POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO
Master of Science in Computer Engineering

Landscape Modelling for Outdoor Photography Analysis, Geolocation and Distance Estimation

Supervisor: Prof. Piero Fraternali
Assistang Supervisor: Roman Fedorov

Master Graduation Thesis by:
Lev Tverdokhlebov
id. 764590

Academic Year 2014/2015

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO
Laurea Magistrale in Ingegneria Informatica

Landscape Modelling for Outdoor Photography Analysis, Geolocation and Distance Estimation

Relatore: Prof. Piero Fraternali
Correlatore: Roman Fedorov

Tesi di laurea di:
Lev Tverdokhlebov
id. 764590

Anno Accademico 2014/2015

Abstract

This work presents a system for rendering of 360° panoramas based on real-world DEM (Digital Elevation Model) data. Given geographic coordinates as input the system generates a panorama image and also a distance map with a distance from the point of view to every pixel of the rendered panorama. The details on estimating these distances and the results of accuracy evaluation and error measurements are provided. It also discusses currently functioning and possible applications of the data generated by the system, in particular edge detection on mountain scenes and Field Of View estimation for outdoor photographs.

Sommario

Questo lavoro di tesi propone un sistema per il rendering di panorami a 360° a base di dati di DEM (Digital Elevation Model o Modello Digitale di Elevazione) del mondo reale. Date come input le coordinate geografiche, il sistema genererà un' immagine panoramica e una mappa di distanza che riporti la distanza dal punto di vista a ogni pixel del panorama renderizzato. Questo lavoro riporta anche i dettagli di stima delle distanze e i risultati di valutazione della precisione e della misurazione degli errori. Vengono altresì presentati scenari d'uso correnti e futuri per i dati generati dal sistema, in particolare il riconoscimento degli edge (bordi) per scene di montagna e la stima del FOV (Field of View o Angolo di Campo) per le fotografie esterne.

Acknowledgements

First, I would like to thank Prof. Piero Fraternali for his guidance during the preparation of this thesis. And for all the talks and meetings with him that I was participating and enjoying during my years in Politecnico as well as for the inspiration he gave me.

I would like to thank Roman Fedorov for the pleasure to work with him on this thesis.

I would like to thank Luca Galli, with whom I failed to prepare any thesis but succeeded in making friendship.

I would like to thank my wife Olga for her support and patience.

I would like to thank my sister Elena for her advice and expertise.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

With rapid development of digital cartography and GIS (geography information systems) a lot of related techniques and approaches aiming at usage of spatial data emerged. The great contribution was made by the NASA's Shuttle Radar Topography Mission (SRTM) giving the public access to highly detailed geospatial data of the whole surface of our planet. Among all the fields of application of digital elevation model (DEM) data it seems rational to highlight the field of comparison between real-life collected data and the model to be able to produce the new, previously non-existent information and conclusions.

1.1 Problem Statement

Given a geographical coordinates, the goal of this work is to build the 360° panorama as seen from the particular point defined by these coordinates.

Together with the panorama the distance mask with the distance to each point of the panorama should be delivered. The distance estimation accuracy should be evaluated against the etalon model.

1.2 Related Work

This work is related to Roman Fedorov's Master graduation thesis [1]. In his work Roman created a platform and algorithms for classification of mountain panoramas from user-generated photographs followed by identification and extraction of mountain peaks from those panoramas. An automatic technique has been developed that receives geo-tagged photograph as an input and estimates the direction of the camera using a matching algorithm on the photograph edge maps and a rendered view of the mountain silhouettes.

As a renderer Roman used an external panorama service the mountain view generator of Dr. Ulrich Deuschle (Figure 1.1; hereinafter referred to as Udeuschle generator because of the domain name udeuschle.selfhost.pro where the service resides). The service accepts as an input several parameters, most important of which are the geographic position of the observer, his altitude and altitude offset, the view direction with the field of view, zoom factor, elevation exaggeration and the range of sight. Latitude and longitude are equal obviously to the geo-tag of the input photograph, and the horizontal extension (field of view) to the round angle, 2π.

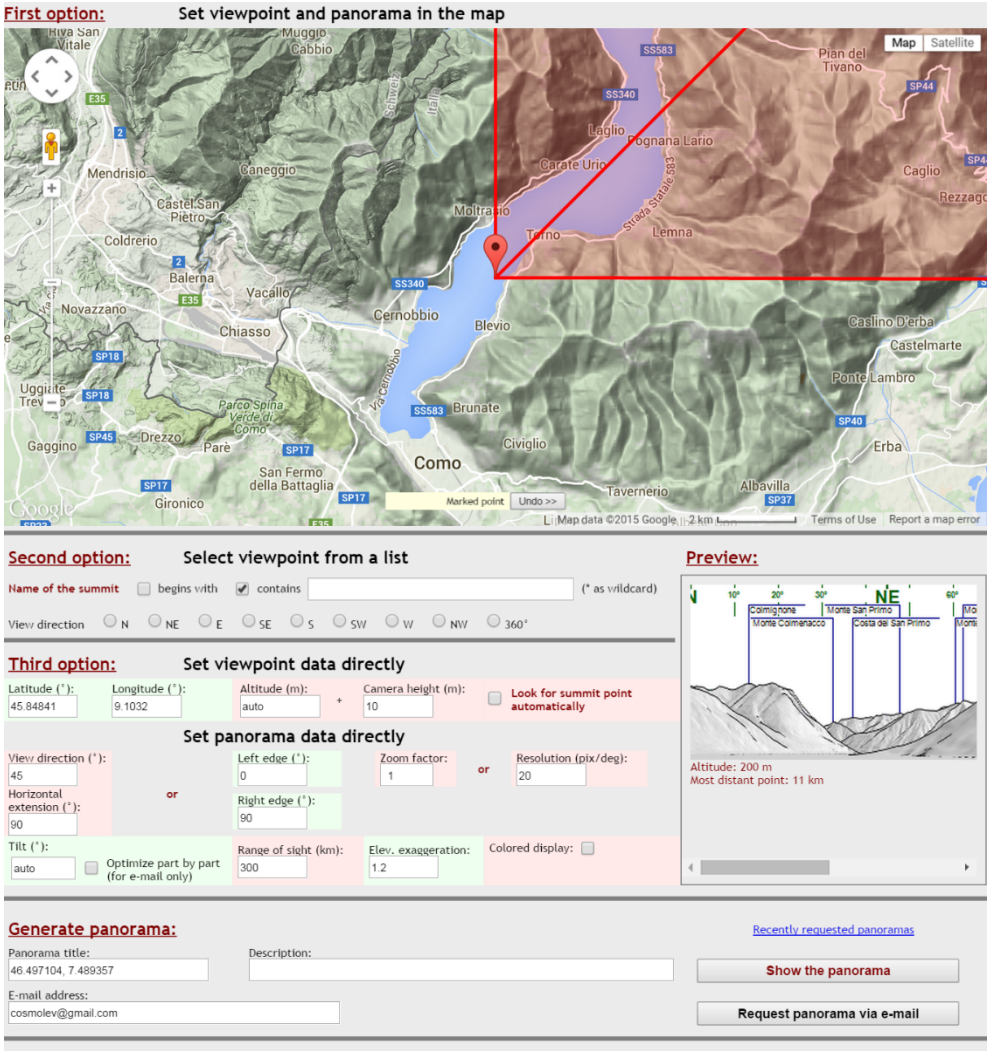

Figure 1.1: A screenshot of the mountain panorama generation web service interface (Udeuschle generator).

The usage of the external service while generally working well poses some serious limitations:

1. The generation of full 360° panorama takes more than a minute of time. In order to cope with this delay the service even uses the mechanism of asynchronous loading of separate image tiles upon completion. However that doesn't help to overcome this limitation as for the Field Of View estimation and other applications full panorama image is needed.

2. Service provides no API or any other automated-friendly way to query and receive the panorama image. Roman overcame it by reverse engineering JavaScript and AJAX scripts of the web interface, but such wise there would be no protection from any subsequent change on the server side of the service.

3. The panorama image provided by the service comes already preprocessed. In particular the contours of some mountains are already highlighted (Figure 1.2). While it's not necessarily bad it's better to have full control on the effects and filters applied to the image. For the purposes of subsequent processing it's always better to have data in as raw representation as possible.

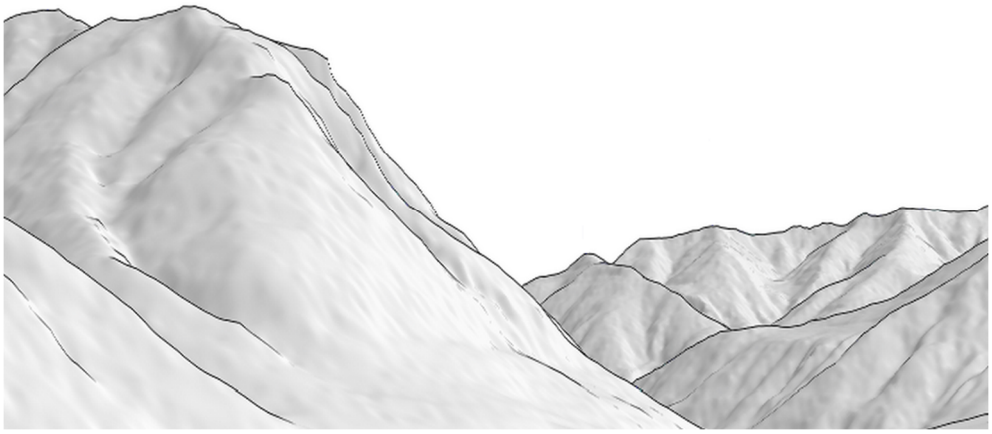

Figure 1.2: Fragment of the panorama generated by Udeuschle generator. Mountain contours are clearly visible.

4. The internet connection is mandatory to perform call to the service. No service downtime is tolerated.

5. The service provides no information about the distance to all the points of the panorama image. Only some mountain peaks are shown in another layer over the panorama image together with distances estimated with orthodromic distance (great-circle distance) algorithm.

1.3 Document Structure

In the next chapter an overview on the theoretical topics touched upon in this thesis is provided.

In the third chapter the implementation of the system is explained including the specific z-buffer data transformation techniques developed.

In the fifth chapter the results of the evaluation against the reference model are listed with the data set structure and error metric description. The directions of usage with respect to the estimated error are given.

Finally in the last chapter the conclusions of this work are given together with possible future improvements.

Chapter 2
Background

2.1 SRTM digital elevation data

STS-99 was a Space Shuttle mission launched on 11 February 2000. The primary objective of the mission was the Shuttle Radar Topography Mission (SRTM) project. The shuttle was equipped with 60 meters displaces antenna. Much like in stereo vision two images of the same earth area with a shift of 60 meters were compared to build a stereogram by analysing which the Digital Elevation Models can be created (Figure 2.1). A great advantage of such measurement principle for data collection is weather and illumination independence.

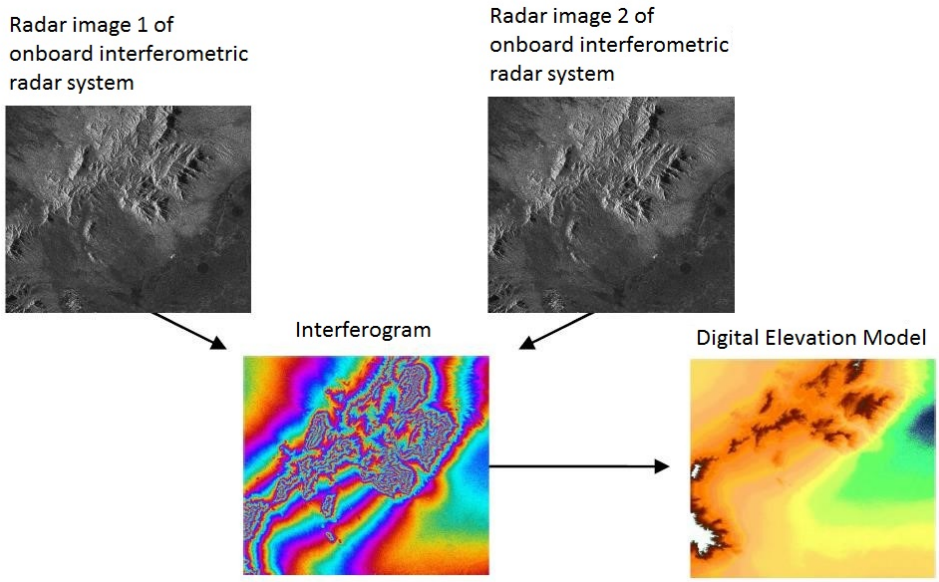

Figure 2.1: Illustration of the interferogram formation principle.

Topographic data between 60°N and 56°S was acquired. Around 8 terabytes of data was gathered. The produced Digital Elevation Model met Interferometric Terrain

Height Data (ITHD)-2 specifications (30 meter x 30 meter spatial sampling with 16 meter absolute vertical height accuracy, 10 meter relative vertical height accuracy and 20 meter absolute horizontal circular accuracy).

At first for most of the world outside the United States the SRTM data publicly available was sampled at a three-arc-second pixel size, which is 1/1200th of a degree of latitude and longitude, or about 90 meters. Later the highest-resolution data was released sampled at one-arc-second pixel size, which is 1/3600th of a degree, which is about 30 meters.

SRTM DEM data is distributed in .hgt format. One HGT file covers an area of 1°x1° (~111x111 kilometers). Its south western corner can be deduced from its file name: for example, n51e002.hgt covers the area between N 51° E 2° and N 52° E 3°, and s14w077.hgt covers S 14° W 77° to S 13° W 76°.

The DEM is provided as 16-bit signed integer data in a simple binary raster. There are no header or trailer bytes embedded in the file. The data are stored in row major order (all the data for row 1, followed by all the data for row 2, etc.). All elevations are in meters referenced to the WGS84/EGM96 geoid. Byte order is Motorola ("big-endian") standard with the most significant byte first. Since they are signed integers elevations can range from -32767 to 32767 meters, encompassing the range of elevation to be found on the Earth.

These data also contain occasional voids from a number of causes such as shadowing, phase unwrapping anomalies, or other radar-specific causes. Voids are flagged with the value -32768.

SRTM DEM data is used as-is as it satisfies our need in our region of interest (Alps). Although some terrace effect [12] was spotted in Italy (Figure 2.2) it was only spotted in pre-Alps region and does not affect peaks and relief.

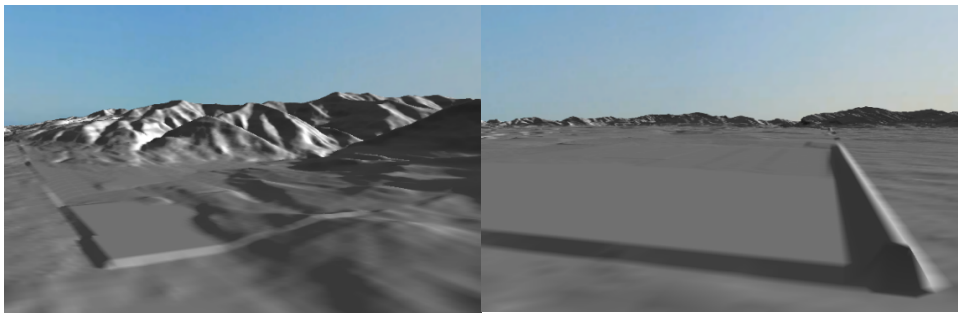

Figure 2.2: Terracing effect.

2.2 Rendering pipeline

In order to use and transform the data from z-buffer as later discussed in the Implementation section it is necessary to fully understand the stages of the graphic pipeline the data goes through on the way from geometric data (collection of vertices) to a two-dimensional image that can be seen on the screen. To do so the transforms and spaces of the graphic pipeline alternating in the order they take place in the processing are shown on the Figure 2.3.

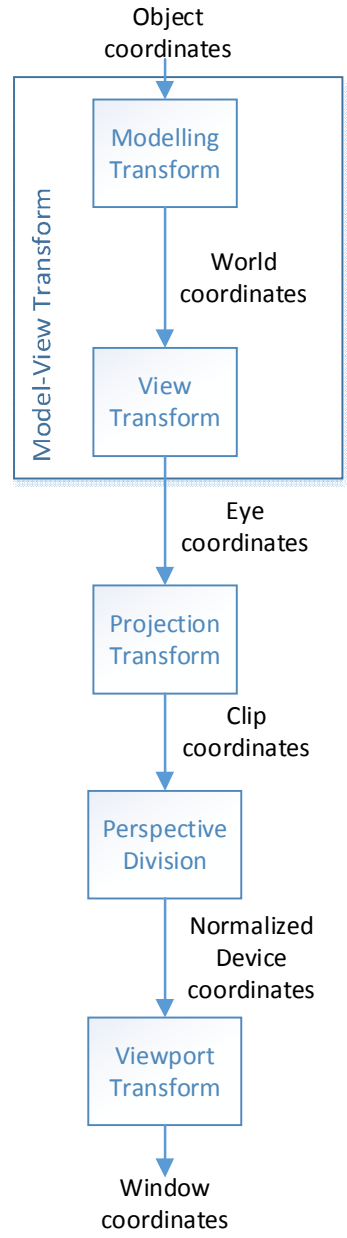

Figure 2.3: Coordinate systems and transforms of the graphics pipeline.

1. Object space (model space)

The initial representation of the data is an object space. The 3D model of the object (Earth surface in our case) is represented as vertex positions in a coordinate system. Object-space 3D vertex positions are usually represented as an <x, y, z> vector. Each vertex may also have an object-space surface normal, also stored as an <x, y, z> vector.

In practice coordinates are stored with an additional 4th component <x, y, z, w>. These coordinates are called homogeneous coordinates or projective coordinates. 3-dimensional position vector <x, y, z> is a special case of 4-dimensional <x, y, z, w> vector where w = 1. 3-dimensional coordinates are also called Cartesian coordinates.

A conversion from Homogeneous to Cartesian coordinates can be always done simply dividing by w:

$$(x, y, z, w) \Leftrightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$$

Usage of homogeneous coordinates has many advantages. The main one is that with homogeneous coordinates all the transformations applied to the scene can be represented as a 4x4 matrix. Moreover two subsequent transformations such as translations, rotations, scales, projections, etc. can be represented by one final matrix which is the multiplication of two corresponding matrices. Also homogeneous coordinates eliminate the need in extensive divisions in perspective transformations.

2. Modelling Transform

This transformation is used to place an object specified in object space to the world space. Each particular transformation used to place one particular object in the scene does not affect other objects of this scene.

3. World space

World space provides some fixed parent coordinate system for all the objects in the scene. This is a global coordinate system that everything is defined relative to.

4. View transform

The typical view transform combines a translation that moves the eye position in world space to the origin of eye space and then rotates the eye appropriately. By doing this, the view transform defines the position and orientation of the viewpoint.

5. Eye space (also known as View space or Camera space)

Eye space is used to choose a particular viewing point from which one looks at the scene. The eye (or camera) is fixed and located at the origin of the coordinate system. Since the camera is fixed the scene needs to be moved on order to reach the desired viewpoint. By the convention the scene is oriented in a way that the camera is looking down one direction of the z-axis. The positive y direction is conventionally chosen to be the up direction.

6. Projection Transform

The transform that converts eye-space coordinates into clip-space coordinates is known as the projection transform.

The projection transform defines a view frustum that represents the region of eye space where objects are viewable. Only polygons, lines, and points that are within the view frustum are potentially viewable when rasterized into an image. OpenGL and Direct3D have slightly different rules for clip space. In OpenGL, everything that is viewable must be within an axis-aligned cube such that the x, y, and z components of its clip-space position are less than or equal to its corresponding w component. This implies that $-w \leq x \leq w$, $-w \leq y \leq w$, and $-w \leq z \leq w$. Direct3D has the same clipping requirement for x and y, but the z requirement is $0 \leq z \leq w$. These clipping rules assume that the clip-space position is in homogeneous form, because they rely on w.

The projection transform provides the mapping to this clip-space axis-aligned cube containing the viewable region of clip space from the viewable region of eye space—otherwise known as the view frustum. You can express this mapping as a 4x4 matrix.

7. Clip space

Clip space coordinate system determines which coordinates are actually viewable in the image or on the screen. Clip space specifies a position in the pixel color render buffer's coordinate system. Values outside the render buffer's coordinate system can't be seen on the screen and are therefore "clipped out" of rendered results.

8. Perspective Division

Dividing x, y, and z by w accomplishes this. The resulting coordinates are called normalized device coordinates. Now all the visible geometric data lies in a cube with positions between <-1, -1, -1> and <1, 1, 1> in OpenGL, and between <-1, -1, 0> and <1, 1, 1> in Direct3D.

9. Normalized Device Coordinates

Here comes a conversion from homogenous coordinates <x, y, z, w> to 2D coordinates <x, y> along with z-buffer value. Normalized Device Coordinates is an intermediate coordinate system independent from the particular graphic device resolution.
The 2D vertex programs in Chapters 2 and 3 output what you now know as normalized device coordinates. The 2D output position in these examples assumes an implicit z value of 0 and a w value of 1.

10. Viewport Transform

This step, called the viewport transform, feeds the GPU's rasterizer. The rasterizer then forms points, lines, or polygons from the vertices, and generates fragments that determine the final image. Another integrated transform, called the depth range transform, scales the z value of the vertices into the range of the depth buffer for use in depth buffering.

11. Window Coordinates

Final coordinate system of the screen or resulting image measured in x and y.

2.3 Z-buffering algorithm

Z-buffering is an algorithm which solves the visibility problem – the problem of deciding which object is visible on the rendered scene and which is blocked by the other object in the foreground. Z-buffering is the de facto standard solution to the problem.

Z-buffer is usually represented as a two-dimensional array with one element for each screen pixel. The algorithm works at the pixel level. At the beginning the z-buffer is initialized with the value which represents infinity, which is the value of the far plane of the viewing frustum. Then for each rasterized primitive and for each pixel of the screen covered by this primitive its z value is calculated. If the value is smaller (means the object is closer to the screen) than the value currently stored in z-buffer for this particular pixel, it is replaced with the new smaller value.

```
1  ZBufferAlgorithm(){
2  forall i,j ZBuffer[i,j] = 1.0;
3  for each primitive pr
4  for each pixel (i,j) covered by pr
5  if(Z[i,j] < ZBuffer[i,j])
6  ZBuffer[i,j] = Z[i,j];
7  }
```

The z-buffering algorithm pseudocode.

Z-buffering algorithm is usually implemented in hardware (GPU). Moreover there is usually no need for the program running on CPU to access a z-buffer as its content is normally requested by the operations performed by GPU (z-culling, visibility problem resolution, shadow mapping).

Chapter 3
Design and Implementation

The whole system is implemented in Java programming language. According to the famous slogan “Write once, run everywhere” Java gives us the advantage of using the code of the system in Desktop applications, web-services and mobile devices.

The rendering of the 3D scenes is performed through the jME3 game engine library with some core classes rewritten in the way that provides the access to the underlying layer of LWJGL library. Access to the underlying layer is needed to be able to directly receive the data from z-buffer of GPU hardware, such low-level functionality is normally not needed to game and conventional 3D software developers.

jME3 itself uses LWJGL (Lightweight Java Game Library). The library accesses native C code through the Java Native Interface (JNI). LWJGL provides bindings to OpenGL. The library acts as a wrapper over the OpenGL native libraries and provides API similar to the OpenGL APIs of other lower lever languages running directly on hardware (as opposed to Java running inside the Java Virtual Machine) such as C\C++.

The presence of the native code libraries in the stack breaks the aforementioned principle “Write once, run everywhere”. Platform-specific lwjgl.dll file should be accessible to the Java code. Though it’s not a big constraint since libraries for most popular operation systems (Linux/Windows) and platforms (x86/x64/ARM) are available.

The first attempt was made to implement the renderer using JOGL OpenGL Java library. Although the 360 degrees panorama rendering was successful, it was not possible to access z-buffer data. It forced us to switch to jME3 and LWJGL libraries. Another advantage of jME3 over the JOGL library is the support of DEM height maps out of the box.

No GUI is provided for the panorama generation functionality since the program is supposed to be integrated to the pipeline of the comprising project. The code can anyway be started through command line.
The system can be split into some modules:

1. DEM operating module

This module reads DEM file and produces the 3D landscape model suitable for renderer. Module takes geographical coordinates representing the centre of the desired area and the desired radius as an input. Since the DEM data is stored in the big amount of files one square degree of latitude\longitude per file the module the module can sew together terrain data from up to 9 files producing the square terrain of the size 333x333km.

2. Renderer module

This module performs rendering of the 3D terrain surface. The usable classes of the module are represented by two different classes: OffscreenRenderer and OnscreenRenderer.
OffscreenRenderer provides the 360 degree panorama generation functionality. An example of panorama rendered by the system can be seen on the Figure 3.2 together with its distance map visual representation (Figure 3.3) and height map visual representation derived from distance map (Figure 3.4).
OnscreenRenderer provides the ability to fly over the 3D landscape in a flight simulator-like way using keyboard and mouse controls. Simple GUI is provided to choose screen resolution and some other rendering parameters. OnscreenRenderer is extremely useful for debug purposes and for detection of anomalies in DEM data. The screenshot of the running application in OnscreenRenderer mode is shown on the Figure 3.1.

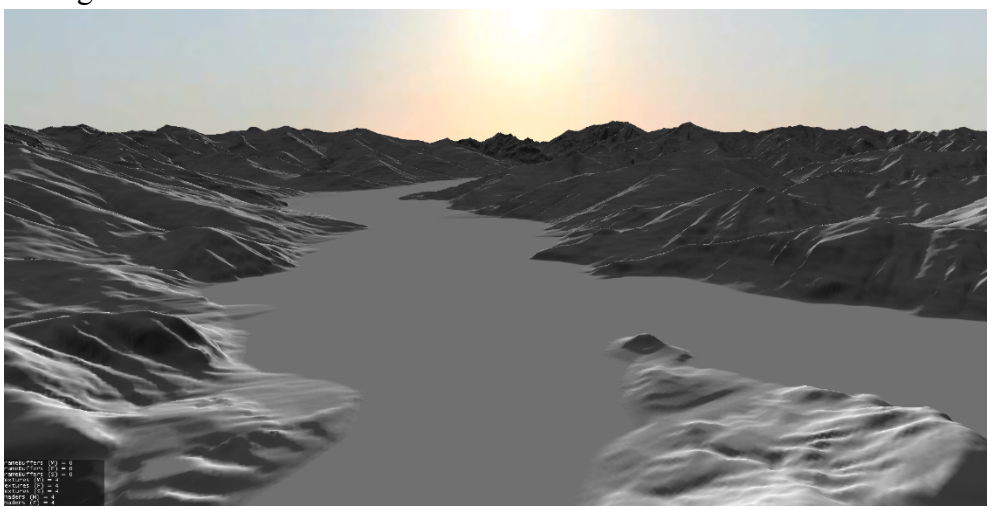

Figure 3.1: Flying over the Bellagio on Como lake in OnscreenRenderer mode screenshot.

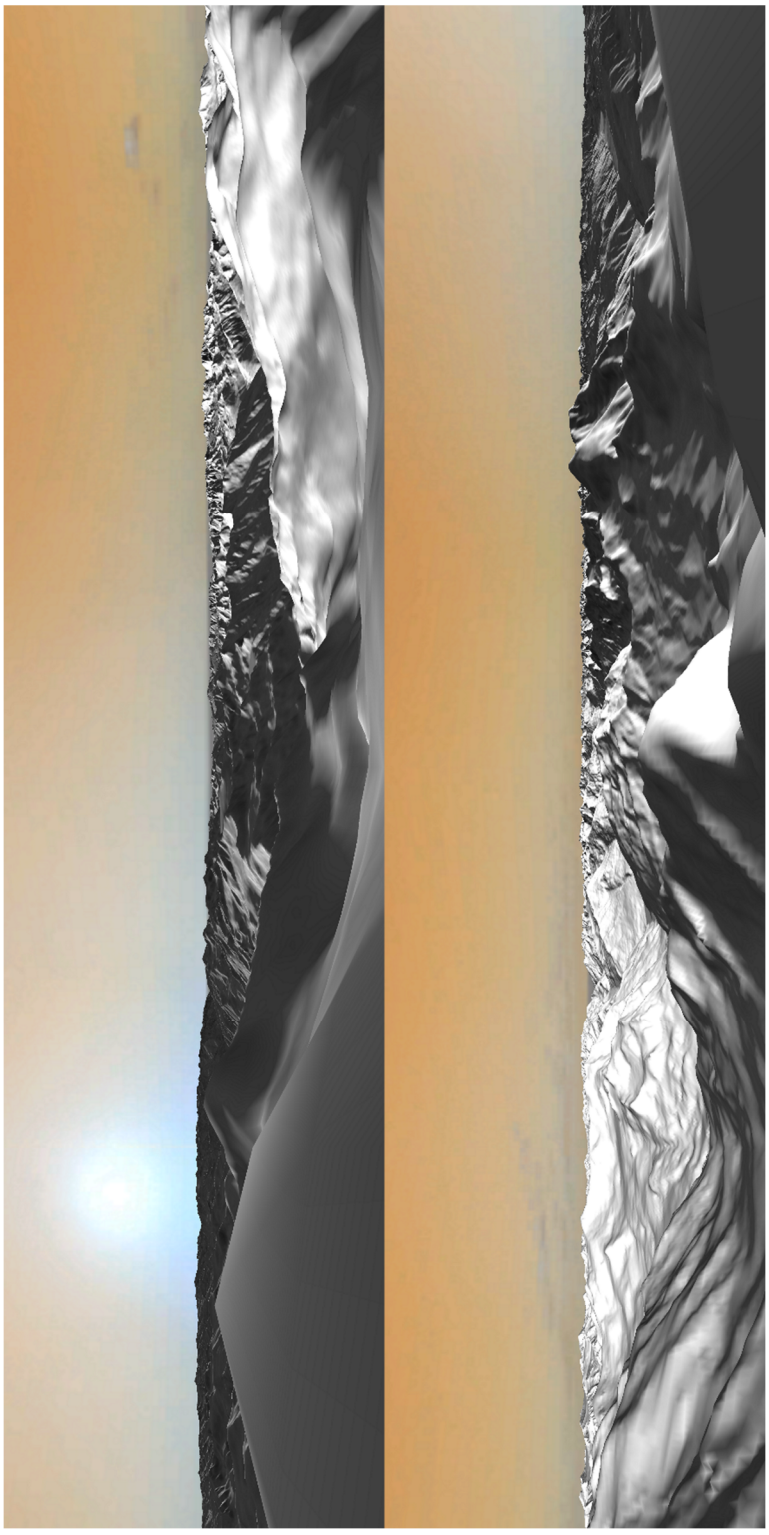

Figure 3.2: 360° panorama rendered by the system. Split into two sections and rotated to fit the page.

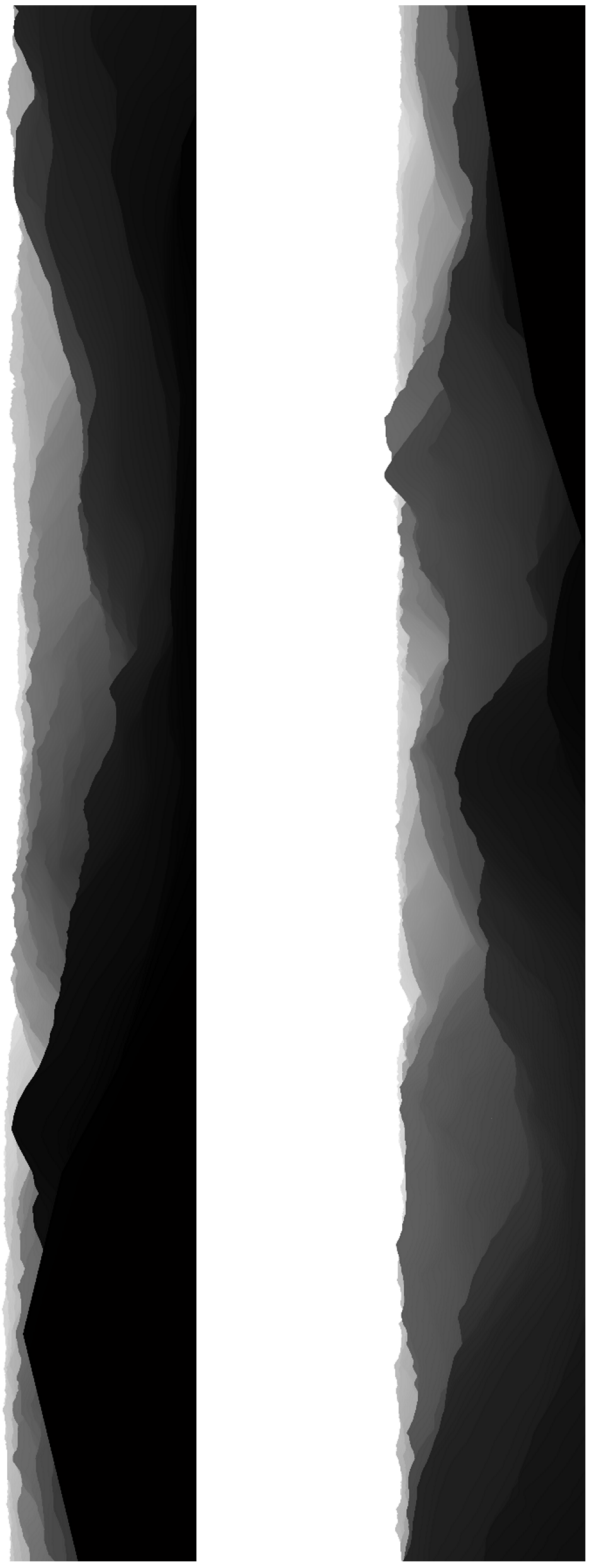

Figure 3.3: Distance map of the panorama from the Figure 3.2.

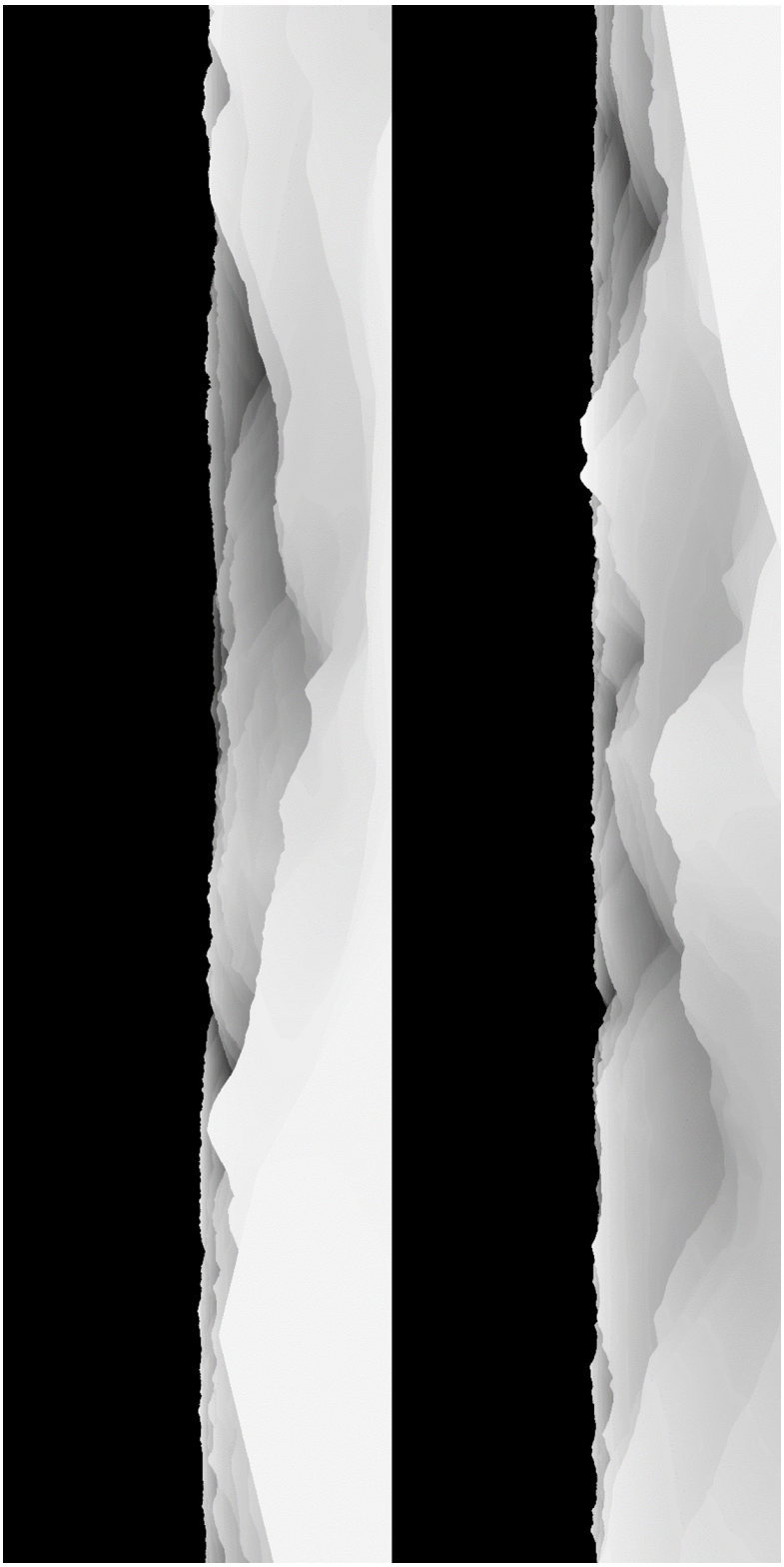

Figure 3.4: Height map derived from distance map from the Figure 3.3.

3.1 Overview of the program workflow

The program workflow diagram is shown in the Figure 3.5.

In the beginning a program receives geographical coordinates (latitude and longitude) as an input. Based on the coordinates the set of files containing relevant DEM data is determined and read into the memory. Then these data are concatenated in the proper order to form one continuous terrain surface. After this the terrain is positioned and cut to the requested coordinate the new centre of the terrain surface. Next the DEM heightmap (two-dimensional array of heights in meters) is translated into the 3D model (vertices, edges, faces, and polygons) and the altitude of the landscape in the desired coordinate is determined based on the DEM data. Then camera is positioned into the desired coordinate and on the determined latitude. Initially the camera is facing north with a clockwise shift of 22.5° (half of 45°). Then the camera is rotated by 45° eight times. After each rotation a snapshot of what camera sees and a snapshot of z-buffer content are made. At the end all the data is sewed up together to form 360° view panorama and distance panorama and this data is returned to the calling function to be used in subsequent processing or just be saved as a file for later use.

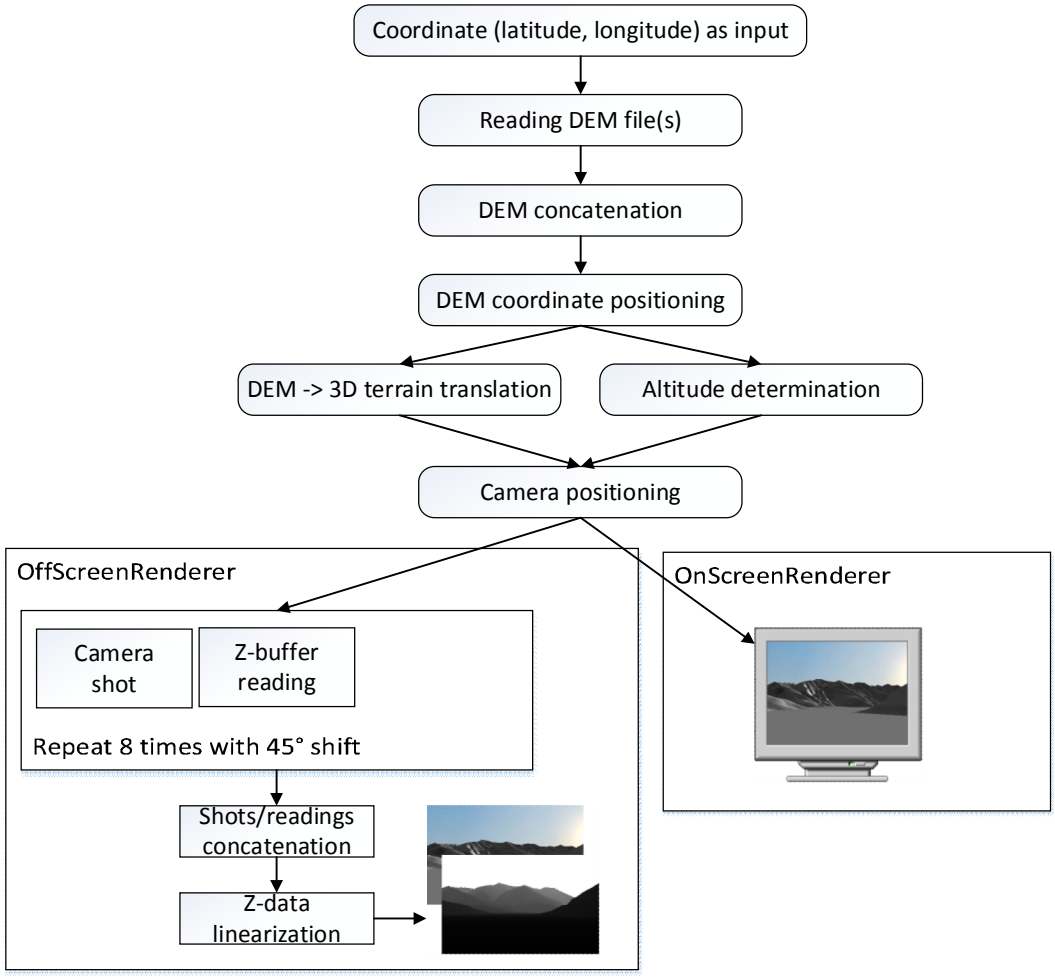

Figure 3.5: Program workflow diagram.

3.2 Altitude selection

The choice of the camera altitude is of particular interest. Ideally the value can be obtained from the DEM model. In real situations the need to tune the obtained altitude arise.

One can imagine an observer standing on a peak or a ridge of a mountain – he has a broad view in front of him, but it's enough for him to take a few steps back and the panorama he was viewing becomes completely covered by the facade of the mountain in front of him. Taking into account the 30 meter precision of the available DEM data it becomes feasible to confirm the need of such functionality. Therefore the manual altitude correction was built into the OnscreenRenderer and OffscreenRenderer as an ability to pass positive or negative integer value in meters

to correct the altitude basing on the altitude determined from the DEM model. Figure 3.6 shows in a simplified way the problem and its possible resolution.

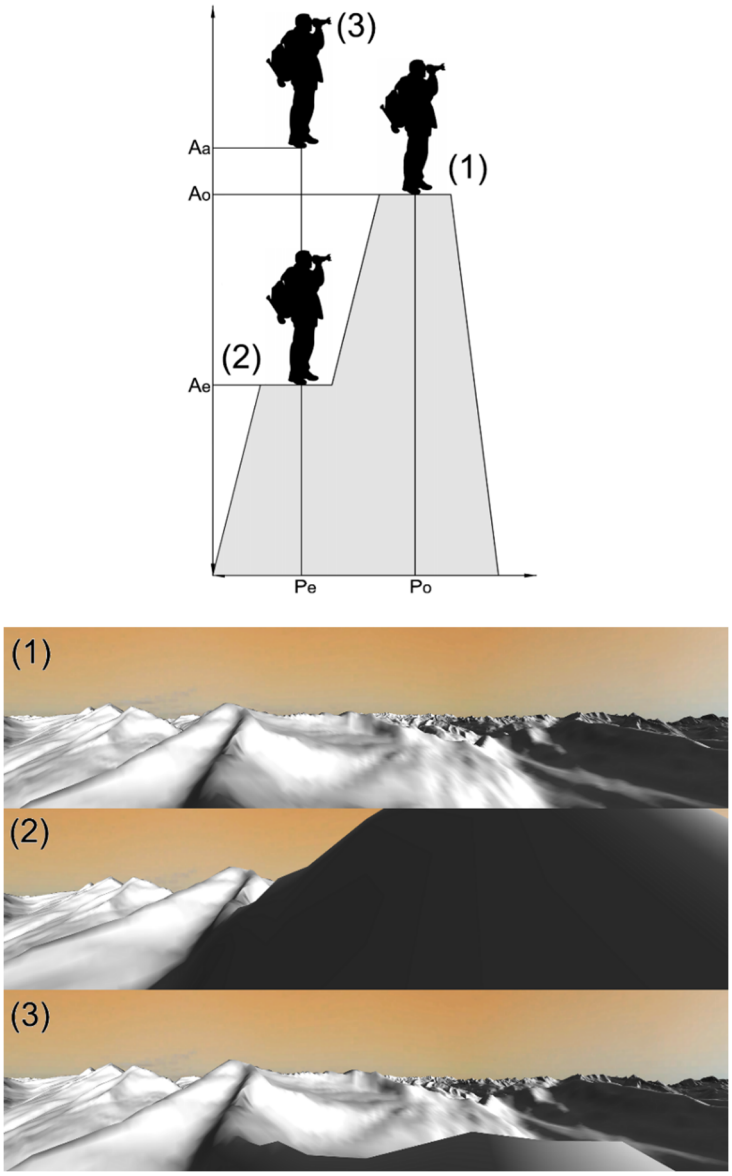

Figure 3.6: Example of wrong position selection. 1 – Camera is positioned on top of the mountain, 2 – Camera is positioned at the slope of the mountain with view covered by the facade of the mountain, 3 – Same position as in (2), but the altitude is corrected to make camera see above the mountain.

3.3 Z-buffer data linearization

In order to reconstruct linear depth values from the values obtained from z-buffer one needs to reverse engineer the pipeline which data goes through before it gets into the z-buffer.

Starting from the basics and follow the pipeline it can be done as:

The OpenGL projection matrix (GL_PROJECTION) looks like:

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

Multiplied by homogeneous point $\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$ in eye space it gives us this point in clip space:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

where

$$x_c = \frac{2n}{r-l} x_e + \frac{r+l}{r-l} z_e$$

$$y_c = \frac{2n}{t-b} y_e + \frac{t+b}{t-b} z_e$$

$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n} w_e$$

$$w_c = -z_e$$

since one only cares about depth component and taking into account the fact that in eye space w_e equals to 1 the result is:

$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n}$$

$$w_c = -z_e$$

Next operation OpenGL does is the division of by its w component (perspective division):

$$z_n = \frac{z_c}{w_c} = \frac{\frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n}}{-z_e} = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z_e}$$

After the perspective division the value is in the range [-1,1] and before it is finally written to the z-buffer it is scaled to the range [0,1]:

$$z_b = \frac{z_n + 1}{2} = \frac{f+n}{2(f-n)} + \frac{fn}{(f-n)z_e} + \frac{1}{2}$$

z_b – is the value obtained from the z-buffer.

Looking at the relation between z_e and z_n one can notice it's a rational function and that the relation is non-linear. Such non-linear values of z-buffer are perfectly suitable for solving the visibility problem. It may even seem natural since the precision is higher at the near plane. But such z-buffer values are completely useless for distance measurements.

So the value from z-buffer should be taken to apply the opposite transformations in order to consequently obtain linear depth values.

First we scale it from the range of [0,1] to [-1,1]:

$$z_n = 2z_b - 1$$

Then we recover z_e:

$$z_n = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z_e}$$

$$\frac{2fn}{(f-n)z_e} = z_n - \frac{f+n}{f-n}$$

$$z_e = \frac{2fn}{(f-n)\left(z_n - \frac{f+n}{f-n}\right)} = \frac{2fn}{z_n(f-n) - f - n} = \frac{-2fn}{f+n-z_n(f-n)}$$

The depths from the formula above will all be negative. So we need to negate the answer for the visualization, since we treat these values as distances. So the function which performs linearization will look like:

```
1  static float linearize(float z_b, float n, float f){
2    float z_n = 2.0f * z_b - 1.0f;
3    float z_e = 2.0f * n * f / (f + n - z_n * (f - n));
4    return z_e;
5  }
```

Depth linearization function.

Where n – distance to the near plane of the frustum, f – distance to the far plane of the frustum.

Chapter 4
Experimental Evaluation

In order to evaluate the accuracy of the distance predictions this work will evaluate the developed model against the Udeuschle model which we take as a reference model. The workflow of the evaluation procedure looks as follows:

1. First three datasets were collected by comparing three pairs of panoramas and handpicking the mountain peaks which were clearly visible and distinguishable both on picture generated by developed system and picture generated by Udeuschle generator. The three datasets are presented 4.1, 4.2 and 4.3. The graphical representation of all three datasets can be found on Figure 4.1.
2. Next the distances from all three datasets were plotted together with the line of perfect correlation.
3. Then the relevant metrics were introduced.
4. Residual histograms were introduced to evaluate the portion of the target that the model is unable to predict.
5. To evaluate the distance predictive power of the model the statistical distribution of the absolute percentage error was plotted.
6. As an intermediate result the table that represents the percentage of the distances that were estimated with the absolute percentage error not more than the particular value was computed.
7. An attempt was made to improve the accuracy of predictions for the cases when the camera is located lower than the peaks by the calculation of systematic error with linear regression.
8. Updated metrics and absolute percentage error table were recomputed and updated with the improvements done by the linear regression.

Observation point geographic coordinates	46.443517, 10.616525 (Monte Cevedale)				
Observation point elevation	3760				
Peak name	**Elevation**	**Elevation gap**	**Our distance to the peak**	**Udeuschle distance to the peak**	**Difference**
Konigspitze [Il Gran Zebra]	3851	91	5400	6273	873
Kreilspitze [Punta Graglia]	3391	-369	4200	4871	671
Ortler [Ortles]	3905	145	9100	9855	755
Thurwieserspitze [Cima Thurwieser]	3652	-108	9100	11207	2107
Piz Linard	3411	-349	57400	68982	11582
Fluchthorn [Piz Fenga]	3399	-361	57900	60738	2838
Ostlicher Gamshorn	3073	-687	59500	62326	2826
Westlicher Gamshorn	2987	-773	59900	62665	2765
Cima della Miniera	3408	-352	6300	7880	1580
Cima Presanella	3558	-202	25100	24227	-873
Cima di Vermiglio	3458	-302	24900	23610	-1290
Monte Care Alto	3463	-297	37300	34260	-3040
Corno di Cavento	3406	-354	34400	31955	-2445
Monte Adamello	3539	-221	33300	33651	351
Cima Plem	3182	-578	34600	35500	900
Monte Giumella	3594	-166	8100	8365	265
Punta San Matteo	3675	-85	8200	8556	356
Corno Baitone	3330	-430	3300	34226	30926
Corno Premassone	3060	-700	33800	34891	1091
Monte Falcone	3456	-304	33100	33351	251
Monte Fumo	3409	-351	33800	32812	-988
Dosson di Genova	3441	-319	33100	31955	-1145
Lobbia Alta	3196	-564	30600	29263	-1337
Cima d'Ambiez	3096	-664	37900	39576	1676
Cima Brenta Bassa	2803	-957	38400	40139	1739
Cima Brenta Alta	2962	-798	38100	39758	1658
Cima Brenta	3150	-610	36500	39161	2661
Cima Groste	2898	-862	35200	38973	3773

Pietra Grande	2937	-823	31900	36590	4690
Vertainspitze [Cima Vertana]	3545	-215	10600	10158	-442
Zufallspitze [Cima Cevedale]	3757	-3	700	749	49
Hintere Eggenspitze [Cima Sternai]	3443	-317	12400	16762	4362
Hasenohrl	3257	-503	21700	28258	6558
Zufrittspitze [Gioveretto]	3439	-321	14300	18885	4585
Cima Mezzena	3172	-588	8500	11328	2828
Cima Zoccolo	2561	-1199	20500	27532	7032
Cima Cavalon	3120	-640	8600	11537	2937
Cima di Bon	2901	-859	23700	24117	417
Cima San Giacomo	3281	-479	6900	7908	1008
Monte Zandila	2936	-824	24400	32961	8561
Cima delle Pozze	2656	-1104	16200	20772	4572

Table 4.1: First dataset.

Observation point geographic coordinates	44.458705, 6.516585				
Observation point elevation	2881				
Peak name	**Elevation**	**Elevation gap**	**Our distance to the peak**	**Udeuschle distance to the peak**	**Difference**
Grande Seolane	2909	28	13700	13227	-473
Tête de l'Estrop	2961	80	19200	17664	-1536
Roche Close	2739	-142	16500	16512	12
Tête de Sautron	3155	274	28800	37141	8341
Monte Viso	3841	960	51100	65176	14076
Brec de Chambeyron	3389	508	27900	36455	8555
Le Laupon	2432	-449	36200	36179	-21
La Moutiere	2596	-285	15000	15228	228
L'Aiguillette	2610	-271	14600	14928	328
Mont Pelvoux	3943	1062	49700	48744	-956

Pic Sans Nom	3913	1032	49500	48813	-687
Le Chaperon	2748	-133	47700	54640	6940
Le Banc du Peyron	2777	-104	48300	56240	7940
Montagne de Faraut	2383	-498	54500	67864	13364
Pic de Crevoux	2649	-232	15300	17673	2373
L'Autapie	2435	-446	28000	27720	-280
L'Ailette	2560	-321	10000	9757	-243
Roche Benite	2415	-466	10600	9872	-728
Neillere	2460	-421	12900	13372	472
Sommet de Cousson	1516	-1365	50100	51582	1482
Le Cimet	3020	139	23900	25685	1785
Le Trou de l'Aigle	2961	80	27200	28680	1480
La Grande Tour	2745	-136	31000	32066	1066
La Petite Tour	2693	-188	30600	31735	1135
La Pelonniere de la Frema	2697	-184	36100	37131	1031
Pic de Bernardez	2430	-451	13100	14206	1106
Vieux Chaillol	3163	282	40200	43969	3769
Grande Autane	2782	-99	27900	30706	2806
Pointe du Vallon des Etages	3564	683	51400	52743	1343
Pic de Rochelaire	3108	227	26800	25776	-1024
La Gueste	2724	-157	22100	20809	-1291
Mont Orel	2565	-316	13800	14114	314
Pic Saint Andre	2863	-18	14300	14758	458

Table 4.2: Second dataset.

Observation point geographic coordinates	46.497104, 7.489357				
Observation point elevation	2763				
Peak name	**Elevation**	**Elevation gap**	**Our distance to the peak**	**Udeuschle distance to the peak**	**Difference**
Gsur	2709	-54	2700	3466	766
Mannliflue	2652	-111	7400	7927	527
Winterhore	2608	-155	7500	8408	908
Erbithore	2508	-255	6900	7817	917

Elsighorn	2341	-422	12200	16531	4331
Bunderspitz	2546	-217	10100	13240	3140
Steghorn	3146	383	11500	12676	1176
Grosstrubel	3242	479	10900	11400	500
Tierhornli	2894	131	11000	12916	1916
Wildstrubel	3243	480	11200	11190	-10
Matterhorn [Monte Cervino]	4477	1714	59300	58900	-400
Dent Blanche	4357	1594	52300	51200	-1100
Dent d'Hérens	4171	1408	59300	57360	-1940
Bouquetins	3838	1075	57400	53622	-3778
Rothorn	2410	-353	4800	4994	194
Seehore	2281	-482	8100	8126	26
Schafarnisch	2107	-656	19900	20747	847
Nuneneflue	2102	-661	23400	22044	-1356
Gantrisch	2175	-588	23300	22368	-932
Ochsen	2188	-575	23100	23040	-60
Mittaghorn	2686	-77	12900	13130	230
Schnidehorn	2937	174	15700	16332	632
Wildhorn	3246	483	18500	19258	758
Hahnen Schritthorn	2834	71	17400	18466	1066
Oldenhorn	3122	359	27800	33635	5835
Les Diablerets	3210	447	31500	37647	6147
Tête de Barme	3185	422	31300	37321	6021
Dent de Brenleire	2353	-410	24800	33577	8777
Stockhorn	2190	-573	22200	21610	-590
Wiriehorn	2304	-459	9700	9934	234
Elsighorn	2341	-422	12200	16531	4331
Eiger	3970	1207	40500	54770	14270
Monch	4099	1336	39400	53169	13769
Jungfrau	4158	1395	36500	48796	12296
Rottalhorn	3975	1212	36800	49021	12221
Bluemlisalphorn	3661	898	21700	28497	6797
Aletschhorn	4195	1432	38800	51505	12705
Doldenhorn	3643	880	19000	25647	6647
Hockenhorn	3293	530	21000	28212	7212
Bietschhorn	3934	1171	30100	40326	10226
Balmhorn	3699	936	17600	23331	5731
Altels	3629	866	16300	21549	5249
Rinderhorn	3453	690	15700	19830	4130

Chli Rinderhorn	3003	240	14200	18203	4003
Dom	4545	1782	53100	55384	2284

Table 4.3: Third dataset.

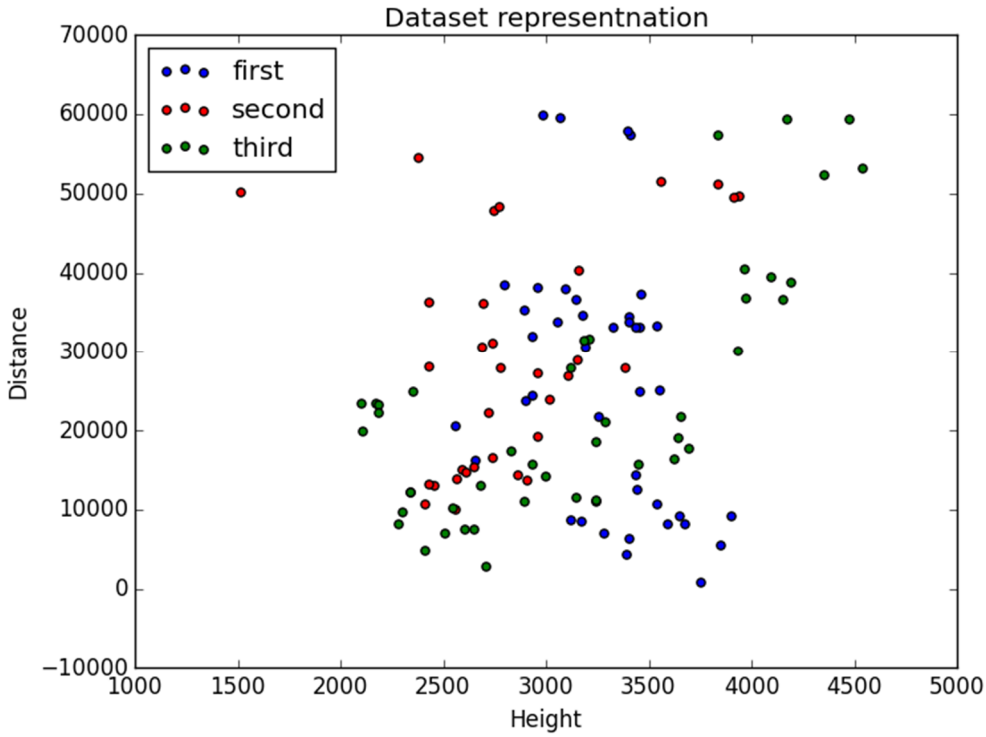

Figure 4.1: Graphical representation of the three datasets.

At Figure 4.2 below one could see the deviation of the predicted value from the etalon value. The closer the point to the line of the perfect correlation the better the model is (according to the perfect model all the point are at this line).

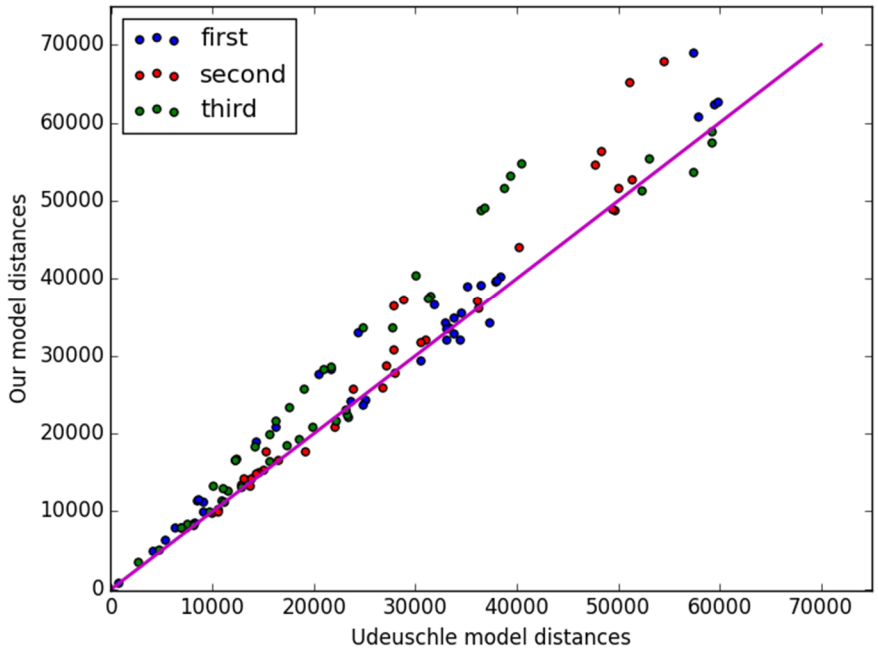

Figure 4.2: Correlation of distance prediction results between our model and Udeuschle model.

Two consequences could be done directly from the graph. As all the points are quite close to the line of the perfect correlation, our model provides accurate predictions of the distances, the second consequence is the fact that our model is likely to overestimate the distances to the mountain as most of the points are above the line.

All the observations could be naturally divided into two groups. The first group includes the observations made by the camera located lower than the peak (Figure 4.4), the second group – observations made by the camera located higher than the peak (Figure 4.3).

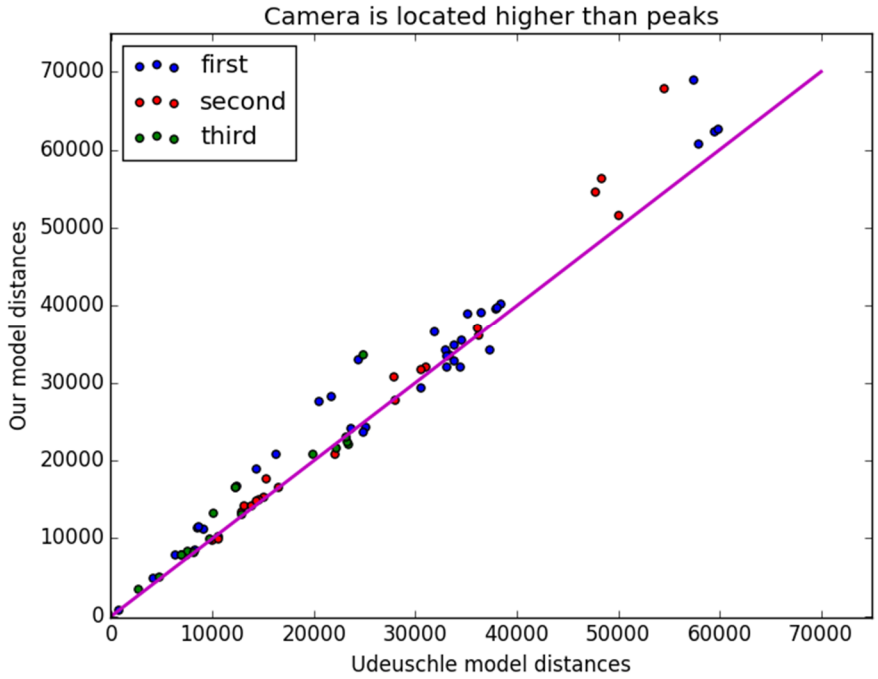

Figure 4.3: Correlation of distance prediction results between our model and Udeuschle model for cases when camera is located higher than peaks.

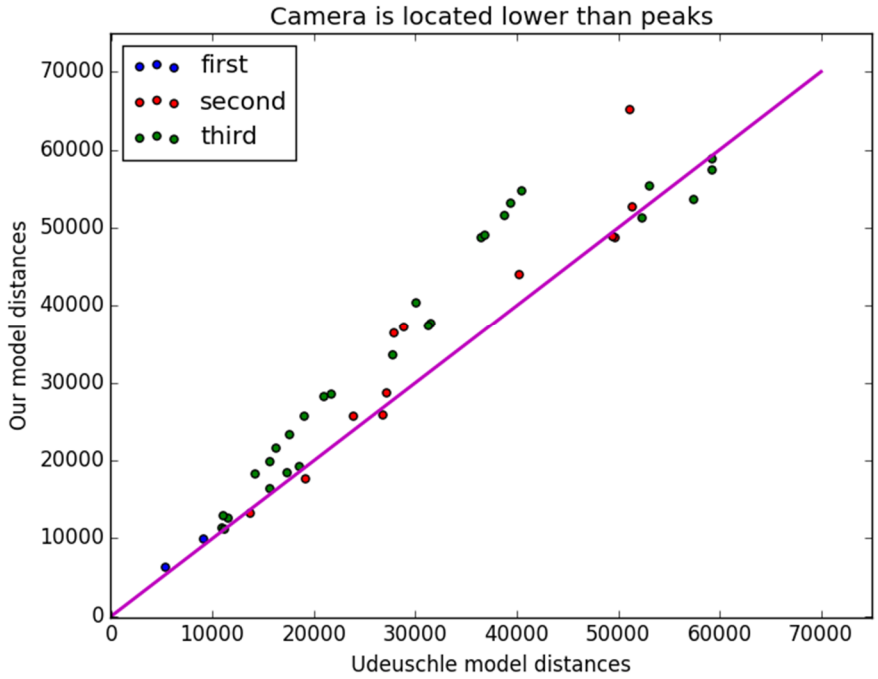

Figure 4.4: Correlation of distance prediction results between our model and Udeuschle model for cases when camera is located lower than peaks.

Basing on this information the metrics to compare these two groups were introduced. The root mean squared error, the mean absolute error, the relative squared error and the coefficient of determination were calculated.

Denotations: n – number of peaks in the dataset

y_i – distance to the i^{th} peak according to the Udeuschle model
$\hat{y}_i$ – distance to the i^{th} peak according to the our model
$\bar{y}$ – mean distance according to the Udeuschle model

1) Root mean squared error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}}$$

2) Mean absolute percentage error

$$MAPE = \frac{\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i}}{n}$$

3) Relative squared error

$$RSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

4) Coefficient of determination (R squared statistics)

$$R^2 = 1 - RSE$$

First metric is able to provide information about goodness of predictions only in comparison, as it is absolute number. Three last values could be valuable considered alone. For example R^2 is a statistic that will give some information about the goodness of fit of a model. An R^2 of 1 indicates that the predictions are perfect (always coincide with the real value).

	Coefficient of determination	RMSE	Relative squared error	Mean Absolute percentage error
Camera is Higher	0.94528821696	3473.95281784	0.0547117830403	0.10913854367
Camera is lower	0.831952360592	6360.7378968	0.168047639408	0.166815799976
All points	0.906195658623	4699.84824246	0.0938043413768	0.129495222366

Table 4.4: Error metrics for different positions of the camera.

Therefore, it is not difficult to see that the camera installed higher than the peaks provides much more accurate observations.

As for the residuals, it can be concluded that a residual for an observation in the evaluation data is the difference between the true target and the predicted target. Residuals represent the portion of the target that the model is unable to predict. A positive residual indicates that the model is underestimating the target (the actual target is larger than the predicted target). A negative residual indicates an overestimation (the actual target is smaller than the predicted target). The histogram of the residuals on the evaluation data when distributed in a bell shape and centered at zero indicates that the model makes mistakes in a random manner and does not

systematically over or under predict any particular range of target values (the histogram follows the form of the normal distribution).

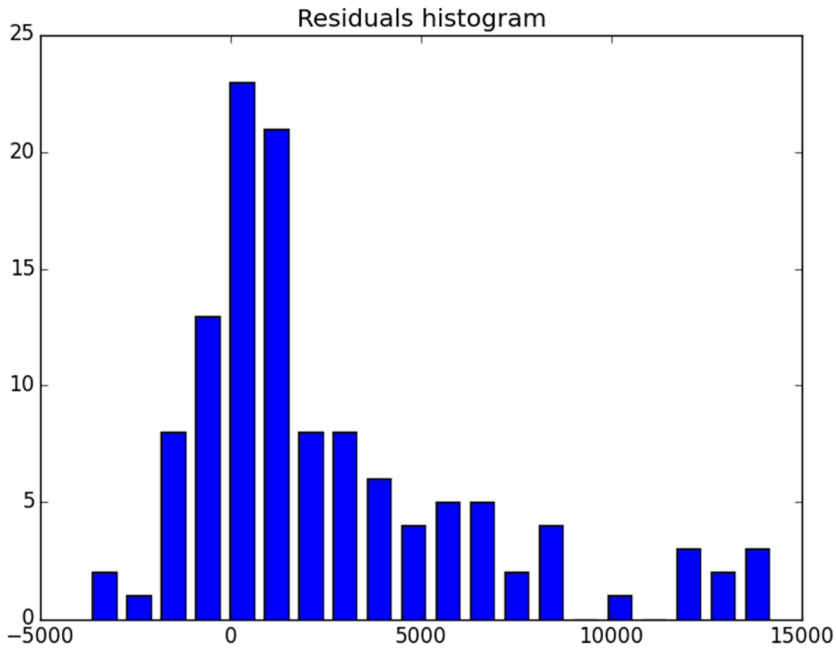

Figure 4.5: The histogram of the residuals on the evaluation data.

The histogram on the Figure 4.5 confirms an assumption made from the first graph; our model is really likely to overestimate the distance to the peak. Thereby it can be helpful to compare the form of this histogram for two cases defined above.

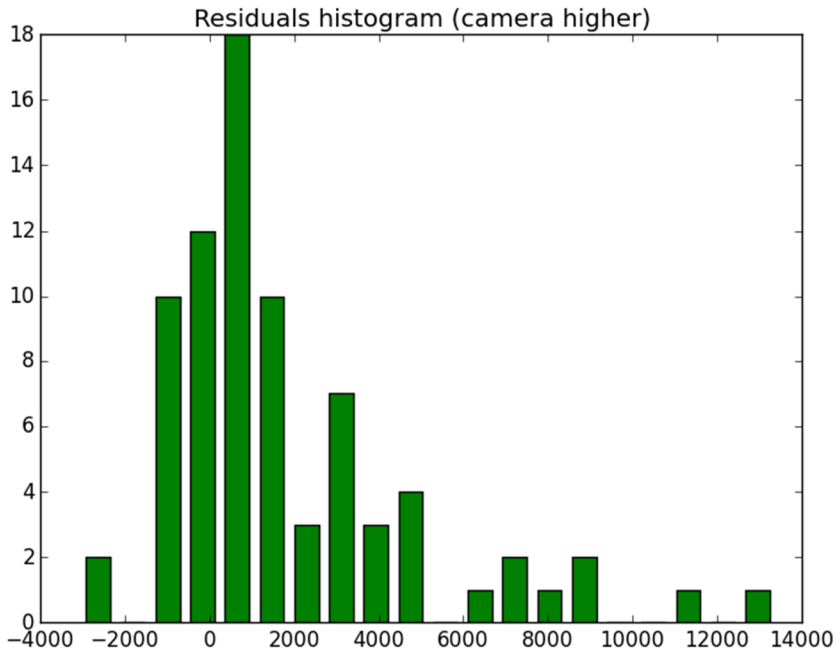

Figure 4.6: The histogram of the residuals on the evaluation data for cases when camera is located higher than peaks.

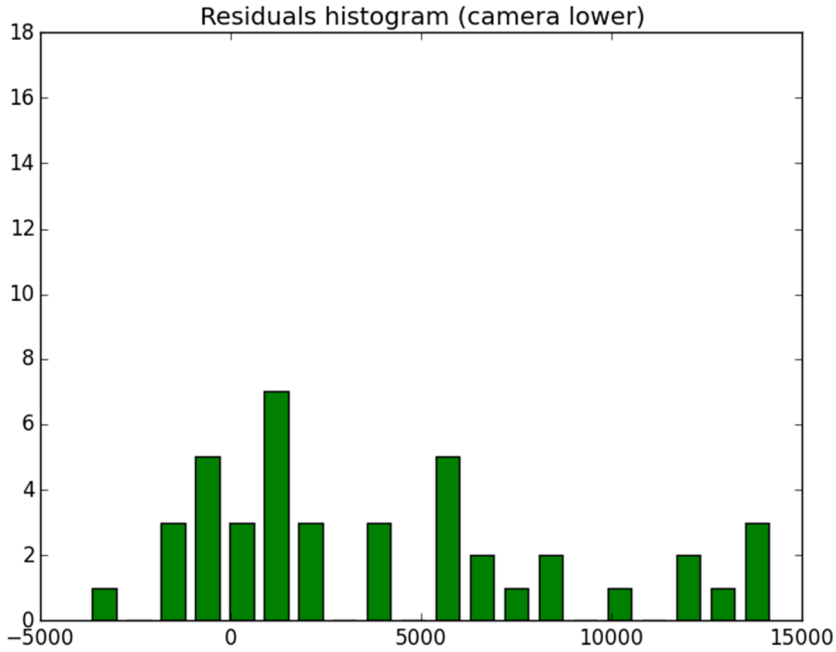

Figure 4.7: The histogram of the residuals on the evaluation data for cases when camera is located lower than peaks.

More or less accurately it could be said, that both histograms on the Figure 4.6 and the Figure 4.7 have a shape of the bell (taking into account the fact, that there are more observations made by the camera located higher), and in both cases we observe overestimation of the distance (the right tail of the distributions is longer). Histograms also confirm the preference to install the camera than the observed peak, because the right tail of the corresponding histogram is thicker.

Coming to the next step the statistical distribution of the absolute percentage error should be considered.

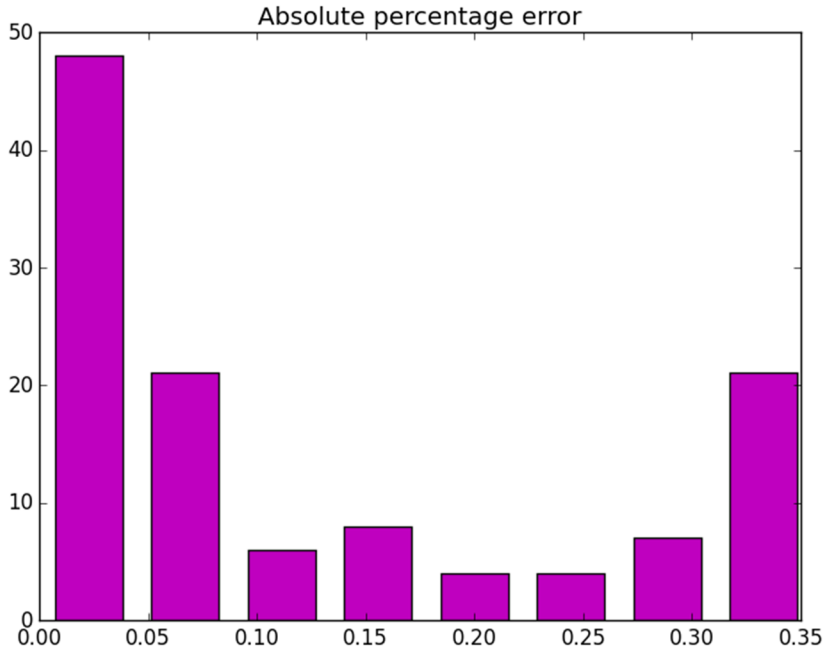

Figure 4.8: Statistical distribution of the absolute percentage error.

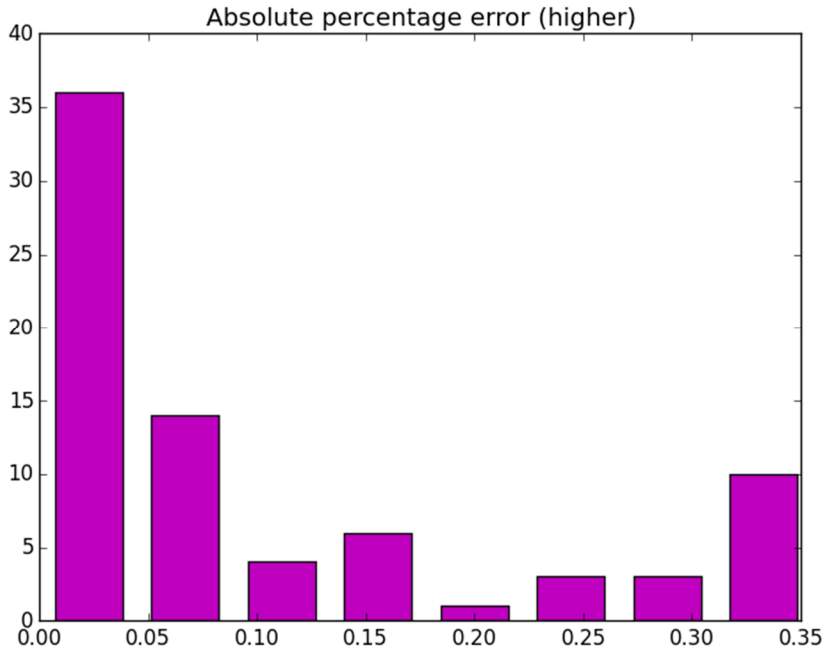

Figure 4.9: Statistical distribution of the absolute percentage error for the cases when camera is higher than peaks.

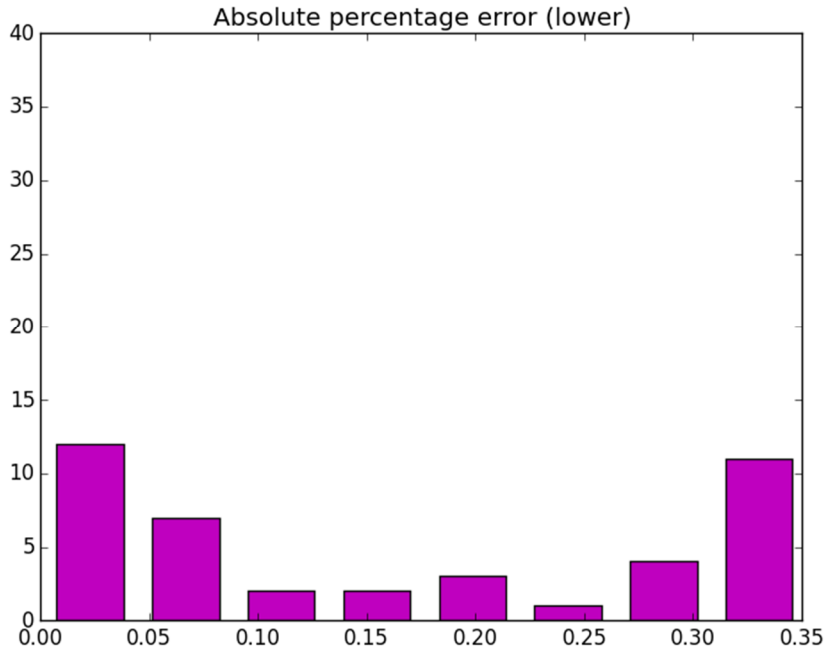

Figure 4.10: Statistical distribution of the absolute percentage error for the cases when camera is lower than peaks.

This histogram on the Figure 4.8 confirms the predictive power of the model. For most of the cases the distance was predicted with the model, as most of the distances are predicted with the percentage error not more than 10% (for most of the points real distance is in the interval $[0.9 * predicted\ value,\ 1.1 * predicted\ value]$).

The Table 4.5 below represents the percentage of the distances that were estimated with the absolute percentage error not more than the particular value.

	<5%	**<10%**	**<15%**	**<20%**	**<25%**	**<30%**	**<35%**	**<40%**
Higher	51.9%	64.9%	74%	77.9%	83.1%	85.7%	93.5%	100%
Lower	30.9%	47.6%	50%	59.5%	61.9%	71.4%	97.6%	100%
All	44.5%	58.8%	65.5%	71.4%	75.6%	80.7%	95%	100%

Table 4.5: Percentage of distances estimated with particular value of the absolute percentage error.

This table could be used for the practical applications of our distance predictions model. Imagine one has done an evaluation L of the distance to the peak from the

panorama. Based on this value you should give the shortest interval of the distances, which will include the real distance with probability 80%. If one knows that the peak is higher than the camera, than one could say that the distance to the mountain is $L \pm O.25L$ with probability 80%. If one has no idea about the height of the peak, he should provide an interval $L \pm O.3L$ in order to be 80% sure.

Now one could try to improve the accuracy of the predictions for the case when the camera is located lower than the peak by calculation of the systematic error.

The dataset was separated into two parts. One part will be used as training set (80%), the second is as test set (20%). The linear regression will be build to find the systematic shift between real and predicted value of the distance. In other the best coefficients b and c will be found, such that
*dist_real= b * dist_predicted + c + ε*,
where:
dist_real – real distance from the peak to the camera
dist_predicted – distance predicted by our model
ε – error

The best b and c are calculated by the least squared estimation method on the training set and the performance is tested on the test set. The best b and c are found using cross validation in order to avoid overfitting. The dataset is divided into k folds (in our case k=7), k-1 folds used as a training set, 1 fold used as a test set. Among this k pair of coefficients the one is chosen, which gives the best value of the R squared statistics.
The best values are b=0.898891146468 and c=-1015.66880806.

	Coefficient of determination	RMSE	Relative squared error	Mean Absolute percentage error
Distance predicted by our model	0.831952360592	6360.7378968	0.168047639408	0.166815799976
Distance predicted by our model + linear regression	0.912449916357	4591.12983698	0.0875500836433	0.119236588191090 04

Table 4.6: Error metrics with and without linear regression.

	<5%	<10%	<15%	<20%	<25%	<30%	<35%	<40%
Higher	51.9%	64.9%	74%	77.9%	83.1%	85.7%	93.5%	100%
Lower	30.9%	47.6%	50%	59.5%	61.9%	71.4%	97.6%	100%
Lower + LR	11.9%	40.5%	66.7%	97.6%	100%	100%	100%	100%

Table 4.7: Percentage of distances estimated with particular value of the absolute percentage error with and without linear regression.

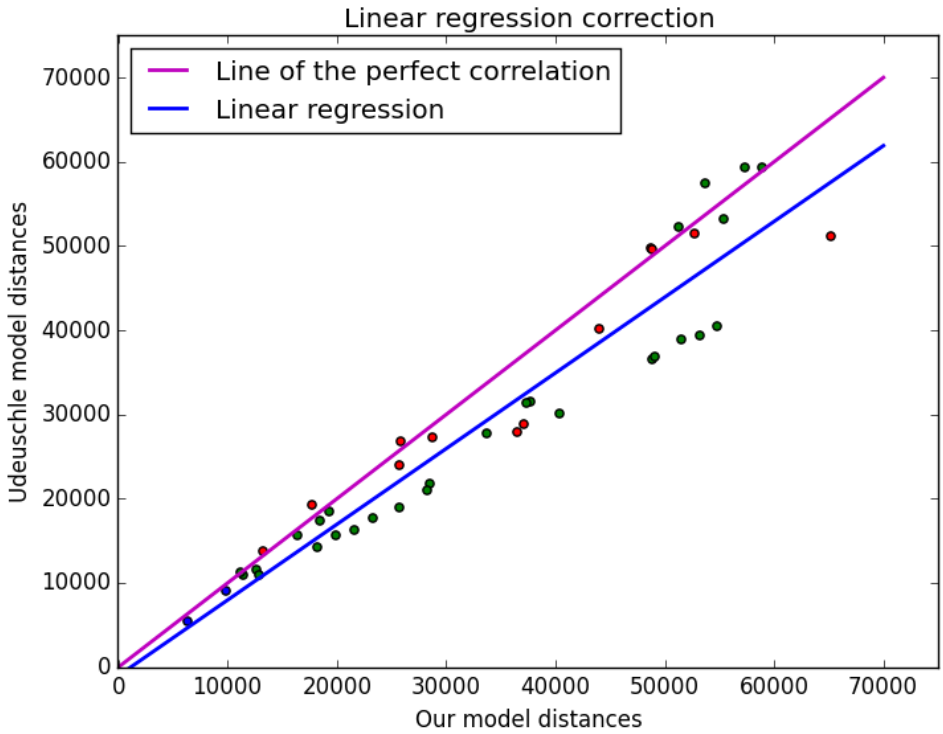

Figure 4.11: Linear regression correction.

The linear regression with computer values *b* and *c* is plotted on the Figure 4.11. The improvement of the predictions can be seen on Table 4.6 and Table 4.7 compared to Table 4.4 and Table 4.5.

So building the linear regression could be a good solution to increase the accuracy of the predictions in cases when it is impossible to install the camera higher than the peaks. But this method could not be used without having some knowledge about real distances to the peak and distances measured by algorithm in order to have enough data to build the regression.

Chapter 5
Conclusions and Future Work

In this work the implementation of the 360° panorama renderer based on real-world digital elevation data was shown. The specific technique of computing the distance mask based on the transformed data from z-buffer is presented and implemented.
The experimental results revealed the correlation between the camera-peak altitude gap and the distance error. The evaluation of the accuracy of distance measurements against the reference model is provided. The tables with applied distance accuracy probabilities are computed and directions on usage are given.
The developed system produces panorama images 10 times faster than the reference Udeuschle system. Together with panorama image it also delivers the distance information for every pixel of it.
Yet there is still a great field for improvements in two different areas: rendering optimisation and accuracy enhancement.

5.1 Rendering optimisations

Although currently the rendering of panorama takes 5-10 seconds and it's up to 10 times faster than Udeuschle renderer there is still a lot of ways the renderer can be optimized up to real-time performance without the decrease of output image quality.
Some pre-rendering stage optimisations can be applied after the digital elevation model was already loaded to the memory, but not yet translated to the 3D model. Even rough algorithms aiming at simplification of the terrain by elimination of unobservable mountains from the panorama coordinate field of view would cut down the rendering time by decreasing the number of triangles to be rendered.
Another possible approach is the simplification of the terrain model for the parts of the terrain which are simultaneously far from the camera and has low altitude. Such terrain is very likely to be obstructed multiple times by other mountains closer to the camera.

Speaking about deep renderer optimisations affecting the mathematical apparatus of picture generation one can come to an interesting idea of simulation of different real-life camera lenses. Together with geographical coordinates EXIF data also contains the information about the camera model and the lens data being used, such as focal length, zoom, etc. [2] Based on this information it becomes possible to simulate the characteristics of the virtual camera to produce rendered panorama image with the same distortions as the real one. This will make images as close as possible to each other and will definitely improve direction of view estimation and other algorithms based on comparison of real and rendered images. Figure 5.1 shows the image of the same location rendered with different virtual camera parameters.

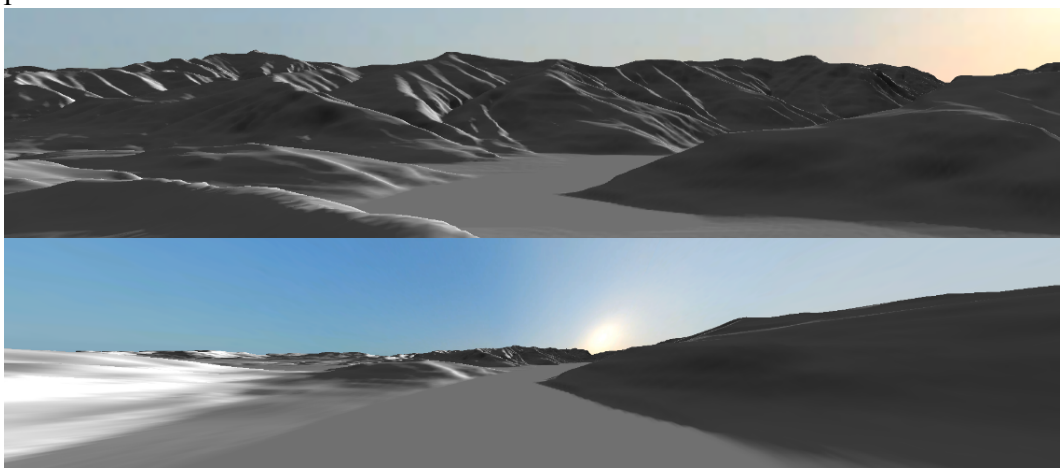

Figure 5.1: Same geographic location rendered with different virtual camera lens parameters.

The system is ready for brute-force optimizations. Given the enough number of graphics cards different sections of the final panorama image can be rendered on different GPU cores in parallel [18]. This exhaustive approach is very hardware intensive, but in theory can dramatically improve rendering time.

5.2 Accuracy enhancement

One idea of distance estimation accuracy enhancement is to increase the resolution of the rendered image (and thereby the resolution of the z-buffer data). This may help to introspect distant edge cases (here literally means the edge of the mountain) when the distance precision was eliminated by the bilinear interpolation during the rasterization stage of the GPU pipeline. Increase of resolution is very RAM and processing time demanding operation, so the way to increase the resolution only in

the limited region of interest should be provided. Figure 5.2 shows a small segment of big panorama in three different resolutions.

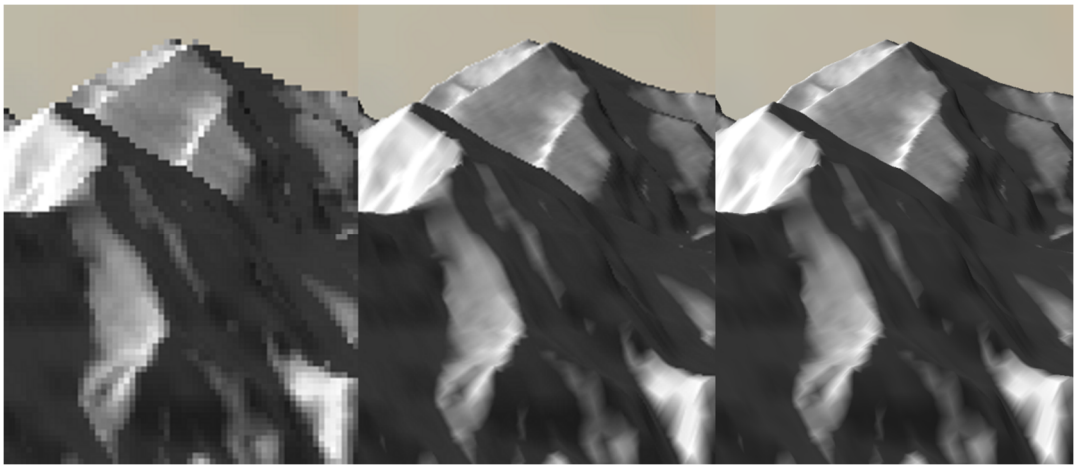

Figure 5.2: The segment of panorama image rendered in low, high and ultrahigh resolutions.

Bibliography

[1] Roman Fedorov. Mountain Peak Detection in Online Social Media. Master thesis, Politecnico di Milano, Polo Territoriale di Como, Italy, 2013.

[2] Roman Fedorov, Alessandro Camerada, Piero Fraternali, Marco Tagliasacchi. Estimating snow cover from publicly available images. CoRR abs/1508.01055. IEEE, 2015.

[3] Technical Standardization Committee on AV & IT Storage Systems and Equipment. Exchangeable image file format for digital still cameras: Exif Version 2.2. Technical Report JEITA CP-3451, April 2002.

[4] The Cg Tutorial. NVIDIA Corporation, 2007.

[5] Edward Angel, Dave Shreiner. Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL (6th Edition). Pearson, 2011.

[6] Fabio Ganovelli, Massimiliano Corsini, Sumanta Pattanaik, Marco Di Benedetto. Introduction to Computer Graphics: A Practical Learning Approach. CRC Press, 2014.

[7] Farr, T.G., M. Kobrick, Shuttle Radar Topography Mission produces a wealth of data, Amer. Geophys. Union Eos, v. 81, p. 583-585, 2000.

[8] Rosen, P.A., S. Hensley, I.R. Joughin, F.K. Li, S.N. Madsen, E. Rodriguez, R.M. Goldstein, Synthetic aperture radar interferometry, Proc. IEEE, v. 88, p. 333-382, 2000.

[9] SRTM Topography. U.S. Geological Survey, 2003.

[10] SRTM Data Editing Rules. U.S. Geological Survey, 2003.

[11] Aaaron E. Walsh, Doug Gehringer. Java 3D API jump-start. Pearson Education, 2002.

[12] Doug Twilleager, Jeff Kesselman, Athomas Goldberg, Daniel Petersen, Juan Carlos Soto, Chris Melissinos. Java technologies for games. Computers in

Entertainment (CIE) Magazine - Theoretical and Practical Computer Applications in Entertainment archive, Volume 2 Issue 2, 2004.
[13] Andrew Davison. Pro Java 6: Game Development with Java: 3D and JOGL. Apress, 2007.
[14] Jene Davis. Learning Java Bindings for OpenGL (JOGL). AuthorHouse, 2004.
[15] M.Hall, D.G. Tragheim. The Accuracy of ASTER Digital Elevation Models, a Comparison to NEXTMap. 2010.
[16] Edwin Catmull. A Subdivision Algorithm for Computer Display of Curved Surfaces, University of Utah, Salt Lake City, 1974.
[17] Wikipedia. Root-mean-square deviation | Wikipedia, the free encyclopedia, 2015.
[18] Wikipedia. Mean absolute percentage error | Wikipedia, the free encyclopedia, 2015.
[19] Wikipedia. Coefficient of determination | Wikipedia, the free encyclopedia, 2015.
[20] Bernhard F. Arnold, Peter Stahlecker. Linear regression analysis using the relative squared error. University of Hamburg, Germany, 2000.
[21] Jeff A. Stuart, Cheng-Kai Chen, Kwan-Liu Ma, John D. Owens. Multi-GPU volume rendering using MapReduce. HPDC '10 Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, p. 841-848, 2010.