Visualization of Radio Networks in a Three Dimensional Environment

Markus Stroot

Bachelor Thesis

November 23, 2016

Examiners

Prof. Dr.-Ing. Marina Petrova
Prof. Dr. Petri Mähönen

Supervisors

Prof. Dr.-Ing. Marina Petrova
Dr. Ljiljana Simić
Dr.-Ing. Janne Riijijärvi

Institute for Networked Systems
RWTH Aachen University

(Markus Stroot)

The present work was submitted to the Institute for Networked Systems

Visualization of Radio Networks in a Three Dimensional Environment

Bachelor Thesis

presented by
Markus Stroot

Prof. Dr.-Ing. Marina Petrova
Prof. Dr. Petri Mähönen

Aachen, November 23, 2016

(Markus Stroot)

Acknowledgements

We thank…

Contents

Abstract

Abstract here.

1

Introduction

In today's world, mobile communication has become extremely popular and highly essential to the social and economical part of our society. Nearly every person living in the industrial countries possesses at least one device capable of wireless communication of some sort. The high density of mobile devices in urban areas has lead to some interesting challenges in the planning and implementation of wireless network infrastructure. On the one hand, the network needs to be able to manage the high data throughput that is created by hundreds or thousands of devices. On the other hand, the network quality needs to be acceptable at any point in the area, because a high throughput is useless if the devices cannot access the network efficiently.

The network coverage problem is especially interesting in urban areas, because of the unique topology. Devices may be mounted atop a very tall skyscraper or far down in the streets, surrounded by buildings. Most of the commonly used wireless communication systems use electromagnetic waves to deliver information, which is no problem on a plane field. Within the rising and falling topology of an urban area however, the propagation of the network gets more difficult to predict. Just like it is with light, obstacles made of different material can absorb, reflect, alter or do nothing to any passing electromagnetic wave. This causes areas of high signal strength, where the antenna has a direct line of sight or a good reflecting path. However, it also crates areas of poor signal strength, for example behind a building (shadow) or in places, where reflections cause too much interference. Over time research institutions have addressed the problem and developed models to predict how network propagation behaves in obscured or confined areas. Many methods however are far to complex to be evaluated by hand for any realistic scenario. Hence, computer supported network simulation emerged. With the help of the computing power of modern computer programs like the WinProp Software Suite [1] are able to forecast the behavior of wireless networks in different environments. Modern simulation applications can predict network propagation for large areas and many different antennas at the same time with decent accuracy. These tools provide a helpful overview over the network coverage, which is especially important for the current trend of infrastructure development.

Currently, the focus is shifting, when it comes to the design of wireless or radio networks in particular. Big macro cells, which provide efficient coverage in rural areas, often struggle in more urban areas, because of the way the buildings interact with the electromagnetic waves. Therefore big cell are being split up, to enhance throughput and directional preferences. Furthermore, newly installed cells are often micro cells, tailored to a specific location in performance and directional properties. This trend however makes it harder to plan where and how to install new infrastructure nodes, in order to satisfy demands. It takes careful analysis of wide range simulations and real life measurements to identify weak points in the coverage and patch them up or to increase signal quality in very demanding areas.

This is where good visualization applications come into play. There are already some tools (e.g. as part of WinProp [1]) that try to help developers by visualizing the signal strength of simulated network scenarios. However, those mainly work in two dimensional space. From a top down perspective the network propagation is shown for a fixed height parameter. This can be helpful for engineers and developers. However, a realistic three dimensional representation of the environment with buildings, maps and the network propagation in between would be even more convenient. Any spacial arrangements and their problems would be easily viewable. For that reason, this thesis tries to first point out the challenges and prerequisites in creating a useful and user-friendly network visualization tool. Thereafter a possible implementation is presented. Finally, the application is analyzed, evaluated and expanded upon.

2

Three Dimensional Visualization and Web Development

As any software project, this visualization tool also relies on many different algorithms and technical standards or conventions. This chapter aims to give an overview of the major external components and ideas, that contribute to the project and the way it works.

2.1 Algorithms

2.1.1 Marching Cubes Algorithm

The marching cubes algorithm was designed in 1987 and published in [2]. The original idea was to create high resolution triangle models of constant density surfaces for medical purposes. However, the algorithm can be applied to any three dimensional scalar data set, not only tissue density. The aim of the algorithm is to find an isosurface within a data set. An isosurface is a two dimensional surface within a three dimensional context, that connects the points in space in which the data set has the same value.[1]

The algorithm uses the "divide and conquer" approach. It divides the whole set into small cubes, which it then iterates or 'marches' through. The cubes are evaluated at the corners and each one gets marked, if it exceeds the isosurface value val_{iso}. That leads to $2^8 = 256$ different ways for a cube to be marked. Based on the marked corners, triangles are inserted within the cube, so they separate the marked corners from the unmarked ones. This is a clever way, because if a corner with a higher value is adjacent to one with a lower value, the isosurface has to intersect the edge somewhere in between. In order to further enhance the accuracy of the surface, a linear interpolation is used, to decide where on each edge of the cube the corners of the triangle(s) should be placed. Using two different symmetries the actual number of triangle arrangements can be reduced to 14. In each of those there is up to four triangles, as can be seen in figure 2.1.

Most implementations of the Marching Cubes Algorithm use lookup tables to store which edges get intersected, based on the marked corners. This makes sense, since an array lookup is a very cheap instruction, when it comes to processing time and an array with 256 entries usually is not too big. Especially in comparison to the number of triangles that is needed for a satisfying reconstruction of any realistic 3D-surface.

[1]Compare contour lines on a two dimensional map.

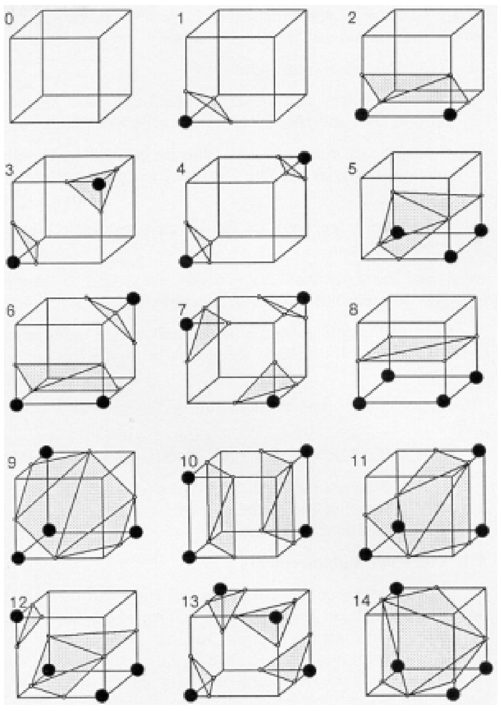

FIGURE 2.1: Tringles arrangements. as seen in [2]

2.2 FILE TYPES

2.2.1 *Shapefile*

A shapefile stores attribute and geometry information. Geometries are represented by shapes consisting of a set of vector coordinates. As it can be seen in its technical description [3] a shapefile actually consists of more than one file. There are at least three parts to every shapefile. A main file, an index file and a dBASE table. The files are all named by the same valid filename. The suffix however distinguishes between the main (.shp), the index (.shx) and the dBase (.dbf) files.

The main file grants direct access to the geometrical information. It consists of a 100 byte header followed by variable-length records. The header contains file management information, like the file code, file length, etc. It also gives information on the data set, like bounding box coordinates and shape type. The records represent the actual shapes. Each record consists of an 8 byte record header and a variable sized list of vertices. The record header simply contains the record number and its length. The length depends on what shape is represented and how many vertices it is made up of. The organization of the main file can be seen in figure 2.2.

The index file also starts with a 100 byte header, which is identical to the main file header. After that, there is a record entry for each of the records in the main file. The i^{th} record entry contains the length of the i^{th} record in the main file and its offset relatively to the beginning of the file.

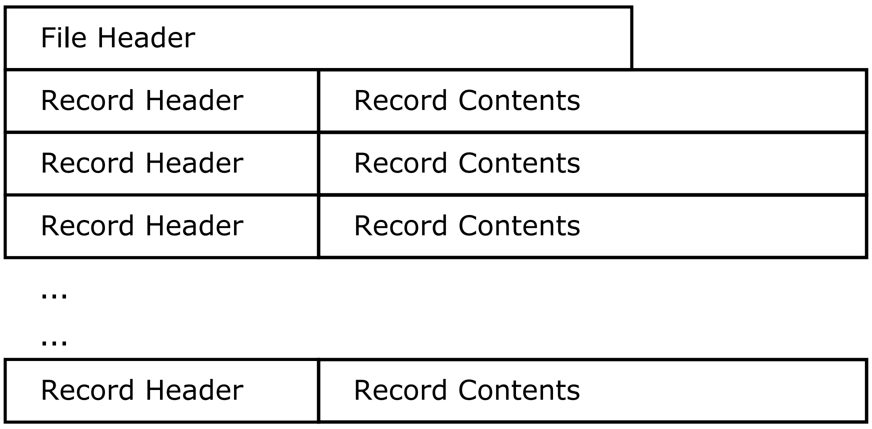

Figure 2.2: Organization of the Main File. as seen in [3]

The dBASE file contains additional attributes concerning the shapes. The format is a standard DBF file, that is used by many applications with only a few extra requirements. The record order for example has to be the same as the order of the shapes in the main file.

2.2.2 *ODA files*

The ODA file format is a simple way to to store geometrical information about urban outdoor environments. This format is especially interesting, because it is used by the WallMan [4] application in the WinProp wireless network planning software package [1]. It is a simple ASCII format.

The file starts with a header of six lines, consisting of five lines of comment and one line of general database settings. The body of the file consists of two blocks. The first block contains some material information, which is referred to in the second block. That block holds the actual outdoor building data. All buildings are represented by an arbitrary number of 2D-coordinates, outlining the base shape, and one height parameter.

2.2.3 *APA file*

The APA file format is an ASCII based format used by the AMan [5] application, which is also part of the WinProp wireless network planing software package [1]. It stores three dimensional antenna gain patterns. APA files mostly consist of data triples. There can only be one triple per line and it must be made up of two angular values (horizontal and vertical) and one gain value (in dB).

2.2.4 *CSV file*

The CSV [6] file format is a widely used, ASCII based, format. It is able to represent any kind of tabular data. Different characters can be used to separate the data values.

The line break character usually signifies the end of a data record. Within each record there can be multiple data fields, separated by a special character[2].

2.3 RENDERING AND OUTPUT

2.3.1 Rendering Basics

2.3.2 Three.js WebGL Framework

Three.js is a JavaScript based API. It uses WebGL [7]. Therefore it provides the opportunity to use hardware accelerated 3D-graphics inside of HTML5 browsers, making it platform independent.

The framework was first published in 2010 by Ricardo Cabello, also called "mr-doob" online. [8] He started with 3D modeling and editing together with other programmers. When he felt, that the tools he used to create his animation scenes were not satisfying his needs, he started do develop his own framework. When finally JavaScript and WebGL support started to get better, he ported this project from ActionScript to JavaScript an published it on GitHub[3]. Since then many contributors are still taking part in the development of that project.

The aim of the framework is to abstract the work and theoretical calculations, that come up when a three dimensional scene is rendered to a two dimensional screen. It provides developers with classes and methods that are convenient for the fast and easy creation of three dimensional scenes. Implemented are many usefull features [9] like:

Different Renderes are available, including WebGl, Canvas and SVG renderer

Scenes can be edited at run-time

Cameras and Controllers in many different varieties.

Lighting can be added to a scene like any object; Shadows are calculated internally

Materials with different shadow and texture options are provided.

Geometries both custom made and predefined (cube, sphere, torus, ...)

Loaders for images, JSON objects and more

Examples are provided on nearly every functionality the framework provides

All this lets the developers working with three.js focus more on designing an creating the scene, rather than on the difficulties of displaying it on the computer screen. However, should a special case arise and the developer needs to work on a more basic level, that is no problem. Three.js can incorporate any self-written shaders and posses a variety of utility math functions, like matrix and vector calculation and projection.

Listing 2.1 shows an easy example on how to use three.js in a JavaScript document. The code creates a Scene, a PerspectiveCamera, and a WebGLRenderer object. The

[2]The comma is usually used for that, but semicolon, colon, tab or space are alternatives.

[3]available on: https://www.github.com/mrdoob/three.js

renderer is then added to the body of the HTML page. Afterwards the predefined BoxGeometry and MeshBasicMaterial classes are used to create the mesh of a simple 1x1x1 cube. Finally, it is added to the scene and the camera is moved. Now the rendering loop is started, which slowly rotates the cube around the x and y axis. This example shows how a relatively complex animation can easily be implemented with just a few lines of code.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.
    innerHeight, 0.1, 1000 );

var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial({
  color: 0x00ff00
  });
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );
camera.position.z = 5;

function render() {
  requestAnimationFrame( render );

  cube.rotation.x += 0.1;
  cube.rotation.y += 0.1;

  renderer.render( scene, camera );
}
render();
```

LISTING 2.1: Three.js Example

2.3.3 *Possible Renderman Pipline (?)*

3

System Architecture

3.1 System Requirements

3.1.1 *Hardware Requirements*

One of the most important hardware elements for the three dimensional visualization is the graphics card[1]. Depending on the size and level of detail of the scene there are a lot of computations that need to be done. Furthermore, the scene is supposed to be movable, so it cannot be a prerendered image. That makes some dedicated graphics hardware nearly indispensable. However, since the application renders simple polygon meshes and point clouds the graphics hardware does not have to be a high end device. It merely needs to relieve the CPU of some work. And with its processing unit made especially for matrix and vector calculation, any modern day graphics card that supports WebGL will meet the expectations.

CPU-only rendering is of course possible, too. The large amount of of points and surfaces in a complex scene however seriously slows down the rendering on an all purpose CPU. This also slows down the whole system, because of all the graphics calculations that are blocking the CPU. For a more detailed analysis see Section **??**.

Another important hardware requirement is the RAM. Visualization data sets, especially from simulation applications, can get large very fast. City wide simulations with multiple antennas and a resolution of a few meters can easily have a few million data points. While the data that is being visualized might not take up much space as a file on the hard drive, within the application that might change. After being loaded and parsed, the data is stored within convenient data structures like arrays or objects. This leads to a less efficient compression of the data and also the addressing schemes and object methods add additional overhead. All that together results in an application that needs to load big chunks of data into the RAM. Therefor it is important that the system has enough memory at its disposal.

3.1.2 *Software Requirements*

Since the main routine of the application, the rendering loop, runs in a web browser as JavaScript code, it is important to have a browser, that supports all the used functionality. Firstly, and most importantly, the renderer used here is a so called ″webGL-rendererÂ´Â´, so it is important that the browser even supports WebGL. All current versions of the commonly used ones (e.g. Chrome, Firefox, Safari, ...) are able to support HTML5, CSS3 and WebGL. It is advisable to use the most recent update of the browser, because the developers always improve the performance or fix bugs. Also the support of WebGL grew over time, so very old versions may not support all

[1]In some cases the hardware acceleration is disabled. See Section 3.1.2 for further information.

the features used in this application. For lesser known browsers the functionality has to be determined individually. Important criteria are the former mentioned HTML5, CSS3 and WebGL support. It is also worth mentioning that the browser needs to allow WebGL to use hardware accelerated rendering, in other word access the graphics card. Microsoft's Internet Explorer for example does not do that. This leads to the problem, that it uses CPU-only rendering even if a graphics card is installed, which in turn, slows down the whole system.

3.2 ARCHITECTURAL DESIGN

In this section I describe the evolution of the project, starting with the implementation of the core functionality. Afterwards, I continue to explain the split between the rendering and the data management.

3.2.1 Initial JavaScript code

After choosing the Three.js to be the framework for the graphics calculations, I made the first steps towards a full application within a simple JavaScript document. Using a very basic HTML document that just executes the script, the work started.

It was clear from the beginning, that some form of file system access was needed. First, the application used the simple HTML5 <*input*>-tag with the file-type attribute. That way, for debugging purposes, files could be chosen freely by the developer an were displayed immediately afterwards. The application parsed the files, saved the processed contents into global variables and set a flag variable. The rendering loop checked the flags in every run and loaded any new data. For that, there are special routines corresponding to the type of data. They process and convert it into different objects, that can be displayed by Three.js.

This approach of handling the data turned out to be helpful in debugging the core functionality. However, when it comes to real usability it has more disadvantages than advantages. While it is more user-friendly to just have to open a simple HTML document, the way of personally choosing the files to be visualized every time one restarts the application is very unhandy. Furthermore, any further file system access is impossible, since JavaScript is not allowed to simply do that for obvious security reasons. Moreover, the reading an processing of the data takes some time for big datasets, which causes the GUI to freeze until the data routine is finished. Since there is no easy way to implement multithreading in JavaScript, there is no easy solution for this problem.

All this reasons led to the conclusion, that a JavaScript only code cannot stisfy the demands I put towards the application. Therefore, I decided to devide the functionality and create a local web app.

3.2.2 Frontend – Backend

In order to make the GUI work more smoothly I needed to devide between the graphical display and the data aquisition. Everything concerning the graphics stayed in JavaScript code, since that is where WebGL and Three.js can be used. The file system access and rudimental data processing however could be outsourced into a small web server backend.

The frontend stayed the same for the most part. Only the functions and classes that handled the different input files had to be updated. Those now used ajax requests provided by the JQuery [**?**] library. Ajax requests are a way to send a request to the corresponding webserver without having to reload the whole page. In other words, the application is able to load new data in the background, while keeping on rendering the present scene.

The backend did not exist until now, so it had to be designed from the begining. For conveinience this application uses an ASP.Net server backend whis is written in C#. This server backend architecture was designed by Microsoft and therefore the Microsoft Visual Studio IDE provides substantial support and many templates for it. Since this thesis and the application are not about web development, this was a easy and quick way to implement the intended functionaslity. However, the backend is not very big or demanding so if another implementation (e.g. in Ruby) should proof to be more convenient in the future, a transition would be reatively easy.

The design of the backend recreates part of a widely known web programming conventions called MVC. In the MVC model the server uses thre different templates to represent differnet abstractions. The models encapsulate different datastructures. They provide a general representation for each type of data, which in many cases come from databanks. The view is a visual representation of the result. It gets delivered to the browser and can be displayed. Finally the Controller is the core of the server. A Controller gets called if the respective request comes in. The controller uses the data in the form of models to create a view, that gets delivered back. Since this application generates its own visual representation the "view" -part can be neglected. Furthermore, there is no data that would call for the use of a database, so the models are simple classes, mimicing the datastructures that the frontend expects. So all in all, when a request comes in from the frontend, it gets routed to a Controller, depending on its URL. The controller then reads the files in question and insert the contents into the right models, which in turn get delivered back to the frontend.

4

Implementation

4.1 Datastructures

Whithin the implementation the data needs to be represented in a way, that is convenient for the algorithms to work on. In this section I will present some of the special datastructures.

4.1.1 Four Dimensional Arrays

The four dimensional array in itself is not an exeptional datastructure, but it has a special meaning in this application. The data delivered by simulation application usually comes as values that can be indexed by the three spacial definitions that define its location.Furthermore, in many simulation circumstances there is more than one antenna, so you get multiple power values for each point in space. Hence, the four dimensional array is a good was to represent this data. In this application, the first index represents the number of the antenna or base station, while the other three indices represent the three spacial directions. Of course, this only works for evenly spaced positions, but since the marching cubes algorithm works best on evenly spaced data points, this restriction is sensible either way.

4.1.2 Data Point Object

The other way to represent data is used for real life measurement data. Here you cannot expect the points to be evenly spaced. They mostly follow steets, or other easly accessible places. Therefore we need exact information on the location of each point. This is realized by a very simple point-object in JavaScript. It has only three members, the x coordinate or ?? and the y coordinate or ??. This is usefull for the inverse distance interpolation used for the real data, because you can easily calculate the distance to any point in space.

A very similar object is used to represent the antenna patterns. Those patterns consist of a horizontal angle, a vertical angle and a amplification for the corresponding direction. Hence, in this case, two of the members represent a direction rather than a point and the value equals the amplification.

4.2 Important algorithms and routines

This section describes some classes or functions in either backend or frontend code that implement the core functionality of the application.

4.2.1 The Ajax Loader

The Loader-class is a Javascript object. It encapsulates four methods, that provide requests for the four different types of data, that are needed in this application. Namely those are building data, measurement data, simulation data and antenna patterns.

Listing ?? shows the examplary implementation of the building loader method. Forming a request is very straight foreward. The $-identifier grants access to the methods provided by the JQuery library. There the ajax method abstracts a generic server request. In the curly braces afterwards, one simply has to specify the options. The 'url' option sets the request url. This is important because it determines which controller the request is mapped to. The 'type' option tells the server what kind of request is send. Requests to retrieve data usually use the 'GET' method. Finally, the 'datatype' option tells the server, what kind of response is expected. Since this application mostly requests objects, it expects a response that contains a JSON object.

4.2.2 Filling Simulation data

4.2.3 The Marching Cubes Algorithm

4.2.4 Inverse Distance Interpolation

4.3 DESCRIPTION OF IMPORTANT PROCEDURES

4.3.1 GUI

4.3.2 Registering and Loading Data

4.3.3 Displaying Measurement Data

4.3.4 Displaying Simulation Data

5

Evaluation

A

Things That Did Not Fit Elsewhere

The appendix is the place to put auxiliary figures, background information, etc. that did not fit into the main part of the thesis.

B

Abbreviations

ASCII American Standard Code for Information Interchange

ODA Outdoor ASCII

APA Antenna Pattern ASCII

CSV Comma Separated Values

HTML Hypertext Markup Language

RAM Random Access Memory

GUI Graphical User Interface

URL Uniform Resource Locator

MVC Model View Controller

List of Tables

List of Figures

BIBLIOGRAPHY

[1] *WinProp, Documentation, Propagation Models and Background Information*, AWE Communications GmbH, May 2002, version 5.43. [Online]. Available: http://www.awe-communications.com/Docs/WinProp_Documentation.pdf

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, Jul. 1987.

[3] *ESRI Shapefile Technical Description*, Environmental Systems Research Institute, Jul. 1998.

[4] *WallMan, User Reference Guide*, AWE Communications GmbH, Feb. 2014. [Online]. Available: http://www.awe-communications.com/Download/Manuals/WallManManual.pdf

[5] *AMan, User Reference Guide*, AWE Communications GmbH, Apr. 2014. [Online]. Available: http://www.awe-communications.com/Download/Manuals/AManManual.pdf

[6] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, SolidMatrix Technologies, Inc., Oct. 2005. [Online]. Available: https://tools.ietf.org/html/rfc4180

[7] *Khronos Releases Final WebGL 1.0 Specification to Bring Accelerated 3D Graphics to the Web without Plug-ins*, Kronos Group, viewed on: 05.10.2016. [Online]. Available: https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification

[8] pnaylor and mrdoob et. al., "Notes on the creation of three.js," viewed on: 05.10.2016. [Online]. Available: https://github.com/mrdoob/three.js/issues/1960

[9] various authors, "Three.js features," viewed on: 05.10.2016. [Online]. Available: https://github.com/mrdoob/three.js/wiki/Features

Eidesstattliche Versicherung

Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift