Stefan Türtscher, BSc

Creation of a toolbox for aeronautical obstacle regulations

MASTER'S THESIS

To achieve the university degree of
Diplom-Ingenieur
Master's degree program: Mechanical Engineering

submitted to

Graz University of Technology

Supervisor
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn Reinhard Braunstingl
Institute of Mechanics

Graz, March 2017

Acknowledgement

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

…….. ……..

Date Signature

Abstract

The objective of this master thesis is the development of a software toolbox for aeronautical obstacle regulations. This toolbox should provide all necessary functions for modelling given areas by different national and international regulations, gathering given aeronautical data and an automatically processing of this input data into obstacle collection surfaces. The modelling of the different regulations should be implement with a script.

TABLE OF CONTENTS

1 Definitions, descriptions and shortcuts

Aerodrome or **airport**. A defined area on land or water (including any buildings, installations and equipment) intended to be used either wholly or in part for the arrival, departure and surface movement of aircraft [1].

Aeronautical data. A representation of aeronautical facts, concepts or instructions in a formalized manner suitable for communication, interpretation or processing [1].

AIXM. Aeronautical Information Exchange Model

BMVIT. Austrian Ministry for Transport, Innovation and Technology

CFD. Computational Fluid Dynamics - a branch of fluid mechanics that uses numerical analysis and data structures to solve and analyze problems.

DB or Database. One or more files of data so structured that appropriate applications may draw from the files and update them [1].

eTOD. Electronic Terrain and Obstacle Database

Eurocontrol. European Organization for the Safety of Air Navigation

FEM. Finite Element Method – numerical method for solving problems of engineering and mathematical physics

FIR. Flight information region

Google Earth. Is a virtual globe, map and geographical information program

GML. Geography Markup Language

ICAO. International Civil Aviation Organization

JS. JavaScript - interpreted programming language

JSON. JavaScript Object Notation – file format

KML. Keyhole Markup Language – file format

Obstacle. All fixed (whether temporary or permanent) and mobile objects, or parts thereof, that: a) are located on an area intended for the surface movement of aircraft; or b) extend above a defined surface intended to protect aircraft in flight; or c) stand outside those defined surfaces and that have been assessed as being a hazard to air navigation [1].

OFM. Open Flightmaps

SRTM. Shuttle Radar Topography Mission

TMA. Terminal Control Area also known as a Terminal Maneuvering Area

XML. Extensible Markup Language – file format

2 Introduction

The purpose of this thesis has been to create a toolbox which provides all necessary functions to reproduce national and international regulations for obstacles in aviation.

Obstacles are a major safety issue in the general aviation. Therefore, it is very essential to have accurate data about the obstacles in a given area. Obstacles are classified and collected depending most important on their position and height under given national and international regulations. In general, and especially ICAO, does not provide a definition of what constitutes an obstacle, rather there is a number of surfaces that must either not be penetrated or, if an object does penetrate the surface, must be recorded as an obstacle. These regulations apply primarily in the area close to airports and are also dependent on local airport specifics. Furthermore, the terrain and airspaces are in most cases relevant to the regulations. Summarized, the regulations are defining a general rule set which is subject to local specifics.

The goal is to combine the general rule set with local specifics to calculate a collection surface, which can then be used to determine if a given object is an obstacle in the sense of the applied rule set.

On the basis of eTOD, which is a project introduced by ICAO for the purpose of providing electronic data of obstacles and terrain under the ICAO regulations, this master thesis aims to solve the problematic of the object validation. Because there are national and international regulations, this thesis is set up not only to solve the particularly ICAO problem, but providing a general software toolbox for various regulations. It is therefore essential to develop a toolbox, which is able to provide all necessary functions for modelling the different rule sets in an easy way. As the regulations are tied to local specifics, it is important to have accesses to the relevant aeronautical data in a way that allows general modelling of regulations.

3 State of the art

The whole eTOD project is a massive undertaking for each state, so that Eurocontrol published a basic manual which provides a general guidance for the project [2].

As this is an international regulation and all countries of the Europe Union are forced to provide electronic obstacle and terrain data according to the ICAO regulations, it is very likely that numerous teams are working on this project at the same time, but because this project is still in progress, there are no working solutions on the market yet.

The Austrian BMVIT started back in 2013 with the kick-off meeting for this project. The project is still in progress without any major achievements. Because the protocols of the meetings are available for the public and Eurocontrol provides a service witch shows the eTOD project progress of each member state in Europe Union, it is easy to follow the development of the project. Important to understand is that this thesis only aims to solve a little part of the whole eTOD project.

The difference of this thesis to the other elaborations lies in three points. First, the purpose of this thesis is not only to solve the requirements of ICAO, but to provide a generic solution for dozens of regulations. Second, it is designed to be cross border capable, which is essential in aviation especially for airports close to country borders. Third, this project has been developed in cooperation with OFM, which provides the necessary aeronautical data in a format that allows cross border calculations.

In preparation to this thesis, experts have been consulted to discuss the difficulties, which derive from this topic and also to get advice concerning hidden problematics. Additionally to this valuable information, experts pointed out, that this problematic dose not only affect obstacles but also drones. Similar to regulations for obstacle there are regulations for drones. As experts explained, the regulations for drones are very complex and more constrictive as most of the people would guess. Furthermore, there is no useable information for drone pilots available, which addresses this issue. This underlined even more the strategy of creating a generic toolbox for all kind of different regulations and purposes.

4 Basic principles

This section explains the essential basics on which this thesis is built on. It also should give a little understanding of the difficulties which derive of this subject and the necessary steps to solve these.

4.1 Coordinate systems

To describe a point in space it is necessary to know in which coordinate system the description relates to. In this thesis are mainly two systems used the Cartesian system and the spherical system. Depending on the kind of calculation one of each system usually benefits the calculation better, so that it is also essential to transform the information from one system to the other system and vice-versa. The basics of these coordinate systems and the transformations are defined in many text books (e.g. [3], [4]).

4.1.1 Cartesian coordinate system

The Cartesian coordinate system is an orthogonal coordinate system which consists of an ordered triplet of axes that are pair-wise perpendicular in three dimensions, have a single unit of length for all three axes and have an orientation for each axis.

Rotation matrix:
A linear map always maps linear subspaces onto linear subspaces. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are finite-dimensional vector spaces and a basis is defined for each vector space, then every linear map from $\boldsymbol{A}$ to $\boldsymbol{B}$ can be represented by

$$\boldsymbol{y} = \boldsymbol{M}_B^A \cdot \boldsymbol{x}\,. \tag{1}$$

A special form of this linear mapping is the linear transformation matrix that is used to perform a rotation in Euclidean space. There are three basic rotations about each of the axes of a coordinate system. The rotation about x-axis by the angle of θ is described by

$$\boldsymbol{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \tag{2}$$

about the y-axis by

$$\boldsymbol{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{3}$$

and about the z-axis by

$$\boldsymbol{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

4.1.2 Spherical coordinate system

As shown in Figure 1, the spherical coordinate system is defined by an origin point O in space and two orthogonal directions called the zenith (z-axis) and the azimuth reference (x-axis).

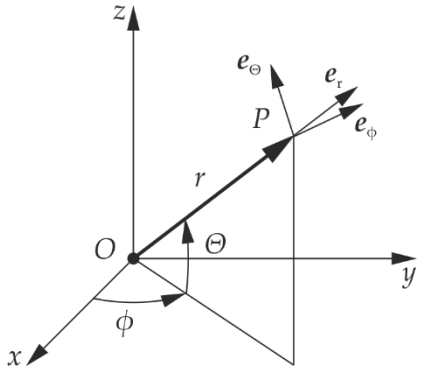

Figure 1: Spherical coordinates

These three definitions determine a plane which contains the origin point and is perpendicular to the zenith. The three coordinates are in the geographic system defined as radius r which describes the distance from the origin O to a point P, the polar angle θ, which is the angle between the perpendicular plane to the zenith and the ray $\overline{OP}$ and the azimuthal angle φ which is the angle between the plane consisting of zenith and azimuth and the ray $\overline{OP}$.

The spherical coordinates may be converted into Cartesian coordinates by

$$x = r\cos\theta\cos\varphi\,, \tag{5}$$

$$y = r\cos\theta\sin\varphi\,, \tag{6}$$

$$z = r\sin\theta\,. \tag{7}$$

The spherical coordinates may be retrieved by

$$r = \sqrt{x^2 + y^2 + z^2}\,, \tag{8}$$

$$\theta = \sin^{-1}\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right) = \sin^{-1}\left(\frac{z}{r}\right)\,, \tag{9}$$

$$\varphi = \begin{cases} \tan^{-1}\left(\frac{y}{x}\right), & x > 0 \\ \operatorname{sgn}(y)\frac{\pi}{2}, & x = 0 \\ \tan^{-1}\left(\frac{y}{x}\right) + \pi\,, & x < 0 \wedge y \geq 0 \\ \tan^{-1}\left(\frac{y}{x}\right) - \pi\,, & x < 0 \wedge y < 0\,. \end{cases} \tag{10}$$

The basis vector of each coordinate describes the direction of the changed position if its coordinate changes by an infinitesimal value. This yields for the basis vector $\boldsymbol{e}_\varphi$ to

$$\boldsymbol{e}_\varphi \sim \frac{\partial P}{\partial \varphi} \tag{11}$$

and further to

$$\begin{aligned} \frac{\partial P}{\partial \varphi} &= \frac{\partial x}{\partial \varphi}\boldsymbol{e}_x + \frac{\partial y}{\partial \varphi}\boldsymbol{e}_y + \frac{\partial z}{\partial \varphi}\boldsymbol{e}_z \\ &= -r\cos\theta\sin\varphi \cdot \boldsymbol{e}_x + r\cos\theta\cos\varphi \cdot \boldsymbol{e}_y\ . \end{aligned} \tag{12}$$

To get the basis vector, the length must be 1, so that

$$\boldsymbol{e}_\varphi = -\sin\varphi \cdot \boldsymbol{e}_x + \cos\varphi \cdot \boldsymbol{e}_y\,. \tag{13}$$

Same is true for the other two directions:

$$\boldsymbol{e}_\theta = -\sin\theta\sin\varphi \cdot \boldsymbol{e}_x - \sin\theta\cos\varphi \cdot \boldsymbol{e}_y + \cos\theta \cdot \boldsymbol{e}_z\,, \tag{14}$$

$$\boldsymbol{e}_r = \cos\theta\cos\varphi \cdot \boldsymbol{e}_x + \cos\theta\sin\varphi \cdot \boldsymbol{e}_y + \sin\theta \cdot \boldsymbol{e}_z\,. \tag{15}$$

The three basis vectors

$$\boldsymbol{e}_r = \begin{pmatrix} \cos\theta\cos\varphi \\ \cos\theta\sin\varphi \\ \sin\theta \end{pmatrix},\ \boldsymbol{e}_\theta = \begin{pmatrix} -\sin\theta\sin\varphi \\ -\sin\theta\cos\varphi \\ \cos\theta \end{pmatrix} \text{ and } \boldsymbol{e}_\varphi = \begin{pmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{pmatrix} \tag{16}$$

are describing the rotation matrix

$$\boldsymbol{S} = \begin{pmatrix} \cos\theta\cos\varphi & -\sin\theta\sin\varphi & -\sin\varphi \\ \cos\theta\sin\varphi & -\sin\theta\cos\varphi & \cos\varphi \\ \sin\theta & \cos\theta & 0 \end{pmatrix}. \tag{17}$$

The rotation matrix can be used to describe the transformation from

$$(\boldsymbol{e}_r, \boldsymbol{e}_\theta, \boldsymbol{e}_\varphi) = \boldsymbol{S} \cdot (\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z) \tag{18}$$

and vice-versa by

$$(\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z) = \boldsymbol{S^{-1}} \cdot (\boldsymbol{e}_r, \boldsymbol{e}_\theta, \boldsymbol{e}_\varphi)\,. \tag{19}$$

4.2 Great circle and small circle

A great circle is the biggest possible circle on a sphere. It is the result of an intersection of a sphere with any plane that passes through the center point of the sphere. It is also the shortest connection of two points on a sphere. A small circle results of an intersection of a sphere with any plane that does not pass through the center point of the sphere as shown in Figure 2.

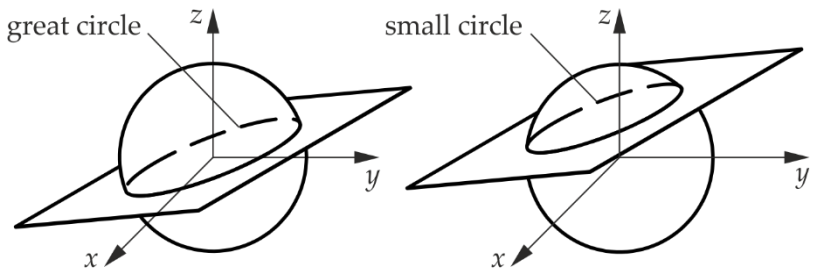

Figure 2: Great and small circle

Every connection of two points in this thesis, if not different stated, is a great circle.

4.2.1 Calculation of the bearing

The bearing, which is used in this thesis, is defined by the angle between a ray in direction of the true North[1] whose starts in the point of observing and a ray starting also in the point of observing to the point the bearing should head to.

[1] The true North is defined by the geographic system, whereat the magnetic North refers to the point where a magnetic compass would point to. The two points are not on the same spot especially as the magnetic North changes its position over time.

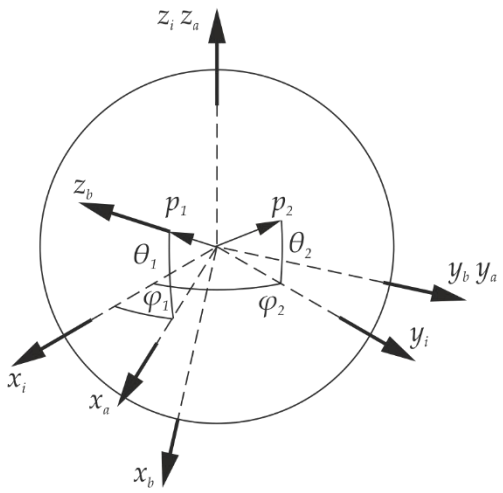

Figure 3: Coordinate transformation to determine the bearing

To calculate the bearing from a point $p_1(r_1, \theta_1, \varphi_1)$ to a point $p_2(r_2, \theta_2, \varphi_2)$, as illustrated in Figure 3, the two points are transformed from spherical coordinates to Cartesian coordinates by the basis vector $\mathbf{e}_\mathrm{r}$ in the way

$$\bar{\boldsymbol{r}}_1^i(r_1, \theta_1, \varphi_1) = \mathbf{e}_\mathrm{r}(\theta_1, \varphi_1) \cdot r_1 \,, \tag{20}$$

$$\bar{\boldsymbol{r}}_2^i(r_2, \theta_2, \varphi_2) = \mathbf{e}_\mathrm{r}(\theta_2, \varphi_2) \cdot r_2 \,. \tag{21}$$

In this case, it can be assumed that

$$r = r_1 = r_2 \,. \tag{22}$$

The coordinate system a is turned around the z_i-axis into the direction of the point p_1 by

$$\bar{\varphi}_1 = 0 \,, \tag{23}$$

$$\bar{\varphi}_2 = \varphi_2 - \varphi_1 \,, \tag{24}$$

$$\bar{\boldsymbol{r}}_1^a(r_1, \theta_1, \bar{\varphi}_1) = \mathbf{e}_\mathrm{r}(\theta_1, \bar{\varphi}_1) \cdot r \,, \tag{25}$$

$$\bar{\boldsymbol{r}}_2^a(r_2, \theta_2, \bar{\varphi}_2) = \mathbf{e}_\mathrm{r}(\theta_2, \bar{\varphi}_2) \cdot r \,. \tag{26}$$

Then coordinate system a is turned around the y_a-axis by the angle

$$\beta = \frac{\pi}{2} - \theta_1 \tag{27}$$

into the point p_1 to get the coordinate system b, so that the z_b-axis points into the same direction as the vector of the point p_1. The vector $\bar{\boldsymbol{r}}_1^b$ can be obtained by

$$\bar{\boldsymbol{r}}_1^b = \boldsymbol{R}_a^b \cdot \bar{\boldsymbol{r}}_1^a = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \cdot \bar{\boldsymbol{r}}_1^a = r \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{28}$$

The same rotation matrix is used for the point p_2 so that the vector $\bar{\boldsymbol{r}}_2^b$ can be calculated by

$$\begin{aligned} \bar{\boldsymbol{r}}_2^b = \boldsymbol{R}_a^b \cdot \bar{\boldsymbol{r}}_2^a &= \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_2 \cos\bar{\varphi}_2 \\ \cos\theta_2 \sin\bar{\varphi}_2 \\ \sin\theta_2 \end{pmatrix} \cdot r \\ &= \begin{pmatrix} \cos\beta \cos\theta_2 \cos\bar{\varphi}_2 + \sin\beta \sin\theta_2 \\ \cos\theta_2 \sin\bar{\varphi}_2 \\ -\sin\beta \cos\theta_2 \cos\bar{\varphi}_2 + \cos\beta \sin\theta_2 \end{pmatrix} \cdot r \\ &= \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix}. \end{aligned} \tag{29}$$

The bearing may be calculated by

$$\begin{aligned} \psi = \tan^{-1}\left(\frac{y_b}{x_b}\right) &= \tan^{-1}\left(\frac{\cos\theta_2 \sin\bar{\varphi}_2}{\cos\beta \cos\theta_2 \cos\bar{\varphi}_2 + \sin\beta \sin\theta_2}\right) \\ &= \tan^{-1}\left(\frac{\sin\bar{\varphi}_2}{\cos\theta_1 \tan\theta_2 - \sin\theta_1 \cos\bar{\varphi}_2}\right) \end{aligned} \tag{30}$$

as exemplified in Figure 4. The view is perpendicular on the plane defined by the x_b-axis and the y_b-axis.

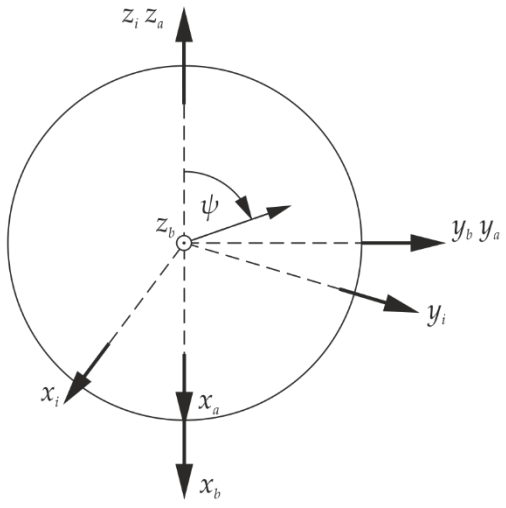

Figure 4: Illustration of the bearing angle ψ

4.2.2 Calculation of the distance between two points

The distance between two points on a sphere depends on the way they relate to each other. Because most of all points, especially all points within a polygon, are connected by a great circle, the distance accords to the arc length of the great circle leg.

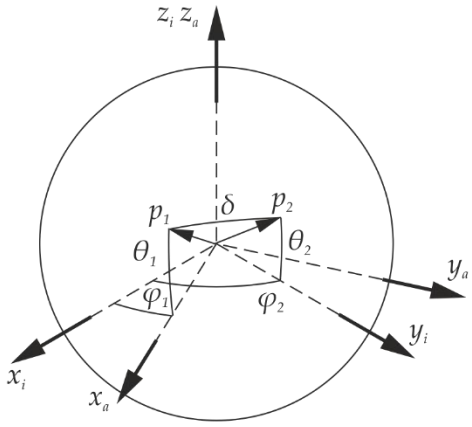

Figure 5: Distance between two points

As shown in Figure 5, the angle between the two points is proportional to the distance, if the radius can be assumed to be constant. The two points can be transformed from spherical coordinates $p(r, \theta, \varphi)$ to Cartesian coordinates $\bar{\boldsymbol{r}}(r, \theta, \varphi)$ by the basis vector $\mathbf{e}_\mathrm{r}$ in the way

$$\bar{\boldsymbol{r}}_{\mathbf{1}}^{\boldsymbol{i}}(r_1, \theta_1, \varphi_1) = \mathbf{e}_\mathrm{r}(\theta_1, \varphi_1) \cdot r_1 = \begin{pmatrix} \cos\theta_1 \cos\varphi_1 \\ \cos\theta_1 \sin\varphi_1 \\ \sin\theta_1 \end{pmatrix} \cdot r_1 \,, \tag{31}$$

$$\bar{\boldsymbol{r}}_{\mathbf{2}}^{\boldsymbol{i}}(r_2, \theta_2, \varphi_2) = \mathbf{e}_\mathrm{r}(\theta_2, \varphi_2) \cdot r_2 = \begin{pmatrix} \cos\theta_2 \cos\varphi_2 \\ \cos\theta_2 \sin\varphi_2 \\ \sin\theta_2 \end{pmatrix} \cdot r_2 \,. \tag{32}$$

In this case, it can be assumed that

$$r = r_1 = r_2 \,. \tag{33}$$

The coordinate system a is turned around the z_i-axis into the direction of the point p_1 by

$$\bar{\varphi}_1 = 0 \,, \tag{34}$$

$$\bar{\varphi}_2 = \varphi_2 - \varphi_1 \,, \tag{35}$$

$$\bar{\boldsymbol{r}}_{\boldsymbol{1}}^{\boldsymbol{a}}(r_1, \theta_1, \bar{\varphi}_1) = \begin{pmatrix} \cos\theta_1 \\ 0 \\ \sin\theta_1 \end{pmatrix} \cdot r \,, \tag{36}$$

$$\bar{\boldsymbol{r}}_{\boldsymbol{2}}^{\boldsymbol{a}}(r_2, \theta_2, \bar{\varphi}_2) = \begin{pmatrix} \cos\theta_2 \cos\bar{\varphi}_2 \\ \cos\theta_2 \sin\bar{\varphi}_2 \\ \sin\theta_2 \end{pmatrix} \cdot r \,. \tag{37}$$

The scalar product of the two vectors yields to

$$\delta = \cos^{-1}\left(\frac{\bar{\boldsymbol{r}}_{\boldsymbol{1}}^{\boldsymbol{a}} \cdot \bar{\boldsymbol{r}}_{\boldsymbol{2}}^{\boldsymbol{a}}}{\|\bar{\boldsymbol{r}}_{\boldsymbol{1}}^{\boldsymbol{a}}\| \cdot \|\bar{\boldsymbol{r}}_{\boldsymbol{2}}^{\boldsymbol{a}}\|}\right) = \cos^{-1}(\cos\theta_2 \cos\bar{\varphi}_2 \cos\theta_1 + \sin\theta_1 \sin\theta_2) \,. \tag{38}$$

The distance d can be calculated by

$$d = \delta \cdot r \,. \tag{39}$$

4.2.3 Calculation of radial coordinates

To determine a point on a sphere with a distance and an inertial bearing the formulas derived in Subsection 4.2.1 and 4.2.2 can be used for the calculation.

The coordinate system is turned into the starting point, so that the z_b-axis is pointing in the same direction as the vector of the starting point, as shown in Figure 3, where

$$\alpha = \varphi_1 \tag{40}$$

and

$$\beta = \frac{\pi}{2} - \theta_1 \tag{41}$$

so that

$$\begin{aligned} \bar{\boldsymbol{r}}_1^b &= \boldsymbol{R}_a^b \cdot \boldsymbol{R}_i^a \cdot \bar{\boldsymbol{r}}_1^i \\ &= \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &\cdot \begin{pmatrix} \cos\theta_1 \cos\varphi_1 \\ \cos\theta_1 \sin\varphi_1 \\ \sin\theta_1 \end{pmatrix} \cdot r = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot r \,. \end{aligned} \tag{42}$$

The to be new calculated point uses the angle ψ, which stands for the given bearing and the angle

$$\delta = d \cdot \frac{1}{r} \tag{43}$$

whereat d is the distance and r the radius. The new point can be described in the coordinate system b by

$$\bar{\boldsymbol{r}}_2^b = \begin{pmatrix} \cos\left(\frac{\pi}{2} - \delta\right)\cos(\pi - \psi) \\ \cos\left(\frac{\pi}{2} - \delta\right)\sin(\pi - \psi) \\ \sin\left(\frac{\pi}{2} - \delta\right) \end{pmatrix} \cdot r\,. \tag{44}$$

By taking the inverse of the rotation matrices, the point $\bar{\boldsymbol{r}}_2^i$ can be obtained by

$$\begin{aligned} \bar{\boldsymbol{r}}_2^i = \boldsymbol{R}_a^i \cdot \boldsymbol{R}_b^a \cdot \bar{\boldsymbol{r}}_2^b = (\boldsymbol{R}_i^a)^{-1} \cdot \left(\boldsymbol{R}_a^b\right)^{-1} \cdot \bar{\boldsymbol{r}}_2^b = (\boldsymbol{R}_i^a)^{\boldsymbol{T}} \cdot \left(\boldsymbol{R}_a^b\right)^{\boldsymbol{T}} \cdot \bar{\boldsymbol{r}}_2^b \\ = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{pmatrix} \\ \cdot \begin{pmatrix} \cos\left(\frac{\pi}{2} - \delta\right)\cos(\pi - \psi) \\ \cos\left(\frac{\pi}{2} - \delta\right)\sin(\pi - \psi) \\ \sin\left(\frac{\pi}{2} - \delta\right) \end{pmatrix} \cdot r \end{aligned} \tag{45}$$

in the inertial system.

4.2.4 Calculation of intersection and inter-common points

A key function of the whole thesis is to determine whether two polygon legs intersect with each other or not. A polygon leg is defined by two points in spherical coordinates, whereat a great circle connects the points with each other. The approach to identify such an intersection, is to get first the intersection points of the great circles and check afterwards if it is a true intersection concerning the two given legs.

As shown in Figure 6, the intersection of two great circles leads always to two intersection points. The two great circles can be transformed into two planes by transforming the leg points from spherical coordinates $p(r, \theta, \varphi)$ to Cartesian coordinates $\bar{\boldsymbol{r}}(r, \theta, \varphi)$ by the basis vector $\mathbf{e}_\mathrm{r}$ in the way

$$\bar{\boldsymbol{r}}(r,\theta,\varphi) = \mathbf{e}_{\mathrm{r}}(\theta,\varphi)\cdot r \tag{46}$$

and calculating the cross product of the two points $\bar{\boldsymbol{r}}_1, \bar{\boldsymbol{r}}_2$ and $\bar{\boldsymbol{r}}_3, \bar{\boldsymbol{r}}_4$ to get

$$\boldsymbol{p}_1 = \bar{\boldsymbol{r}}_1 \times \bar{\boldsymbol{r}}_2\,, \tag{47}$$

$$\boldsymbol{p}_2 = \bar{\boldsymbol{r}}_3 \times \bar{\boldsymbol{r}}_4\,. \tag{48}$$

By calculating the cross product of the two new vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, which are representing the planes of the great circles,

$$\boldsymbol{a} = \boldsymbol{p}_1 \times \boldsymbol{p}_2 \tag{49}$$

the resulting vector $\boldsymbol{a}$ equals the axis of the intersection of both planes. This vector can be transformed back into spherical coordinates $\bar{a}(r,\theta,\varphi)$.

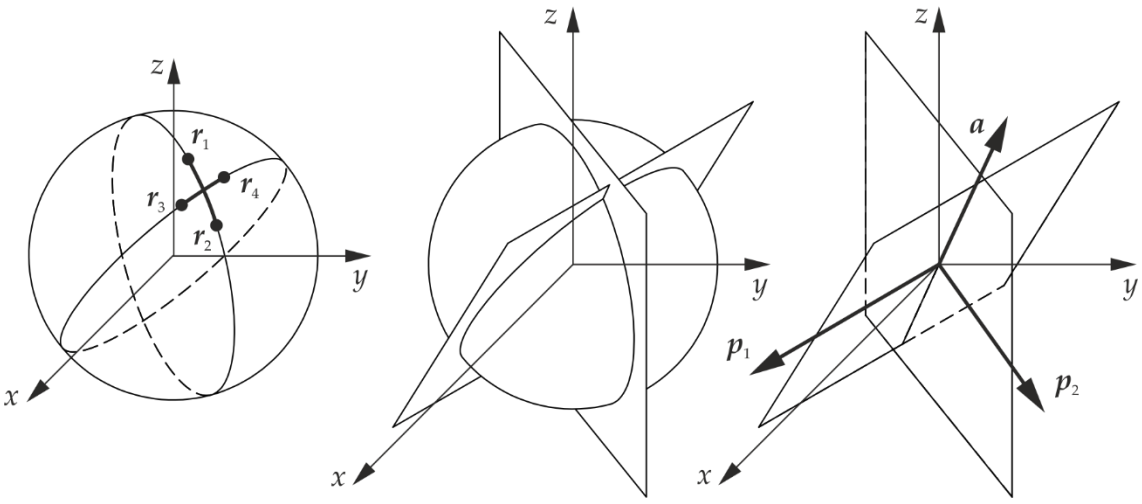

Figure 6: Intersection of two great circles

By calculating the average distance from the four points $\bar{\boldsymbol{r}}_1, \bar{\boldsymbol{r}}_2, \bar{\boldsymbol{r}}_3$ and $\bar{\boldsymbol{r}}_4$ to both intersection points $\bar{\boldsymbol{a}}$ and $-\bar{\boldsymbol{a}}$ the real intersection point can be determined.

There are four different types of intersections to distinguish. Figure 7 shows a true intersection and an untrue intersection of two polygon legs. Figure 8 illustrates the possible special cases. The first case is the inter-common point, which means that one of the two leg points lies directly on the great circle of the other leg. This is not considered to be a true intersection. The second special case derives, if one of each leg points are common. This is also not to be considered as a true intersection. The first special case is a hybrid of a common point and an intersection and is therefore called inter-common point.

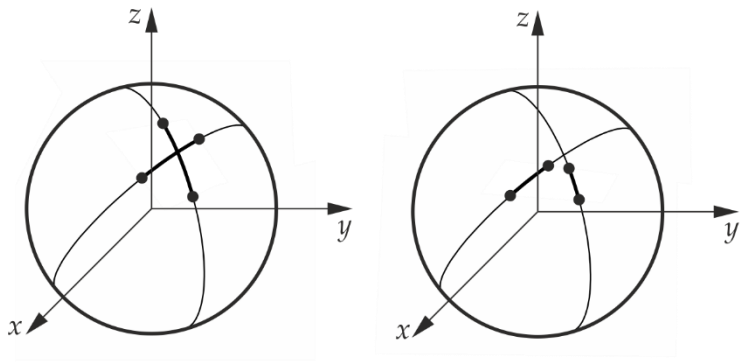

Figure 7: Types of intersection – a) true intersection, b) no intersection

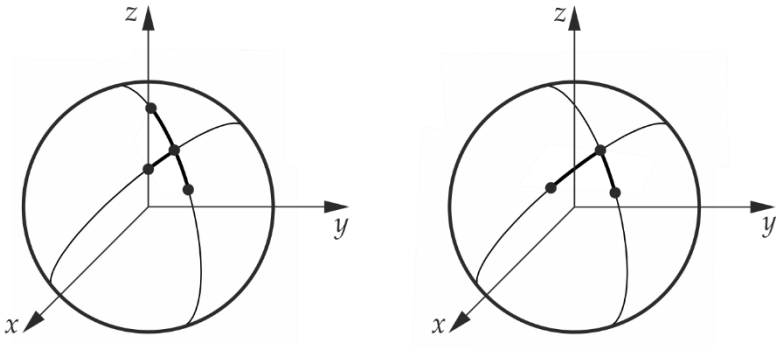

Figure 8: Types of common points – a) inter-common point, b) common point

As the calculation of the intersection takes place on a perfect sphere, the height of the intersection point can be determined afterwards. The height is obtained by a linear interpolation of the two heights from the points of the first leg. This means that it is crucial which leg is taken as the first one. This is an important issue in the polygon clipping algorithm.

4.3 Polygon

Usually a polygon is a plane geometry which is bounded by several straight-line segments in a loop to form a closed chain.

In this thesis, the word polygon has a slightly different definition, as there are no straight lines which are connecting the vertices with each other but great circles. Also, all polygons are so called simple polygons which demands that the boundary of the polygon does not cross itself. Simple polygon does not imply convex polygon, so that all applied functions are capable of dealing with concave polygons as well.

4.3.1 Calculation of the polygon area

To calculate the area of a simple concave polygon the Gauss's area formula

$$A = \frac{1}{2}\sum_{i=0}^{n}(y_i + y_{i+1})(x_i - x_{i+1}) \tag{50}$$

is usually used, whose vertices are described in a Cartesian coordinate system in the plane. Where y_i and x_i are the coordinates of the vertices, n is the total number of vertices and A is the area of the polygon. Figure 9 shows the several areas of the polygon legs (A_0, A_1, A_2 and A_3), which sum up to the total area of the polygon. A convenient side effect is, that the sign of the result can be taken as verification for the direction of the vertices. If the result is positive the direction is clockwise.

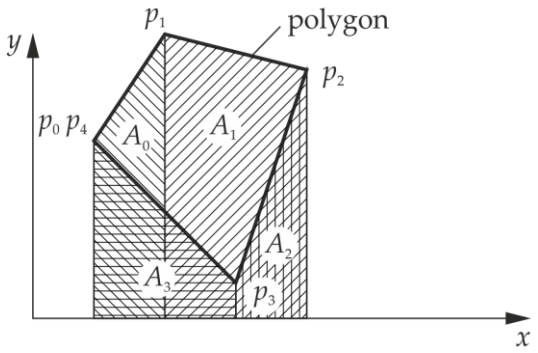

Figure 9: Illustration of the Gauss's area formula

As the to be calculated polygon area is on a sphere, little adjustments to the formula must be made. The derivation of the surface formula of a sphere with constant boundaries described by

$$A = \int_{\varphi_1}^{\varphi_2}\int_{\theta_1}^{\theta_2} r^2 \cdot \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\varphi \tag{51}$$

which yields to

$$A = r^2(\cos\theta_1 - \cos\theta_2)(\varphi_2 - \varphi_1) \tag{52}$$

can be used for the calculation. Where r is the radius of the sphere, θ the polar angle and φ the azimuthal angle. Figure 10 shows a polygon leg on a sphere. Its lower boundary is the equator, its upper boundary is the polygon leg described by a great circle and the left and right boundaries are great circles which are positioned perpendicular to the equator surface.

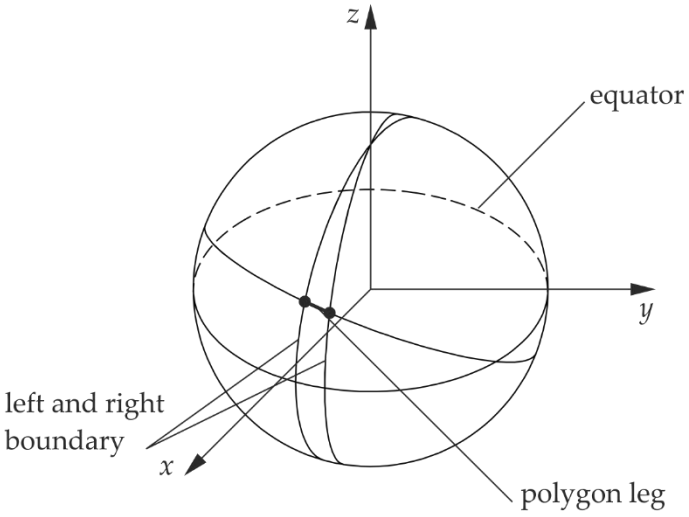

Figure 10: Polygon leg on a sphere

Depending on the accuracy of the result, the leg is subdivided into a various number of areas which can be assumed to have an upper boundary, which is a small circle parallel to the equator. Therefore, the formula

$$A_{leg} = r^2 \sum_{j=0}^{m} (\cos\theta_1 - \cos\theta_{j+1})(\varphi_{j+1} - \varphi_j) \tag{53}$$

can be applied for the calculation of the subdivided areas. The variable m represents the number of subdivided areas of one leg. As the lower boundary is given by the equator and the upper boundary can be written as the average of the two polar angle of one part, the equation derives to

$$A_{leg} = r^2 \sum_{j=0}^{m} \left(1 - \cos\left(\frac{\theta_{j+1} + \theta_j}{2}\right)\right)(\varphi_{j+1} - \varphi_j)\,. \tag{54}$$

The summation of all these areas in the way of

$$A = r^2 \sum_{i=0}^{n} \sum_{j=0}^{m} \left(1 - \cos\left(\frac{\theta_{j+1,i} + \theta_{j,i}}{2}\right)\right)(\varphi_{j+1,i} - \varphi_{j,i}) \tag{55}$$

adds up to the total area of the polygon. Figure 11 shows the approximation of the area with a small circle.

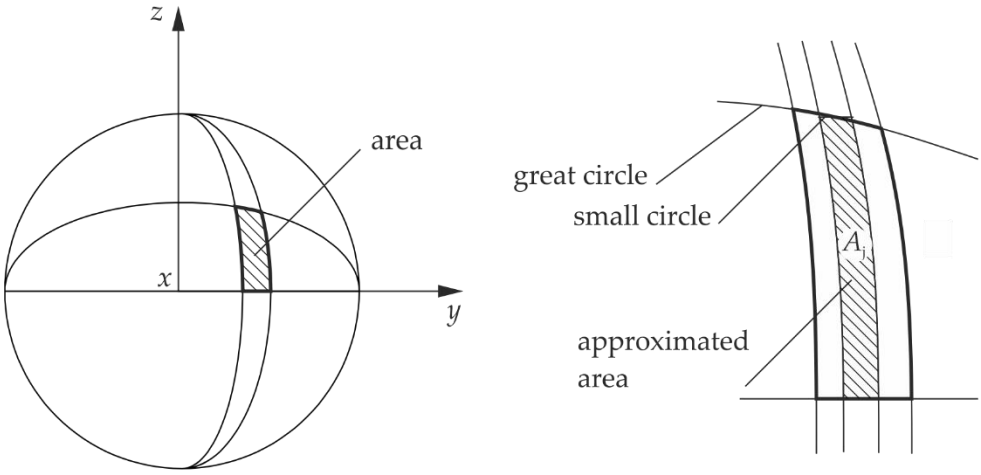

Figure 11: Approximated area

To ensure an accurate result and a fast calculation of the area the number of subdivided areas per polygon leg depends on the longitude spread of the leg. There is at least a number of five areas per leg and a maximum size of 0.1 degrees' longitude of a subdivided area.

4.3.2 Point in polygon check

The function to determine whether a point is in a given polygon or not uses the Jordan Polygon Theorem [5]. As shown in the Figure 12, if a ray, which starts at the to be verified point and is heading in a random direction, the number of intersections can be taken for the verification. If the number of intersection points is odd, the point is in the polygon. This function uses the intersection function explained in Subsection 4.2.4 which causes that a point lying on the polygon boundary will be stated as being inside of the polygon.

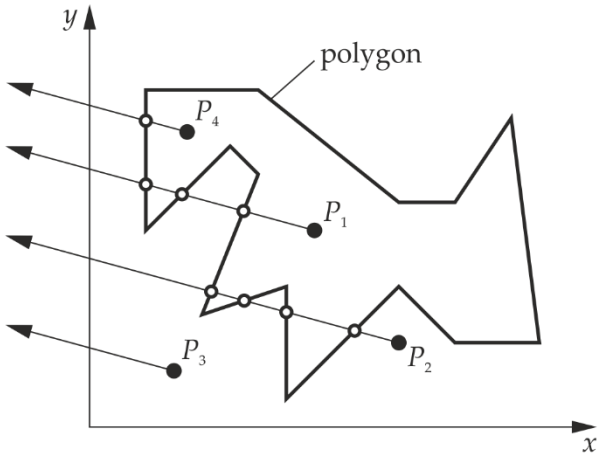

Figure 12: Point in polygon check

To avoid invalid results, the random heading of the ray is not allowed to point towards any vertex. The theorem does not work on a sphere, because instead of a ray a great circle is been used, which would always intersect the polygon at an even number, if the point would be in the polygon. Therefore, an end point must be set to avoid this problem. By getting the bounding rectangle of the given polygon an end point can be defined, which is located outside of the polygon, as shown in the Figure 13. These two points, P_{end} and P_1, define the leg, which is used for intersection function.

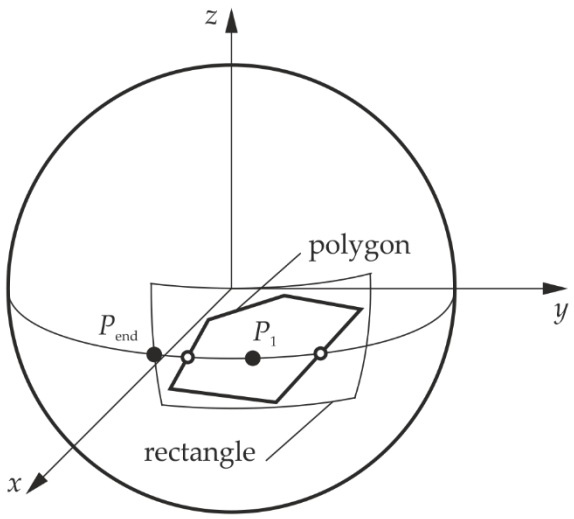

Figure 13: Bounding rectangle of a polygon

To avoid numerical errors by getting too close to a vertex with the random picked great circle, the function uses the heading which points between the biggest gap of the whole polygon, as shown in Figure 14. A big advantage of this approach is that the function normally tries automatically to get the closest exit out of the polygon or if the point is lying outside, it usually uses a heading which does not intersect at all with the polygon.

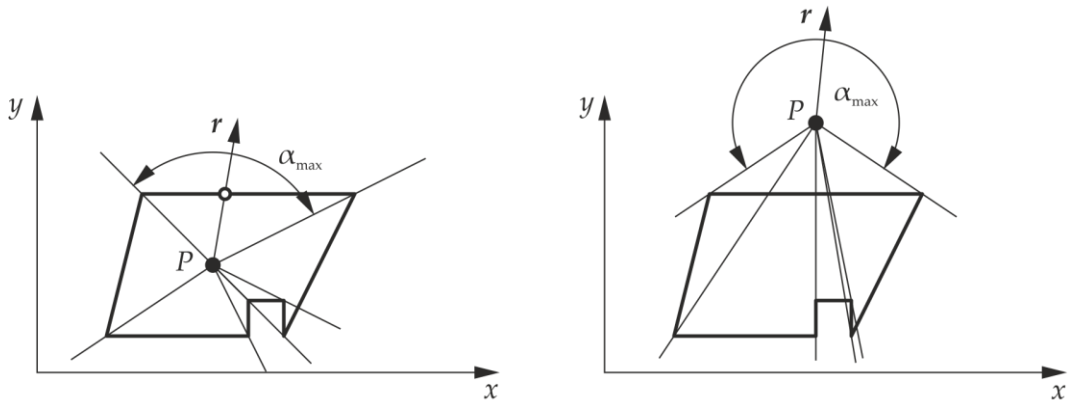

Figure 14: Choosing the direction of the ray

Other algorithms like the winding number algorithm have shown to be insufficient accurate for polygons with a large number[2] of vertices and if the to be verified point lies close to the polygon boundary. Another drawback of this algorithm is given by the fact, that it would recognize a point on the opposite side of the sphere as being inside of the polygon. Although this case is highly unlikely arrangements would have to be taken.

4.3.3 Clipping polygon algorithm

The algorithm for clipping concave polygons with each other is a key element of this thesis. The applied algorithm is based on an algorithm for convex polygons [6] and a more advanced algorithm [7] which is capable of managing concave and self-intersecting polygons as well. Unfortunately, these algorithms are too little accurate for this thesis as they are not using common or inter-common points, but displace the vertex slightly which causes a perturbation[3]. Therefore, a more accurate algorithm had to be developed on the basis of these two algorithms.

---

[2] Tests have shown that the implemented algorithm is starting to be too little accurate at a number of about 300 vertices or more. There has been no examination of the problem in detail, because it could be solved by using the Jordan Polygon Theorem.

[3] This can be the case if there are two points of each polygon on the very same spot. In these algorithms are only two types of points permitted interior and exterior. Therefore, one of the points must be displaced. The same is necessary for the so called inter-common point. Because there are often a various number of clippings necessary to obtain the final result, an undefined number of displacements may have happened, which would lead to a unsatisfied accuracy of the result.

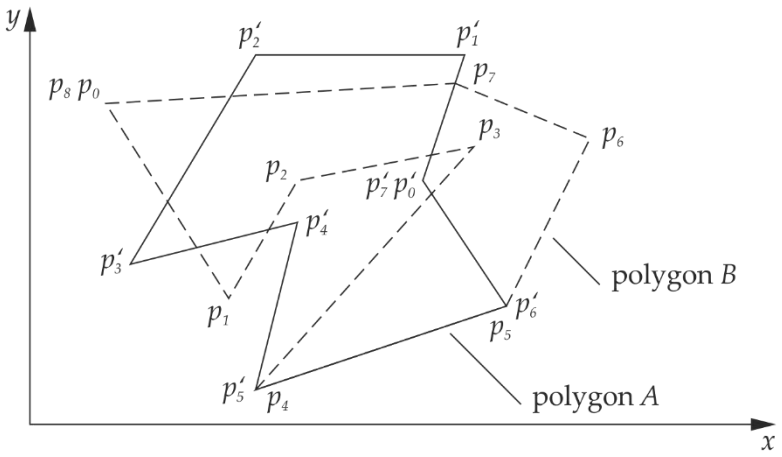

Figure 15: Two simple polygons

As shown in Figure 15, there are two simple polygons with each a several number of vertices. Important for the applied algorithm is that the direction of the vertices of both polygons is counter clockwise and all other requirements which are given by using simple polygons, as explained in Subsection 4.3, are complied.

Figure 16 shows the basic steps of the polygon clipping algorithm. In the first step, all points of both polygons are going to be classified depending on their type. There are three types: in, out and common. In and out means, if the to be assessed point lies inside or outside of the other polygon. If two points of both polygons are exactly on the same spot, the points are classified as common points.

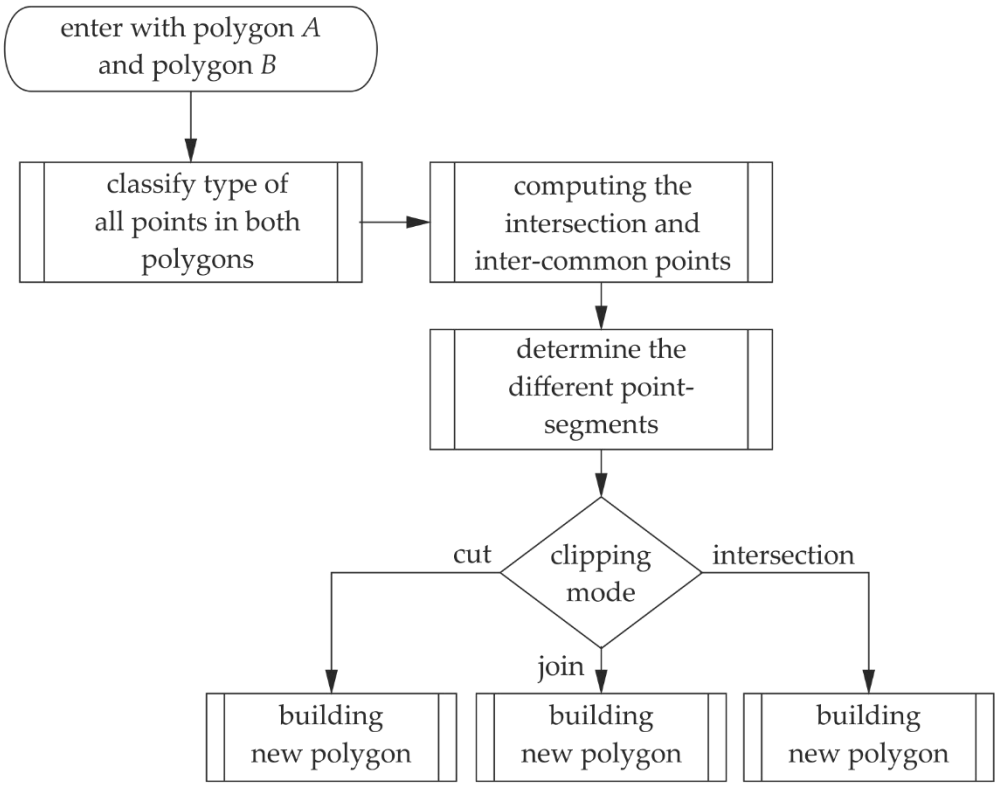

Figure 16: Overview of the clipping algorithm

In the next step, all intersection and inter-common points are being calculated. By intersecting every leg of each polygon with each other, all intersection points and inter-common points can be found and marked as shown in Figure 17. The labeling of the position of each point depending not on their coordinates but on the position in the polygon is very essential for the next step.

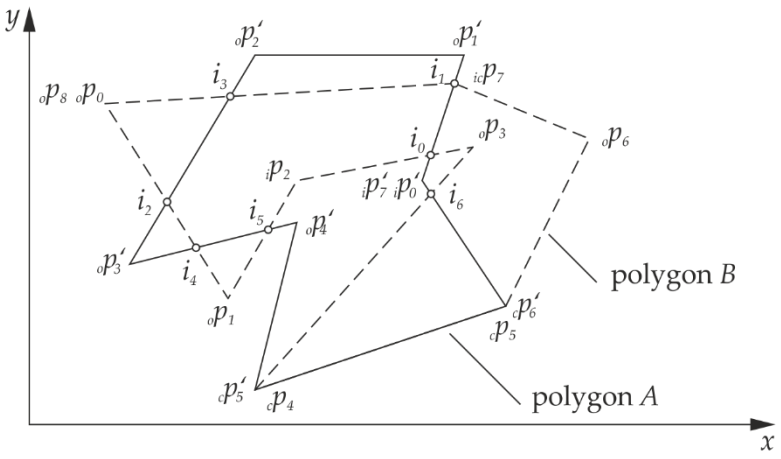

Figure 17: Intersection and inter-common points

Before the different point segments can be obtained, the intersection and inter-common points must be merged into the two polygons in correct order. Afterwards the first polygon is being scanned for so called point segments or legs. The characteristics of a leg is that on its boundaries the state changes always from interior to exterior or vice-versa, so that there are several inside and outside legs which alternate in their order. The boundary points from the different legs of the first polygon are used to get the legs of the second polygon. This is the most difficult step in the whole algorithm, because there is various number of different possibilities. Figure 18 shows the five different cases which are used to determine the point segments. The most problematic part of obtaining the legs is to find a starting leg which can ambiguously assigned as being interior or exterior.

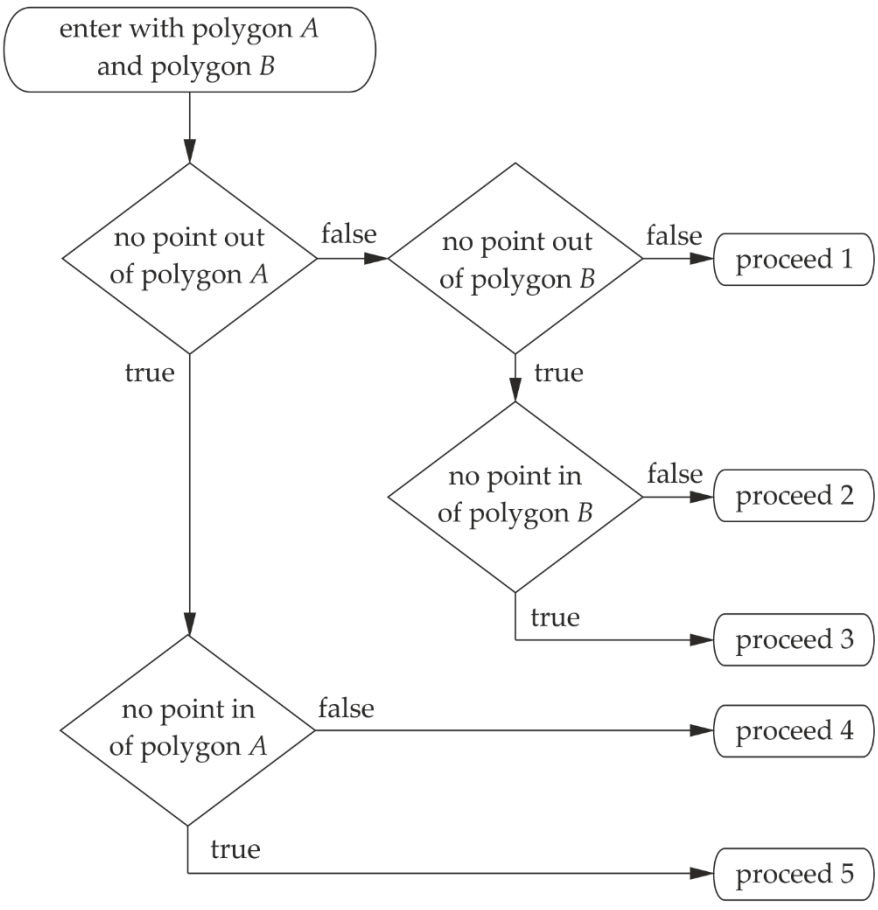

Figure 18: Determine the different point segments

While classifying the types of the vertices of both polygons in the first step, the number of the different types of both polygons is notified. This allows the illustrated case structure in Figure 18 to distinguish among the different cases.

Proceed with case 1:

In this case, there is at least in both polygons one point stated as being exterior. This is considered to be the normal case and assures that there is a clearly marked exterior leg with at least one point stated as being outside. Therefore, the starting leg is exterior. However, there are several cases which should be mentioned at this point. As shown in Figure 19 the possibility a) and c) are for the algorithm the same because there are several points stated outside and a number of common points. In this case, all common points are assessed as interior, because the first leg is exterior and whenever there is a new leg it changes its type to the opposite. Case b) shows that there has not to be necessarily two intersection or inter-common points for an intersection of two polygons. These three cases are revealing the problematic with common points. Depending on the sequence of common points with other points, the leg boundaries can differ very strongly.

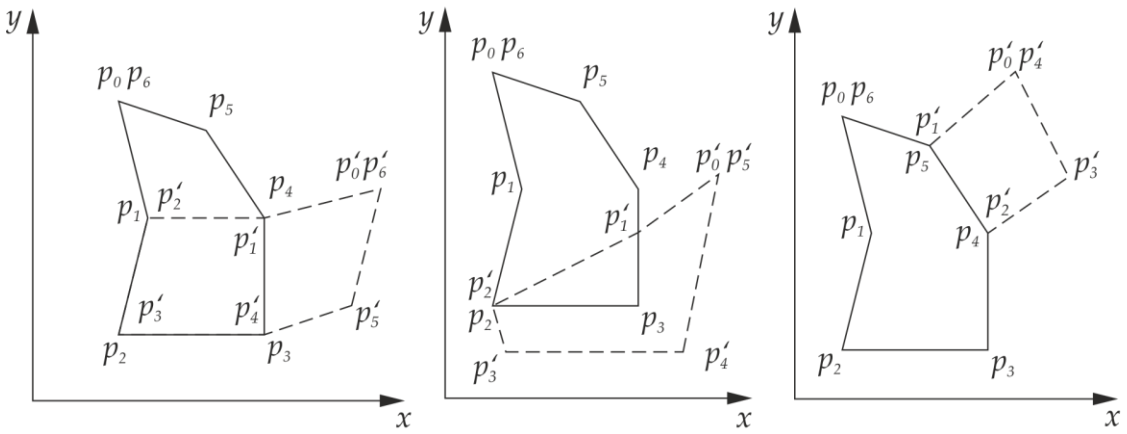

Figure 19: Case 1 – possibility a), b) and c)

Proceed with case 2:

Figure 20 shows a possible case where there is no point stated out of polygon *B*, but there is an intersection with polygon *A*. In this case the starting leg would be of the type interior because there is at least one point stated as being inside of the polygon *B*.

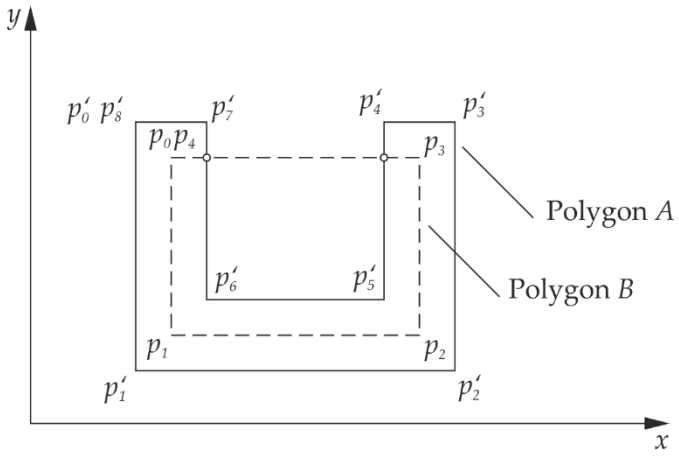

Figure 20: Case 2

Proceed with case 3:

In this case, all points of polygon *B* must be common points because there are no points stated as being outside or inside, as shown in Figure 21. To distinguish between the two illustrated cases a characteristic need to be defined, which is unique for either one of each layout. One may see that there are just two possibilities of a random point which lies in polygon *B*, either the point lies inside of polygon *A* or it lies outside. For getting a correct starting leg it is important to differ between these two cases.

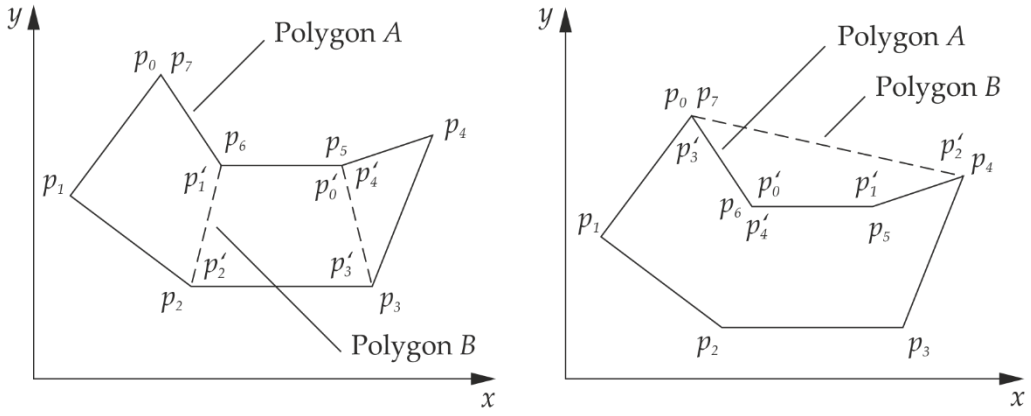

Figure 21: Case 3 – possibility a) and b)

Proceed with case 4:

This case is similar to the case 2 except that the polygon *A* becomes to polygon *B* and vice-versa.

Proceed with case 5:

This case is similar to the case 3 except that the polygon *A* becomes to polygon *B* and vice-versa.

Figure 22 shows the result after determine the different legs. The bold lines represent the exterior legs of both polygons and the regular lines the interior legs. Polygon *B* is represented by a dashed line. As mentioned before the interior and exterior legs alternate. Also recognizable is that between the two common points, the legs of the polygons are not the same. This is caused by the implemented algorithm and is mandatory for all common section to make it possible to support all three clipping modes.

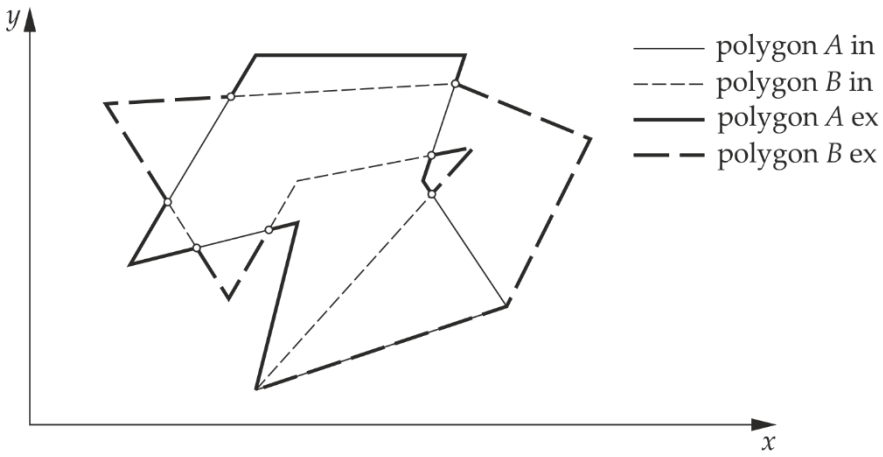

Figure 22: Interior and exterior legs of both polygons

The last step is to assemble the new polygon depending on the clipping mode. Figure 23 shows the three different solutions for the three modes. One can recognize that the solution for the join mode is derived by connecting all exterior legs with each other. The solution for the intersection mode can be obtained by connecting all interior legs with each other. Depending on which polygon gets cut out of which one, one just must alternate with connecting exterior legs from polygon *A* and interior legs from polygon *B* or exactly the opposite, if polygon *B* gets cut of polygon *A*.

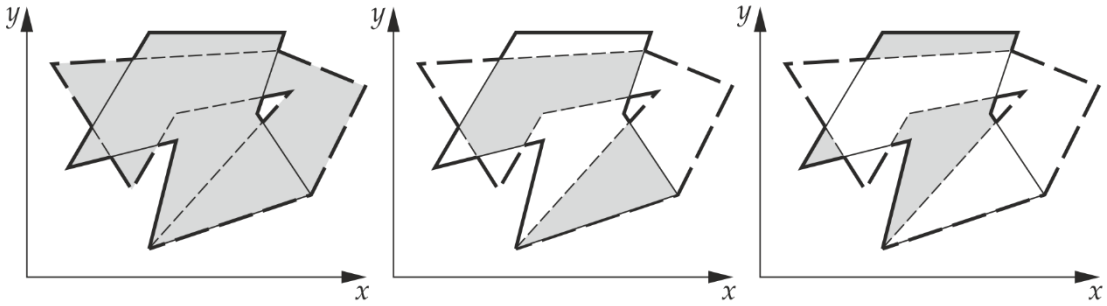

Figure 23: Clipping mode – a) join, b) intersection and c) cut

This reveals the big benefit of the developed and applied algorithm. Irrespectively of the clipping mode the first three steps are always the same. Only in the last step the method splits into three different cases. Additional to this advantage, the assembly of the polygon with the preprocessed legs could not be any easier and causes therefore very little potential for errors in the program code.

5 Solution

The developed solution consists basically out of two parts. A main processing software and a post processing software. The reason of dividing is due to the different use cases of each part. The main processing is only necessary if the aeronautical data or the applied rule set has changed. The post processing is designed to have minimal execution time and can be executed by different platforms.

5.1 Programming language

The whole program is written in JavaScript, also known as JS. The requirements for the developed toolbox, regarding the programming language, has been very little. There is no time critical factor involved neither is the dimension of the project very large. The key requirement has been to use a well-known and easy to handle language, because the script, which reproduces the aeronautical regulations, uses also JS. As JS is a very widespread programming language, especially because of the use for web applications, a lot of people are familiar with it.

Besides that, JS is supported by an open source project called Node.js, which developed a runtime environment for the interpreted programming language JS. This project fits exactly the needs of this toolbox and has been the second reason for using JS as programming language.

5.2 Basic software architecture

In Figure 24 is the basic software architecture for the user illustrated. The program provides a main script file, where the to be computed scripts can be included. It is possible to open this application several times to subdivide the different calculation steps, so that one may have an application for calculating the rule sets and another application to process the validation of the objects and to export the data to a KML file.

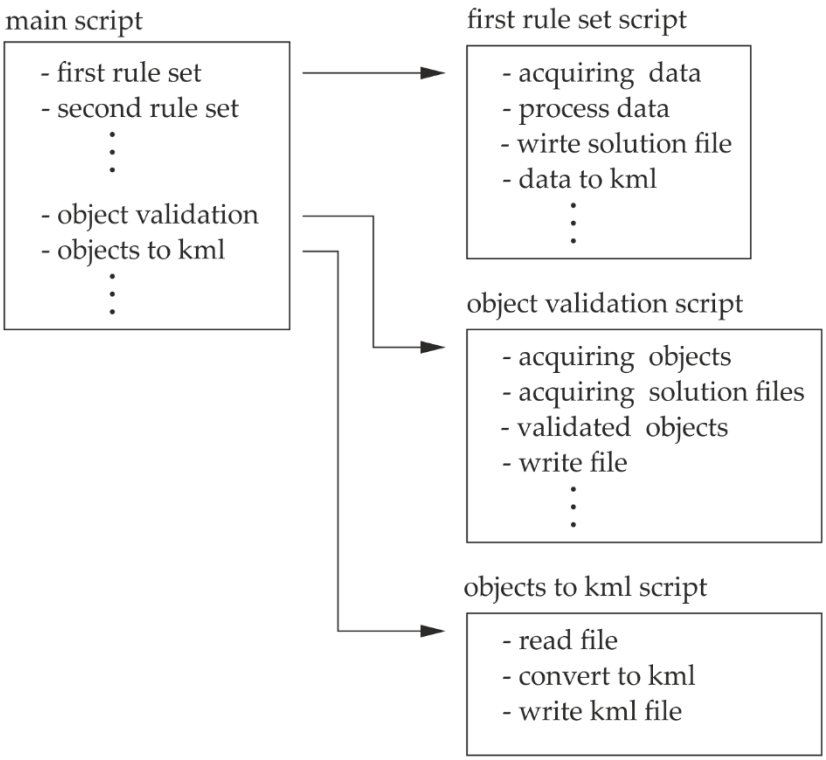

Figure 24: Basic architecture for the user

As the objective of this thesis is to provide a toolbox for aeronautical obstacle regulations, the methods are designed for modelling all possible cases of rules. Usually the first step, in the script for a specific rule set, is to acquire the necessary aeronautical data. Afterwards this data is processed by the scripted rules. The collection surfaces resulting of this process can then be exported as a text file or as a KML file. One way of a simple but effective examination of the data is to visualize it in Google Earth with the help of a KML file.

Another important part of the script is the validation of objects with the calculated collection surfaces. The validation itself is a rather simple function, as the whole toolbox has been designed to enable an easy of object validation.

Figure 25 shows an overview of the developed classes for covering the introduced problematic. As this is not a classic software for a specific purpose but rather a cluster of functions which are necessary for modelling aeronautical obstacle regulations, the architecture differs to typical programs since the requirements are also different. Although there are parts of object-oriented code, most of the classes are just providing the necessary functions for modelling regulations and are gathered in a reasonable way. The majority of the classes provide directly functions for being used in the script. Just a few basic classes are needed to help getting a clear architecture.

The Clipping class takes care of all interactions that are concerning polygons. This class provides four public functions (cut, join, intersection and thicken) that are necessary to cover all possible ways of handling polygons with each other.

The Polygon class is necessary for creating polygons. The main public functions offer methods for designing polygons in various ways, examine them regarding to the demands described in Subsection 4.3 and verify if a point lies in the polygon or not.

A polygon is defined by numerous vertices. Those are described in the Vertex class. Most of the functions take vertices as parameter and return vertices. The class itself consist of some basic methods to set parameters and the attributes of the vertex.

A point in space can also be described by a vector. The Vector class is basically the same as the Vertex class except that vectors are mainly used for calculations in the background. Normally vertices are getting transformed to vectors for different kinds of calculations and afterwards back transformed to vertices. The use of vertices appears to be circuitous when vectors are needed for the calculations, but for reviewing intermediate results or checking the functionality of methods it is very helpful. Furthermore, the calculations are not time critical so that the extra time required for the transforming does not matter.

Another rather important class is the CoordianteFunctions class. It handles all basic methods to work with vertices on a sphere. As distances and directions cannot be obtained directly from the coordinates, proper functions are necessary. The class covers pretty much all sections described in Subsection 4.2.

Besides that, the handling of data is a major issue for the toolbox. The DataFunctions class handles everything that is related to data. The most important public function is

the getAeronauticalData() method which is designed to acquire the sought data in an XML file, that is standardized in the AXIM 4.5 format.

The examination of the resulting collection surfaces is an important issue. The Kml class exports the calculated collection surfaces to a KML file, which can be opened with Google Earth. There are different settings possible to differ among the different rule sets. Also, the classified objects can be displayed in Google Earth to check if the result is in any way plausible. As GML is a widely-used format for describing points in space the Gml class provides some basic functions for converting different formats in the GML format or vice-versa.

The xt_Math class is an extension to the usually provided math functions in most software languages. It covers some special features as for example the coordinate transformation from spherical to Cartesian coordinates and vice-versa.

To gather all different types of vertex objects the VertexObject class is being used. Yet there are just two types polygons and corridors, whereat corridors are only very rarely used, it designed to hold different types as there is might something added in the future.

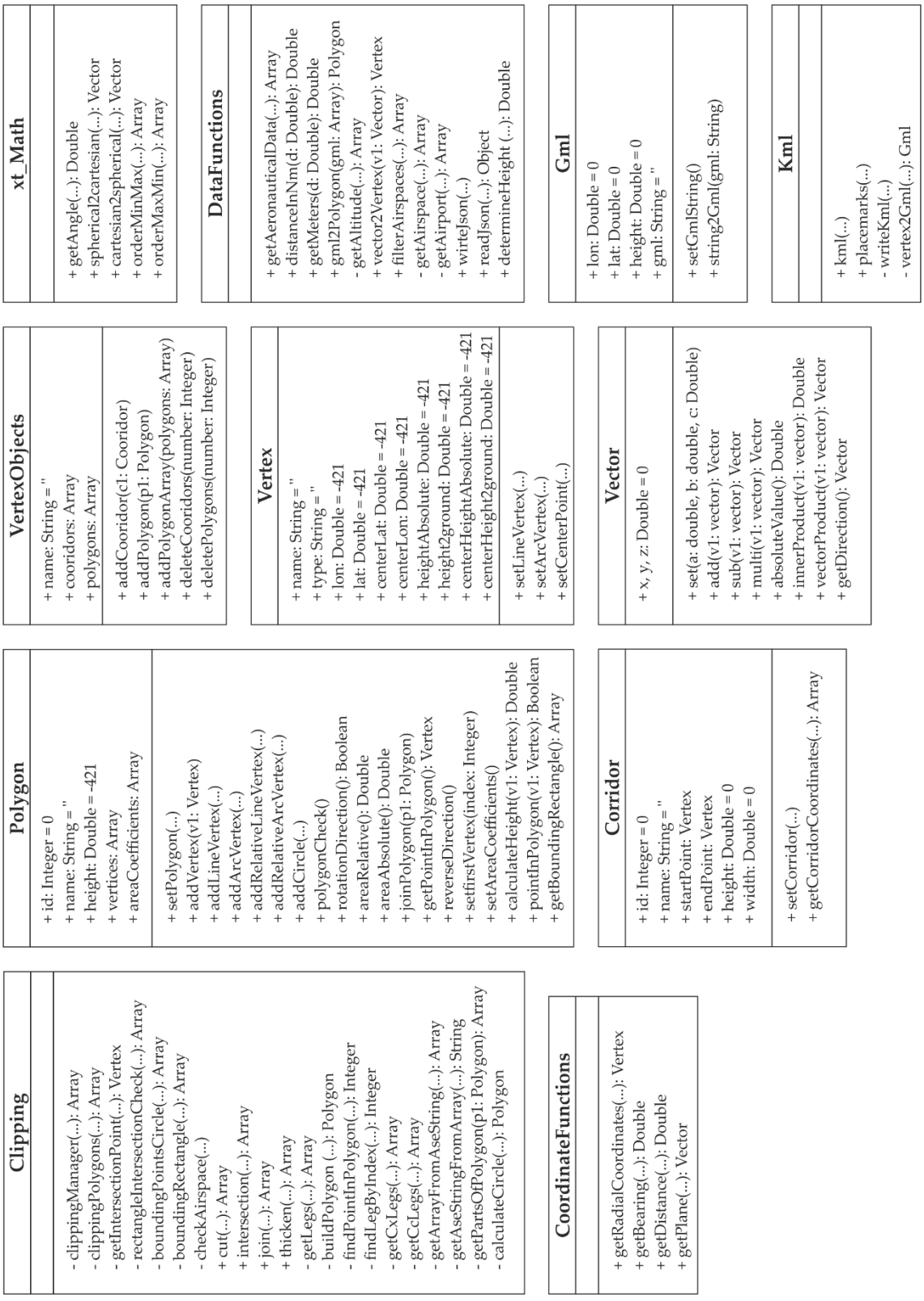

Figure 25: Overview of classes

Figure 26 shows in a rough way the dependencies of the developed classes. The layout also illustrates the hierarchy of the different classes.

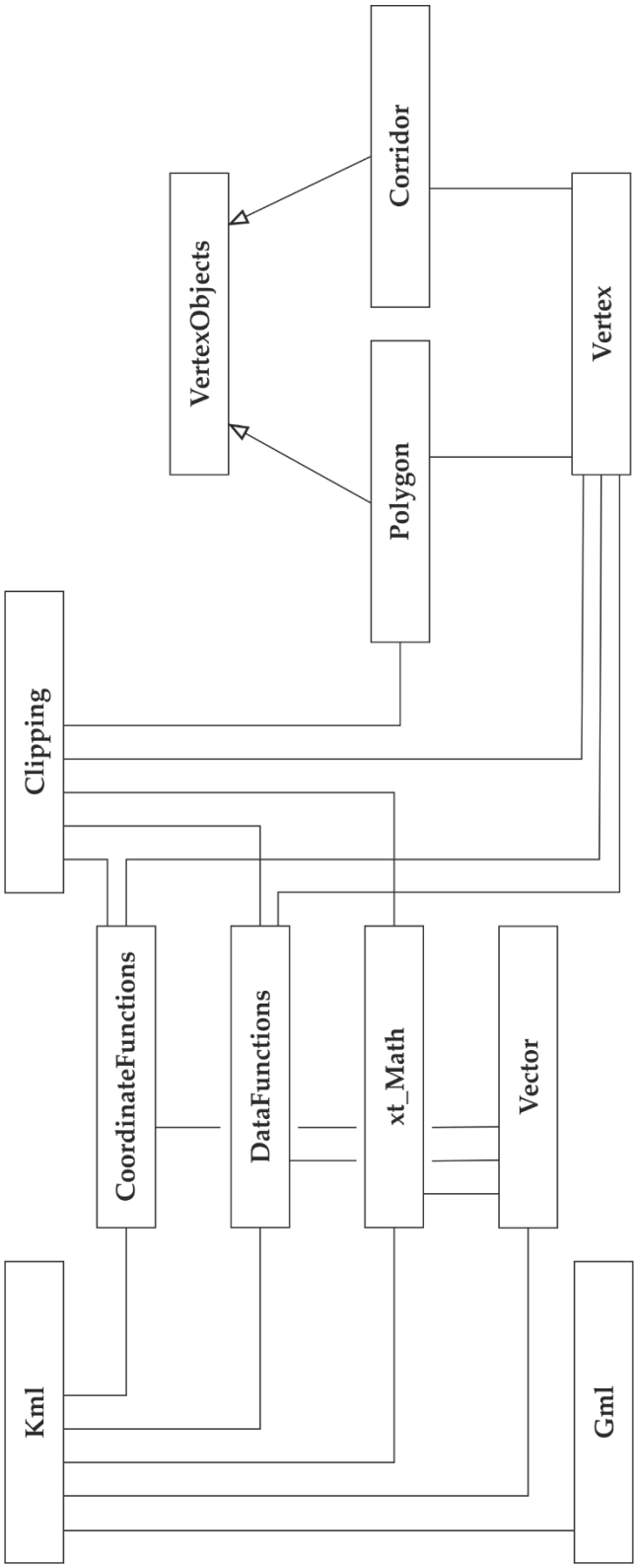

Figure 26: Class diagram

5.3 Main processing software

The main processing software has the task to calculate and provide collection surfaces, which can then be used by simple post processing software's, so that there is very little effort necessary to obtain useable results. The benefit of this does not only result in a simple post processing software, but also in an application which uses only very little time for executing a query.

5.3.1 Acquiring aeronautical data

The aeronautical data is being acquired from a XML file which is formatted by the international standard of AIXM 4.5. The file can be accessed locally or downloaded from the OFM network. The last option is the favored one as it can be guaranteed that the latest data is being used. A file contains usually one specific FIR whereat a distinction of two types is made. One contains only the FIR data and nothing more and the other covers the surrounding data of all FIRs which share a boundary to the be processed FIR. This is in this case preferred, as for example neighboring airports can have an impact to the classification of objects in a bordering FIR.

The acquired data comes in two objects where the first one contains the data of airports and the second one holds all airspaces. The objects are sharing the same structure as the XML, which makes it easy to access the data if one is familiar with the AIXM 4.5 format. To decrease the size of the acquired data only necessary information is being hold by the two objects. For example, the information of taxi lanes, gates and other miscellaneous objects are not relevant for any known regulation concerning the classification of obstacles and can therefore be excluded.

The DataFunctions class holds the function which takes care of acquiring the aeronautical data. The function also provides several parameters to confine the data. For example, one can distinguished among different types of airspaces to be included in the resulting object or choose specific airports by their ICAO code.

5.3.2 Modelling surfaces

Before it is possible to calculate collection surfaces, it is necessary to define the different kind of surfaces that are described in the regulatoins, which can then be processed regarding to airspaces and other constrains.

There are two different types of surfaces to distinguish. The first one is a surface which follows the shape of the terrain surface with a constant distance to it, as shown in Figure 27. The lateral limits are defined by a polygon on a sphere.

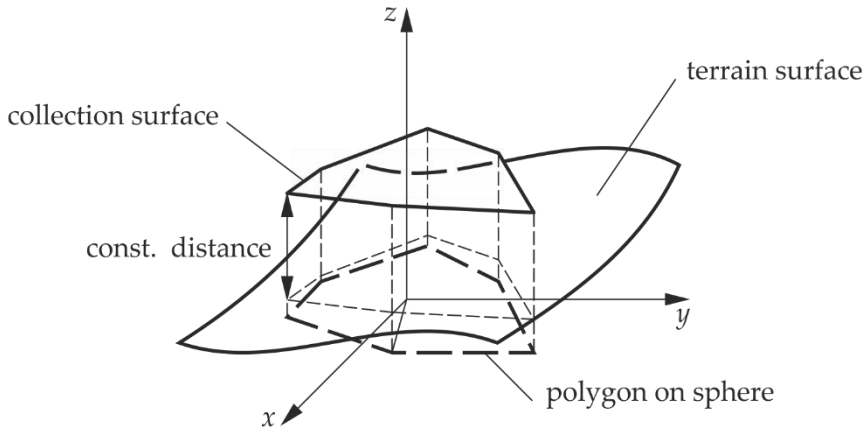

Figure 27: Collection surface as terrain surface

The second one uses absolute values for defining points on a geometrical surface. The lateral limits are determined identically for both types, just the vertical limits differ to each other. The Vertex object knows two definitions of height an absolute and a relative value. The absolute one follows the same rules as the height of the terrain is being measured, zero at the mean sea level. The relative value is zero at the local terrain surfaces.

Figure 28 shows the absolute definition of the vertical limit from a random picked shape. Similar to the polygon shown in Figure 27, the lateral limits of the polygon are described on a sphere. The absolute definition demands that all points of a polygon are lying on either a plane or on a cone surface[4]. This constrain allows a further processing of the polygons. Every polygon comes with an attribute called areaCoefficients, which holds the required coefficients to describe either a plane or a cone in a geometrical way.

The first three points of a polygon are used to evaluate the coefficients for the geometrical description of the plane surface. In case that the first two vertices are different of height and the second vertex holds the coordinates for a center point, the geometrical description for a cone is used.

[4] In preparation of the thesis itself a study of international and national regulations has been conducted. The outcome has shown that only plane and cone surfaces are needed to cover the given regulations. However, it is possible to add other geometrical surfaces to the existing program code very easily if it is necessary.

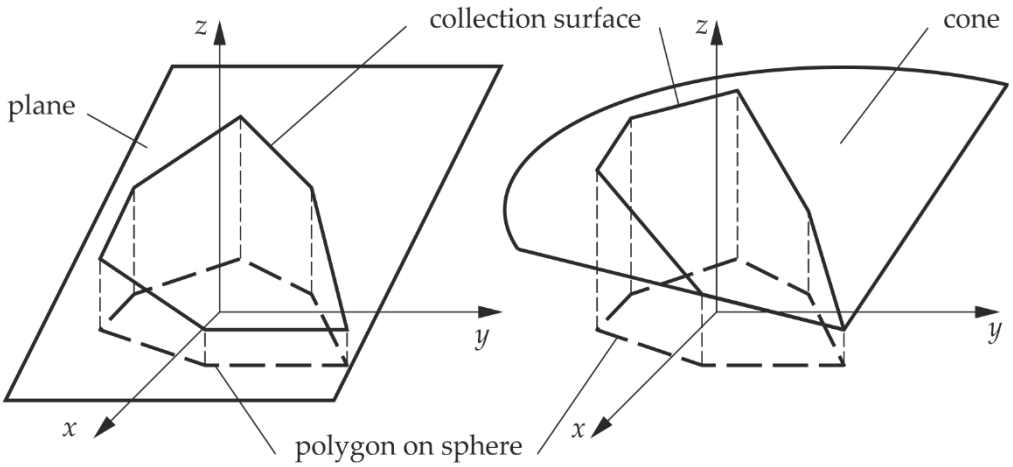

Figure 28: a) plane collection surface, b) cone collection surface

A plane described by the spherical coordinates longitude and latitude on a sphere does not result in a plane surface but in a curved surface, respectively a cone described by spherical coordinates does not result in a cone shaped surface. One can see, that the description of a line in the two-dimensional space

$$y = k \cdot x \tag{56}$$

turns into the Archimedean spiral

$$r = k \cdot \varphi \tag{57}$$

if the argument x gets replaced by the angle φ and the function value turns into the radius, as shown in Figure 29. The same is true for the three-dimensional case.

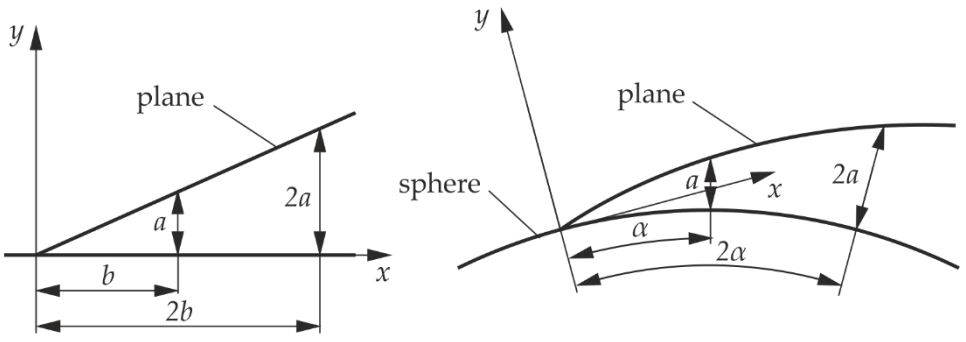

Figure 29: Geometrical plane on a sphere

This may seem like an error caused by insufficient accuracy of the transformation from Cartesian coordinates to spherical coordinates, but one can see that the proportional relation between height (radius) and distance (angle) is still in existence, which is the important characteristic for this instance.

For most regulations, it is necessary not to define locally fixed polygons, but to model polygons which depend on a little number of reference points. This enables a general layout that can be used for similar cases by just modifying the coordinates of the references points. To do so, the Polygon class provides several functions to create relative relations to the vertices of a random shape. Considering the example shown in Figure 30 a). One can assume that the vertices P_1 and P_2 are two reference points. The method addRelativeLineVertex(…) allows to create a new vertex, which is positioned relative to the bearing from the vertex P_1 to the vertex P_2 by the angle γ, is away by the distance d and elevated relative to the vertex P_1 by the height h. One can see that by this method a random shaped polygon can be created, which depends only on the first two reference points.

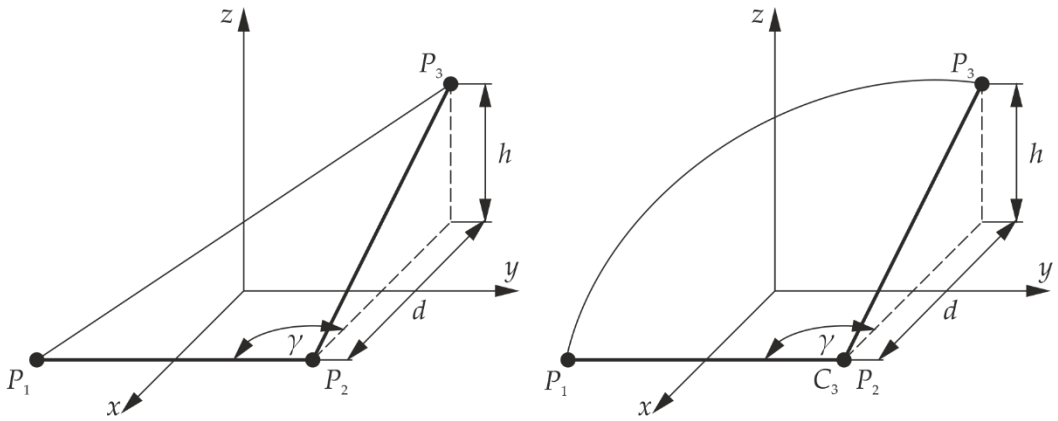

Figure 30: a) relative line vertex, b) relative arc vertex

This program code, as most other common software's, knows only polygons. A circle is described by numerous vertices. To facilitated the creation of an arc, one may use the addRelativeArcVertex(…) method, which defines a certain vertex as the starting point of a circle. This point holds also the coordinates for the center point of the circle, as shown in Figure 30 b). The abstract definition of an arc gets evaluated by the function polygonCheck(). The error caused by the discretization can be adjusted either relative to the size of the arc or absolute by the maximum distance the polygon differs to the arc.

The absolute definition of height demands, that all points of the polygon are lying on the applied geometrical surface. One can see that the creation of a more advanced polygon can cause complex calculations for the height of each point. Furthermore, the shape itself can get very complex so that it is easier to divide the shape into parts. To handle the mentioned problems, one can use the functions provided by the Clipping

class. This class offers the three operations described in Subsection 4.3.3. These functions not only take care of the lateral limits but also of the vertical limit, so that if a polygon P_1 gets joined with a second polygon P_2 all vertices of the polygon P_2 are located on the height specified by the first polygon P_1.

The pictured use of the Clipping class is just a minor application of this class. As some regulations are depending heavily on the given airspace structure, the major use is the clipping of airspaces with the pre-defined polygons.

The introduced algorithm for clipping polygons is just capable of handling two simple polygons with each other, as described in Subsection 4.3.3. As given regulations, can cause multiple clippings in succession, the resulting polygon may derive to a more complex polygon, which does not meet the needed requirements for the developed algorithm, as shown in Figure 31. A simple, convex polygon P_2 gets cut out of another simple, convex polygon P_1, which causes, that the solution is given by two polygons. One for the outside and one for the inside boundary. Afterwards the polygon P_3 gets cut of the object P_1' which results in two separated polygons.

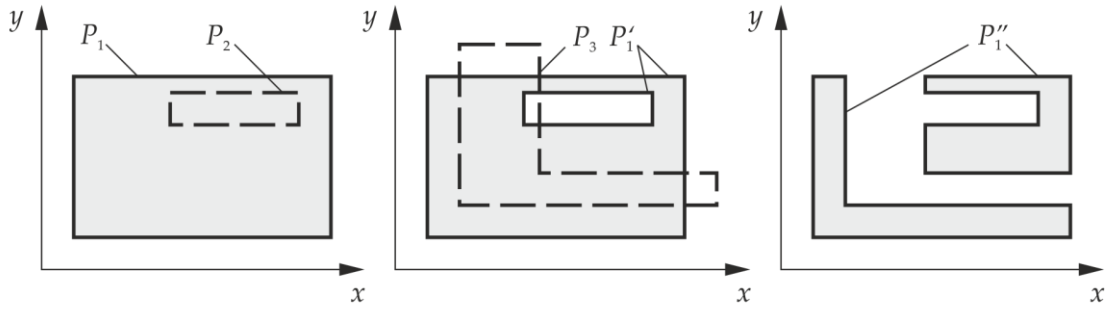

Figure 31: Derivation of a complex polygon

To handle these kind of processes, a clipping manager has been added to the class, which handles complex polygons. The main task is to disassemble a complex polygon (also called object) into an array of simple polygons, managing the different clippings of the simple polygons and reassemble the resulting polygons to one object.

In the case, illustrated in Figure 31, the clipping manager joins the inside polygon of the object P_1' and the polygon P_3 and cuts afterwards the resulting polygon out of the outside polygon of the object P_1'.

The so far explained relations of lateral and vertical limits shows that the lateral and vertical limits can be separated from each other in the calculation. This simplifies the whole process, as it is not a real three-dimensional problem.

5.3.3 Calculating collection surfaces

In the easiest way, a locally fixed collection surface is obtained by modelling a surface, setting the area coefficients for this particular surface and evaluating the arcs in the polygon.

In most cases, it is a little bit more complex. The to be modelled surface may depend on various local parameters, so that the actual shape varies among the different locations. Not only the shape is different, but also the airspace structure differs, which is highly relevant for the final collection surface and as most regulations are binding the shape of the collection surfaces to the local airspace structure.

Most of the time the three provided functions of the Clipping class are sufficient to calculate the common rules regarding the combination of airspace structure and pre-modelled surfaces.

5.3.4 Export of processed data

At the end of the main processing software stands the export of the data for other applications. The gathered collection surfaces, saved as an array of polygons, are exported in a JSON file. This might be considered as a drawback, as the aeronautical data comes in an XML file and it causes twice the effort to support two formats, but the JSON file format is especially developed for JS and is therefore very easy to handle. The XML format or other formats could be implemented very easily, if there would be a request for it. The data is being saved in the same structure as the objects are related to each other, which makes it very comfortable to access the data, if one is familiar with objects and software architecture of this code.

These kinds of exports share the purpose of just publishing the resulting data in a very simple way. A therefore different kind of publishing data is in the KML format, which is used to display the calculated data in Google Earth and needs an additional processing of the data. An own class has been developed to take care of creating a KML file. The public function kml(…) takes the same polygon array which is used for the other formats and also a path. The path not only contains the information, where the file should be saved at, but also information concerning settings for the appearance of the collection surfaces in Google Earth. This may be helpful, if there are displayed different rule sets at the same time or it is necessary to distinguish among different collec-

tion surfaces. Besides that, the main reasons for using Google Earth the quality management.

5.3.5 Quality management

As for every calculation, it is important to assess the correctness and accuracy of the result. Especially in numerical methods (e.g. FEM, CFD, …) it is not assured that a working calculation without any errors does lead to a correct or useable result. Therefore, a similar approach as the common numerical methods are using has been deployed to assess the result.

The approach consists of two parts. First, a log of the ongoing calculation is being displayed, which shows what is calculated at the moment and also if this part of the calculation is executed successfully or not. It is important to mention at this point, that the resulting collection surfaces are not necessarily correct although every calculation step was completed successfully. This log shows only rough errors, which should not happen in the first place and assumptions which must be made because of incomplete data. This has the convenient side effect to discover missing data. The information concerning missing data can be forwarded to the responsible organization.

The second step is only sensible, if the first step has been completed without any errors and a very little number of assumptions[5] have been made. If that is the case, the data may be exported in a KML file and displayed with the help of Google Earth. As the used airspaces, can also be displayed in Google Earth, it is a very easy and an effective way to discover errors. Google Earth also comes with the big benefit, that it is a well renowned software, which is for several years on the market and is used by various organizations. As well as it is desirable to have a complete different software to assess the result, then to come up with an own development, which might cover internal errors.

If this software is used for its dedicated purpose, it is highly recommended to export the collection surfaces in the standardized AIXM 4.5 format to verify the data with existing, especially for aeronautical data developed software's.

[5] Assumption does not mean guessing, but that the data has been completed as a certain regulation would specify it. Because there are a lot of special cases, this is just an assumption and not audited aeronautical data.

5.4 Post processing software

The development of this thesis focused on a post processing software, which is as little and simple as possible. As already mentioned, the post processing software should be very fast and very stable at the same time. These objectives are best met, if the preprocessing takes care of most of the calculations, so that the post processing does not need to process the data any further.

5.4.1 Data pre-treatment

To meet the postulated requirements a treatment of the collection surfaces would be necessary. The idea is to create a mesh, where all collection surfaces have been evaluated, so that there is a consistently mesh all over a specific FIR, instead of having an array of polygons without any assignment. This approach is supposed to raise very little possibility for errors and provides a fast way to handle requests.

There are two different approaches possible. First, a mesh created out of triangles would guarantee a very little error compared to the original collection surfaces as all plane and terrain following surfaces could be met exactly and only the cone shaped surfaces would lead to an error depending on the triangle size. Because a lot of numerical methods (e.g. FEM, CFD, …) need to provide a mesh for the calculation, there are well developed algorithm for creating a triangle mesh available (e.g. Delaunay Triangulator [8]). A drawback of the triangulation of a surface is the limited assignment of the triangles to their location, which would require to bundle them for a fast access.

The second way is to use a structured mesh, which comes with the big benefit of a direct assignment of each node of an element, which does not need any searching for the matching element. The big drawback of a structure grid is that the height of a point in an element must be interpolated, which leads to an undefined error. The mesh needs to be refined on the boundary of the collection surface to follow the surface border better. The other way is clip the boundary with the concerning element, but this would lead to general shaped polygons at the collection surface boundary, which represents a total discrepancy to the mentioned advantages of a structured mesh.

An unstructured mesh with irregular polygons combines the drawbacks of the already mentioned tessellation techniques.

After assessing the advantages and disadvantages of the two different approaches to create a mesh, none of the two methods showed a vast benefit compared to the other

one without having a big drawback itself. In an addition to this tie of approaches, one must consider, that the additional calculation effort causes room for errors. Furthermore, does the verification of the correctness and accuracy of the mesh require a new tool, then displaying these meshes in Google Earth. The illustration leads to an overcrowded picture, with only very little chance of discovering errors. Therefore, the pretreatment of the data has been cut out.

5.4.2 Terrain elevation

Depending on the kind of collection surface, the local terrain elevation is necessary to calculate the permitted height of an object. One can see, that surfaces which use absolute definition of height need a local elevation of the terrain to determine the distance in between. The relative definitions have the convenient benefit not to require any additional data.

This demands that every request contains not only the coordinates of the position but also the terrain elevation. If it is not possible to obtain the terrain elevation of a to be verified position, the post processing software needs a tool to make a sensible assumption of the terrain elevation in this area.

This can be accomplished by using the SRTM data. These DB holds terrain elevation data for wide parts of the earth. The data is split into binary files, where the name of the file reveals the covered area. There are two accuracy modes available. One supports a 90 m grid and the second one a 30 m grid, although the 90 m grid covers more of the earth than the 30 m grid. This DB enables a rough supposition of the local terrain elevation. One may see, that for a more accurate result, an exact elevation is necessary.

5.4.3 Applying collection surface for classifying objects

The mandatory parameters for a request are only the coordinates. To increase the accuracy the exact elevation can be added to the request. The post processing software works in two steps. First, the position of the requested point is compared with all polygons in the concerning FIR. If the point lies in a polygon, this specific polygon is marked. Important to mention that a point can lie in several polygons at the same time, so that every matching polygon must be verified afterwards. In the second step the area coefficients are used to calculate the maximum permitted height of the object for each polygon. The lowest height is being returned.

Tests have shown, that this approach is very fast[6] without having any meshed data available. One of the reasons for the fast verification is the smart point in polygon check, which would cause normally a very big calculation effort, but by a preselection based on the bounding rectangle of each polygon, the effort can be minimized for most of the checks.

Furthermore, this approach does not cause any additional numerical errors as it is the case for the mentioned triangle and the structured mesh procedure.

[6] The FIR of Austria with the applied ICAO ANNEX 15 rule set has been used for testing. The time between a request and the return of the verified distance on a mobile device is below one second.

6 Results

This section should illustrate the developed software with the help of two different rule sets. An international rule set (ICAO ANNEX 15) and a national rule set (ZFV § 35 - 42) have been chosen to demonstrate the developed solution. The international rule set has been applied for most of the testing, as the description of the rule set is easier to interpret for loans and it is more complex regarding the dependencies between airspace structure and the modelled surfaces.

6.1 ICAO Annex 15 regulation

In Addenda 8.1 a short summary of the ICAO ANNEX 15 regulation is described. This regulation focuses on the restriction areas for obstacles. These collection surface consists out of four areas, whereat the Area 2 is subdivided in the areas Area 2a, Area 2b, Area 2c and Area 2d. The Area 2 also represents the most complex area for modelling, why this area has been chosen for testing the toolbox. Furthermore, it is the most important area of all four regarding the restrictions causing by this area.

With respect to legal matter, as the interpretation of regulations and applicable law has been not subject of this thesis, there is no claim of correctness regarding the modelled collection surfaces.

6.1.1 Script

The script represents the work bench on which the regulations can be modelled. The custom-made script for each regulation is being executed by the program similar to other script languages (e.g. Python, …). The following code extractions should illustrate the process of modelling the collection surfaces.

Area 2a:
A new polygon `area2ap` respectively a new corridor `area2a` is defined. Also, the three reference points `thr1`, `thr2`, and `arp`, which most of the collection surfaces depend on are being initialized. The points `thr1` and `thr2`, are representing the position of the thresholds of the particular runway. The point `arp` indicates the airport reference point.

```
var area2a = new corridor();
var area2ap = new polygon();
```

```
var thr1 = new vertex();
var thr2 = new vertex();
var arp = new vertex();

var dispTres1 = 0;
var dispTres2 = 0;
var vertices = [];
```

By using the setLineVertex(…) function the attributes of the vertices are being set. The array `airports` holds the result of the data acquiring process. This part of the script is for every item in the array `airports` executed, so that the relevant data can be assigned to the three reference points for every item in the airports array.

```
arp.setLineVertex('arp', airports[ab].attributes.geoLong, air-
ports[ab].attributes.geoLat, dataFunc-
tions.getMeters(airports[ab].attributes.valElev, air-
ports[ab].attributes.uomElev), 0);

thr1.setLineVertex('thr1', airports[ab].attrP1[ac].rdn[0].geoLon, air-
ports[ab].attrP1[ac].rdn[0].geoLat, dataFunc-
tions.getMeters(airports[ab].attributes.valElev, air-
ports[ab].attributes.uomElev), 0);

thr2.setLineVertex('thr2', airports[ab].attrP1[ac].rdn[1].geoLon, air-
ports[ab].attrP1[ac].rdn[1].geoLat, dataFunc-
tions.getMeters(airports[ab].attributes.valElev, air-
ports[ab].attributes.uomElev), 0);
```

As the Area 2a covers the whole runway, the displacement of the thresholds need to be considered with the values `dispTres1` and `dispTres2`.

```
dispTres1 = dataFunc-
tions.distanceInNm(airports[ab].attrP1[ac].rdn[0].xt_valDispTres, air-
ports[ab].attrP1[ac].rdn[0].xt_uomDispTres);

dispTres2 = dataFunc-
tions.distanceInNm(airports[ab].attrP1[ac].rdn[1].xt_valDispTres, air-
ports[ab].attrP1[ac].rdn[1].xt_uomDispTres);
```

The start and end point of the runway are getting calculated with the following function, which is a part of this script and introduced to provide a more readable script.

```
vertices = getPointsOfArea2a(thr1, thr2, dispTres1, dispTres2);
```

Afterwards the Area 2a corridor can be set with the pre-calculated points. The width of the corridor is also obtained from the acquired data.

```
area2a.setCorridor(0, 'area2a', vertices[0], vertices[1], 0, dataFunc-
tions.getMeters(airports[ab].attrP1[ac].valWidStrip, air-
ports[ab].attrP1[ac].uomDimStrip));
```

As it is easier for the following modelling process to work with a polygon, the corridor is being converted to the polygon `area2ap`. The attribute height in the polygon object represents the maximum permitted height of an obstacle without requiring a verification.

```
var vertices = area2a.getCooridorCoordinates(2);
area2ap.vertices = vertices;
area2ap.name = 'area2a';
area2ap.height = 3;
area2ap.setAreaCoefficients();
```

In the last step, the Area 2a polygon is added to the VertexObjects object `area2` by considering that areas where flying is prohibited need to be excluded of the area. The array `airspaces` holds all acquired airspaces of this region and the string `airspaceString` defines the to be excluded types of airspaces.

```
area2.addPolygonArray(clipping.cut(area2ap, dataFunc-
tions.filterAirspaces(airspaces, airspaceString)));
vertices = area2a.getCooridorCoordinates(1);
```

Area 2b:

The Area 2b is the next surface, which is being modelled. As it depends directly on the Area 2a, the calculated vertices of Area 2a are being used as reference for the Area 2b. The Area 2b needs to be subdivided in six subareas. Firstly, because there is on both sides of the runway an Area 2b and secondly each area consist of two cone shaped surfaces and a plane shaped surface. The first area is the plane shaped, rectangle area in the middle, which uses the two corner points `vertices[0]` and `vertices[3]` of the Area 2a as reference. The other two points of the rectangle are positioned 10,000 m away in runway direction and 120 m above the reference points `vertices[0]` and `vertices[3]`. The maximum permitted height of an object in this area, without requiring a verification is 3 m above the ground level.

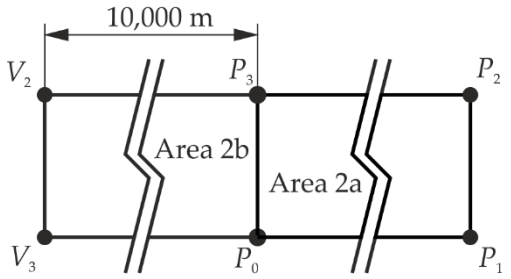

Figure 32: Area 2b

Figure 32 shows the existing Area 2a with the four, counter clockwise ordered vertices P_0, P_1, P_2 and P_3. The first two points to add for the Area 2b are P_0 and P_3. The next

vertices are added by using the addRelativeLineVertex(…) function. It is important to mention, that the order in which the points are being added needs to be either clockwise or, preferred, counter clockwise.

```
var area2b_11 = new polygon();
area2b_11.setPolygon(0, 'area2b_11', 3, [])

area2b_11.addVertex(vertices[0]);
area2b_11.addVertex(vertices[3]);
area2b_11.addRelativeLineVertex(vertices[3], vertices[2], 0, dataFunc-
tions.distanceInNm(10000, 'm'), 120, 1);
area2b_11.addRelativeLineVertex(vertices[0], vertices[1], 0, dataFunc-
tions.distanceInNm(10000, 'm'), 120, 1);
area2b_11.setAreaCoefficients();
```

Before adding the subarea to the VertexObject `area2` the for flight prohibited areas get excluded of the area.

```
area2.addPolygonArray(clipping.cut(area2b_11, dataFunc-
tions.filterAirspaces(airspaces, airspaceString)));
```

The adjacent part works similar to the just explained part of the Area 2b, except of the difference that the surface is cone shaped. To obtain this kind of surface the addRelativeArcVertex(…) function is being used.

```
var area2b_12 = new polygon();
area2b_12.setPolygon(1, 'area2b_12', 3, [])

area2b_12.addVertex(vertices[0]);
area2b_12.addRelativeArcVertex(vertices[0], vertices[1], 0, dataFunc-
tions.distanceInNm(10000, 'm'), 120, 1);
area2b_12.addRelativeLineVertex(vertices[0], vertices[1], -xt_Math.getAngle(15,
100), dataFunctions.distanceInNm(10000, 'm'), 120, 1);
area2b_12.setAreaCoefficients();

area2.addPolygonArray(clipping.cut(area2b_12, dataFunc-
tions.filterAirspaces(airspaces, airspaceString)));
```

The other four missing parts of the Area 2b are being modelled in the same way as explained for the first two parts.

Area 2c:

The Area 2c does not need to be described in detail as the only difference between the Area 2b and Area 2c lies in the fact that the maximum permitted height for an obstacle, without requiring a verification, is 15 m instead of 3 m. Other than that, the procedure is similar to the Area 2b.

Area 2d:

This area uses a relative definition of the height compared to the Area 2a, Area 2b and Area 2c, which are using an absolute definition. The basic collection surface is de-

scribed by a circle with a radius of 45,000 m. The circle can be added to the `area2d` polygon with the addCircle(…) function. The Area 2d collection surface differs to the other areas in the way, that it is the result of the intersection with the specific airspace type TMA and MTMA and the prior to this created circle. The for flight prohibited areas are being also excluded of the intersection result.

```
var area2d = new polygon();
area2d.setPolygon(1, 'area2d', 120, [])
area2d.addCircle(arp, dataFunctions.distanceInNm(45000, 'm'), 0, 2);
area2d.setAreaCoefficients();

var result = clipping.intersection(area2d, dataFunc-
tions.filterAirspaces(airspaces, 'TMA,MTMA'));

var result1 = clipping.cut(result, dataFunctions.filterAirspaces(airspaces, air-
spaceString))

area2.addPolygonArray(result1);
```

The so calculated and afterwards collected collection surfaces in the `area2` object are going to be exported in a JSON file and also post processed in the Kml class to obtain a KML file, which can be displayed in the Google Earth program.

6.1.2 Google Earth

This subsection should illustrate the results of the prior to this created script. The script is held generic, so that it could be applied for any region. To get specific results the FIR of Austria (LOVV) has been chosen for application.

Figure 33 shows the Area 2d of the airport LOWW in Vienna. It is clearly recognizable how the airspace structure impacts the final collection surface. The structure of the TMA airspaces combined with the 45,000 m in radius big circle are defining the outer boundary of the Area 2d. The border of the country derives also from the TMA airspaces, which are limited by state border. The exclusions derive from the prohibited areas, in this case mostly of the type restricted airspace.

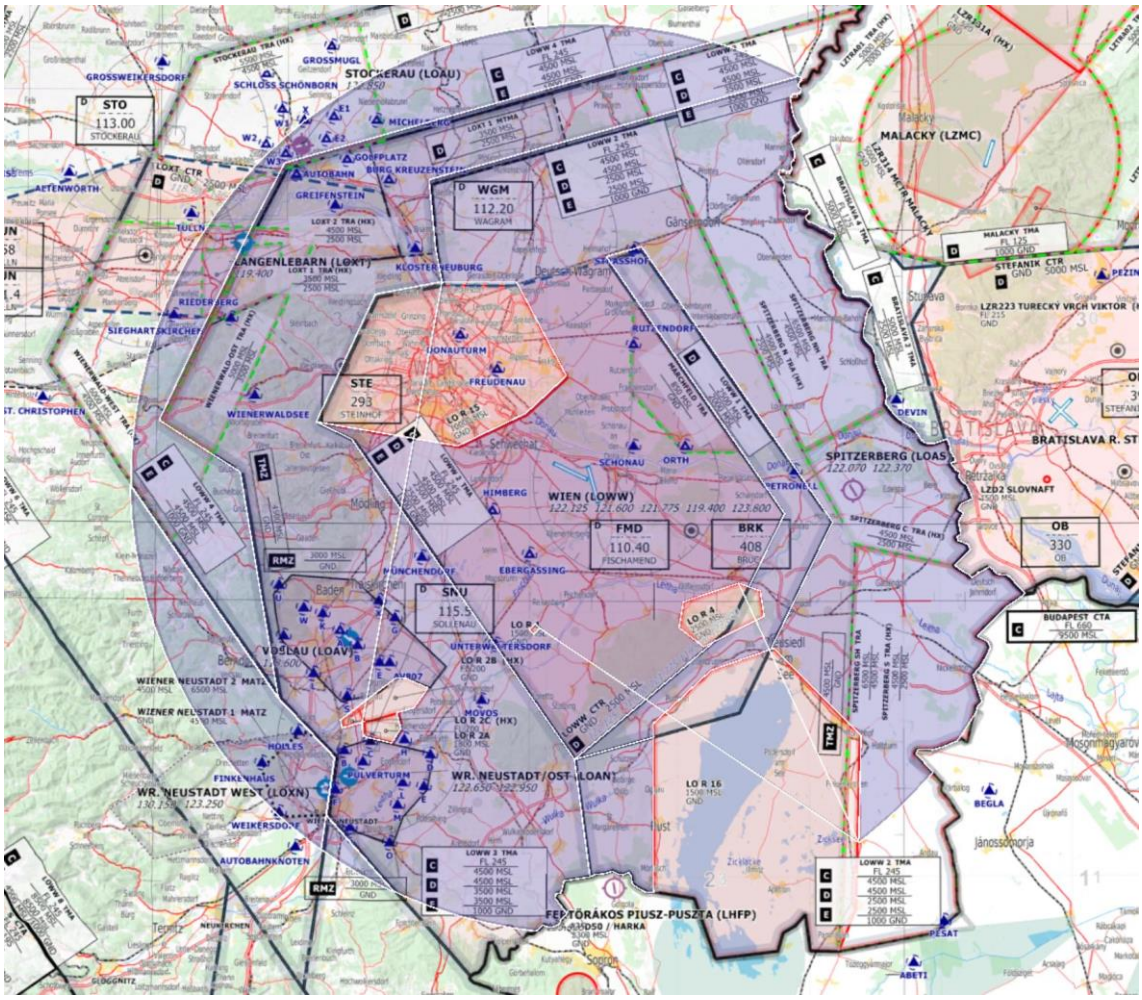

Figure 33: Area 2d – LOWW (Vienna)

Figure 34 shows the Area 2a, Area 2b, and Area 2c. The described script has been applied for both runways. One can see, that the restricted area around the town has been excluded of the collection surface as defined in the script.

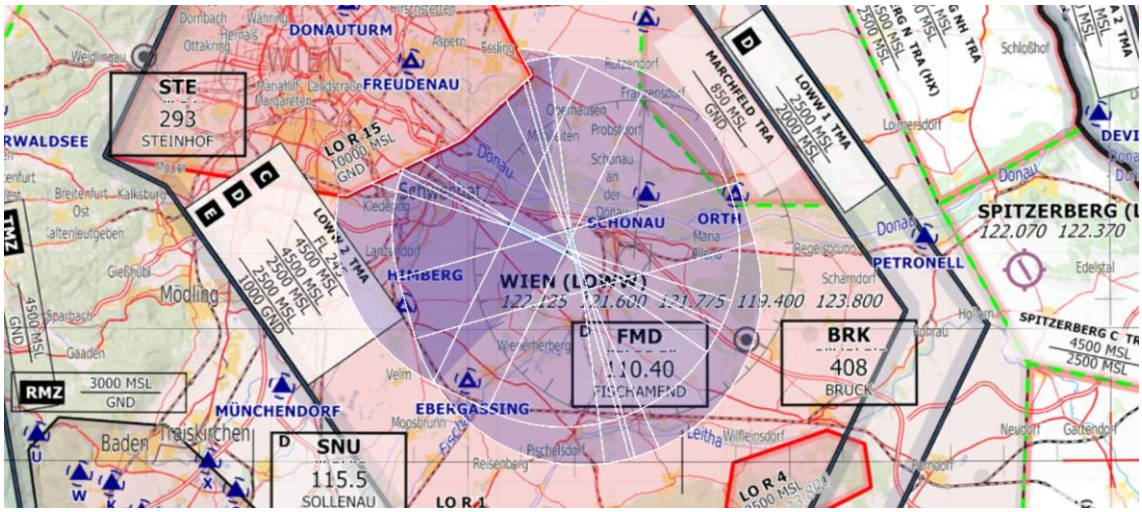

Figure 34: Area 2a, 2b, and 2c – LOWW (Vienna)

In the Figure 35 is the collection surface around the airport LOWG displayed. The brighter oval in the middle represents the Area 2a, Area 2b and Area 2c. The Area 2d is not excluded by these areas.

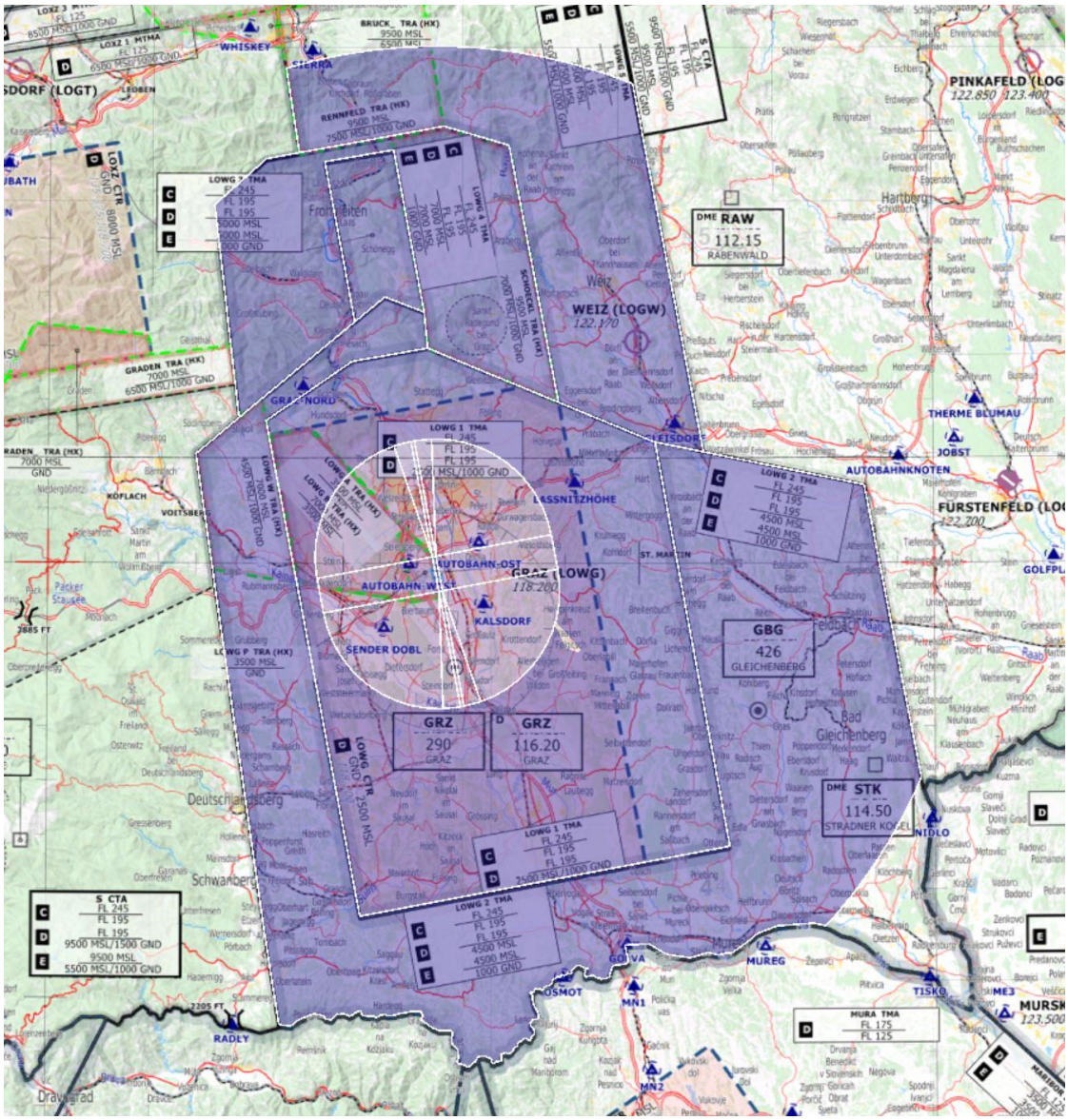

Figure 35: The whole Area 2 – LOWG (Graz)

Figure 36 shows the variegated possibilities of Google Earth. With the help of the displayed terrain elevation, it is possible to assess the results of the obstacle validation and the correctness of the collection surfaces itself. Of course, only plausibility checks are feasible.

Figure 36: Northern boundary of the Area 2b – LOWG (Graz)

6.2 ZFV protection area

In Addenda 8.2 the directly enforceable law is enclosed. This paragraph handles the protection zones of an airport. This zone consists of several surfaces around the runway.

6.2.1 Script

The basic structure of this script is similar to the one described in Subsection 6.1.1. However, there are differences. The tables illustrated in in Addenda 8.2 are included in the script as a function, which returns the values of the different areas depending on the runway class. This shows who the different regulations can be easily implemented, although there are very detailed specifics to consider.

6.2.2 Google Earth

Figure 37 shows the protection area of the airport LOWS (Salzburg). The two trapezoid areas in direction of the runway are the so called Anflugsflächen. These two areas are connected to the Sicherheitsstreifen, which is the rectangle area covering the whole runway. The surfaces to left and right of the runway are the Übergangsflächen, which are enclosed by the Anflgusflächen. The inner circle is the Horziontalfläche. This surface is plane shaped. The surface between the inner and outer circle is called Kegelfläche and is shaped like a cone.

This example illustrates the working combination of aeronautical data and pre-defined surfaces, which are depending on the class of the runway.

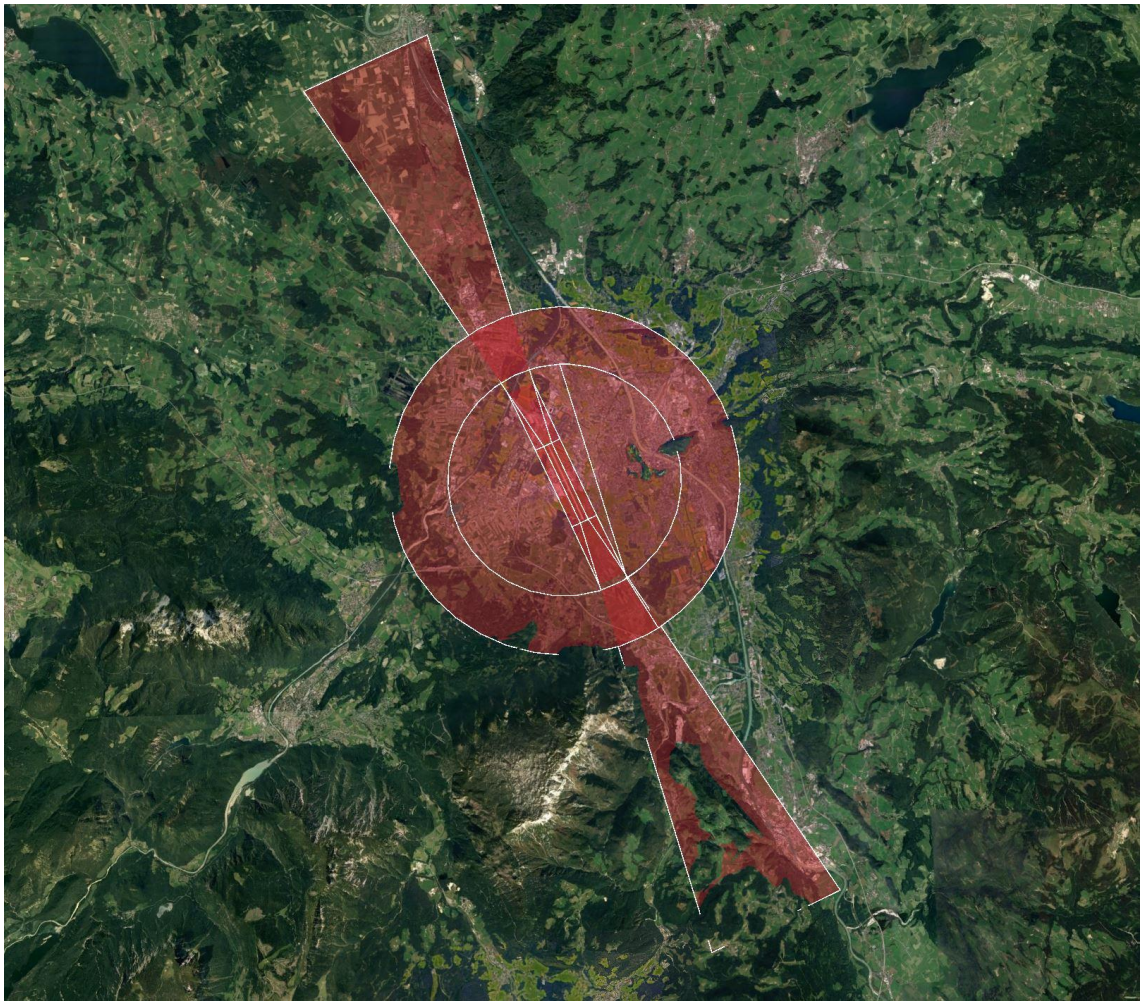

Figure 37: ZFV protection area – LOWS (Salzburg)

Figure 38 shows the protection area of the airport LOIH (Hohenems). One can distinguish the differences between the two protection areas very easily, which have been calculated fully automatic.

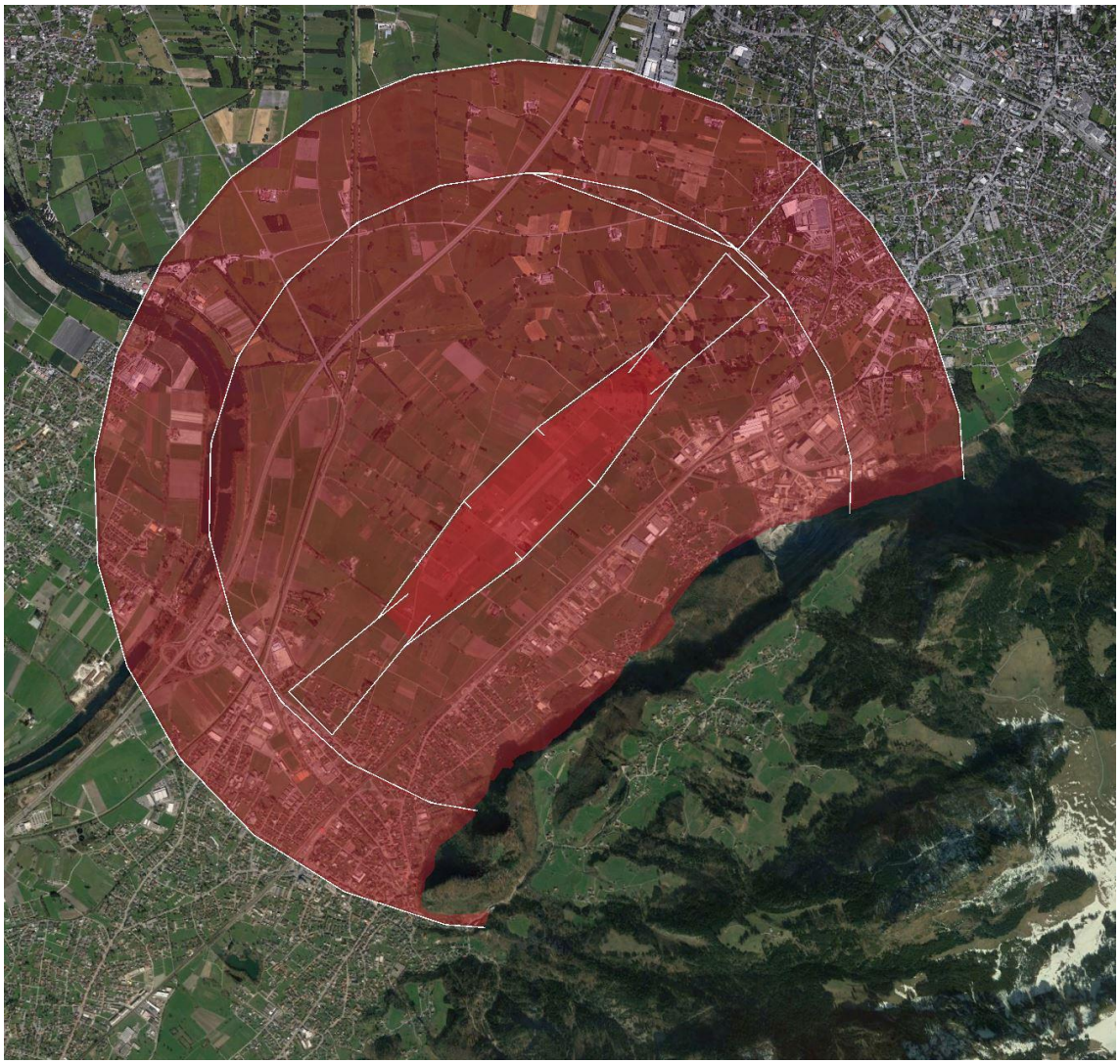

Figure 38: ZFV protection area – LOIH (Hohenems)

7 Conclusion

Prior to this thesis, a number of experts have been consulted to discuss the problematics of this thematic and to define the must haves of to be developed toolbox. A very detailed target specification has been the result of these meetings.

Based on this specification a basic software architecture of the toolbox has been developed. On the one hand, important class dependencies have been defined, but on the other hand the assigned tasks of these classes have been vague enough to have plenty of space to respond to changing requirements due to hidden problematics.

A rather big issue has been the implementation of the toolbox for the user. Either a toolbox would have been developed, which uses an abstract language that would have been interpreted by the developed code, or a solution which would have executed directly the created script. The second option has been chosen for the developed solution. Firstly, the user has a wider range of possibilities to implement given regulations and to consider special cases. Secondly, this approach is used by other well-known software's, which underlines the practical and convenient application.

The technical and mathematical basics, which this thesis is based on, are well described in many text books and have been therefore not much of an effort to implement. A more complex issue has been the transformation of solutions, which have been developed for a plane surfaces and are required to work on a spherical surface. As the collection surfaces, can reach very wide dimensions, a partial plain assumption of the surface would have led to noticeable errors. One of the most difficult task has been to develop a polygon clipping algorithm, which works on spherical shaped surface. Not only the transformation has led to different kinds of problems, but also to develop a more accurate algorithm than the ones that are published.

The last step has included to find a capable tool for testing the results. The capabilities of Google Earth (Pro) meet the requested requirements as a testing environment. For more advanced testing a software particularly for aviation designed should be used.

8 Addenda

8.1 Summary ICAO ANNEX 15, Chapter 10 - Electronic Terrain and Obstacle Data

General information

Subdivision of the coverage area.

- Area 1 – the entire territory of a state
- Area 2 – the area around an aerodrome divided in
    - Area 2a
    - Area 2b
    - Area 2c
    - Area 2d
- Area 3 – the area bordering an aerodrome movement area
- Area 4 – the area extending prior to the runway threshold

Terrain data collection surfaces

Within the area covered by a 10 km radius from the ARP, terrain data shall comply with the Area 2 numerical requirements. In the area between 10 km and the TMA boundary or 45 km radius, whichever is smaller, data on terrain that penetrates the horizontal plane 120 m above the lowest runway elevation shall comply with the Area 2 numerical requirements. The terrain which does not penetrate the horizontal plane 120 m shall comply with the Area 1 numerical requirements. In those portions of Area 2 where flight operations are prohibited, terrain data shall comply with the Area 1 numerical requirements.

Obstacle data collection surfaces

Obstacle data shall be collected and recorded in accordance with the Area 2 numerical requirements.

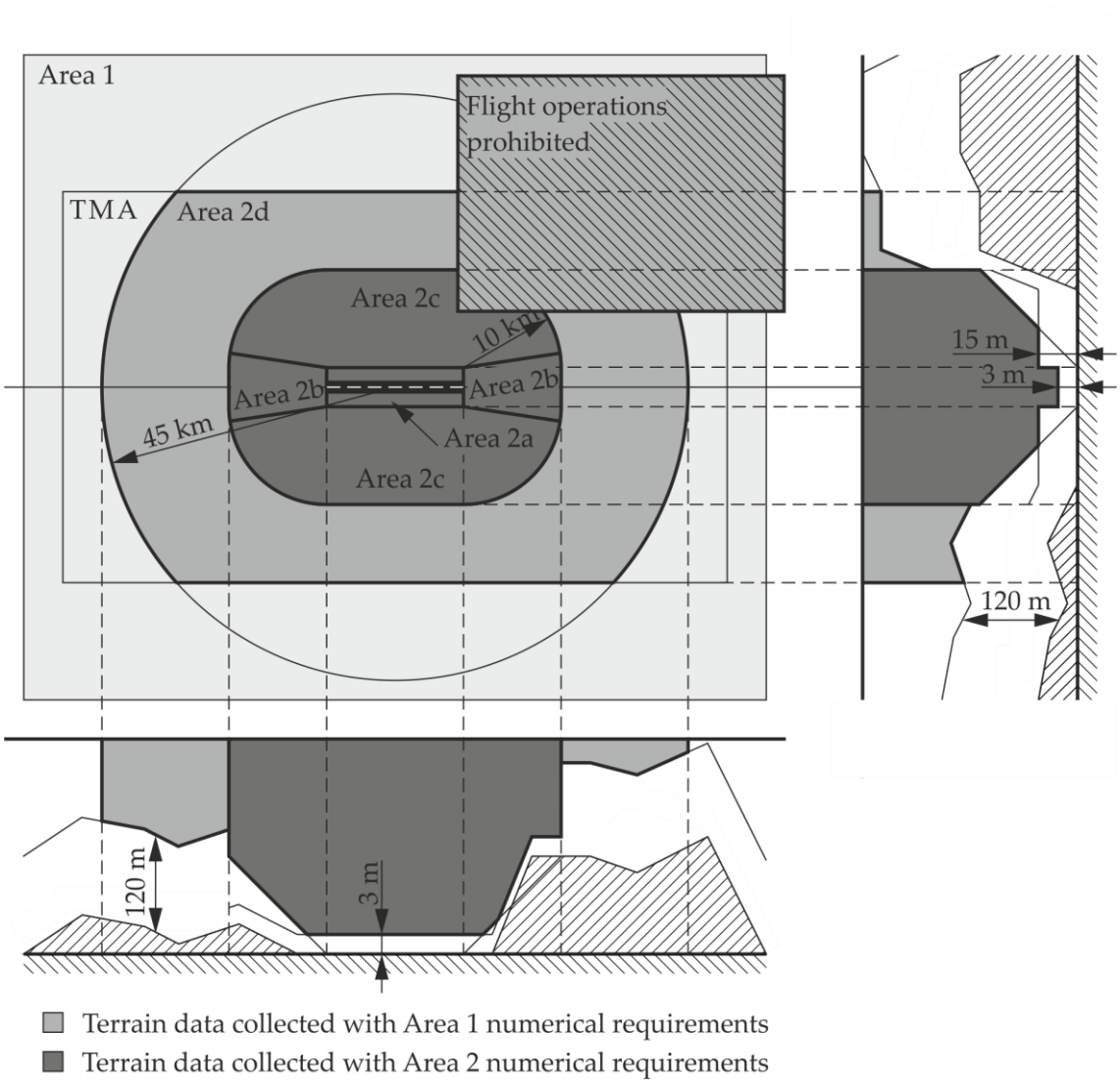

Figure 39: Obstacle data collection surfaces

Area 2a:

A rectangular area around a runway that comprises the runway strip plus any clearway that exists, as seen on Figure 2. The Area 2a obstacle collection surface shall have height of 3 m above the nearest runway elevation measured along the runway center line and for those portions related to a clearway, if one exists, at the elevation of the nearest runway end.

Area 2b:

An area extending from the ends of Area 2a in the direction of departure, with a length of 10 km and a splay of 15% (8.5°) to each side. The Area 2b obstacle collection surface has a 1.2% (0.68°) slope extending from the ends of Area 2a at the elevation of the run-

way end in the direction of departure. Obstacles less than 3 m in height above ground need not to be collected.

Area 2c:
An area extending outside of Area 2a and 2b at a distance of less than 10 km from the boundary of Area 2a. The Area 2c obstacle collection surface has a 1.2% (0.68°) slope extending from the ends of Area 2a and 2b at a distance of less than 10 km from the boundary of Area 2a. The initial elevation of Area 2c shall be the elevation of the point of Area 2a at which it starts. Obstacles less than 15 m in height above ground need not be collected.

Area 2d:
An area outside the Areas 2a, 2b and 2c up to a distance of 45 km from the aerodrome reference point, or to an existing TMA boundary, whichever is nearest. The Area 2d obstacle collection surface has a height of 100 m above ground.

In those portions of Area 2 where flight operations are prohibited, obstacle data shall be collected and recorded in accordance with Area 1 requirements.

Data on every obstacle within Area 1 whose height above the ground is 100 m or higher shall be collected and recorded in the database in accordance with the Area 1 numerical requirements.

Terrain and obstacle data collection surface – Area 3
An area extending 90 m from the runway center line in both sides or 50 m from the edge of all parts of the aerodrome movement area. The data collection surface for terrain and obstacles extends 0.5 m above the horizontal plane passing through the nearest point of the aerodrome movement area.

Terrain and obstacle data in Area 3 shall comply with the numerical requirements from Area 3.

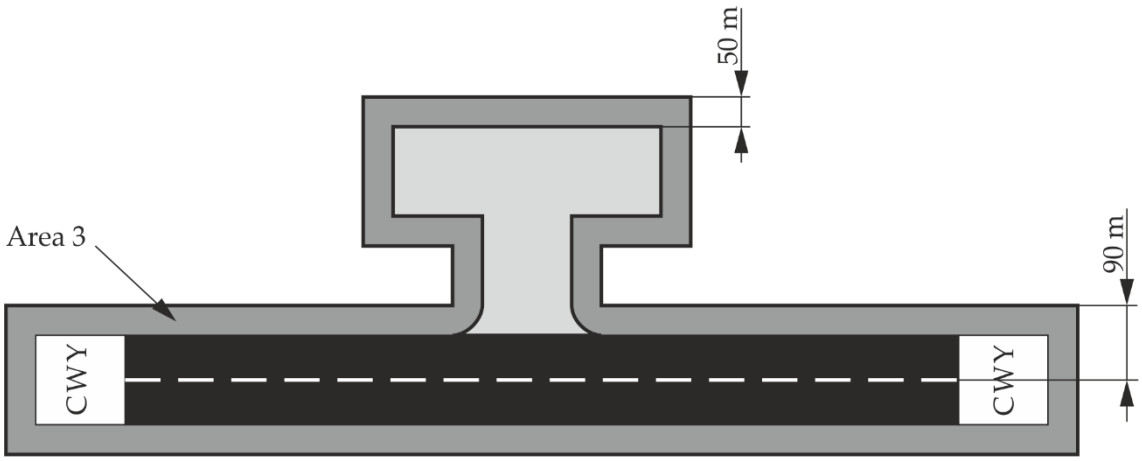

Figure 40: Area 3

Terrain and obstacle data collection surface – Area 4

The area extending 900 m prior of the runway threshold and 60 m each side of the extended runway center line in the direction of the approach on a precision approach runway, category II or III.

Recommendation: Where the terrain at a distance greater than 900 m from the runway threshold is mountainous or otherwise significant, the length of Area 4 should be extended to a distance not exceeding 2000 m from the runway threshold.

Terrain and obstacle data in Area 4 shall comply with the numerical requirements from Area 3.

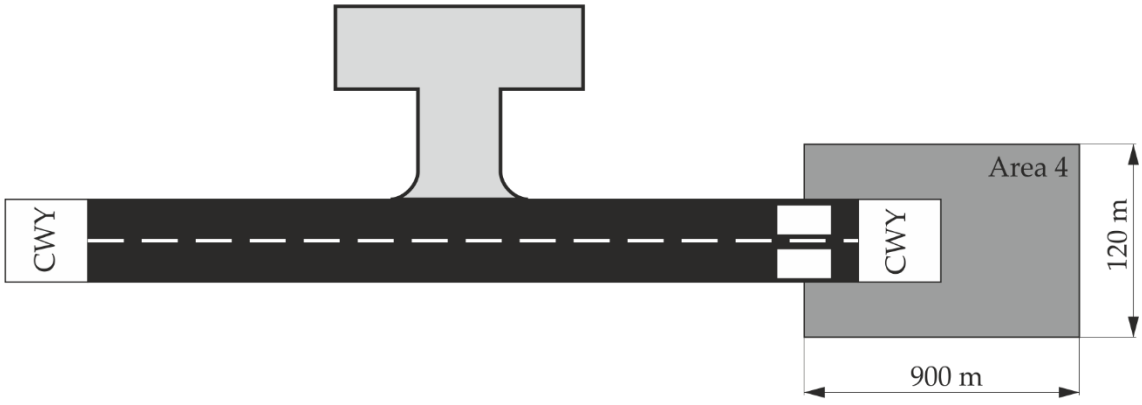

Figure 41: Area 4

8.2 ZFV - III. Teil: Schutzbereich und Freiflächen

1. Abschnitt - Schutzbereich

§ 35. Allgemeines

(1) Zivilflugplätze dürfen nur betrieben werden, wenn der Schutzbereich der für den An- und Abflug bestimmten Bewegungsflächen frei von Hindernissen ist, welche die Sicherheit der Luftfahrt beeinträchtigen, oder wenn durch Beseitigung solcher Hindernisse oder durch ihre Kennzeichnung beziehungsweise Befeuerung die Sicherheit der Luftfahrt gewährleistet wird.

(2) Als Hindernis gelten insbesondere Bauwerke, Bäume, Sträucher, verspannte Seile und Drähte sowie Bodenerhebungen, soweit sie folgende, den Schutzbereich nach unten begrenzende Flächen (Grenzflächen) überragen:

a) die Erd- beziehungsweise Wasseroberfläche im Bereich der für den Start und für die Landung bestimmten Bewegungsflächen,

b) die Erd- beziehungsweise Wasseroberfläche des Pistenvorfeldes in den Anflugsektoren, und zwar:

aa) bei Präzisionsanflugpisten der Kategorie II und III innerhalb von 1000 m, bei anderen Pisten der Klassen A, B und C innerhalb von 300 m vom Ende des Sicherheitsstreifens beziehungsweise vom Ende der Wasserpiste,

bb) bei Pisten der Klassen D, E, F und Flugfeldern ohne Pisten innerhalb von 150 m vom Ende des Sicherheitsstreifens,

c) die Anflugflächen,

d) die Übergangsflächen,

e) die Horizontalfläche und

f) die Kegelflächen

(3) Als Hindernis gelten in den im Abs. 2 lit. a und lit. b bezeichneten Bereichen außerdem Gruben, Kanäle und ähnliche Bodenvertiefungen. Ferner gelten in diesen Bereichen Verkehrswege, welche während des Flugbetriebes nicht gesperrt werden können, als Hindernis mit jener Höhe, welche darauf verkehrende Fahrzeuge maximal aufweisen.

§ 36. Darstellung des Schutzbereiches

Der Schutzbereich ist nach dem Muster der Anlage 3 (Anm.: Anlage nicht darstellbar) in einem geeigneten Plan (bei Flugfeldern womöglich ein Katasterplan) maßstabgetreu darzustellen. In diesem Plan sind die Hindernisse im Sinne des § 35 einzutragen und deren größte Höhe über dem mittleren Meeresspiegel anzugeben.

§ 37. Verpflichtung zur Meldung von Hindernissen

(1) Die Grundrisse der Anflugsektoren müssen trapezförmig sein. Die kleine Parallelseite des Trapezes muss mit den Enden des Sicherheitsstreifens beziehungsweise bei Wasserpisten mit dem Pistenende zusammenfallen.

(2) Bei Landflugplätzen, mit Ausnahme von Hubschrauberplätzen, müssen die Grundrisse der Anflugsektoren folgende Ausmaße aufweisen:

Table 1: Trapezoid dimensions depending on runway class

	Kleine Parallelseite	Große Parallelseite	Trapezhöhe
Bei Instrumentenpisten der Klassen A, B und C	300 m	4.800 m	15.000 m
Bei anderen Pisten der Klassen A, B und C	180 m	3.180 m	12.000 m
Bei Pisten der Klasse D	80 m	580 m	2.500m
Bei Pisten der Klasse E	60 m	380 m	1.600 m
Bei Pisten der Klasse F	60 m	300 m	1.200 m
Bei Landeflächen für Segelflugzeuge	50 m	200 m	500 m

(3) Bei Hubschrauberplätzen müssen die Grundrisse der Anflugsektoren aufweisen:

Table 2: Trapezoid dimensions depending on helipad class

	Kleine Parallelseite	Große Parallelseite	Trapezhöhe
Bei Pisten der Klasse A	60 m	660 m	2.000m
Bei Pisten der Klasse B	40 m	490 m	1.500 m

Bei Pisten der Klasse C	20 m	320 m	1.000 m

(4) Flugfelder ohne Pisten müssen an ungefähr gegenüberliegenden Seiten zwei Anflugsektoren aufweisen, wobei die Mittellinien der Grundrisse der Anflugsektoren miteinander keinen größeren Winkel als 20 Grad einschließen dürfen. Die Bestimmungen des Abs. 2 gelten mit der Maßgabe sinngemäß, daß anstelle der Pistenklasse der Durchmesser des Flugfeldes zwischen den Anflugsektoren zugrunde zu legen ist.

(5) Bei Wasserflugplätzen müssen die Grundrisse der Anflugsektoren folgende Ausmaße aufweisen:

Table 3: Trapezoid dimensions depending on water runway

	Kleine Parallelseite	Große Parallelseite	Trapezhöhe
Bei Instrumentenpisten	300 m	4.800 m	15.000 m
Bei der Klasse A	225 m	825 m	3.000 m
Bei der Klasse B	180 m	780 m	3.000 m
Bei der Klasse C	150 m	750 m	3.000 m

§ 39. Anflugflächen

(1) Die Grundrisse der Anflugflächen müssen mit den Grundrissen der Anflugsektoren zusammenfallen, wobei als Basis der Anflugfläche eine horizontale Gerade in der Höhe der Schwelle über dem mittleren Meeresspiegel gilt.

(2) Die Neigung der Anflugflächen darf nicht übersteigen:
2,0% bei Instrumentenpisten sowie bei Wasserpisten der Klassen A und B,
2,5% bei anderen Landpisten der Klassen A und B,
3,3% bei Land- und Wasserpisten der Klasse C,
4,0% bei Landpisten der Klasse D,
5,0% bei Landpisten der Klassen E, F, Hubschrauberpisten der Klasse A und Landeflächen für Segelflugzeuge,
10,0% bei Hubschrauberpisten der Klassen B und C.

(3) Würde eine Anflugfläche von einem Hindernis überragt werden, welches die Sicherheit der An- und Abflüge gefährdet, dann ist die Schwelle soweit gegen die Pis-

tenmitte zu versetzen, dass die auf die versetzte Schwelle bezogene Anflugfläche von keinem Hindernis überragt wird. Ist der vor der versetzten Schwelle liegende Pistenteil als Sicherheitsstreifen gemäß § 23 nicht benützbar (zum Beispiel schadhafte Oberfläche), so muss die Schwelle im Ausmaß des festgelegten Sicherheitsstreifens entsprechend weiter pisteneinwärts versetzt werden.

§ 40. Übergangsflächen

(1) Die Neigung der Übergangsflächen, gemessen in der Vertikalebene senkrecht zur Pistenmittellinie, darf nicht größer sein als:
14,3% bei Land- und Wasserpisten der Klassen A, B und C, 20,0% bei Landpisten der Klassen D und E,
25,0% bei Landpisten der Klasse F und bei Hubschrauberpisten, 30,0% bei Landeflächen für Segelflugzeuge.

(2) Bei Flugfeldern ohne Pisten sind die Übergangsflächen an die Verbindungsgeraden der Eckpunkte der Basen der Anflugflächen anzuschließen. Die Bestimmungen des Abs. 1 gelten mit der Maßgabe sinngemäß, daß anstelle der Pistenklasse der Durchmesser des Flugfeldes zwischen den Anflugsektoren zugrunde zu legen ist.

§ 41. Horizontalfläche

(1) Der Radius der Horizontalfläche, deren Mittelpunkt lotrecht über dem Flugplatzbezugspunkt 45 m über der Flugplatzbezugshöhe festzulegen ist, muß mindestens betragen:
4.000 m bei Land- und Wasserpisten der Klassen A, B und C,
2.500 m bei Landpisten der Klasse D,
2.000 m bei Landpisten der Klasse E,
800 m bei Landpisten der Klasse F und bei Hubschrauberpisten der Klasse A,
600 m bei Hubschrauberpisten der Klasse B,
400 m bei Hubschrauberpisten der Klasse C.

(2) Bei Flugplätzen mit mehreren Pisten bestimmt sich der Mindestradius der Horizontalfläche nach der Klasse der längsten Piste.

§ 42. Kegelfläche

(1) Die Kegelfläche muss eine Neigung von 5% aufweisen.

(2) Der äußere Rand der Kegelfläche muß über der Horizontalfläche in einer Höhe liegen von:

100 m bei Land- und Wasserpisten der Klassen A und B,
75 m bei Land- und Wasserpisten der Klasse C,
55 m bei Landpisten der Klasse D,
35 m bei Landpisten der Klassen E und F.

2. Abschnitt - Freiflächen

§ 43. Freiflächen

(1) Freiflächen müssen innerhalb der Flugplatzgrenzen liegen und so breit sein wie der Sicherheitsstreifen der zugehörigen Piste. Ist eine Stoppfläche vorhanden, so bildet diese einen Teil der Freifläche.

(2) Innerhalb einer Freifläche dürfen keine Bodenerhebungen oder sonstige Hindernisse eine mit 1,25% ansteigende Ebene überragen, deren Basis an das Pistenende anschließt.

9 List of figures

10 List of tables

11 List of references

[1] INTERNATIONAL CIVIL AVIATION ORGANIZATION, Annex 15, Aeronautical Information Services, Montréal: ICAO, 2010.

[2] EUROCONTROL , EUROCONTROL Terrain and Obstacle Data Manual, Brussels, 2015.

[3] L. Papula, Mathematik für Ingenieure und Naturwissenschaftler Band 3, Weisbaden: Springer Fachmedien, 2016.

[4] J. Liesen and V. Mehrmann, Lineare Algebra, Wiesbaden: Springer, 2015.

[5] M. Shimrat., Algorithm 112: Position of point relative to polygon., 1962.

[6] I. E. Sutherland and G. W. Hodgman, Reetrant Polygon Clipping, Salt Lake City: Association for Computing Machinery, 1974.

[7] G. Greiner and K. Hormann, Efficient Clipping of Arbitrary Polygons, Erlangen: Friedrich Alexander University, 1997.

[8] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator," Carnegie Mellon University , Pittsburgh, Pennsylvania, 1996.