der Objekte der virtuellen Welt verbessert werden. Dafür kann die Funktionalität eingebaut werden, dass beim Bewegen des Benutzers Objekte außerhalb der Umgebung beziehungsweise des Sichtfeldes ausgeblendet und dafür neue Objekte eingeblendet werden.

Wenn eine realistischere Repräsentation der realen Welt in der virtuellen Welt gewünscht ist, kann zusätzlich der Einfall des Sonnenlichtes als Lichtquelle auf die Sphäre, je nach Uhrzeit, angepasst werden. Weiterhin ist es möglich, Wetterdaten miteinzubeziehen, die für Wettereffekte sorgen können. Außerdem kann durch die Benutzung von Höhendaten, der Erdoberfläche ein Relief verliehen werden.

Nach umfangreichen Anpassungen kann ein derartiges System auch für mehrere Anwendungsgebiete genutzt werden. Eines dieser Anwendungsfelder könnte die Stadtplanung sein. Da das System einen guten Überblick über ein gewünschtes Gebiet liefert, ist es möglich die Infrastrukturpositionierung und Planung durch dieses System zu unterstützen. Ein weiteres Anwendungsfeld ist die Simulation des autonomen Fahrens. Da die Straßennetze in der virtuellen Welt enthalten sind, könnte eine Simulation für Systeme des autonomen Fahrens in sehr vielen Situationen erstellt werden. Außerdem wäre es möglich, die Simulation von Verkehrsaufkommen mit einem derartigen System zu betreiben. Dazu könnten die Häuser als Startpunkte der Verkehrsteilnehmer und die Zentren der großen Städte als Endpunkte genutzt werden. Die Simulation kann anschließend die Konzentration der Verkehrsteilnehmer auf den Straßen bestimmen.

Zusammenfassend kann gesagt werden, dass ein derartiges System viele Erweiterungsmöglichkeiten und Anwendungsbereiche besitzt. Durch die zunehmende Vernetzung der realen Welt mit der virtuellen Welt werden auch in Zukunft viele weitere Anwendungsgebiete dieser Applikation entstehen.

# Literaturverzeichnis

- [Bas15] L. Bassett. *Introduction to JavaScript Object Notation: A to-the-point Guide to JSON*. Ö'Reilly Media, Inc.", 2015 (zitiert auf S. 26).
- [Ben10] J. Bennett. *OpenStreetMap*. Packt Publishing Ltd, 2010 (zitiert auf S. 18).
- [Dir13] J. Dirksen. *Learning Three.js: the JavaScript 3D library for WebGL*. Packt Publishing Ltd, 2013 (zitiert auf S. 22).
- [Kai17] R. Kaiser. *VR Map: Virtual Reality Map (WebVR experiment)*. <https://vrmap.kairo.at>. 2017 (zitiert auf S. 16).
- [Mar18] J. Marsch. *OSMBuildings: 3d building geometry viewer based on OpenStreetMap data*. <https://osmbuildings.org>. 2018 (zitiert auf S. 15).
- [McC17] D. McCurdy. *movement-controls: Collection of locomotion controls, which can switch between input devices as they become active*. <https://github.com/donmccurdy/aframe-extras/tree/master/src/controls>. 2017 (zitiert auf S. 40).
- [Rbr08] Rbrundritt. *Conversion between Spherical and Cartesian Coordinates Systems*. <https://rbrundritt.wordpress.com/2008/10/14/conversion-between-spherical-and-cartesian-coordinates-systems/>. 2008 (zitiert auf S. 17).
- [Tre18] M. Treitler. *spherical-controls-component: Spherical movement controls for A-Frame*. <https://github.com/mattrei/aframe-spherical-controls-component>. 2018 (zitiert auf S. 40).

Alle URLs wurden zuletzt am 11.11.2019 geprüft.



### **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift