Assessment of Terrain Database Correlation Using Line-Of-Sight Measurements

2015

Leonardo Oyama

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


https://stars.library.ucf.edu/etd/1395

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
ASSESSMENT OF TERRAIN DATABASE CORRELATION USING LINE-OF-SIGHT MEASUREMENTS

by

CAPT LEONARDO SEIJI OYAMA
B.S. Instituto Militar de Engenharia, 2006

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Modeling and Simulation in the College of Graduate Studies at the University of Central Florida Orlando, Florida

Fall Term
2015

Major Professor: Brian Goldiez
ABSTRACT

The uncountable number of tools for the creation of synthetic terrains poses as a challenge for simulation interoperability. The permutations of tools, elevation maps, and software settings leads to combinations of poorly correlated virtual terrains. An important issue in distributed simulations is the lack of line-of-sight correlation. For example, in military networked simulations, consistent intervisibility between simulated entities is crucial for a fair-fight, especially when simulations include direct-fire weapons. The literature review presented in the Chapter Two discusses a multitude of interoperability issues caused by discrepant terrain representations and rendering engines noncompliant to any standard image generation process. Furthermore, the literature review discusses past research that strived for measuring (or mitigating) the correlation issues between terrain databases. Based on previous research, this thesis proposes a methodology for analysis of line-of-sight correlation between a pair of terrain databases. All the mathematical theory involved in the methodology is discussed in the Chapter Three. In addition, this thesis proposes a new method for measuring the roughness of a visual terrain database. This method takes into account the 3D dispersion of the vectors normal to the polygons in the terrain’s mesh. Because the vectors normal to the polygons are conveniently stored in most visual databases, the roughness calculation suggested here is fast and does not require sampling the terrain’s elevation. In order to demonstrate the proposed method, twin terrain databases and a tool were created as part of this thesis. The goal of this tool is to extract data from the terrain databases for statistical analysis. The tool is open source and its source code is provided with this thesis. The Chapter Four includes an example of statistical analysis using an
open source statistic software. The line-of-sight correlation analysis discussed here includes the terrain’s geometry only (terrain’s culture is not addressed). Human factors were not taken into consideration.
“In the beginning was the Word, and the Word was with God, and the Word was God. The same was in the beginning with God. All things were made by him; and without him was not any thing made that was made. In him was life; and the life was the light of men. And the light shineth in darkness; and the darkness comprehended it not.” (John 1:1-5, King James Version)
ACKNOWLEDGMENTS

I would like to thank the Brazilian Army’s Ground Operation Command for funding my studies. I cannot fail to express my gratitude to the Brazilian Army Aviation Command for providing support to me before my travel to the United States. I am thankful to the Brazilian Military Attaché and the Brazilian Army Commission for the administrative support they provided since I arrived in the United States. I would like to thank my advisor, Dr. Brian Goldiez, for his criticism and for providing me the guidance to write this thesis. I must express my appreciation for Nina Rothstein’s insightful reviews. I am grateful to ANACOM, Bruno Falcão for providing me technical support with the software tool used to generate the terrain databases tested in this research. I would also like to thank my parents, Tadao & Sonia, and my sisters, Priscila & Beatriz, who always helped me out in stormy situations. Finally, I would like to thank my beloved wife Juliana, who kept the home fires burning with love and devotion for our kids while I was dedicating most of my time to conclude this work.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................... ix
LIST OF TABLES ............................................................................................................. x
LIST OF ACRONYMS ....................................................................................................... xi

CHAPTER ONE: INTRODUCTION .................................................................................. 1

CHAPTER TWO: LITERATURE REVIEW ........................................................................... 5
  The Development of New Image Generators and the Arising of New Interoperability Issues .. 5
  Solutions for Mitigating TDB Correlation Issues in Distributed Simulations .................. 8
  TDB correlation analysis tools .................................................................................... 9
    LOSIM ................................................................................................................... 10
    ZCAP .................................................................................................................. 11
    LightBox ............................................................................................................. 15
    Other tools .......................................................................................................... 16
  Conclusion ............................................................................................................... 17

CHAPTER THREE: METHODOLOGY ............................................................................... 19
  Reasoning ............................................................................................................... 19
  General Structure of the Methodology ...................................................................... 21
  The Tool’s GUI ....................................................................................................... 23
  First Step: Store the Terrain’s Geometry and Calculate TDB’s Roughness ............... 27
    Reading the Database File ...................................................................................... 28
    Roughness calculation .......................................................................................... 33
    Determining Limits to the LOS Rays ................................................................... 35
  Second Step: Divide the TDB in Blocks .................................................................. 36
  Third Step: Seed the Sample Points ....................................................................... 37
    Line-to-Triangle Intersection ............................................................................... 39
    Seeding Method .................................................................................................. 46
  Fourth Step: Set up the LOS rays .......................................................................... 47
  Fifth Step: Generate Data for Correlation Analysis ............................................... 51

CHAPTER FOUR: DATA COLLECTION AND ANALYSIS ............................................. 55
  Generation of the TDBs ......................................................................................... 55
  Characteristics of the TDBs .................................................................................... 58
  LOS Test Results and Data Analysis ...................................................................... 61
    Block 4_7 Analysis .............................................................................................. 67
    Block 8_9 Analysis .............................................................................................. 74
  3D Output ............................................................................................................. 81

CHAPTER FIVE: CONCLUSION ..................................................................................... 84
  Performance .......................................................................................................... 84
  Terrain Features .................................................................................................... 85
  Use of Terrain’s Roughness as an Indicator of LOS Correlation Issues .................. 85
  Statistical Analysis, Human Factors, and More TDB Formats .............................. 86

APPENDIX: SOURCE CODE ............................................................................................ 87
  VTDBCorrelationAssessmentTool.pro .................................................................... 88
  main.cpp .............................................................................................................. 88
  util.h .................................................................................................................... 89
LIST OF FIGURES

Figure 2-1: Roughness ambiguity ........................................................................ 14
Figure 3-1: Interface for loading the TDBs ............................................................ 24
Figure 3-2: Tab widget for visualization of the baseline TDB .............................. 25
Figure 3-3: Tab widget for visualization of the subject TDB ............................... 25
Figure 3-4: Widget for setting the parameters for the LOS correlation analysis ...... 26
Figure 3-5: Simplified view of the OpenFlight hierarchical tree structure ............ 29
Figure 3-6: Data structures used to store the terrain geometry ............................ 31
Figure 3-7: TDBs with different size extents and their translucent grey bounding boxes 36
Figure 3-8: TDB divided in one hundred blocks .................................................. 37
Figure 3-9: Line’s parametric representation ....................................................... 40
Figure 3-10: Plane’s parametric representation .................................................. 41
Figure 3-11: The angles between the line segments are used to determine triangle inclusion 45
Figure 3-12: 10,000 sampling points (magenta dots) over the TDB’s surface ........ 47
Figure 3-13: GUI for configuring the attributes of the LOS tests ...................... 49
Figure 3-14: Sampling point, eye-points and directional vectors ....................... 51
Figure 3-15: Disagreement with respect to bounding box intersection ............... 53
Figure 3-16: LOS rays created in the Fifth Step .................................................. 54
Figure 4-1: Baseline TDB generated from NED 1/3 arc-second resolution ........... 57
Figure 4-2: Flat TDB generated from NED 1/3 arc-second resolution .................. 58
Figure 4-3: Terrain roughness per block ......................................................... 63
Figure 4-4: Average length difference of corresponding the LOS rays per block .... 65
Figure 4-5: Average length absolute difference of corresponding the LOS rays per block .... 66
Figure 4-6: LOS Ray Length Difference histogram for the Block 4_7 ..................... 69
Figure 4-7: LOS Ray Length Difference Q-Q normal plot for the Block 4_7 .......... 69
Figure 4-8: LOS Ray Length Absolute Difference histogram for the Block 4_7 ....... 72
Figure 4-9: LOS Ray Length Absolute Difference Q-Q normal plot for the Block 4_7 .... 72
Figure 4-10: LOS Ray Length Difference histogram for the Block 8_9 ................. 76
Figure 4-11: LOS Ray Length Difference Q-Q normal plot for the Block 8_9 .......... 76
Figure 4-12: LOS Ray Length Absolute Difference histogram for the Block 8_9 ....... 78
Figure 4-13: LOS Ray Length Absolute Difference Q-Q normal plot for the Block 8_9 .... 79
Figure 4-14: Largest uncorrelated LOS rays .................................................... 82
Figure 4-15: Colored posts indicating LOS correlation issues per block ............. 83
LIST OF TABLES

Table 4-1  Comparing an undulated TDB with a flat TDB .............................................................. 59
Table 4-2  Preliminary analysis of the TDBs.................................................................................. 60
Table 4-3  Statistics for the Block 4_7............................................................................................ 68
Table 4-4  Normality tests for the LOS Ray Length Difference in the Block 4_7 ...................... 70
Table 4-5  Wilcoxon tests for the LOS Ray Length Difference in the Block 4_7 ..................... 71
Table 4-6  Normality tests for the LOS Ray Length Absolute Difference in the Block 4_7 ...... 73
Table 4-7  Wilcoxon tests for the LOS Ray Length Absolute Difference in the Block 4_7 ...... 73
Table 4-8  Test for the Bounding Box Intersection Agreement Proportion in the Block 4_7 .... 74
Table 4-9  Statistics for the Block 8_9............................................................................................ 75
Table 4-10 Normality tests for the LOS Ray Length Difference in the Block 8_9 ..................... 77
Table 4-11 Normality tests for the LOS Ray Length Difference in the Block 8_9 ..................... 77
Table 4-12 Normality tests for the LOS Ray Length Absolute Difference in the Block 8_9 .... 80
Table 4-13 Wilcoxon tests for the LOS Ray Length Absolute Difference in the Block 8_9 ..... 80
Table 4-14 Test for the Bounding Box Intersection Agreement Proportion in the Block 8_9 .. 81
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGL</td>
<td>Above Ground Level</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>CDLOD</td>
<td>Continuous Distance-Dependent Level of Detail</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Interactive Simulation</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unity</td>
</tr>
<tr>
<td>IG</td>
<td>Image Generator</td>
</tr>
<tr>
<td>IST</td>
<td>Institute for Simulation and Training</td>
</tr>
<tr>
<td>LOD</td>
<td>Level of Detail</td>
</tr>
<tr>
<td>LOS</td>
<td>Line-of-sight</td>
</tr>
<tr>
<td>LOSIM</td>
<td>Line of Sight Intervisibility Measurement</td>
</tr>
<tr>
<td>LVC</td>
<td>Live, Virtual and Constructive</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MTM</td>
<td>Multiresolution Terrain Model</td>
</tr>
<tr>
<td>NED</td>
<td>National Elevation Dataset</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>RI</td>
<td>Roughness Index</td>
</tr>
<tr>
<td>RTDB</td>
<td>Runtime Terrain Database</td>
</tr>
<tr>
<td>STF</td>
<td>SEDRIS Transmittal Format</td>
</tr>
<tr>
<td>TDB</td>
<td>Terrain Database</td>
</tr>
<tr>
<td>USGS</td>
<td>United States Geological Survey</td>
</tr>
</tbody>
</table>
CHAPTER ONE: INTRODUCTION

In the 90’s, the effectiveness of training on networked computer simulations proved to be valuable in the preparation of military teams (Gorman & McMaster, 1992). Hence, the usage of networked simulation for collective training was leveraged. The first generation of networked simulators used shared memory to exchange data (Goldiez, 1993). However, interoperability challenges arose with the creation of asynchronous networked simulators (Schiavone, Sureshchandran & Hardis, 1997). One of the main issues that directly affects the outcomes of a distributed simulation is the differences in line-of-sight (LOS) intervisibility between networked simulated entities and/or the simulation participants (Tracy, 2004). Regarding human-in-the-loop simulations, intervisibility differences can cause biased results when the people participating in the simulation have access to direct fire weapons (Hoffman, Horan, McDonald, Paris & Uliano, 1994). Schiavone et al. (1997) wrote a comprehensive article reporting the major problems related to terrain database (TDB) correlation issues in Distributed Interactive Simulation (DIS) systems. In that article, the authors describe ZCAP, which is a suite of software tools developed to perform interoperability investigations in TDBs. Those authors state that the major motivation to eliminate TDB correlation problems is to avoid negative training transfer. The study of Schiavone et al. (1997) and the ZCAP final report (Schiavone, Sakude, Graniela, Morelos-Borja & Cortes, 1998) served as guidance to the research described here.

As the demand for new simulation technologies increased, so did simulation analysts’ concern about the validity of the collective training simulations. In 2007, a study group sought to
define the U.S. Army Aviation Strategy to improve collective training and interoperability in simulation. The study group pointed out some interoperability issues, such as the lack of terrain correlation in distributed simulation, and the absence of tools to detect TDB correlation issues in rendered images (Goldiez & Sottilare, 2008). These deficits were linked to challenges in creating “fair fight” scenarios for users. The report produced by that study group indicated that the terrain correlation research field was still a recurrent interoperability issue at that time. Indeed, to this date, the problem of integration between Live, Virtual and Constructive simulations remains without a final solution. Additionally, the lack of correlation between the distributed TDBs is still a barrier for full simulation interoperability (Barry, 2013; Peres & Moreira, 2014).

Following ZCAP, few articles describing TDB correlation tools have been published in the open literature (considering only articles published and available from archival sources). Some simulation companies (e.g., CAE) and government organizations (e.g., SEDRIS and U.S. Department of Army) developed tools addressing the TDB correlation issues (Palmer & Boyd, 2011), but few publications are available outside of those organizations. The research proposed in the present paper intends to bridge this gap (at least partially).

Three years after Goldiez & Sottilare (2008) wrote a document criticizing the absence of a tool capable to identify correlation errors in the rendered terrain images automatically, Palmer & Boyd (2011) described a method to uncover TDB errors using the Graphics Processor Unity (GPU) in large amounts of data. Although using GPU capabilities to uncover correlation issues among multiple TDBs was not an innovative technique (Tracy, 2004), this method was different from the previous ones because it did not rely on human intervention to detect errors in the TDB. In order to describe their methodology to uncover TDB correlation issues, Palmer & Boyd (2011) used a prototype tool dubbed “LightBox”. According to the authors, the LightBox could
fill the gap foreseen by Goldiez & Sottilare (2008) by checking the TDBs for existence of correlation issues that would not allow a “fair fight”. Even though the article written by Palmer & Boyd (2011) truly addresses the correlation issues in TDBs in distributed simulations, the tool they developed to accomplish that task (the LightBox) is neither available to test nor is purchasable.

The tool developed for this thesis is capable of comparing two different TDBs that represent the same gaming area, in order to help simulation analysts find problematic areas regarding LOS correlation. It does not produce statistical data for judging if the tested TDBs can provide good scenery for a “fair fight”, but it can show analysts the location of the top uncorrelated LOS test points. Details of the implementation of the tool will be further explained in this document.

Some ideas presented by Palmer & Boyd (2011) were used in the development of the software created for this research (e.g., the configurable LOS test volume). However, the tool is different than the software described by Hoffman et al. (1994), since it does not take in consideration human factors (e.g., the maximum range of human vision and the maximum rendering distance of the target objects were not considered). This thesis describes an open source implementation that merges several ideas described by researchers in the past. Since it is an open source development, it is hoped that others will improve it incrementally in the future to serve personal or organizational needs.

In addition, this study suggests a method to characterize the TDB geometry by taking into account its “roughness”. It is quite intuitive that undulated portions of the TDB will present more LOS correlation issues than nearly flat sub areas of it. For this reason, it is suggested here that the value of roughness could be used to reduce the number of test points in the sub areas where
the terrain’s roughness is nearly insignificant. Therefore, the calculation of the roughness of smaller areas of a TDB could provide the discernment of the quantity of LOS rays that should be cast to uncover intervisibility issues. The term “LOS ray” is extensively used along this thesis. A LOS ray is a straight line defined by one eye-point (or starting point of the LOS ray), one directional vector and one ending point, which is determined when the ray is cast from the eye-point through the direction defined by the directional vector. The ending point is located at the terrain or at the internal surface of the bounding volume that surrounds the terrain. However, this study does not define a relationship between the roughness of a sub area of a TDB and the number of LOS rays that should be cast to uncover correlation issues in that region. The methodology for calculating the roughness of the terrain geometry will be detailed in the Chapter 3. The terrain roughness is actually another way to check the correlation between two TDBs.
CHAPTER TWO: LITERATURE REVIEW

This chapter aims to survey the relevant publications addressing interoperability issues regarding terrain correlation. The first section describes the interoperability issues that emerged along with the development of new rendering techniques. The second section refers to works dedicated to mitigate TDB correlation issues in distributed simulations. Finally, the third section of this chapter is dedicated to describe tools that were developed aiming to analyze the correlation between TDBs.

The Development of New Image Generators and the Arising of New Interoperability Issues

The evolution of image rendering techniques and the development of new hardware to generate computer images led to the creation of numerous image rendering software and vendor unique runtime terrain database (RTDB) formats. Moreover, some techniques used to improve the performance of the image generation process (and some methods employed to improve the quality of the rendered images) can cause interoperability inconsistencies.

Despite the evolution of the technology, personal computers (PCs) have a limited amount memory dedicated to image generation; leaving them hardly capable of rendering an extensive and highly detailed TDB with high resolution. The Multiresolution Terrain Model (MTM) is a common method used to overcome the PC’s memory limitation (Xu, 2003), but has a propensity for interoperability issues. Multiresolution TDBs are composed of two or more levels of detail
(LODs). Each LOD is composed of a set of polygons. In order to increase the realism in realtime simulations, it is essential to represent portions of the terrain close to the observer with a large number of polygons (high LOD). However, in order to save computer memory, pieces of the terrain that are far from the virtual camera can be represented with a small number of polygons (low LOD). In networked simulations, each simulator is liable for managing its own image generation. Consequently, each simulator is responsible for transitioning the LODs in its own scene. Differences between the LOD transitioning methods might lead to interoperability issues. Moreover, interoperability inconsistencies occur if the networked simulators have TDBs composed of dissimilar number of LODs.

In order to make better use of the computer memory, current rendering techniques utilize various methods to determine which parts of the rendered terrain must be displayed with higher resolution. Usually, the position of the observer (or the camera position) with respect to the terrain mesh is a criterion to determine which areas of the terrain will require refined polygonization. Strugar (2009) proposed a method (dubbed Continuous Distance-Dependent Level of Detail or CDLOD) that uses higher resolution on pieces of the terrain that are close to the camera, based on the continuous distance from the camera position to the terrain mesh. Previous techniques did not consider the height of eye point with respect to the terrain mesh (which is the major advantage of CDLOD over the previous similar methods). The image generator (IG) developed to demonstrate the CDLOD algorithm renders the terrain directly from a heightmap (or elevation map) stored into the computer’s hard disk. A heightmap is a matrix data structure used to store elevation values of a geographic region. The terrain mesh is triangulated in runtime based on a regular grid of heights that represents the terrain. Thus, eliminating the need to store the triangulated mesh in the hard drive. The runtime triangulation of
the terrain can cause more interoperability issues in a distributed simulation, since there is not a safe way to ensure that the networked IGs are using similar methods to render the terrain at the same instance in time.

Usually, the RTDBs are composed of multiple LODs. The IGs can make use of several methods to render a terrain with multiple resolutions. For example, even if two networked IGs ingest the same TDB in a given simulation exercise, there is no guarantee that they will produce correlated images due to differences between their rendering engines. Similarly, there is a lack of assurance for interoperability for constructive and virtual simulators participating on the same training environment (even if they are sharing the same TDB). As in the previous case, it is not possible to ensure that the simulators will use the TDB in the same way. Lastly, the differences in the usage of the TDB may also cause a bias to the simulation results. For example, the geometry of the terrain is crucial for stimulation of the sensors of the agents controlled by the constructive simulator and this is often very different from a visual scene. Although the differences between rendering algorithms may cause interoperability inconsistencies, these issues will not be addressed in this thesis, because they are strictly related to the IGs and are out of the scope of this study. In addition, this work will not address the relation between TDBs and the behavior of simulation agents in constructive simulations.

Although polygon culling techniques can increase the performance of IGs, they also can lead to interoperability issues. Covelli, Rolland, Proctor, Kincaid & Hancock (2010) researched the effects of field of view on pilot’s performance in flight. In this study, they suggested the usage of the concepts “field of view” and “field of regard” to cull out the polygons that will be loaded to the computer memory in flight simulators. Polygon culling techniques also can cause interoperability issues in distributed simulations, because each simulator can use its own method
for selecting the polygons to be culled. However, as stated before, this study is not concerned with the runtime representation of the TDBs. This thesis is only concerned with correlation issues of TDBs representing the same area using distinct polygonal meshes.

Solutions for Mitigating TDB Correlation Issues in Distributed Simulations

Several authors described TDB generation methods to reduce interoperability issues due to TDB correlation problems (Schiavone & Goldiez, 2000; Hutchinson & Lewis, 2009; Johnson, Stanzione, & Lashlee, 2010; Hieb & Maxwell, 2012; Graniela & Proctor, 2012).

Schiavone & Goldiez (2000) proposed a detailed specification of TDB requirements in order to extend the Terrain Common Data Model to low cost training simulations. They criticized the lack of attention given to interoperability during the generation of the TDBs, which interferes with the seamless exchange and reuse of data between simulators.

Hutchinson & Lewis (2009) argued that most of interoperability problems caused by lack of correlation between TDBs can be solved by creating the terrain in real-time. They argue that government sponsored formats (e.g., SEDRIS) cannot solve interoperability inconsistencies, because those formats do not aim to reduce TDB correlation issues. They are only intended to reduce costs in TDB generation pipeline.

The paper written by Johnson et al. (2010) refers to the U.S. Army Geospatial Center efforts to reduce the interoperability issues caused by lack of terrain correlation between modeling and simulation systems and battle command systems. The article describes the GIS Enabled Modeling and Simulation project. This project aimed to permit entity based simulation to interact in the GIS environment directly during the training simulation.
Hieb & Maxwell (2012) reported that the generation of consistent TDBs that are suitable to modeling and simulation (M&S) systems and command and control (C2) systems figure as a challenge for the U.S. Army Simulations and Mission Command Systems. The authors described the efforts made by a team dubbed Overarching Integrated Product Team to mitigate TDB inconsistencies and redundancies that would not allow the integration between the simulators. This team identified that the major part of the resources expended in terrain generation were consumed in the initial steps of the process. Hieb & Maxwell (2012) claim that the U.S. Army could avoid needless wastes of resources by improving the terrain generation process. The team concluded that exchanging information between the stakeholders and the TDB producers can help to improve the standardization of the terrain creation process, since the TDB producers would have to certify their creation process with both simulation domains (M&S and C2).

Graniela & Proctor (2012) described a network-centric terrain generation process in replacement of the traditional terrain creation pipeline. The advantage of this is the modifications made to a single database are replicated automatically to analogous TDBs in the network in near real-time. The correlation errors between the TDBs are mitigated by generating terrains from a common TDB representation.

Although these studies are related to the research proposed in this thesis, the mitigation of correlation issues between TDBs is out of the scope of this thesis.

**TDB correlation analysis tools**

This section is intended to describe tools developed to assess correlation issues in TDBs representing the same terrain. The tools discussed here are publically available through archival
sources. Government organizations and companies may have created tools but they are not available for research purposes. Non-archival tools were not described in this thesis because it was not possible to retrieve detailed information about the work enclosed with companies or organizations. Due to the ability to discern fair fight scenarios, the assessment of correlation between TDBs in a networked simulation is important for providing insight on training effectiveness. The first tool discussed in this section is dubbed Line of Sight Intervisibility Measurement (LOSIM). The second tool described here is ZCAP (actually, ZCAP is a set of tools). The third tool portrayed in this section is the LightBox. Finally, the last subsection of this section is dedicated to discuss other tools that were created to measure (or uncover) correlation issues between TDBs. These additional tools have served as a base of knowledge for the development of this thesis. For this reason, they are thoroughly described in following subsections.

LOSIM

Developed by the Institute for Simulation and Training (IST) in the 90’s, LOSIM evolved as a metric for the assessment of the correlation level of the rendered images of a pair of TDBs with respect to the intervisibility between the simulation entities. According to Hoffman et al. (1994), intervisibility is the capability of an observer positioned on the TDB to see a target located in another position on the same TDB. In order to obtain a correlation index, the LOSIM software seeds the TDB with sample points and checks for the existence of a LOS between the observer’s sample point and the target located at another sample point. This process is repeated on the second TDB. Then the results are collected for statistical analysis.
One of the strengths of the LOSIM software is that it considers the attributes of the IG (e.g., the IG’s screen resolution) and some human factors (e.g., naked human eye’s maximum visibility range) when checking for the existence of a straight line between the observer and the target. For example, the height of the position eye of a dismounted infantry unit (with respect to the ground level) is considered zero, while the same attribute for a helicopter is considered 15.25 meters. The dismounted infantry unit cannot be visualized (by a human naked eye) if its distance to the observer is greater than 1.2 kilometers while a helicopter cannot be seen if its distance to the observer is greater than 10.2 kilometers (Hoffman et al., 1994).

The LOSM generates an output file as result of the data collection. This file can be read by the IBM SPSS Statistics software. The study conducted by Hoffman et al., (1994) demonstrated a method to discern a fair fight based on hypothesis testing. However, it did not define a tolerance value for the LOS dissimilarity between the pair of analyzed TDBs, letting this tolerance level to be set by the user of the software.

ZCAP

ZCAP is a set of software tools developed by the IST’s Visual Systems Lab for the U.S. Army Simulation, Training and Instrumentation Command. ZCAP was designed to assess the correlation between two databases representing the same terrain in their native format. The report by Schiavone et al. (1998) and the article by Sakude et al. (1998) contain valuable information about ZCAP.

The Grid Method Sampling is the algorithm used by ZCAP to calculate the terrain elevation. Superior to its contemporary counterparts, it consumes less of the computer memory
and it is quicker than similar algorithms in that date. ZCAP performs statistical analysis to assess the correlation between the baseline TDB and the subject TDB. This includes mean, median, variance, standard deviation, skewness and more. Additionally, using Kappa statistic ZCAP could perform culture and LOS correlation tests. The LOS tests performed by ZCAP are different from the LOS tests executed by the tool developed for this thesis. ZCAP uses fixed length LOS rays to create categorical data: the ray may or may not intersect a polygon in the terrain mesh. The length of the ray is calculated in such a way that one-half of the rays are likely to intersect the terrain (or its features). Finally, in order to reduce the correlation issues between the baseline TDB and the subject TDB, ZCAP has a remediation tool based on the constrained least square fitting method.

According to Schiavone et al. (1998), innovations in the TDB correlation research were pointing to the analysis of the terrain roughness and to the research of terrain polygonization algorithms in that date. The tool that would be part of the ZCAP suite uses three different methods to assess the roughness of the terrain: sigma-t, a method based on multiple linear regressions, and the roughness index (RI). The sigma-t method analyzes the standard deviation of the elevation of the terrain. The method based on multiple linear regression fits a plane to the grid sample points in order to calculate the Mean Square Error, which is a measure of standard deviation with respect to the fitting plane. The third method used by this tool to assess the terrain roughness is based on the RI. The RI is interpreted as the average of the discrete norm of the second order gradient of the terrain elevation. As stated by Schiavone et al. (1998), the purpose for calculating the terrain roughness was to classify and select portions of a terrain for terrain correlation analysis.
The terrain roughness is calculated in a different way in this thesis because it takes into account the normal vectors of the polygons of the terrain mesh. Since the terrain’s roughness is an indicator of possible LOS correlation issues in distributed simulations, the next subsection is dedicated to the discussion of methods for determining the terrain roughness.

*Terrain Roughness Determination*

The concept of terrain roughness is important in several research fields. Biologists calculate the terrain roughness for the analysis of the habitats of some animals. Terrain roughness is also important for predicting the wind profile near to the terrain surface. In computer graphics, the computation of terrain roughness is important for calculating the number of polygons required to generate a thorough synthetic representation of the terrain. Flat areas can be represented with few polygons, whereas the accurate representation of rough areas require a larger number of polygons.

Besides the roughness computation methods implemented in ZCAP, there are many other ways for determining the terrain ruggedness. According to Hoffman & Krotkov (1989), the roughness calculation methods developed prior to 1989 use a single terrain parameter for calculating the roughness value. In addition, Hoffman & Krotkov (1989) argue that the roughness concept is ambiguous. The Figure 2-1 illustrates the roughness ambiguity described by these authors in terms of frequency, amplitude and repetition pattern. In order to address this issue, they proposed a method capable of discriminating the surface roughness with respect to amplitude, frequency and correlation. They also argue that terrain roughness could not be
represented by a single number. Instead, they state that the terrain roughness should be expressed in terms of a vector.

Figure 2-1: Roughness ambiguity

The roughness calculation method implemented in this thesis follows the same principle of the technique proposed by Sappington, Longshore, & Thompson (2007). Sappington et al., (2007) created a software for calculating the terrain roughness based on the theory developed by Hobson (1972). According to Sappington et al., (2007), it is possible to determine terrain roughness by computing the 3D dispersion of vectors orthogonal to planar surfaces on the
terrain. The software created by Sappington et al., (2007) assumes that the terrain is divided into contiguous rectangular cells forming a grid. Each cell in the grid has a height value assigned to it (the grid is an elevation map). The goal of the software created by Sappington et al., (2007) is to assign a roughness value to each cell in the grid. The roughness of a cell in the grid is proportional to the 3D dispersion of the normal vectors to the cells neighboring to it. The neighborhood size can be set by the software’s user, but according to the authors, the use of large neighborhoods results in low roughness values.

It is noteworthy that the method for computing terrain roughness implemented in this thesis assumes that the terrain’s model is a polygonal mesh, not an elevation map.

LightBox

The LightBox is a tool developed by SAIC. The LightBox claimed superiority to the previous correlation analysis tools because it utilizes the processing power of the GPU to uncover correlation issues between TDBs. Palmer & Boyd (2011) also claim that the LightBox is capable of detecting correlation issues in the TDBs without human intervention.

However, the usage of the GPU processing power to uncover correlation issues in TDBs was a concept previously proposed by Tracy (2004). The LightBox relies on the SEDRIS spatial reference model for coordinate system conversions and for TDB projection (Palmer & Boyd, 2011).

The LightBox utilizes a methodology that is worth mentioning. In order to reduce the number of LOS tests that are required for assessing the correlation between TDBs, the tool is capable of detecting the location of the terrain’s features. The eye-points are cleverly placed
around the terrain features, because LOS correlation issues are more likely to occur near terrain features (trees and buildings). To date, the LightBox is the property of the company Leidos.

Similar to areas near to terrain features, ridges and valleys are terrain locations where LOS correlation errors are likely to occur. Palmer & Boyd (2011) did not mention any method implemented in the LightBox for identifying ridges and valleys in visual TDBs. The identification of ridges and valleys in elevation maps is an important topic in geomorphology. In the past, the identification of ridges and valleys was subject to human visual analysis of the maps, but to date there are algorithms for addressing this issue computationally (Chang, Song & Hsu, 1998; Bangay, de Bruyn & Glass, 2010).

Since the LOS correlation issues are more likely to be present in areas near to ridges and valleys, the detection of these geomorphological features could reduce computational effort for measuring LOS correlation issues between TDBs. Although, it is worth mentioning that the terrain roughness was parameter used as indication of LOS correlation issues in this thesis.

Other tools

SEDRIS has not only developed tools to convert several terrain databases formats to and from its native format, but also a tool dubbed SEE-IT (Sedris.org, 2014). SEE-IT is capable of performing consistency checks between TDBs. However, the format of the TDBs being tested must be SEDRIS native format. Besides this limitation, SEE-IT development and source code is limited to SEDRIS associates. Anyone can download and use SEDRIS tools, but the development of existing tools is limited to a restricted group of developers.
According to Kang et al. (2014), in order to achieve interoperability among distinct databases, the native databases should be converted to a neutral database format (SEDRIS Transmittal Format, or STF). Kang et al. (2014) argue that detecting flaws in the conversion process to/from STF is not a simple task because of the absence of tools to compare the TDBs in their native formats with its STF counterparts. For this reason, they developed a tool capable of visualizing simulation scenarios in many native database formats. This tool can be used to investigate data conversion discrepancies to/from STF. The solution these authors created is based in human visualization, thus it is error prone. Moreover, the source code of SEDRIS’s tools are available only to its associates (Sedris.org, 2014).

Palmer & Boyd (2011) reported the existence of other tools (e.g., CAE Side-by-Side and U.S. Army OneSAF® Automated Test Tool/Visual Test Tool). Nevertheless, no information about these tools could be found in archival sources.

Conclusion

Since the beginning of the 90’s much research addressed interoperability issues regarding the correlation of distributed TDBs. This work developed tools capable of assessing the correlation between the TDBs used in networked simulations. Only one of these tools is available (e.g., SEDRIS SEET-IT), while all the others are commercial or restricted to governmental organizations.

The tool developed in this thesis was designed to demonstrate a methodology for assessing the correlation between TDBs using line-of-sight measurements. The methodology demonstrated here is similar to the methodology employed in the design of previous tools, but
there are some important differences, which will be highlighted in the next chapters. The key differences are the numerical data produced as the result of the LOS tests and the method used to compute the terrain roughness. It is noteworthy that the tool is published in this thesis. It provides an opportunity for others to add more features to its utility.
CHAPTER THREE: METHODOLOGY

Reasoning

From a review of the literature, the methodology discussed in this thesis uses LOS measurements for assessing the correlation between two TDBs. The first TDB loaded into computer memory is henceforth referred to as baseline TDB and the second one is henceforth referred to as subject TDB. The LOS measurements take into consideration the terrain geometry.

In addition, the tool for this thesis is capable of computing the terrain roughness without sampling its elevation. The 3D dispersion of the vectors normal to the terrain’s polygons is used for determining the roughness of the terrain’s surface. This method for calculating the roughness takes advantage of the fact that the normal vectors of the polygonal surfaces are stored in the hard drive in most visual TDB formats.

Correlation assessment for the present study excluded the characteristics of IGs and GPU-based assessment techniques. Unlike the methodology reported by Tracy (2004) and the LightBox software (Palmer & Boyd, 2011), no assumptions on IG’s rendering capabilities were taken into consideration in the development of the tool described in this thesis. The tool described herein is intended to have a broad base of use as well as be open source so others can add to it.

This thesis will focus on the terrain’s geometry to assess the correlation between two TDBs. As the GPU-based correlation methods mentioned before, the correlation assessment
methodology proposed in this thesis is a necessary but not sufficient condition for interoperability.

This research is concerned with the intersections of the LOS rays with the terrain mesh. LOS correlation is important in distributed military simulations involving direct fire weapons because the effectiveness of these weapons depend on the existence of a LOS between the sighting entity and the target entity. It is also important in non-military applications such as in modeling airports for flight and air traffic operations. The lack of LOS correlation affects the simulation’s results. The unreliable nature of the simulation results as a lack of LOS correlation poses a liability for negative learning situations. In addition, the lack of LOS correlation causes visual disturbance in distributed simulations (e.g., power lines crossing the terrain). Visual defects in the scene contribute for distracting the trainees from the goals of the simulation.

Although the correlation between the geometry of surfaces of virtual terrains is insufficient for ensuring conditions for a fair fight in training simulations, it is noteworthy that all the features of the terrain (e.g., rivers, roads, trees, and buildings) are drawn over the surface of the terrain. Thus, in order to determine intervisibility between two entities in simulation it is always necessary to check if the terrain mesh does not occlude the target entity (terrain’s blockage is always checked first). Additionally, the terrain mesh is composed exclusively of opaque polygons. Therefore, the transparency of the polygons is not an issue for determining intervisibility with respect to the terrain geometry, which makes the correlation analysis simpler and faster than it would be if terrain’s features had taken into account.
General Structure of the Methodology

In order to improve the performance of the correlation analysis, the tool for this thesis needs to reorganize the data stored in the TDBs in a more convenient manner. Sequentially, the roughness of the databases must be computed. The calculation of the terrain’s roughness is important in the LOS correlation analysis because it allows the concentration of computational and subsequent analysts’ efforts in LOS tests on non-flat portions of the terrain. The roughness computation demonstrated in this thesis takes advantage of the vectors orthogonal to the polygons stored in the databases. In sequence, aiming to detect areas of the TDBs where there are more correlation issues, the TDBs are divided in smaller portions (blocks) by the user of the tool. After the blocks’ creation, the user must determine the number of sampling points and the attributes of the LOS tests to be performed. In short, sampling points are points positioned exactly on the terrain surface, which means that the elevation of a sampling point with respect to the ground is zero. The starting points of the LOS rays (henceforth referred to as eye-points) must be placed above ground level (AGL). Therefore, the sampling points are essential for positioning the eye-points. Finally, the LOS tests are executed and the data resulting from the tests is stored in hard drive for statistical analysis.

For the reasons mentioned in the previous paragraph, the tool developed for this thesis executes five steps for assessing the correlation between TDBs. This following paragraphs summarizes these steps and provides an insight into the mathematical approach used for creating the tool.

The first step is to store the terrain’s geometry in data structures that will facilitate its use in the next steps. The roughness of the entire database is calculated in this step. In addition, a
A rectangular parallelepiped is drawn surrounding the geometry of each TDB. The purpose of this bounding volume is to serve as a barrier to the LOS rays that do not hit the terrain geometry. The second step is to divide the TDB in blocks. The user selects the number of blocks by setting the number of lines and columns of blocks. A polygon is considered to be within a block if its centroid is inside the block. Additionally, the roughness of the each block is calculated in this step. The third task is to “seed” the TDBs with sample points. The sample points are distributed within each block in a grid format. These sample points are where LOS rays will be generated. The terrain’s elevation is calculated for each sample point in this step. The elevations are stored for further use. The fourth step is determine how the LOS rays will be cast. The direction of the LOS rays and starting point’s height of each ray with respect to the terrain are set in this task. It is worth mentioning that in this step the user indirectly determines the sample size by setting the number of rays that will be cast per sample point. Actually, the user’s intervention is essential from the second to the fourth steps. The fifth step is to generate the data for correlation analysis. The length of each LOS ray and the difference between each pair of rays (each ray in the baseline database has a counterpart in the subject database) is calculated in this step. This numerical data is stored for computing the correlation between the databases. Categorical data is also stored for each ray (the ray may hit or may not hit the terrain geometry). The data produced in this task can be saved to a text file.

The user can visualize the largest uncorrelated LOS rays in the visualization widgets created as part of this thesis when the tests are concluded. The tool includes a graphical user interface (GUI), which is presented in the next section. The tool’s GUI is related to the methodology workflow, therefore it is easier to understand the methodology’s steps by associating each step with a part of the GUI.
The Tool’s GUI

The tool’s GUI was created with the intention of providing an interactive visual input/output to the user. In addition, at the beginning of the development, the tool’s visual output served as an aid for visualizing the TDBs and for debugging the software. The GUI was created with Qt (Qt, version 4.8.6). Qt is a cross-platform C++ application framework that can be seamlessly integrated with most of the libraries written in C programming language (e.g., the OpenFlight API).

The GUI is composed of four tab widgets. The first tab (Figure 3-1) offers the required interface for loading the TDBs. Although each TDB is loaded into computer memory in a dedicated thread, only one TDB can be loaded at a time due to a limitation of the OpenFlight API (OpenFlight API Documentation, 2014). The motivations for handling the TDBs with the OpenFlight API are discussed later in this chapter. In short, all the activities performed in the First Step of the methodology, which is described in detail ahead in this chapter, are related to this tab widget. The results of the preliminary analysis of the TDBs are displayed in text boxes located right below the push buttons used for loading the TDBs. The content of the text boxes and the terrain geometry can be saved to a text file using the button “Save Terrain Geometry”. In order to enable the other three tab widgets and proceed to the next steps in the correlation analysis, the user must select one LOD for each TDB using the select boxes and push buttons immediately below the text boxes. The LOD number ‘0’ represents the most detailed LOD present in the TDB. Once the LOD is selected, this selection cannot be undone.
The second and the third tab widgets (Figures 3-2 and 3-3) are visualization widgets. The user can visualize the TDBs and other geometric entities from the correlation analysis (e.g., LOS rays). These tab widgets are not related to any specific methodology’s step, but they can be used for visualizing the TDBs or other geometric entities at any time. The user can navigate through the TDBs using the keyboard and the mouse. The navigation keys for translation of the camera are ‘A’, ‘W’, ‘S’, ‘D’, ‘Shift’, ‘Space’. The keyboard arrows can be used for rotating the camera.
Figure 3-2: Tab widget for visualization of the baseline TDB

Figure 3-3: Tab widget for visualization of the subject TDB
The fourth tab widget (Figure 3-4) is related to the Second, Third, Fourth, and Fifth steps in the methodology. This tab widget is divided into four frames. The frame named ‘Block Size’ is related to the Second Step. The frame named ‘Number of Eye-Points Starting Positions per Block’ is related to the Third Step. The frame named ‘Line of Sight Test Attributes Setup’ is related to the Fourth Step. Finally, the frame named ‘Start Line of Sight Tests’ is related to the Fifth Step of the methodology.

Figure 3-4: Widget for setting the parameters for the LOS correlation analysis
First Step: Store the Terrain’s Geometry and Calculate TDB’s Roughness

The first step performed by the correlation assessment tool consists of loading the TDBs to computer memory. The user selects the TDBs by using the GUI. In order to demonstrate the methodology for assessing the correlation between two TDBs, the tool is endowed with the ability to manipulate OpenFlight TDBs. Although several current runtime visualization software do not use OpenFlight data structure for handling the scene graph, many simulators convert OpenFlight databases to their own internal formats. In addition, there are numerous legacy databases in OpenFlight format. For this reason, the OpenFlight format was chosen for this research. There are several TDB formats commonly used in simulations, but the tool developed for this thesis supports only OpenFlight databases because its central intention is to demonstrate a correlation assessment methodology. Since the source code for the tool is published within this thesis, support to different database formats may be included in the future.

Because of its extensive documentation (available within the software package) and its relative ease of use, the OpenFlight Application Programming Interface (API) is incorporated into the tool for manipulating OpenFlight databases. According to OpenFlight API Documentation (2014), the API supports seven types of projections (Flat Earth, Trapezoidal, Round Earth, Lambert, UTM, Geodetic, and Geocentric) and five Earth ellipsoid models (WGS 1984, WGS 1972, Bessel, Clark, NAD 1927). It also has functions for converting projection coordinates to and from database coordinates. These conversion functions are useful for determining matching locations within the baseline and the subject TDBs. Although the OpenFlight API is maintained by a private company (Presagis), it is freely available for developers. Open source libraries for handling OpenFlight databases (e.g., OpenSceneGraph
OpenFlight Plugin) exist, but the documentation is inadequate compared to the OpenFlight API’s documentation.

Reading the Database File

In order to demonstrate the methodology proposed in this thesis, TDBs are required. Because OpenFlight databases are used as sources of terrain geometry, a brief explanation of OpenFlight format may help the reader to understand details of the methodology proposed here.

In OpenFlight databases, the data is organized in a hierarchical tree structure (also known as graph). The graph is composed of nodes that are classified by type. Each node stores data according to its type and according to its function in the database. As stated in the OpenFlight Specification (2014), the most important node types are header (the root of the tree), group, object, LOD, polygon, and vertex (this list is not exhaustive). The Figure 3-5 illustrates the simplified structure of an OpenFlight database.
OpenFlight databases are designed for visualization purposes. For this reason, they offer a way for the user to load the data into memory. However, the correlation assessment method proposed here is based on the geometry of the terrain only. Therefore, the OpenFlight scene graph is not appropriate for this research. In order to optimize the correlation analysis, it is necessary to reorganize the data in a more convenient way.

In OpenFlight databases, LOD nodes are subsets of the database that can be switched on and switched off. The OpenFlight API has functions for switching the LOD and functions that allow traversing the tree hierarchy (OpenFlight API Documentation, 2014). The traverse
functions are intended to retrieve data from the database. If a LOD node is switched off, none of the data under its hierarchy is retrieved by the traverse function. Usually, runtime applications switch the LOD nodes on and off according to the distance between the LOD node’s center and the viewer (virtual camera). According to the OpenFlight Specification (2014), the switch distance and the position of the center of the LOD node are attributes of the LOD node that indicate to visualization software how to render the scene. However, some IGs do not use these OpenFlight specifications for displaying images due to performance optimizations.

The tool developed for this thesis does not make use of these LOD node’s attributes. In order to organize the terrain geometry in a convenient way, the database’s tree hierarchy is traversed and polygons that belong to the same level of resolution are stored in the same array. The database is traversed several times, until the all content of the OpenFlight LOD nodes is stored into the LOD arrays. The Figure 3-6 illustrates the LOD arrays used to store the terrain geometry. Each LOD array contains enough polygons for representing the entire gaming area, but with different resolutions.
All the polygons stored into the LOD arrays must be triangles. If the database has non-triangle polygons, they must be triangulated in order to be stored into the arrays. If all the polygons are planar, the triangulation will not cause geometric changes to the terrain’s shape. The triangulation is necessary because the LOS algorithm described in this thesis is only valid for triangular polygon meshes. The TDBs used in this thesis are composed of triangles only. Therefore, it was not necessary to implement triangulation functions. The polygon triangulation functions may be added to the tool in the future.

Figure 3-6: Data structures used to store the terrain geometry
Before storing the polygons into the LOD arrays, the tool computes each polygon’s centroid. The centroid of the polygon is used to determine if the polygon is within a block (the blocks are created in the next step). The centroid is a point in the three-dimensional (3D) space, thus it can be represented by a 3D vector. The computation of a triangle’s centroid is quite simple. Each component of the centroid is the average of the components of the triangle’s vertices. For example, the centroid of a triangle whose vertices are \((A_x, A_y, A_z), (B_x, B_y, B_z)\) and \((C_x, C_y, C_z)\) is calculated using the following formula:

\[
\overrightarrow{G} = \left(\frac{A_x + B_x + C_x}{3}, \frac{A_y + B_y + C_y}{3}, \frac{A_z + B_z + C_z}{3}\right).
\] (1)

After the polygon’s centroid is computed, it is stored into the polygon data structure. In turn, the polygon data structure is stored into the LOD array. The polygon data structure used in this thesis is composed of an array of vertices, the normal vector to the polygon surface, and the position of its centroid. OpenFlight databases have the polygons’ normal vectors stored into the polygon data structure (as most visual databases do). For this reason, it was not necessary to calculate the normal vectors of the polygons in this research. However, if the database does not provide the normal vectors for its polygons, they can be quickly computed. The normal vector for a planar a polygon can be obtained by computing the cross product of two sides of the polygon (the ordering method used for storing the vertices matter). It is worth mentioning that the normal vector stored into the polygon data structure are normalized (length equals one).
Roughness calculation

The methodology for calculating the roughness of the TDBs is based on the 3D dispersion of the normal vectors of the terrain’s polygon mesh. As mentioned before, several methods take into account the normal vectors of the terrain mesh’s polygons for characterizing the roughness of the terrain. However, whilst these methods rely on a regular grid of heights (also known as height field or heightmap) for determining the terrain’s roughness, the technique described here can be applied to any polygonal mesh.

The roughness determination method for this research takes advantage of the TDB’s polygonal mesh. Many visual TDBs store the normal vectors to its polygons and vertices in order to avoid runtime calculation. Typically, virtual simulators (e.g., flight simulators) require high performance image generators. Thus, pre-calculated normal vectors can help to improve the speed of image generation process by eliminating rearward facing polygons.

The advantage of using the technique described here is that it does not require sampling the TDB in order to retrieve the terrain’s elevation in a regular grid format. Sampling the TDB for retrieving the elevation of the sampled points is a computer-intensive task. For this reason, the method proposed in this thesis can avoid lengthy computer calculations. The computation is not complex and can be executed quickly once the TDB is loaded to the computer memory. The roughness is calculated for each the LOD array. However, it is worth mentioning that only one LOD array per TDB will be used in the next steps described in this chapter. All the polygons in the LOD array are taken into account. The distribution of the normal vectors to the polygons of the terrain mesh is computed based on the standard deviation of the normal vectors.
For example, assuming that a given LOD array has \( N \) polygons, the average vector of normal vectors of the polygons was obtained by the formula:

\[
(\bar{x}, \bar{y}, \bar{z}) = \left( \frac{\sum_{i=1}^{N} x_i}{N}, \frac{\sum_{i=1}^{N} y_i}{N}, \frac{\sum_{i=1}^{N} z_i}{N} \right).
\] (2)

Note that the average normal vector is calculated in a manner similar to that is used to determine the vector strength, created by Hobson (1972). However, in Hobson (1972) the components of the vector strength are not divided the number terrain blocks used to calculate the average. In the method described in this study, the 3D dispersion of the normal vectors is measured by length of the standard deviation of the entire “population of polygons” in the LOD array. The vector standard deviation can be obtained by the Equation 3.

\[
(\sigma_x, \sigma_y, \sigma_z) = \left( \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N}}, \sqrt{\frac{\sum_{i=1}^{N} (y_i - \bar{y})^2}{N}}, \sqrt{\frac{\sum_{i=1}^{N} (z_i - \bar{z})^2}{N}} \right).
\] (3)

Finally, the terrain roughness is obtained by computing the length of the vector standard deviation. It can be obtained by the following formula:

\[
| (\sigma_x, \sigma_y, \sigma_z) | = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2}.
\] (4)

According to this method, the length of the standard deviation method is zero if, and only if, the terrain is completely flat. A steep terrain can be considered flat if the normal vectors to its
polygons point to the same direction. In other words, the roughness calculated by using this method is not affected by the slope of the terrain. It is noteworthy that this method captures only the variability of the normal vectors; it does not capture the amplitude of the roughness.

Determining Limits to the LOS Rays

The last task in the first step is determining limits to the LOS rays. It is important to impose limitations to the length of the LOS rays because the length of the LOS rays is used to calculate the correlation between the TDBs.

Limitations for the LOS rays are important because without limitations, rays that do not intersect the terrain geometry would not have a measurable length. The goal of the methodology described in this thesis is to determine the correlation between the TDBs comparing (measuring the difference) the length of the LOS rays cast on both TDBs. For this reason, a rectangular parallelepiped is created surrounding the terrain geometry. The size of this bounding box must be the maximum of the extents of both TDBs in order to ensure that all polygons of the terrain are inside the bounding box. The Figure 3-7 illustrates two TDBs representing the same terrain. These databases were intentionally built with different size extents. The bounding box is large enough to wrap the geometry of both.

It is noteworthy that the world coordinate system (latitude, longitude and altitude) must be used in order to determine the vertices of the rectangular parallelepiped. The local coordinate systems of each database are meaningless for determining correlation between the TDBs.
The second step consists of dividing the TDB in blocks. The motivation for this step is calculating the roughness of each block. As mentioned before, the roughness of the TDB is an indicator for possible LOS interoperability issues. Although the tool does not give to the user the opportunity to analyze each block with more sampling points after the primary analysis is finished, this feature is important and may be added to it in the future.

In order to divide the TDBs in blocks, the user must determine the number of blocks by changing the input values in corresponding user interface fields (lines and columns). After this selection is done, the TDB is divided in block data structures. The block is a subset of the TDB. Each polygon stored in the LOD array has an index that varies from zero to the number of polygons stored in the LOD array minus one. In order to save memory, only the indices of the polygons are stored into the block data structure. As stated before, a polygon is considered to be inside the block if its centroid is within the block. The roughness of each block is calculated as

Second Step: Divide the TDB in Blocks
described in the previous step, but only the polygons whose centroids are within each block are taken into account for this computation.

It is worth mentioning that the block’s extents are determined based on the baseline database extents. The world coordinates (latitude, longitude, altitude) of the extents of each block are the same for both baseline and subject TDBs. The Figure 3-8 illustrates the TDB divided in one hundred blocks (ten lines and ten columns).

![Figure 3-8: TDB divided in one hundred blocks](image)

**Third Step: Seed the Sample Points**

The third step consists of seeding the TDBs with sampling points. The sampling points are points in space where sampling for LOS will occur. Eye points are created from sample
points and are orthogonal projections of the sample points onto the terrain mesh (note that the eye-points are created in the Fourth Step). The sample and resulting eye-points must be placed above the terrain surface. Otherwise, LOS tests would be meaningless. The eye-points are placed somewhere above the sampling points. The sampling points and their respective eye-points share the same latitude and longitude coordinates. This step is an essential part of the methodology because the terrain’s elevation at a given latitude and longitude location is unknown a priori.

In order to determine the terrain’s elevation at a given latitude and longitude, a downward ray is cast from a starting position above the terrain mesh (the latitude and longitude are constant along the ray). The Z coordinate of the ray’s starting position is set equal to the maximum elevation found in both TDBs. This Z value is taken from the bounding box described in the First Step. The intersection between this ray and the terrain mesh determines the terrain elevation at the given latitude and longitude location.

In short, the task of seeding the TDBs with sampling points is achieved by determining the intersection between the downward ray and the terrain’s polygonal mesh. Since the mesh is composed of triangles, it is necessary to determine if the ray intersects the triangles in the mesh. For this reason, the triangles are tested iteratively until an intersection point is determined. When the first intersection is found, the testing algorithm stops. It is not necessary to test more triangles, because the triangles of the terrain mesh do not overlap.

There are numerous methods for determining the intersection between a line and a triangle. The method described in this section hereinafter is implemented in the tool for this thesis.
Line-to-Triangle Intersection

The first step for determining if a line intersects a triangle is to discover the location where the line intersects the plane defined by the triangle. The intersection between a line and a plane is determined with help of analytic geometry and vector algebra. The line and the plane are represented by their parametric equations. The intersection between the line and the plane is obtained by solving the linear system from these parametric equations.

The set of all the points in a given line can be expressed in a parametric manner. For example, if the point \( P \) is in the line and the line’s direction is determined by the nonzero vector \( A \), then the line can be parametrized in \( t \) (\( t \) being a real number), such that the set of all points in this line can be expressed by the formula:

\[
\vec{P} + t \cdot \vec{A}.
\]  

(5)

This set is called a line through \( \vec{P} \) spanned by \( \vec{A} \) (Apostol, 1967, Chapter 13). The Figure 3-9 illustrates this case.
Analogously, the set of all points in a given plane also can be denoted parametrically. For example, if the point $\vec{P}$ is in the plane and the plane is parallel to the plane spanned by the linearly independent nonzero vectors $\vec{A}$ and $\vec{B}$, then the plane can be parametrized in $t$ and $s$ ($t$ and $s$ being real numbers), such that the set of all points in this plane can be expressed by the formula:

$$\vec{P} + s \cdot \vec{A} + t \cdot \vec{B}.$$  \hspace{1cm} (6)
This set is called a plane through $\vec{P}$ spanned by $\vec{A}$ and $\vec{B}$ (Apostol, 1967, Chapter 13). The Figure 3-10 illustrates this case.

![Diagram of a plane and vectors](image)

**Figure 3-10: Plane’s parametric representation**

Based on these parametric definitions, it is possible to determine the intersection between a line and a plane by solving a linear system in three variables. In practice, the ray’s starting position is denoted by $\vec{S}$, the ray’s directional vector is denoted by $\vec{D}$, and the triangle in the
terrain mesh is described by its vertices $\vec{A}$, $\vec{B}$ and $\vec{C}$ (henceforth referred to as triangle ABC), such that:

$$\vec{A} = (A_x, A_y, A_z).$$  \hspace{1cm} (7)

$$\vec{B} = (B_x, B_y, B_z).$$  \hspace{1cm} (8)

$$\vec{C} = (C_x, C_y, C_z).$$  \hspace{1cm} (9)

$$\vec{D} = (D_x, D_y, D_z).$$  \hspace{1cm} (10)

$$\vec{S} = (S_x, S_y, S_z).$$  \hspace{1cm} (11)

According to the theory previously discussed, every point $(x, y, z)$ in the ray can be expressed by the following parametrical formula in $t$ ($t$ being a real number):

$$(S_x, S_y, S_z) + t \cdot (D_x, D_y, D_z) = (x, y, z).$$  \hspace{1cm} (12)

Similarly, every point $(x, y, z)$ in the plane defined by the triangle ABC can be expressed by the following parametrical formula in $r$ and $s$ ($r$ and $s$ being real numbers):

$$(A_x, A_y, A_z) + r \cdot (B_x - A_x, B_y - A_y, B_z - A_z) + s \cdot (C_x - A_x, C_y - A_y, C_z - A_z) = (x, y, z).$$  \hspace{1cm} (13)

Consequently, the intersection point between the ray and the plane is obtained by the following equation:
\[(S_X, S_Y, S_Z) + t \cdot (D_X, D_Y, D_Z) =
= (A_X, A_Y, A_Z) + r \cdot (B_X - A_X, B_Y - A_Y, B_Z - A_Z) + s \cdot (C_X - A_X, C_Y - A_Y, C_Z - A_Z).\]  

(14)

The Equation 14 results in a linear system in \(t\), \(r\) and \(s\). The matrix representation of this system is denoted by the following equation:

\[
\begin{bmatrix}
D_X & A_X - B_X & A_X - C_X \\
D_Y & A_Y - B_Y & A_Y - C_Y \\
D_Z & A_Z - B_Z & A_Z - C_Z
\end{bmatrix} \cdot 
\begin{bmatrix}
t \\
r \\
s
\end{bmatrix} = 
\begin{bmatrix}
A_X - S_X \\
A_Y - S_Y \\
A_Z - S_Z
\end{bmatrix}.
\]  

(15)

In order to define the intersection between the ray and the plane, it is sufficient to determine the real number \(t\), which is achieved by using the Cramer’s rule:

\[
t = \frac{\begin{vmatrix}
A_X - S_X & A_X - B_X & A_X - C_X \\
A_Y - S_Y & A_Y - B_Y & A_Y - C_Y \\
A_Z - S_Z & A_Z - B_Z & A_Z - C_Z
\end{vmatrix}}{\begin{vmatrix}
D_X & A_X - B_X & A_X - C_X \\
D_Y & A_Y - B_Y & A_Y - C_Y \\
D_Z & A_Z - B_Z & A_Z - C_Z
\end{vmatrix}}.
\]  

(16)

Note that if the denominator of the fraction in the Equation 16 is zero and the numerator is nonzero, then the ray will never intersect the plane defined by the triangle, because they are parallel but not coplanar.

If both the numerator and the denominator of the Equation 16 are zero, then the values of \(r\) and \(s\) must be calculated. If all of these three variables (\(r\), \(s\) and \(t\)) are indeterminate (“zero over zero”), then the ray and the triangle ABC are coplanar. Else, the ray will never intersect the plane defined by the triangle, because they are parallel but not coplanar.
If the ray and the plane are coplanar, it is necessary to determine the intersection between the ray and the triangle’s edges in order to find the intersection closest to the starting point of the ray. In the case that the ray does not intersect any of the triangle’s edges, then the ray does not intersect the triangle at all.

It is noteworthy that a negative value of $t$ implies that the line containing the ray intersects the plane in a point located behind the ray’s starting point. Therefore, in this case, the ray does not intersect the triangle.

Finally, if the value of $t$ is greater than zero, it is necessary to discover if the intersection point is within the triangle. If the point is within the triangle, the problem is solved and the sampling point can be successfully stored in the computer memory. Else, if the intersection point is outside the triangle, then more triangles in the terrain’s polygonal mesh must be tested.

The task of determining if the intersection point is inside the triangle is quite simple because the point and the triangle are coplanar. The method implemented in the tool is described hereinafter.

In order to determine if the point $\vec{P}$ is within the triangle $ABC$, it is necessary to analyze the angles between the line segments $\vec{AP}$ and $\vec{BP}$ ($\theta_1$), $\vec{AP}$ and $\vec{CP}$ ($\theta_2$), and $\vec{CP}$ and $\vec{BP}$ ($\theta_3$).

If $\theta_1 + \theta_2 + \theta_3 < 2\pi$, the point $\vec{P}$ is outside of the triangle defined by the points $\vec{A}$, $\vec{B}$ and $\vec{C}$.

If $\theta_1 + \theta_2 + \theta_3 = 2\pi$, the point $\vec{P}$ is inside of the triangle. The Figure 3-11 illustrates how the position of $\vec{P}$ with respect to the triangle affects the sum of the angles $\theta_1$, $\theta_2$ and $\theta_3$. The angle $\theta_1$ is obtained by the equation:
\[ \theta_1 = \arccos \frac{(\vec{AP} \cdot \vec{BP})}{\|\vec{AP}\| \cdot \|\vec{BP}\|}. \] (17)

The angles \( \theta_2 \) and \( \theta_3 \) can be obtained analogously.

Figure 3-11: The angles between the line segments are used to determine triangle inclusion

Therefore, all line-to-triangle intersection cases are covered, except for one. It is possible that the intersection point is located at one of the edges of the triangle. For the sake of completeness, it is necessary to check if the intersection point is in the edges of the triangle. If the denominator of the Equation 17 is zero, then the point \( \vec{P} \) is one of the vertices of the triangle. If one of the three angles (\( \theta_1, \theta_2 \) or \( \theta_3 \)) is equal to \( \pi \), then the point \( \vec{P} \) is in one of the edges of the triangle.
**Seeding Method**

The user must interact with the tool’s graphical interface in order to specify the number of sampling points to be placed over the terrain’s surface. Currently, the user is restricted to select only square numbers, such as 4, 9, 16, etc. The sampling points are spread evenly over the terrain’s surface within each block.

Sampling the terrain is a computer intensive task. In the worst case, the time efficiency of the algorithm implemented in the tool is $O(m \times n)$, where $m$ is the number of polygons in the database and $n$ is the number of sampling points.

The only optimization implemented in the tool for this algorithm is the use of a View-Projection matrix to cull out some triangles. This matrix is used for determining if a polygon is “in the way” of the LOS ray. In short, the View-Projection matrix works as an exclusion window. If the triangle is out of the window, line-to-triangle intersection algorithm is not executed for that triangle. The vertices and centroid of the triangle are used for checking if it is in the View-Projection window (Foley, van Dam, Feiner, & Hughes, 1997). For this reason, the exclusion window cannot be excessively small, or else all the polygons will be excluded.

The Figure 3-12 illustrates a TDB with 100 blocks and 100 sampling points per block. The sampling points are represented in magenta. Note that the sampling points are never placed on the borders of the TDB’s blocks to avoid overlap in the LOS tests performed in the Fifth Step.
Fourth Step: Set up the LOS rays

In order to cast a ray, two pieces of information are required. The first one is the starting point of the ray. The starting point of a LOS ray is the eye-point. The second information necessary to cast a ray is the direction of the ray. The direction of the LOS ray is determined by its directional vector. The Fourth Step consists of setting the eye-point positions and the directional vectors of the LOS rays. The tool creates the eye-points and the directional vectors of the LOS rays according to user’s inputs.
For each sampling point generated on the Third Step, the tool creates one or more eye-points (depending on the user input). As stated before, the eye-point and its respective sampling point share the same latitude and longitude coordinates. The user must configure three parameters for creating the eye-points:

- The height of the eye-point that is positioned closest to the sampling point (henceforth referred to as first eye-point).
- The number of additional eye-points. Besides the first eye-point, the tool can generate additional eye-points.
- The distance between the eye-points.

The Figure 3-13 highlights in red the tool’s input fields for generating the eye-points. For the input parameters depicted in the Figure 3-13, the tool will create three eye-points per sampling point. The first eye-point will be positioned 15 meters (AGL), the second one will be positioned 35 meters AGL, and the last one will be placed 55 meters AGL.
For each eye-point, the tool creates a list of directional vectors. This list of vectors determines the angular range covered by the LOS tests and the amount of tests performed per eye-point. The user must configure three parameters for creating the list of directional vectors:

- The number of azimuth variations per eye point. The first ray is always pointing to the North. If the number of azimuth variations equals two, then the second ray will be pointing to the South. If the number of azimuth variations equals four, then the second ray will be pointing to the East, the third one will be pointing the south, and the last one will be point to the West. In short, the azimuth variations are evenly distributed around the eye-point.

- The number of pitch variations for each azimuth direction. This number defines the number of angular variations with respect to the horizon line. If the number of pitch
variations equals zero, then the ray is casted only in the horizon line. If the number of pitch variations equals one, then in addition to the ray casted in the horizon line, two more rays are casted. The first one ray is casted above the horizon line and the last one is casted below the horizon line.

- The angle between the pitch variations. This number defines angle (in degrees) between two neighbor pitch variations.

The Figure 3-13 highlights in blue the tool’s input fields for generating the list of directional vectors for each eye-point. For the input parameters depicted in the Figure 3-13, the tool will create a list with 40 directional vectors for each eye-point. In this case, eight azimuth directions will be tested per eye-point, namely, North (zero), Northeast (45°), East (90°), Southeast (135°), South (180°), Southwest (225°), West (270°), and Northwest (315°). In addition to the directional vectors created in the horizon line, for each azimuth direction, the tool will create two directional vectors above the horizon line (+15° and +30°) and two directional vectors below the horizon line (-15° and -30°).

The Figure 3-14 illustrates the eye-points and the list of directional vectors created by with the inputs depicted in the Figure 3-13. Note that the sampling point corresponding to the newly created eye-points is represented in the Figure 3-14 by a small magenta dot over the grass texture.
Fifth Step: Generate Data for Correlation Analysis

The Fifth Step consists of determining the ending points of the LOS rays and recording preliminary data for statistical analysis.

At this point, the entire environment is ready for the LOS tests. The method used for testing the intersection between the LOS ray and the terrain mesh is similar to the method described in the Third Step. Although, there are some important differences that are described hereinafter.
The rays cast for sampling the terrain’s elevation always intersect at least one triangle in the terrain’s polygonal mesh (assuming that the TDBs are not of different sizes), whereas the LOS rays may never intersect the terrain geometry. If the LOS ray misses terrain surface, it will intersect the bounding box created in the First Step. Due to the bounding box, the LOS ray always has a measurable length. This is important for generating numerical data for statistical analysis.

Another important difference between the rays cast for sampling the terrain and the LOS rays is the condition for exiting the line-to-triangle intersection check looping. If the ray used to sample the terrain intersects a triangle, then the intersection checking routine stops, because the polygons in the terrain mesh do not overlap. However, it is possible that the LOS ray intersects more than one triangle. It is worth mentioning that only the triangle closest to the eye-point is significant for the intersection test, because this triangle occludes all the other triangles behind it. Since the conditions for exiting the intersection testing routine are more restrictive to the LOS rays, its checking algorithm is more computer intensive. In order to reduce the number of intersection tests, the polygons that are back facing the eye-point are culled out of the testing routine. If the dot product between the LOS ray’s directional vector and the normal vector to the triangle surface is greater or equal to zero, the triangle is excluded from the intersection test.

The tool developed for this thesis does not include a complete statistical analysis for the data it produces. However, it generates data that can be used as input for statistical tools, because the data produced is a simple uncompressed text file. After the LOS tests are concluded, the tool produces numerical and categorical data for correlation analysis.

The numerical data is composed of a data array that stores the differences between the lengths of LOS rays in the baseline database and the lengths of their counterparts in the subject
database. The tool computes the average, the standard error, the maximum and the minimum values of the sample.

The categorical data is composed of a binary number array. These binary numbers represent the “agreement” between the LOS rays in the baseline database and their counterparts in the subject database with respect to bounding box intersection. If the LOS ray intersects the bounding box in one of the databases but its counterpart does not, the “agreement” is zero. Otherwise, it is one. The tool computes the agreement proportion as a metric for correlation. The Figure 3-15 shows a pair of corresponding LOS rays that does not agree with respect to bounding box intersection criterion. The ray depicted in the left picture hits the bounding box, whereas its counterpart intersects the terrain’s surface. In this illustration, the eye-points are shown in blue and the ending points are shown in red. The purple line represents the LOS ray.

The Figure 3-16 depicts the LOS rays in the tool’s visualization widget. The intersection points are represented by red dots. The LOS rays are represented by green lines. Note that the rays that do not intersect the terrain surface are intersecting the bounding box.

![Figure 3-15: Disagreement with respect to bounding box intersection](image)
Figure 3-16: LOS rays created in the Fifth Step
CHAPTER FOUR: DATA COLLECTION AND ANALYSIS

This chapter illustrates a use case of the tool. It is divided into four sections. In order to demonstrate the usability of the tool and the applicability of the proposed methodology, two TDBs were created. Additionally, a third TDB was created with the intention of validating the roughness calculation method implemented in the tool. The first section briefly discusses the creation of the TDBs used in this research. The second section describes preliminary characteristics of these TDBs. The third section shows the results of the LOS tests and describes an example of statistical analysis of the data retrieved from these tests. The last section in this chapter shows the visual outputs provided by the tool in a 3D environment.

Generation of the TDBs

Two OpenFlight TDBs were created for demonstrating the methodology discussed in Chapter 3 of this thesis (the reasons for choosing OpenFlight format were discussed in the previous chapter). It is noteworthy that for demonstrating the methodology proposed herein, the TDBs should represent the same area of the world, but they should not be identical, since identical TDBs would have the same geometry.

The elevation data files used for generating the TDBs were retrieved from United States Geological Survey (USGS) National Elevation Dataset (NED) website. The NED is a USGS raster product developed for providing the U.S. elevation in a seamless manner (USGS NED
According to the USGS NED Official Site (2015), the NED elevation data is available in several layers with different horizontal resolutions, namely 1/3 arc-second (approximately 10 meters), 1 arc-second (approximately 30 meters) and 2 arc-second (approximately 60 meters).

The software used for creating the TDBs from the USGS decimated values was Presagis Terra Vista (Terra Vista, version 14), henceforth referred to as TDB Generator. The TDB Generator’s configurations for the generation of the TDBs were identical for both databases. Layers with different resolutions of the NED were selected as input elevation files for the TDB software with the intention of generating non-identical TDBs. The resolution of the input file for generating the baseline TDB was 1/3 arc-second, whereas the resolution of the input file for the subject TDB was 1 arc-second. As expected, even using the same settings for TDB generation, the resulting databases were not identical. Figure 4-1 shows a picture of the baseline TDB captured from the TDB Generator software. Intentionally, the area selected for demonstrating the functionality of the tool is undulated and it is mostly composed of rough sub-areas, but it also contains some flat sub-areas.
Additionally, another TDB representing a visually flat area of the world was generated in order to investigate the coherence between the roughness calculation methodology and the visual analysis of the roughness. The Figure 4-2 shows a picture of the flat terrain captured from the TDB Generator software.
Characteristics of the TDBs

As described in the previous chapter, the tool’s preliminary analysis (First Step) is composed of the following tasks:

- Revealing the number of LODs present in each database. This procedure is important because the user must select one LOD for analysis in each TDB. The higher the polygon count in the LOD, the slower will be the analysis.
- Organizing the terrain geometry in LOD arrays.
- Computing the centroid of each polygon.
- Counting the polygons in each LOD array.
- Revealing the extents of each TDB in the world and local coordinate systems.
- Calculating the roughness of each TDB.

In order to check the reasonability of the roughness calculation method implemented in this thesis, the roughness values of the TDBs depicted in the Figures 11 and 12 were assessed with the tool. The reader can recall that the terrain shown in the Figure 4-1 is undulated, whereas the terrain shown in the Figure 4-2 is flat. The Table 4-1 shows the results of the preliminary analysis on the TDBs. The roughness of the TDB depicted in the Figure 3-16 (TDB no. 1) is more than 15 times greater than the roughness of the TDB depicted in the Figure 4-1. This result matches the reader’s expectations (the value of the roughness of the undulated terrain is substantially greater than the value of the roughness of the flat terrain).

Table 4-1  
Comparing an undulated TDB with a flat TDB

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Undulated TDB (TDB no. 1)</th>
<th>Flat TDB (TDB no. 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Polygons</td>
<td>131,072</td>
<td>131,052</td>
</tr>
<tr>
<td>North Extent</td>
<td>36.745433°</td>
<td>37.077183°</td>
</tr>
<tr>
<td>South Extent</td>
<td>36.450545°</td>
<td>36.782295°</td>
</tr>
<tr>
<td>West Extent</td>
<td>-118.556645°</td>
<td>-120.728330°</td>
</tr>
<tr>
<td>East Extent</td>
<td>-118.189803°</td>
<td>-120.359089°</td>
</tr>
<tr>
<td>Minimum Elevation</td>
<td>1,286.71 m</td>
<td>29.62 m</td>
</tr>
<tr>
<td>Maximum Elevation</td>
<td>4,411.91 m</td>
<td>142.54 m</td>
</tr>
<tr>
<td>Roughness</td>
<td>0.57164</td>
<td>0.03710</td>
</tr>
</tbody>
</table>

After this quick verification of the roughness calculation method was concluded, the baseline and the subject TDBs were subject to further analysis using the tool. The data retrieved from the preliminary analysis of these TDBs is presented in the Table 4-2.
Table 4-2

Preliminary analysis of the TDBs

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Baseline TDB (TDB no. 1)</th>
<th>Subject TDB (TDB no. 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of LODs</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**LOD 0 (maximum detail)**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Baseline TDB (TDB no. 1)</th>
<th>Subject TDB (TDB no. 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Polygons</td>
<td>131,072</td>
<td>131,072</td>
</tr>
<tr>
<td>North Extent</td>
<td>36.745433°</td>
<td>36.745433°</td>
</tr>
<tr>
<td>South Extent</td>
<td>36.450545°</td>
<td>36.450545°</td>
</tr>
<tr>
<td>West Extent</td>
<td>-118.556645°</td>
<td>-118.556645°</td>
</tr>
<tr>
<td>East Extent</td>
<td>-118.189803°</td>
<td>-118.189803°</td>
</tr>
<tr>
<td>Minimum Elevation</td>
<td>1,286.71 m</td>
<td>1,287.22 m</td>
</tr>
<tr>
<td>Maximum Elevation</td>
<td>4,411.91 m</td>
<td>4,411.90 m</td>
</tr>
<tr>
<td>Roughness</td>
<td>0.57164</td>
<td>0.55145</td>
</tr>
</tbody>
</table>

**LOD 1 (minimum detail)**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Baseline TDB (TDB no. 1)</th>
<th>Subject TDB (TDB no. 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Polygons</td>
<td>16,640</td>
<td>16,654</td>
</tr>
<tr>
<td>North Extent</td>
<td>36.745433°</td>
<td>36.745433°</td>
</tr>
<tr>
<td>South Extent</td>
<td>36.450545°</td>
<td>36.450545°</td>
</tr>
<tr>
<td>West Extent</td>
<td>-118.556645°</td>
<td>-118.556645°</td>
</tr>
<tr>
<td>East Extent</td>
<td>-118.189803°</td>
<td>-118.189803°</td>
</tr>
<tr>
<td>Minimum Elevation</td>
<td>1,286.71 m</td>
<td>1,287.22 m</td>
</tr>
<tr>
<td>Maximum Elevation</td>
<td>4,410.32 m</td>
<td>4,409.68 m</td>
</tr>
<tr>
<td>Roughness</td>
<td>0.54951</td>
<td>0.53680</td>
</tr>
</tbody>
</table>

The data retrieved from the preliminary analysis tells the analyst the number of polygons in each LOD present in each TDB, the boundaries of the TDBs within each LOD, and the roughness of the TDBs within each LOD. The quantity of polygons is an important measure, because the time and the computer memory consumed for finishing the correlation analysis is proportional to the quantity of polygons. The boundaries of the TDBs being compared are
expected to be nearly the same. If there is a large difference between the boundaries of the TDBs, then this pair of TDBs do not cover the same gaming area. Therefore, the LOS correlation analysis between these TDBs is meaningless in this case. The Table 4-2 shows that the TDBs tested in this thesis represent the exact same gaming area. Roughness is another important parameter of the TDBs, because it provides insight for the correlation between the TDBs. If one of the TDBs is flat and the other one is rough, it is an indication of poor correlation. The Table 4-2 shows that the roughness of the TDBs tested is not too large.

LOS Test Results and Data Analysis

In order to test the tool, the most detailed polygonal meshes from each TDB were selected. The TDBs were divided in 100 blocks (10 lines and 10 columns), and 36 sampling points were evenly distributed in each block. Three eye-points were created for each sampling point. The lowest eye-points were positioned 15 meters AGL. The eye-points right above the lowest eye-points were positioned 35 meters AGL. The highest eye-points were positioned 55 meters AGL. Eight azimuth directions were tested per eye-point and five pitch directions were tested per azimuth variation. The angle between the pitch directions was 15º. For each azimuth direction, two rays were cast above the horizon line (30º and 15º), one ray was cast in the horizon line (0º), and two rays were cast below the horizon line (-15º and -30º). In total, 432,000 LOS tests were executed in each TDB, which means that 4,320 tests were executed per block.

The Figure 4-3 shows the value of roughness of each block. The name of the blocks are highlighted in blue (the block name is denoted by Block I_J, where I is the row index and J is the column index). The Block 0_0 is located at the extreme southwest corner of the TDB, whereas
the Block 9_9 is located at the extreme northeast corner of the TDB. The cells in the row right below the block name contain the value of the block’s roughness in the baseline TDB. The cells located two lines below the block name contains the values of the block’s roughness in subject TDB. The cells highlighted in green indicate that the roughness of the corresponding terrain block is less than 0.2 in at least one of the TDBs. The cells highlighted in red indicate that the roughness of the corresponding terrain block is greater than 0.65 in at least one of the TDBs. Considering both TDBs, the Block 4_7 is the most rough block, whereas the Block 8_9 is the most flat one. The value of the roughness of these blocks is written in yellow numbers in Figure 4-3.
The data generated in the LOS tests was used for statistical analysis. The first variable of interest in this analysis was the average difference between the lengths of corresponding LOS rays in both TDBs. This difference and all the other variables of interest discussed later in this

<table>
<thead>
<tr>
<th>North</th>
<th>West</th>
<th>East</th>
<th>South</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 9.0</td>
<td>Block 9.1</td>
<td>Block 9.2</td>
<td>Block 9.3</td>
</tr>
<tr>
<td>0.542269</td>
<td>0.581074</td>
<td>0.590970</td>
<td>0.621880</td>
</tr>
<tr>
<td>0.501611</td>
<td>0.568545</td>
<td>0.577293</td>
<td>0.600901</td>
</tr>
<tr>
<td>Block 8.0</td>
<td>Block 8.1</td>
<td>Block 8.2</td>
<td>Block 8.3</td>
</tr>
<tr>
<td>0.408717</td>
<td>0.621446</td>
<td>0.386553</td>
<td>0.652800</td>
</tr>
<tr>
<td>0.384983</td>
<td>0.597849</td>
<td>0.569405</td>
<td>0.634300</td>
</tr>
<tr>
<td>Block 7.0</td>
<td>Block 7.1</td>
<td>Block 7.2</td>
<td>Block 7.3</td>
</tr>
<tr>
<td>0.616317</td>
<td>0.560075</td>
<td>0.655100</td>
<td>0.469061</td>
</tr>
<tr>
<td>0.384412</td>
<td>0.539964</td>
<td>0.641100</td>
<td>0.453306</td>
</tr>
<tr>
<td>Block 6.0</td>
<td>Block 6.1</td>
<td>Block 6.2</td>
<td>Block 6.3</td>
</tr>
<tr>
<td>0.651180</td>
<td>0.569470</td>
<td>0.579140</td>
<td>0.472161</td>
</tr>
<tr>
<td>0.625515</td>
<td>0.541147</td>
<td>0.560869</td>
<td>0.453799</td>
</tr>
<tr>
<td>Block 5.0</td>
<td>Block 5.1</td>
<td>Block 5.2</td>
<td>Block 5.3</td>
</tr>
<tr>
<td>0.612447</td>
<td>0.585912</td>
<td>0.598543</td>
<td>0.438365</td>
</tr>
<tr>
<td>0.570556</td>
<td>0.579857</td>
<td>0.574097</td>
<td>0.530255</td>
</tr>
<tr>
<td>Block 4.0</td>
<td>Block 4.1</td>
<td>Block 4.2</td>
<td>Block 4.3</td>
</tr>
<tr>
<td>0.625960</td>
<td>0.488865</td>
<td>0.520696</td>
<td>0.616323</td>
</tr>
<tr>
<td>0.614089</td>
<td>0.466505</td>
<td>0.511830</td>
<td>0.605590</td>
</tr>
<tr>
<td>Block 3.0</td>
<td>Block 3.1</td>
<td>Block 3.2</td>
<td>Block 3.3</td>
</tr>
<tr>
<td>0.606924</td>
<td>0.645745</td>
<td>0.490965</td>
<td>0.499928</td>
</tr>
<tr>
<td>0.575262</td>
<td>0.618411</td>
<td>0.483939</td>
<td>0.476716</td>
</tr>
<tr>
<td>Block 2.0</td>
<td>Block 2.1</td>
<td>Block 2.2</td>
<td>Block 2.3</td>
</tr>
<tr>
<td>0.522292</td>
<td>0.561390</td>
<td>0.554182</td>
<td>0.531477</td>
</tr>
<tr>
<td>0.492984</td>
<td>0.543087</td>
<td>0.541989</td>
<td>0.521824</td>
</tr>
<tr>
<td>Block 1.0</td>
<td>Block 1.1</td>
<td>Block 1.2</td>
<td>Block 1.3</td>
</tr>
<tr>
<td>0.433715</td>
<td>0.432541</td>
<td>0.321022</td>
<td>0.396038</td>
</tr>
<tr>
<td>0.417929</td>
<td>0.425598</td>
<td>0.310338</td>
<td>0.378765</td>
</tr>
<tr>
<td>Block 0.0</td>
<td>Block 0.1</td>
<td>Block 0.2</td>
<td>Block 0.3</td>
</tr>
<tr>
<td>0.659999</td>
<td>0.450318</td>
<td>0.517783</td>
<td>0.395218</td>
</tr>
<tr>
<td>0.639040</td>
<td>0.453891</td>
<td>0.499573</td>
<td>0.386021</td>
</tr>
</tbody>
</table>

Figure 4-3: Terrain roughness per block
chapter were computed for the entire TDB and for each terrain block as well. The second variable of interest was the average absolute value of the difference between the lengths of corresponding the LOS rays. This variable is closely related to the average difference between the lengths of corresponding LOS rays, because they derive from the same data. However, since the absolute difference is always positive, the histogram for the absolute difference between the lengths of corresponding LOS rays variable is right-skewed. This fact becomes more evident in the statistical analysis described later in this chapter. A low average indicates a good LOS correlation between the TDBs. The third variable of interest in this analysis was the proportion of bounding box visibility agreement in both TDBs. If there is an unobstructed LOS between one eye-point and the bounding box through a certain direction, it is said that the bounding box is visible from that eye-point through that direction. As stated in Chapter One, a LOS ray is a straight line composed of one eye-point (starting point), one directional vector and one ending point, which is the intersection between the ray and the surface of the terrain (or the bounding volume). It is said that a pair of LOS rays do not agree with respect to the bounding box visibility if, and only if, one of the LOS rays intersects the bounding box in one of the TDBs and its counterpart intersects the terrain in the other TDB. Otherwise, it is said that the pair of rays agree with respect to the bounding box visibility. All the “agreeing” LOS ray pairs were counted in the computation of the proportion of bounding box visibility agreement variable. A small agreement proportion indicates a poor LOS correlation between the TDBs.

The Figure 4-4 shows the average difference between the lengths of corresponding LOS rays in each of the analyzed blocks. The blocks highlighted in green have average difference between -1 meter and 1 meter. The blocks highlighted in red have average difference less than -10 meters or greater than 10 meters.
The Figure 4-5 shows the average absolute difference between the lengths of corresponding LOS rays in each of the analyzed blocks. The blocks highlighted in green have
average absolute difference lesser than 15 meters. The blocks highlighted in red have average absolute difference greater than 50. The analyst may choose any threshold to highlight low or high levels of LOS agreement between TDB’s.

Figure 4-5: Average length absolute difference of corresponding the LOS rays per block
It is noteworthy that all the terrain blocks that are highlighted in green in the Figure 4-3 (low roughness level) are also highlighted in green in the Figure 4-5 (lower LOS correlation issues).

The procedure for analyzing the collected data is the same for each terrain block. Currently, the tool does not perform the statistical analysis automatically. This task was accomplishable using alternative statistic software capable of accepting the text file produced by the tool. In this thesis, the R Project for Statistical Computing (Dragulescu, 2014; Meyer, Dimitriadou, Hornik, Weingessel, and Leisch, 2015; Gross & Ligges, 2015; Lemon, 2006; R Core Team, 2015; Verzani, 2015) was used.

For illustrative purposes, this thesis presents the analysis for only two blocks. The blocks selected for analysis were the Block 4_7 and the Block 8_9, the former having the highest roughness value, whereas the latter having the lowest value. It is worth mentioning that the process described to analyze the blocks is identical to the process used to analyze the entire TDB. The output text file generated with the tool was transformed into Microsoft Excel Workbook files (“Block 4_7.xlsx” and “Block 8_9.xlsx”). In sequence, these files were imported into RStudio for statistical analysis. The results of this analysis is detailed in the next two subsections.

**Block 4_7 Analysis**

The Table 4-3 contains the summary of the results of the LOS tests in the *Block 4_7*. 
Table 4-3

Statistics for the Block 4_7

<table>
<thead>
<tr>
<th>Statistic Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>4,320</td>
</tr>
<tr>
<td>Minimum (LOS Ray Length Difference)</td>
<td>-26,447.67</td>
</tr>
<tr>
<td>Maximum (LOS Ray Length Difference)</td>
<td>23,822.75</td>
</tr>
<tr>
<td>Average (LOS Ray Length Difference)</td>
<td>-19.53</td>
</tr>
<tr>
<td>Trimmed Average (5%) (LOS Ray Length Difference)</td>
<td>0.06</td>
</tr>
<tr>
<td>Median (LOS Ray Length Difference)</td>
<td>0.00</td>
</tr>
<tr>
<td>Standard Deviation (LOS Ray Length Difference)</td>
<td>1,066.87</td>
</tr>
<tr>
<td>Standard Error (LOS Ray Length Difference)</td>
<td>16.23</td>
</tr>
<tr>
<td>Skewness (LOS Ray Length Difference)</td>
<td>-2.52</td>
</tr>
<tr>
<td>Kurtosis (LOS Ray Length Difference)</td>
<td>261.96</td>
</tr>
<tr>
<td>Minimum (LOS Ray Abs. Length Difference)</td>
<td>0.00</td>
</tr>
<tr>
<td>Maximum (LOS Ray Abs. Length Difference)</td>
<td>26,447.67</td>
</tr>
<tr>
<td>Average (LOS Ray Abs. Length Difference)</td>
<td>127.60</td>
</tr>
<tr>
<td>Trimmed Average (5%) (LOS Ray Abs. Length Diff.)</td>
<td>13.63</td>
</tr>
<tr>
<td>Median (LOS Ray Abs. Length Difference)</td>
<td>8.60</td>
</tr>
<tr>
<td>Standard Deviation (LOS Ray Abs. Length Diff.)</td>
<td>1059.39</td>
</tr>
<tr>
<td>Standard Error (LOS Ray Abs. Length Difference)</td>
<td>16.12</td>
</tr>
<tr>
<td>Skewness (LOS Ray Abs. Length Difference)</td>
<td>14.64</td>
</tr>
<tr>
<td>Kurtosis (LOS Ray Abs. Length Difference)</td>
<td>262.57</td>
</tr>
<tr>
<td>Bounding Box Hit Agreement (total)</td>
<td>4,249</td>
</tr>
<tr>
<td>Bounding Box Hit Agreement (percentage)</td>
<td>98.36%</td>
</tr>
</tbody>
</table>

The Figure 4-6 shows the histogram for the LOS Ray Length Difference variable in the Block 4_7. The Figure 4-7 shows the Q-Q Normal plot for the same variable.
Figure 4-6: LOS Ray Length Difference histogram for the Block 4_7

Figure 4-7: LOS Ray Length Difference Q-Q normal plot for the Block 4_7
Assuming that the LOS Ray Length Difference is nearly normal distributed, the confidence intervals are 95% CI [-51.34, 12.29] and 90% CI [-46.23, 7.17]. Both confidence intervals include the zero value, regarding the respective confidence intervals. However, the Q-Q normal plot depicted in the Figure 4-7 shows that the distribution of this variable has thick tails. For this reason, three normality assessment tests were performed in R, namely Kolmogorov-Smirnov, Anderson-Darling and Shapiro-Wilk. Table 4-4 summarizes the results of these tests. The null hypothesis for these three tests is that the distribution is normal. The p-values in the Table 4-4 shows that the null hypothesis could be rejected in all of these three tests, regarding a statistical significance of 5%. Hence, there was strong evidence that the distribution of this variable was not normal. Therefore, this case was more suitable for non-parametric statistical tests. The Wilcoxon signed rank test with continuity correction was performed in order to test some hypotheses about the variable. The results of this non-parametric tests are summarized in the Table 4-5.

Table 4-4

<table>
<thead>
<tr>
<th>Normality Assessment Test Name</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov-Smirnov</td>
<td>0.4524</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>1474.4</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Shapiro-Wilk</td>
<td>0.1083</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
</tbody>
</table>
Table 4-5

Wilcoxon tests for the LOS Ray Length Difference in the Block 4_7

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean is not equal to 0</td>
<td>4,450,000</td>
<td>0.4123</td>
</tr>
<tr>
<td>Mean is greater than -1</td>
<td>5,098,600</td>
<td>6.87 x 10^-8</td>
</tr>
<tr>
<td>Mean is lesser than 1</td>
<td>4,363,600</td>
<td>0.0001093</td>
</tr>
<tr>
<td>Mean is lesser than -10</td>
<td>7,286,500</td>
<td>1</td>
</tr>
<tr>
<td>Mean is greater than 10</td>
<td>2,090,200</td>
<td>1</td>
</tr>
</tbody>
</table>

The Table 4-5 shows that there was evidence that the average of the variable of interest was in between -1 and 1. The statistical significance of the tests was 5%. This results confirmed that the difference was within a small interval close to zero. Therefore, this pair of terrain blocks would not favor simulation entities in any of these TDBs with respect to LOS (regarding this [-1, 1] acceptance interval). Note that this test could be repeated using an even more restrictive acceptance interval or a different statistical significance.

The Figure 4-8 shows the histogram for the LOS Ray Length Absolute Difference in the Block 4_7. The Figure 4-9 shows the Q-Q Normal plot for the same variable.
Figure 4-8: LOS Ray Length Absolute Difference histogram for the Block 4_7

Figure 4-9: LOS Ray Length Absolute Difference Q-Q normal plot for the Block 4_7
The histogram depicted in the Figure 4-8 and Q-Q normal plot depicted in the Figure 4-9 shows that the distribution of this variable is skewed. In order to ensure that the variable of interest is not normal, the three normality assessment tests were performed in R. The Table 4-6 summarizes the results of this tests. The null hypothesis for the three tests is that the distribution is normal. The p-values in the Table 4-6 support the rejection of the null hypothesis in all three tests, regarding a statistical significance of 5%. Therefore, there was strong evidence that the distribution of this variable was not normal. For this reason, this case was more suitable for non-parametric statistical tests. The Wilcoxon signed rank test with continuity correction was performed in order to test some hypothesis about the variable. The results of this non-parametric tests are summarized in the Table 4-7.

Table 4-6

Normality tests for the LOS Ray Length Absolute Difference in the Block 4_7

<table>
<thead>
<tr>
<th>Normality Assessment Test Name</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov-Smirnov</td>
<td>0.47538</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>1514.8</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Shapiro-Wilk</td>
<td>0.09080</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
</tbody>
</table>

Table 4-7

Wilcoxon tests for the LOS Ray Length Absolute Difference in the Block 4_7

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean is lesser than 15</td>
<td>3,669,000</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Mean is greater than 50</td>
<td>765,730</td>
<td>1</td>
</tr>
</tbody>
</table>
The Table 4-7 shows that there was evidence that the average of the variable of interest was in between 0 and 15, regarding a statistical significance of 5%.

A quick analysis was performed on the variable Bounding Box Intersection Agreement Proportion. The results of the hypothesis test are shown in the Table 4-8. The results indicate that the proportion is greater than 98%, given a statistical significance of 5%. This findings confirmed that more than 98% of the LOS rays agreed with respect to bounding box intersection (this agreement was defined in the last section of the Chapter Four and the Figure 3-5 helps to clarify it). Note that the bounding box intersection agreement test could be repeated with a proportion greater than 98% or a different statistical significance.

Table 4-8

Test for the Bounding Box Intersection Agreement Proportion in the Block 4_7

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportion is greater than 98%</td>
<td>0.04710</td>
</tr>
</tbody>
</table>

*Block 8_9 Analysis*

The Table 4-9 synthetizes the results of the LOS tests in the *Block 8_9.*
Table 4-9

Statistics for the Block 8_9

<table>
<thead>
<tr>
<th>Statistic Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>4,320</td>
</tr>
<tr>
<td>Minimum (LOS Ray Length Difference)</td>
<td>-6,910.27</td>
</tr>
<tr>
<td>Maximum (LOS Ray Length Difference)</td>
<td>7,844.88</td>
</tr>
<tr>
<td>Average (LOS Ray Length Difference)</td>
<td>0.03</td>
</tr>
<tr>
<td>Trimmed Average (5%) (LOS Ray Length Difference)</td>
<td>0.20</td>
</tr>
<tr>
<td>Median (LOS Ray Length Difference)</td>
<td>0.00</td>
</tr>
<tr>
<td>Standard Deviation (LOS Ray Length Difference)</td>
<td>159.55</td>
</tr>
<tr>
<td>Standard Error (LOS Ray Length Difference)</td>
<td>2.43</td>
</tr>
<tr>
<td>Skewness (LOS Ray Length Difference)</td>
<td>8.71</td>
</tr>
<tr>
<td>Kurtosis (LOS Ray Length Difference)</td>
<td>2,168.22</td>
</tr>
<tr>
<td>Minimum (LOS Ray Abs. Length Difference)</td>
<td>0.000</td>
</tr>
<tr>
<td>Maximum (LOS Ray Abs. Length Difference)</td>
<td>7,844.88</td>
</tr>
<tr>
<td>Average (LOS Ray Abs. Length Difference)</td>
<td>6.49</td>
</tr>
<tr>
<td>Trimmed Average (5%) (LOS Ray Abs. Length Diff.)</td>
<td>1.52</td>
</tr>
<tr>
<td>Median (LOS Ray Abs. Length Difference)</td>
<td>0.58</td>
</tr>
<tr>
<td>Standard Deviation (LOS Ray Abs. Length Diff.)</td>
<td>159.41</td>
</tr>
<tr>
<td>Standard Error (LOS Ray Abs. Length Difference)</td>
<td>2.43</td>
</tr>
<tr>
<td>Skewness (LOS Ray Abs. Length Difference)</td>
<td>46.36</td>
</tr>
<tr>
<td>Kurtosis (LOS Ray Abs. Length Difference)</td>
<td>2,167.87</td>
</tr>
<tr>
<td>Bounding Box Hit Agreement (total)</td>
<td>4,318</td>
</tr>
<tr>
<td>Bounding Box Hit Agreement (percentage)</td>
<td>99.95%</td>
</tr>
</tbody>
</table>

The Figure 4-10 shows the histogram for the LOS Ray Length Difference variable in the Block 8_9. The Figure 4-11 shows the Q-Q Normal plot for the same variable.
Figure 4-10: LOS Ray Length Difference histogram for the Block 8_9

Figure 4-11: LOS Ray Length Difference Q-Q normal plot for the Block 8_9
Three normality assessment tests were performed in R on the variable LOS Ray Length Difference for the Block 8_9. The Table 4-10 summarizes the results of this tests. As stated before, the null hypothesis for the three tests is that the distribution is normal. The p-values in the Table 4-10 supports the rejection of the null hypothesis in the three tests (given a statistical significance of 5%). Hence, strong evidence that the distribution of this variable was not normal. This case was more suitable for non-parametric statistical tests. The Wilcoxon Signed Rank Test with Continuity Correction was performed in order to validate some hypothesis about the variable. The results of this non-parametric tests are summarized in the Table 4-11.

Table 4-10
Normality tests for the LOS Ray Length Difference in the Block 8_9

<table>
<thead>
<tr>
<th>Normality Assessment Test Name</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov-Smirnov</td>
<td>0.4512</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>1,559.5</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Shapiro-Wilk</td>
<td>0.0149</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
</tbody>
</table>

Table 4-11
Wilcoxon tests for the LOS Ray Length Difference in the Block 8_9

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean is not equal to 0</td>
<td>2,894,600</td>
<td>3.37 x 10^{-12}</td>
</tr>
<tr>
<td>Mean is greater than -1</td>
<td>7,418,500</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Mean is lesser than 1</td>
<td>2,539,100</td>
<td>&lt; 2.2 x 10^{-16}</td>
</tr>
<tr>
<td>Mean is lesser than -10</td>
<td>8,996,100</td>
<td>1</td>
</tr>
<tr>
<td>Mean is greater than 10</td>
<td>220,680</td>
<td>1</td>
</tr>
</tbody>
</table>
The Table 4-11 shows that there was evidence that the average of the variable of interest was in between -1 and 1, but it was definitely non-zero (given a statistical significance of 5%). As in the analysis of the Block 4_7, the results confirmed that the difference was within a small interval close to zero. Therefore, this pair of terrain blocks would not favor simulation entities in any of these TDBs with respect to LOS (regarding this [-1, 1] acceptance interval). Note that this test could be repeated using an even more restrictive acceptance interval or a difference statistical significance.

The Figure 4-12 shows the histogram for the LOS Ray Length Absolute Difference in the Block 8_9. The Figure 4-13 shows the Q-Q Normal plot for the same variable.

Figure 4-12: LOS Ray Length Absolute Difference histogram for the Block 8_9
Three normality assessment tests were performed in R on the variable LOS Ray Length Absolute Difference for the Block 8_9. The Table 4-12 summarizes the results of this tests. According to the p-values on the Table 4-12, the distribution is not normal regarding a statistical significance of 5%. Non-parametric tests were required again. The Wilcoxon Signed Rank Test with Continuity Correction was performed in order to test some hypothesis about the variable. The results of this non-parametric tests are summarized in the Table 4-13.
Table 4-12

Normality tests for the LOS Ray Length Absolute Difference in the Block 8_9

<table>
<thead>
<tr>
<th>Normality Assessment Test Name</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov-Smirnov</td>
<td>0.4838</td>
<td>&lt; 2.2 x 10^-16</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>1,572.4</td>
<td>&lt; 2.2 x 10^-16</td>
</tr>
<tr>
<td>Shapiro-Wilk</td>
<td>0.0125</td>
<td>&lt; 2.2 x 10^-16</td>
</tr>
</tbody>
</table>

Table 4-13

Wilcoxon tests for the LOS Ray Length Absolute Difference in the Block 8_9

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>Statistic</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean is lesser than 15</td>
<td>360,040</td>
<td>&lt; 2.2 x 10^-16</td>
</tr>
<tr>
<td>Mean is greater than 50</td>
<td>54,086</td>
<td>1</td>
</tr>
</tbody>
</table>

The Table 4-13 shows that there was evidence that the average of the variable of interest was greater than zero and lesser than 15.

A quick analysis was performed on the variable Bounding Box Intersection Agreement Proportion in the Block 8_9. The results of the hypothesis test are shown in the Table 4-14. The results indicate that the proportion is greater than 98% regarding a statistical significance of 5%. As in the analysis of the Block 4_7, this findings confirmed that more than 98% of the LOS rays agreed with respect to bounding box intersection (this agreement was defined in the last section of the Chapter Four and the Figure 3-5 helps to clarify it). Note that the bounding box intersection agreement test could be repeated with a proportion greater than 98% or a different statistical significance.
Table 4-14

Test for the Bounding Box Intersection Agreement Proportion in the Block 8_9

<table>
<thead>
<tr>
<th>Alternative Hypothesis</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportion is greater than 98%</td>
<td>2.32 x 10^{-20}</td>
</tr>
</tbody>
</table>

3D Output

The goal of this section is to show the 3D outputs of the software developed for this thesis.

The tool is capable of highlighting the largest uncorrelated pairs of LOS rays in a 3D environment. The largest uncorrelated pairs of LOS rays are rated by the difference between their lengths. The user can choose the number of rays that will be drawn in the scene. The Figure 4-14 illustrates the twelve top uncorrelated pairs of LOS rays generated in the analysis described in this chapter. The left picture shows the rays in the baseline TBD, whereas the right picture illustrates the counterpart of that rays in the subject TDB. The length of the rays in the subject TDB are, in most of the cases, longer than their counterparts in the baseline TDB. The concentration of the longer rays in one of the TDBs indicates interoperability issues and conditions for an unfair fight in the aforementioned area(s).
The tool also provides a 3D visualization of the TDBs with posts indicating the magnitude of the average absolute difference between the lengths of corresponding LOS rays in each terrain block. The height of the post is proportional to the value of the average absolute difference between the lengths of corresponding LOS rays in the block. The block whose post is colored with “pure green” color has the least average absolute difference among all the terrain blocks, whereas the block whose post is colored with “pure red” color has the greatest average absolute difference. The colors of all the other posts vary from green to red according to the value of the average absolute difference of the lengths of corresponding LOS rays in the block. The Figure 4-15 shows the output of the tool for the data analyzed in this chapter.

Figure 4-14: Largest uncorrelated LOS rays
Figure 4-15: Colored posts indicating LOS correlation issues per block
CHAPTER FIVE: CONCLUSION

This thesis described a methodology for assessing the correlation between two TDBs using LOS measurements. An open source tool was developed in order to demonstrate this method. The approach discussed here relies only on the terrain’s geometry and it does not make any assumptions about the rendering process used to display the databases. For this reason, it is applicable in a wide variety of contexts so long as the terrain is triangulated. Triangulation algorithms must be added to the methodology in order to include TDBs composed of non-triangle polygons. Since the tool’s source code is provided with this thesis, there is opportunity to improve (or modify) it.

This chapter presents suggestions for overcoming the limitations faced in this thesis.

Performance

Because the tool was developed with the purpose of demonstrating the methodology for this thesis, performance was not the focus of the tool’s design. Therefore, improvements can be done in order to increase the LOS speed calculation. The tool implements a CPU based algorithm. GPU based techniques are likely to outperform the CPU based approaches with respect to LOS computation.
Terrain Features

The method discussed in this thesis focused on the geometry of the terrain surface, because everything in a synthetic natural environment is built upon the terrain. Hence, the terrain geometry is a necessary condition for determining whether there is a LOS between the sighting entity and the target entity. However, features of the terrain (e.g., building, trees, power lines, rivers, and roads) play an important role in many simulations. Poor correlation of the terrain features also hurts distributed simulation results. Therefore, the LOS correlation assessment discussed in this thesis can be improved in order to take the terrain’s features into account. However, the consideration of the terrain’s features in the correlation assessment would greatly increase the complexity of the analysis, since many details of the features must be checked one by one (e.g., orientation, size, and opacity).

Use of Terrain’s Roughness as an Indicator of LOS Correlation Issues

The Figures 4-3 and 4-5 appear to indicate that flat terrains are less likely to present LOS correlation issues. Because the LOS computation is a computer intensive process, the usage of the terrain roughness as an indicator for LOS correlation issues could allow the simulation analysts to concentrate their attention on not flat portions of the TDBs.

In addition, more roughness computation methods could be implemented in the tool. Currently, the tool calculates the roughness in a very efficient way (it is not necessary to sample the terrain in order to calculate the roughness). However, the method implemented in the tool is by no means the only existent roughness calculation approach. The usage of more than one
method for computing the terrain’s roughness would improve the accuracy of the roughness diagnosis.

Statistical Analysis, Human Factors, and More TDB Formats

The statistical software used in the Chapter 4 is not embedded in the tool. The integration of the statistical software with the tool would allow more flexibility in the hypothesis testing process.

Human factors were not taken into consideration in this thesis. Because the range of the human vision is limited to some extent, some pairs of LOS rays could be eliminated from the statistical analysis. The analysis proposed in this thesis is merely geometric, thus it does not take into consideration the characteristics (e.g., size) of the sighting and the target entities. A simulation analyst might use the tool for detecting areas poorly correlated with respect to LOS in a pair of TDBs regarding a desired level of correlation. This information could be used during the development of the training scenario. The scenery developer could prevent the encounter between simulation entities in poorly correlated areas. By the other hand, the tool could be used after the simulation is concluded. In the analysis of after action reports, the simulation results in poorly correlated areas must be interpreted with care, since the results might have bias.

Currently, the tool accepts only one format of TDB as input for terrain geometry. The usability of the tool can be expanded by adding support to other formats of TDB.
APPENDIX: SOURCE CODE
VTDBCorrelationAssessmentTool.pro

QT += core gui opengl
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = VTDBCorrelationAssessmentTool
TEMPLATE = app

INCLUDEPATH += ../OpenFlight_API/include  # include
debug{
  LIBS += -L../OpenFlight_API/lib/debug  # path to static libraries
  LIBS += -lmgapilib -lmgdd  # libs
}

release{
  LIBS += -L../OpenFlight_API/lib/release  # path to static libraries
  LIBS += -lmgapilib -lmgdd  # libs
}

DEFINES += WIN32 API_LEV4  # Pre-processor

SOURCES += main.cpp\
  widget.cpp \ myopenglwidget.cpp \ tdbanalyzer.cpp \ threadheavycalculations.cpp \ threadtraversescenegraph.cpp \ tdbcomparison.cpp

HEADERS += widget.h \ myopenglwidget.h \ tdbanalyzer.h \ threadheavycalculations.h \ threadtraversescenegraph.h \ util.h \ tdbcomparison.h

FORMS += widget.ui

RESOURCES += resources.qrc

main.cpp

#include "widget.h"
#include <QApplication>
#include <QMessageBox>

class thesisApplication : public QApplication
public:
    thesisApplication(int &argc, char ** argv): QApplication(argc, argv)
    {
    }
    // ~MyApplication();
private:
    bool notify(QObject *receiver_, QEvent *event_)
    {
        try
        {
            return QApplication::notify(receiver_, event_);
        } catch (std::exception & e )
        {
            QMessageBox errorMsgBox;
            QString exceptionStr;
            exceptionStr = "Exception: ";
            exceptionStr += e.what();
           errorMsgBox.setText(exceptionStr);
            errorMsgBox.setIcon(QMessageBox::Critical);
            errorMsgBox.exec();

            exceptionStr = "\nObject name(receiver): ";
            exceptionStr += receiver_->objectName();
            errorMsgBox.setText(exceptionStr);
            errorMsgBox.setIcon(QMessageBox::Critical);
            errorMsgBox.exec();

            return false;
        }
    }
};

int main(int argc, char *argv[])
{
    thesisApplication a(argc, argv);
    //QApplication a(argc, argv);
    Widget w;
    w.show();

    return a.exec();
}

util.h

#ifndef UTIL_H
#define UTIL_H

#include <QVector>
#include <QVector3D>

89
struct oyPolygon
{
    QVector<QVector3D> vertices;
    QVector3D Normal;
    QVector3D Centroid;
};

struct oyLOD
{
    QVector<oyPolygon> polygons;
    QVector<oyPolygon> boundingBoxExternal;
    QVector<oyPolygon> boundingBoxInternal;
    QVector<QVector3D> boundingBoxEdges;
    double maxX, maxY, maxZ;
    double minX, minY, minZ;
    double centerX, centerY, centerZ;
    double minLat, minLon, minHeight;
    double maxLat, maxLon, maxHeight;
};

struct oyLOSTestDataset
{
    QVector3D eyePosXYZ;
    double actualHeightAGL;
    QVector<QVector3D> upDirXYZList;
    QVector<QVector3D> dirXYZList;
    QVector<double> azimuthAngleList; //degrees
    QVector<double> pitchAngleList; //degrees
    QVector<QVector3D> intersectPntXYZList; //position in GL coordinate system
    QVector<double> eyeToIntersectPtDistList; //length
    QVector<bool> isRayIntersectingBBList;
};

struct oyProblematicRays
{
    QVector3D eyePosXYZ;
    QVector3D rayDirXYZ;
};

struct oyTerrainBlock
{
    int lineIndex, columnIndex;
    int EyePointOrthoProjectionCount;
    double northBoundDegrees; // decimal degrees
    double southBoundDegrees; // decimal degrees
    double westBoundDegrees; // decimal degrees
    double eastBoundDegrees; // decimal degrees
    double maxX; // GL units
    double minX; // GL units
    double maxY; // GL units
    double minY; // GL units
    double maxZ; // GL units
    double minZ; // GL units
    double centerLat;
double centerLon;
QVector<QVector3D> cornersXYZ;
QVector<QVector3D> eyePointSkyXYZList;
QVector<QVector3D> eyePointSkyLLEList;
QVector<QVector3D> eyePointGroundXYZList;
QVector<QVector3D> eyePointGroundLLEList;
QVector<oyLOSTestDataset> LOSDatasetList;
QVector<oyProblematicRays> problematicRayList;
QList<int> polyIndexList;
QVector3D blockNormalVectorAverage;
QVector3D blockNormalVectorStdDev;
int LOD;

oyTerrainBlock& operator =(const oyTerrainBlock& o)
{
    northBoundDegrees = o.northBoundDegrees; // decimal degrees
    southBoundDegrees = o.southBoundDegrees; // decimal degrees
    westBoundDegrees = o.westBoundDegrees; // decimal degrees
    eastBoundDegrees = o.eastBoundDegrees; // decimal degrees
    maxX = o.maxX; // GL units
    minX = o.minX; // GL units
    maxY = o.maxY; // GL units
    minY = o.minY; // GL units
    maxZ = o.maxZ; // GL units
    minZ = o.minZ; // GL units
    centerLat = o.centerLat;
    centerLon = o.centerLon;
    cornersXYZ = o.cornersXYZ;
    lineIndex = o.lineIndex;
    columnIndex = o.columnIndex;
    EyePointOrthoProjectionCount = o.EyePointOrthoProjectionCount;
    eyePointSkyXYZList = o.eyePointSkyXYZList;
    eyePointSkyLLEList = o.eyePointSkyLLEList;
    eyePointGroundXYZList = o.eyePointGroundXYZList;
    eyePointGroundLLEList = o.eyePointGroundLLEList;
    LOSDatasetList = o_LOSDatasetList;
    polyIndexList = o.polyIndexList;
    blockNormalVectorAverage = o.blockNormalVectorAverage;
    blockNormalVectorStdDev = o.blockNormalVectorStdDev;
    LOD = o.LOD;
    return *this;
}
```c
int blockLOSRayDiffSampleSize;
double blockElevAbsDiffMax;
double blockElevAbsDiffMin;
double blockElevAbsDiffAvg;
double blockElevAbsDiffStdDev;
double blockElevAbsDiffStdErr;
double blockElevDiffMax;
double blockElevDiffMin;
double blockElevDiffAvg;
double blockElevDiffStdDev;
double blockElevDiffStdErr;
int blockElevDiffSampleSize;
QVector<int> blockLOSRayDiffIndexList;
QVector<int> blockElevDiffIndexList;

struct oyGrndPosAnalysisData{
    double lat;
    double lon;
    double grndPosRayAbsDiffMax;
    double grndPosRayAbsDiffMin;
    double grndPosRayAbsDiffAvg;
    double grndPosLOSRayAbsDiffStdDev;
    double grndPosLOSRayAbsDiffStdErr;
    double grndPosRayDiffMax;
    double grndPosRayDiffMin;
    double grndPosRayDiffAvg;
    double grndPosLOSRayDiffStdDev;
    double grndPosLOSRayDiffStdErr;
    int grndPosLOSRayDiffSampleSize;
    QVector<int> grndPosLOSRayDiffIndexList;
};

struct oyRaySegment
{
    QVector3D eyePosXYZ;
    QVector3D intersecPtXYZ;
    bool isRayIntersectingBB;
};

struct oyLOSRayDiffUnity
{
    double AGLHeight;
    double azimuth;
    double pitch;
    double absDiffLOS;
    double diffLOS;
    bool BBIntersectingCorresp;
    QVector3D groundProjLLE;
    oyRaySegment TDB1RayXYZ;
    oyRaySegment TDB2RayXYZ;
};

struct oyTDBComparisonData
{
```
QVector<oyTerrainBlockAnalysisData> blockAnalysisData;
QVector<oyGrndPosAnalysisData> grndPosAnalysisData;
QVector<double> LOSRayDiffList;
QVector<double> elevDiffList;
QVector<oyLOSRayDiffUnity> LOSRayDiffUnityOrderedList;

double overallLOSRayAbsDiffMax;
double overallLOSRayAbsDiffMin;
double overallLOSRayAbsDiffAvg;
double overallLOSRayAbsDiffStdDev;
double overallLOSRayAbsDiffStdErr;
double overallLOSRayDiffMax;
double overallLOSRayDiffMin;
double overallLOSRayDiffAvg;
double overallLOSRayDiffStdDev;
double overallLOSRayDiffStdErr;
int overallLOSRayDiffSampleSize;

double overallElevationAbsDiffMax;
double overallElevationAbsDiffMin;
double overallElevationAbsDiffAvg;
double overallElevationAbsDiffStdDev;
double overallElevationAbsDiffStdErr;
double overallElevationDiffMax;
double overallElevationDiffMin;
double overallElevationDiffAvg;
double overallElevationDiffStdDev;
double overallElevationDiffStdErr;
int overallElevationDiffSampleSize;

};

#ifdef // UTIL_H

#endif

// UTIL_H

widget.h

#ifndef WIDGET_H
#define WIDGET_H

#include <QtCore>
#include <QWidget>
#include <QFileDialog>
#include <QMessageBox>
#include <QFile>
#include <QList>
#include <QTextStream>
#include <QDateTime>
#include <math.h>

#include "threadtraversescenegraph.h"
#include "tdbanalyzer.h"
#include "threadheavycalculations.h"
#include "tdbcomparison.h"
namespace Ui {

class Widget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

    QFile resumeFile, polygonInfoFile, LOSTestsResultsFile;
    QTextStream resumeStream, polyInfoStream, LOSTestsResultsStream;
    QMessageBox msgBoxInfoWarn;
    QMessageBox msgBoxError;
    QString OpenFlightCompleteFileName1, OpenFlightFileName1,
    OpenFlightFilePath1, OpenFlightCompleteFileName2,
    OpenFlightFileName2, OpenFlightFilePath2;
    ThreadTraverseSceneGraph *myTraverseThread1,
        OpenFlightCompleteFileName2, OpenFlightFileName2,
    OpenFlightFilePath2;
    ThreadTraverseSceneGraph *myTraverseThread2;
    bool isFirstTimeRedenringDB1, isFirstTimeRedenringDB2;
    bool isFirstCallToMultigenAPI;
    int LODToUseDB1, LODToUseDB2;
    TDBAnalyzer analyzer1, analyzer2;
    TDBComparison *compareTDBObj;
    QVector3D camPosLLE, camUpwardDirLLE, camLookAtLLE;
    QDateTime startSetEyepointsTDB1, startSetEyepointsTDB2,
    endSetEyepointsTDB1, endSetEyepointsTDB2;
    QDateTime startLOSTDB1, startLOSTDB2, endLOSTDB1, endLOSTDB2;
    ThreadHeavyCalculations *TDB1CalcThread1, *TDB1CalcThread2,
    *TDB2CalcThread1, *TDB2CalcThread2;
    bool isTDB1CalcThread1Ready, isTDB2CalcThread1Ready;
    bool isTDB1CalcThread2Ready, isTDB2CalcThread2Ready;

    void DrawTopUncorrRaysVisWidgets1();
    void DrawTopUncorrRaysVisWidgets2();
    void ClearLOSRaysVisWidget1();
    void ClearLOSRaysVisWidget2();

public slots:
    void HandleThreadTraverseSceneGraphReady1();
    void HandleThreadTraverseSceneGraphReady2();
    void HandleChildThreadError(QString error);
    void HandleTraverseSceneGraphMsgBox(QString info);
    void HandleTraverseSceneGraphTextEdit(QString out);
    void HandleWidgetVisualization1CameraUpdated();
    void HandleWidgetVisualization2CameraUpdated();
    void HandleThreadCalculationsEyepointProjectionsReady();
    void HandleThreadCalculationsLOSRready();

private slots:
}
void on_pushButtonOpenDB_1_clicked();
void on_pushButtonSaveGeometry_clicked();
void on_pushButtonOpenDB_2_clicked();
void on_tabWidget_currentChanged(int index);
void on_pushButtonCreateBlocks_clicked();
void on_pushButtonSetEyepoints_clicked();
void on_checkBoxDrawBlockBorders1_clicked();
void on_checkBoxDrawBlockBorders2_clicked();
void on_checkBoxDrawEyepoints1_clicked();
void on_checkBoxDrawEyepoints2_clicked();
void on_checkBoxDrawGroundEyepoints1_clicked();
void on_checkBoxDrawGroundEyepoints2_clicked();
void on_pushButtonSetupLOSTestAtt_clicked();
void on_pushButtonEnableTabDB1_clicked();
void on_pushButtonEnableTabDB2_clicked();
void on_doubleSpinBoxPitchIncrement_valueChanged(double arg1);
void on_spinBoxPitchCount_valueChanged(int arg1);
void on_checkBoxDrawLOSDirVec1_clicked();
void on_checkBoxDrawLOSDirVec2_clicked();
void on_pushButtonStartLOSTests_clicked();
void on_pushButtonClearSettings_clicked();
void on_checkBoxDrawLOS Rays1_clicked();
void on_checkBoxDrawLOS Rays2_clicked();
void on_checkBoxBoundingBox1_clicked();
void on_checkBoxBoundingBox2_clicked();
void on_pushButtonSaveLOSTestsResults_clicked();
void on_checkBoxDrawOriginAxes1_clicked();
void on_checkBoxDrawOriginAxes2_clicked();
void on_checkBoxDrawLOSAbsDiffPosts1_clicked();
void on_checkBoxDrawLOSAbsDiffPosts2_clicked();
void on_checkBoxDrawElevAbsDiffPosts1_clicked();
void on_checkBoxDrawElevAbsDiffPosts2_clicked();
void on_checkBoxDrawTopUncorrelatedRays1_clicked();
void on_checkBoxDrawTopUncorrelatedRays2_clicked();
void on_spinBoxTopUncorrelatedRays1_valueChanged(int arg1);
void on_spinBoxTopUncorrelatedRays2_valueChanged(int arg1);
void on_checkBoxDrawTopUncorrEyepointsOnly1_clicked();
void on_checkBoxDrawTopUncorrEyepointsOnly2_clicked();

private:
    Ui::Widget *ui;
};
#endif // WIDGET_H

widget.cpp

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    //setWindowFlags(Qt::CustomizeWindowHint | Qt::WindowMinimizeButtonHint | Qt::Window);
    // progress bars
    ui->progressBar->setVisible(false);
    ui->progressBar_6->setVisible(false);

    // tabs
    ui->tabWidget->setTabEnabled(1, false);
    ui->tabWidget->setTabEnabled(2, false);
    ui->tabWidget->setTabEnabled(3, false);

    // combo boxes
    QStringList comboBoxList;
    for(int i = 1; i <= 10; i++)
    {
        comboBoxList.append(QString::number(i*i));
    }
ui->comboBoxEyePointsNumber->addItem(comboBoxList);
ui->comboBoxEyePointsNumber->setCurrentIndex(2);

// frames
ui->frame_2->setEnabled(false);
ui->frame_3->setEnabled(false);
ui->frame_4->setVisible(false);
ui->frame_5->setVisible(false);
ui->frame_6->setEnabled(false);
ui->frame_7->setVisible(false);
ui->frame_9->setVisible(false);
ui->frame_10->setEnabled(false);
ui->frame_11->setEnabled(false);

// check boxes
ui->checkBoxDrawBlockBorders1->setEnabled(false);
ui->checkBoxDrawBlockBorders2->setEnabled(false);
ui->checkBoxDrawEyepoints1->setEnabled(false);
ui->checkBoxDrawEyepoints2->setEnabled(false);
ui->checkBoxDrawGroundEyepoints1->setEnabled(false);
ui->checkBoxDrawGroundEyepoints2->setEnabled(false);
ui->checkBoxDrawLOSDirVec1->setEnabled(false);
ui->checkBoxDrawLOSDirVec2->setEnabled(false);
ui->checkBoxDrawLOSRays1->setEnabled(false);
ui->checkBoxDrawLOSRays2->setEnabled(false);
ui->checkBoxDrawOriginAxes1->setChecked(true);
ui->checkBoxDrawOriginAxes2->setChecked(true);

// push buttons
ui->pushButtonOpenDB_2->setEnabled(false);
ui->pushButtonSaveGeometry->setVisible(false);
ui->pushButtonSaveLOSTestsResults->setEnabled(false);

// child widgets attributes
ui->widgetVisualization_1->enableDrawOriginAxes = true;
ui->widgetVisualization_2->enableDrawOriginAxes = true;

// attributes of this class
isFirstCallToMultigenAPI = true;
isFirstTimeRenedingDB1 = true;
isFirstTimeRenedingDB2 = true;
LODToUseDB1 = 0;
LODToUseDB2 = 0;

connect(&analyzer1, SIGNAL(error(QString)), this, SLOT(HandleChildThreadError(QString)));
connect(&analyzer2, SIGNAL(error(QString)), this, SLOT(HandleChildThreadError(QString)));
} 

Widget::~Widget()
{
    delete ui;
}

void Widget::on_pushButtonOpenDB_1_clicked()
{
    ui->checkBoxSaveTraverseLog->setEnabled(false);
    QStringList auxStringList;
    OpenFlightCompleteFileName1 = "";

    OpenFlightCompleteFileName1 = QFileDialog::getOpenFileName(this, tr("Open Terrain Database"),
"C:/MyTDBs/2ndAttempt/RoughTerrain/NED_1_third_arc_sec_2_LOD/flight/", tr("OpenFlight Files (*.flt)")));  

    OpenFlightCompleteFileName1 = QFileDialog::getOpenFileName(this, tr("Open Terrain Database"),
"C:/MyTDBs/2ndAttempt/Montrey/ASTER_GDEM_2_3_LOD/flight/", tr("OpenFlight Files (*.flt)")));  

    if (OpenFlightCompleteFileName1.size() == 0)
    {
        ui->textEdit->append("Invalid file name for database 1.
    }
    else
    {
        myTraverseThread1 = new ThreadTraverseSceneGraph(this,
OpenFlightCompleteFileName1);
        myTraverseThread1->isOpenFlightFirstCall = isFirstCallToMultigenAPI;
        myTraverseThread1->isOpenFlightLastCall = false;
        myTraverseThread1->dataBaseOrderNumber = 1;
        connect(myTraverseThread1, SIGNAL(resultReady()), this,
SLOT(HandleThreadTraverseSceneGraphReady1()));
        connect(myTraverseThread1, SIGNAL(error(QString)), this,
SLOT(HandleChildThreadError(QString)));
        connect(myTraverseThread1, SIGNAL(updateProgress(int)), ui-
>progressBar, SLOT(setValue(int)));
        connect(myTraverseThread1, SIGNAL(infoReady(QString)), this,
SLOT(HandleTraverseSceneGraphMsgBox(QString)));
        connect(myTraverseThread1, SIGNAL(standardOut(QString)), ui-
{textEdit, SLOT(append(QString))});
        ui->pushButtonClose->setEnabled(false);
        ui->textEdit->clear(); // clear the text box
        ui->textEdit->append("Opening database 1.
    }
    auxStringList = OpenFlightCompleteFileName1.split("/",
QString::KeepEmptyParts, Qt::CaseSensitive);
    OpenFlightFileName1 = auxStringList[auxStringList.size()-1];
    OpenFlightFilePath1 = "";
    for (int i = 0; i < (auxStringList.size()-1); i++)
    {
        OpenFlightFilePath1.append(auxStringList[i]);
        OpenFlightFilePath1.append("/");
    
    }
void Widget::on_pushButtonOpenDB_1_clicked()
{
    ui->checkBoxSaveTraverseLog->setEnabled(false);
    myTraverseThread1->saveTraverseLog = (ui->checkBoxSaveTraverseLog->isChecked()) ? true : false;
    ui->progressBar->setVisible(true);
    myTraverseThread1->start();
}

void Widget::on_pushButtonOpenDB_2_clicked()
{
    ui->checkBoxSaveTraverseLog->setEnabled(false);
    QStringList auxStringList;
    OpenFlightCompleteFileName2 = ""
;
    OpenFlightCompleteFileName2 = QFileDialog::getOpenFileName(this, tr("Open Terrain Database"),
        tr("C:/MyTDBs/2ndAttempt/RoughTerrain/NED_1_arc_sec_2_LOD/flight/"),
        tr("Open Flight Files (*.flt)");
    
    auxStringList = OpenFlightCompleteFileName2.split("/", Qt::KeepEmptyParts, Qt::CaseSensitive);
    OpenFlightFileName2 = auxStringList[auxStringList.size() - 1];
    OpenFlightFilePath2 = "";
    for (int i = 0; i < (auxStringList.size()-1); i++)
    {
        OpenFlightFilePath2.append(auxStringList[i]);
    
    auxStringList = OpenFlightCompleteFileName2.split("/", Qt::KeepEmptyParts, Qt::CaseSensitive);
    OpenFlightFileName2 = auxStringList[auxStringList.size()-1];
    OpenFlightFilePath2 = "";
    for (int i = 0; i < (auxStringList.size()-1); i++)
    {
OpenFlightFilePath2.append("/");
ui->pushButtonOpenDB_2->setEnabled(false);
myTraverseThread2->saveTraverLog = (ui->checkBoxSaveTraverseLog->isChecked()) ? true : false;
ui->progressBar_6->setVisible(true);
myTraverseThread2->start();

void Widget::on_pushButtonSaveGeometry_clicked() // needs to be adapted to handle the DB2's thread
{
    int saveFilesOk = 0;
    QString resultsFilePath, suggestedFileName;
    ui->pushButtonSaveGeometry->setEnabled(false);
    QDateTime now = QDateTime::currentDateTime();
    suggestedFileName = "SUMMARY_";
    suggestedFileName.append(now.toString("MMM.dd.yyyy.h.mm.ap"));
    suggestedFileName.replace(".", "_庭");
    suggestedFileName.append(".txt");
    resultsFilePath = suggestedFileName;
    resumeFile.setFileName(resultsFilePath);
    if(!resumeFile.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        msgBoxInfoWarn.setText("The summary file was not saved!");
        msgBoxInfoWarn.setIcon(QMessageBox::Warning);
        msgBoxInfoWarn.exec();
    }
    else
    {
        resumeStream.setDevice(&resumeFile);
        QString resultsContent;
        resultsContent = ui->textEdit->toPlainText();
        resumeStream << resultsContent;
        resumeStream << "\n\n";
        resultsContent = ui->textEdit_2->toPlainText();
        resumeStream << resultsContent;
        resumeFile.close();
        saveFilesOk++;
    }
    suggestedFileName = "POLY_INFO_";
    suggestedFileName.append(OpenFlightCompleteFileName1);
    now = QDateTime::currentDateTime();
    suggestedFileName.replace(":"", "庭");
    suggestedFileName.replace("/", "_庭");
    suggestedFileName.replace(".", "_庭");
    suggestedFileName.replace("_庭", "_庭");
    suggestedFileName.append("_庭");
    suggestedFileName.append(now.toString("MMM.dd.yyyy.h.mm.ap"));
    suggestedFileName.replace(":"", "庭");
    suggestedFileName.replace("/", "_庭");
    suggestedFileName.replace(".", "_庭");
    suggestedFileName.append(".txt");
polygonInfoFile.setFileName(suggestedFileName);
if (!polygonInfoFile.open(QIODevice::WriteOnly | QIODevice::Text))
{
    msgBoxInfoWarn.setText("The polygons' description file for Database #2 was not saved!");
    msgBoxInfoWarn.setIcon(QMessageBox::Warning);
    msgBoxInfoWarn.exec();
} else {
    if (ui->progressBar->isHidden())
    {
        ui->progressBar->setVisible(true);
    }
    polyInfoStream.setDevice(&polygonInfoFile);
    polyInfoStream << "Description of the polygons in the file: " << 
OpenFlightCompleteFileName1 << "\n";
    for (int count = 0; count < analyzer1.LODList.size(); count++)
    {
        polyInfoStream << "############################################################################
          
LOD = " << count << " " "############################################################################\n"
        polyInfoStream << "############################################################################
          
for (int n = 0; n < analyzer1.LODList[count].polygons.size(); n++)
        {
            QCoreApplication::processEvents();
            polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].Normal.x(), 'f', 6) << "\t";
            polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].Normal.y(), 'f', 6) << "\t";
            polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].Normal.z(), 'f', 6) << "\t";
            for (int vertexCount = 0; vertexCount < 
analyzer1.LODList[count].polygons[n].vertices.size(); vertexCount++)
            {
                polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].vertices[vertexCount].x( 
), 'f', 6) << "\t";
                polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].vertices[vertexCount].y( 
), 'f', 6) << "\t";
                polyInfoStream << 
QString::number(analyzer1.LODList[count].polygons[n].vertices[vertexCount].z( 
), 'f', 6) << "\t";
                polyInfoStream << "\n";
            }
            ui->progressBar->setValue(100 * ((double)(n + 1)) / 
((double)analyzer1.LODList[count].polygons.size()));
        }
{ }
polygonInfoFile.close();
saveFilesOk++;
}
ui->progressBar_6->setVisible(false);
suggestedFileName = "POLY_INFO_";
suggestedFileName.append(OpenFlightCompleteFileName2);
now = QDateTime::currentDateTime();
suggestedFileName.replace(":", "_");
suggestedFileName.replace("/", "_");
suggestedFileName.replace("_", "_");
suggestedFileName.append("_");
suggestedFileName.append(now.toString("MMM.dd.yyyy.h.mm.ap"));
suggestedFileName.replace(".", "_");
suggestedFileName.replace("_", "_");
suggestedFileName.append("\.txt");
polygonInfoFile.setFileName(suggestedFileName);
if(!polygonInfoFile.open(QIODevice::WriteOnly | QIODevice::Text))
{
    msgBoxInfoWarn.setText("The polygons\' description file for Database 
#2 was not saved!");
    msgBoxInfoWarn.setIcon(QMessageBox::Warning);
    msgBoxInfoWarn.exec();
} else
{
    if(ui->progressBar_6->isHidden())
    {
        ui->progressBar_6->setVisible(true);
    }
polyInfoStream.setDevice(&polygonInfoFile);
polyInfoStream << "Description of the polygons in the file: 
OpenFlightCompleteFileName2 << "\n";
    for (int count = 0; count < analyzer2.LODList.size(); count++)
    {
        polyInfoStream << "############################################################################
Lod = " << count << " 
";
        polyInfoStream << "###########################################
for (int n = 0; n < analyzer2.LODList[count].polygons.size(); n++)
        {
            QCoreApplication::processEvents();
            polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].Normal.x(), 'f', 6) << "\t";
            polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].Normal.y(), 'f', 6) << "\t";
polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].Normal.z(), 'f', 6) << " \t";

    for (int vertexCount = 0; vertexCount < analyzer2.LODList[count].polygons[n].vertices.size(); vertexCount++)
    {
        polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].vertices[vertexCount].x( ), 'f', 6) << " \t";
        polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].vertices[vertexCount].y( ), 'f', 6) << " \t";
        polyInfoStream <<
QString::number(analyzer2.LODList[count].polygons[n].vertices[vertexCount].z( ), 'f', 6) << " \n";
    }
    polyInfoStream << "\n";
ui->progressBar_6->setValue( 100 * ((double)(n +1)) / ((double)analyzer2.LODList[count].polygons.size()));
}
}
polygonInfoFile.close();
saveFilesOk++;

if (saveFilesOk == 3)
{
    ui->pushButtonSaveGeometry->setEnabled(false);
    msgBoxInfoWarn.setText("The polygons\' description file and the resume file were saved successfully.\n");
    msgBoxInfoWarn.setIcon(QMessageBox::Information);
    msgBoxInfoWarn.exec();
}
else
{
    ui->pushButtonSaveGeometry->setEnabled(true);
}
ui->progressBar_6->setVisible(false);

void Widget::HandleThreadTraverseSceneGraphReady1()
{
    // fill the analyzer object with the data obtained from the thread
    analyzer1.fileName = OpenFlightCompleteFileName1;
    analyzer1.dbProjection = myTraverseThread1->dbProjection;
    analyzer1.LODCount = myTraverseThread1->dbMaxLOD;

    for(int i = 0; i <= analyzer1.LODCount; i++)
    {
        oLOD currentLOD;

        currentLOD.minX = myTraverseThread1->minX[i];
        currentLOD.minY = myTraverseThread1->minY[i];
        currentLOD.minZ = myTraverseThread1->minZ[i];
        currentLOD.centerX = myTraverseThread1->centerX[i];
currentLOD.centerY = myTraverseThread1->centerY[i];
currentLOD.centerZ = myTraverseThread1->centerZ[i];
currentLOD.maxX = myTraverseThread1->maxX[i];
currentLOD.maxY = myTraverseThread1->maxY[i];
currentLOD.maxZ = myTraverseThread1->maxZ[i];
currentLOD.minLat = myTraverseThread1->minLat[i];
currentLOD.minLon = myTraverseThread1->minLon[i];
currentLOD.minHeight = myTraverseThread1->minHeight[i];
currentLOD.maxLat = myTraverseThread1->maxLat[i];
currentLOD.maxLon = myTraverseThread1->maxLon[i];
currentLOD.maxHeight = myTraverseThread1->maxHeight[i];

for (int j = 0; j < myTraverseThread1->polygonInfolList[i].size(); j++) {
    QCoreApplication::processEvents();
    QPolygon currentPolygon;
    currentPolygon.Normal.setX(myTraverseThread1->polygonInfolList[i][j].normal_i);
    currentPolygon.Normal.setY(myTraverseThread1->polygonInfolList[i][j].normal_j);
    currentPolygon.Normal.setZ(myTraverseThread1->polygonInfolList[i][j].normal_k);
    for (int k = 0; k < myTraverseThread1->polygonInfolList[i][j].vertices.size(); k++) {
        QVector3D currentVertex;
        currentVertex.setX(myTraverseThread1->polygonInfolList[i][j].vertices[k].coord[0]);
        currentVertex.setY(myTraverseThread1->polygonInfolList[i][j].vertices[k].coord[1]);
        currentVertex.setZ(myTraverseThread1->polygonInfolList[i][j].vertices[k].coord[2]);
        currentPolygon.vertices.push_back(currentVertex);
    }
    QVector3D currentCentroid;
    currentCentroid.setX((currentPolygon.vertices[0].x() +
    currentPolygon.vertices[1].x() + currentPolygon.vertices[2].x()) / 3);
    currentCentroid.setY((currentPolygon.vertices[0].y() +
    currentPolygon.vertices[1].y() + currentPolygon.vertices[2].y()) / 3);
    currentCentroid.setZ((currentPolygon.vertices[0].z() +
    currentPolygon.vertices[1].z() + currentPolygon.vertices[2].z()) / 3);
    currentPolygon.Centroid = currentCentroid;
    currentLOD.polygons.push_back(currentPolygon);
}
}
analyzer1.LODList.push_back(currentLOD);
}
delete myTraverseThread1;

//update GUI
QStringList comboBoxList;
for (int i = 0; i <= analyzer1.LODCount; i++)
comboBoxList.append(QString::number(i));
ui->comboBoxLODDB1->addItems(comboBoxList);
ui->comboBoxLODDB1->setCurrentIndex(analyzer1.LODCount);
ui->progressBar->setVisible(false);
ui->pushButtonOpenDB_2->setEnabled(true);
ui->checkBoxSaveTraverseLog->setEnabled(true);
ui->frame_4->setVisible(true);
}

void Widget::HandleThreadTraverseSceneGraphReady2()
{
    // fill the analyzer object with the data obtained from the thread
    analyzer2.fileName = OpenFlightCompleteFileName2;
analyzer2.dbProjection = myTraverseThread2->dbProjection;
analyzer2.LODCount = myTraverseThread2->dbMaxLOD;

    for(int i = 0; i <= analyzer2.LODCount; i++)
    {
        oyLOD currentLOD;
currentLOD.maxX = myTraverseThread2->maxX[i];
currentLOD.maxY = myTraverseThread2->maxY[i];
currentLOD.maxZ = myTraverseThread2->maxZ[i];

currentLOD.minX = myTraverseThread2->minX[i];
currentLOD.minY = myTraverseThread2->minY[i];
currentLOD.minZ = myTraverseThread2->minZ[i];

currentLOD.centerX = myTraverseThread2->centerX[i];
currentLOD.centerY = myTraverseThread2->centerY[i];
currentLOD.centerZ = myTraverseThread2->centerZ[i];

    currentLOD.minLat = myTraverseThread2->minLat[i];
currentLOD.minLon = myTraverseThread2->minLon[i];
currentLOD.minHeight = myTraverseThread2->minHeight[i];

currentLOD.maxLat = myTraverseThread2->maxLat[i];
currentLOD.maxLon = myTraverseThread2->maxLon[i];
currentLOD.maxHeight = myTraverseThread2->maxHeight[i];

    for(int j = 0; j < myTraverseThread2->polygonInfolList[i].size(); j++)
    {
        QCoreApplication::processEvents();
    oyPolygon currentPolygon;
currentPolygon.Normal.setX(myTraverseThread2->polygonInfolList[i][j].normal_i);
currentPolygon.Normal.setY(myTraverseThread2->polygonInfolList[i][j].normal_j);
currentPolygon.Normal.setZ(myTraverseThread2->polygonInfolList[i][j].normal_k);
        for (int k = 0; k < myTraverseThread2->polygonInfolList[i][j].vertices.size(); k++)
        {
    }
QVector3D currentVertex;
currentVertex.setX(myTraverseThread2->polygonInfolList[i][j].vertices[k].coord[0]);
currentVertex.setY(myTraverseThread2->polygonInfolList[i][j].vertices[k].coord[1]);
currentVertex.setZ(myTraverseThread2->polygonInfolList[i][j].vertices[k].coord[2]);
currentPolygon.vertices.push_back(currentVertex);
}
QVector3D currentCentroid;
currentCentroid.setX((currentPolygon.vertices[0].x() + currentPolygon.vertices[1].x() + currentPolygon.vertices[2].x()) / 3);
currentCentroid.setY((currentPolygon.vertices[0].y() + currentPolygon.vertices[1].y() + currentPolygon.vertices[2].y()) / 3);
currentCentroid.setZ((currentPolygon.vertices[0].z() + currentPolygon.vertices[1].z() + currentPolygon.vertices[2].z()) / 3);
currentPolygon.Centroid = currentCentroid;
currentLOD.polygons.push_back(currentPolygon);
}
analyzer2.LODList.push_back(currentLOD);
}
delete myTraverseThread2;

//update GUI
QStringList comboBoxList;
for (int i = 0; i <= analyzer2.LODCount; i++)
{
    comboBoxList.append(QString::number(i));
}
ui->comboBoxLODDB2->addItems(comboBoxList);
ui->comboBoxLODDB2->setCurrentIndex(analyzer2.LODCount);
ui->pushButtonSaveGeometry->setVisible(true);
ui->progressBar_6->setVisible(false);
ui->frame_5->setVisible(true);

void Widget::HandleChildThreadError(QString error)
{
    QMessageBox msgBox;
    msgBox.setText(error);
    msgBox.setIcon(QMessageBox::Critical);
    msgBox.exec();
}

void Widget::HandleTraverseSceneGraphMsgBox(QString info)
{
    QMessageBox msgBox;
    msgBox.setText(info);
    msgBox.setIcon(QMessageBox::Information);
    msgBox.exec();
}

void Widget::HandleTraverseSceneGraphTextEdit(QString out)
{
ui->textEdit->append(out);
ui->textEdit->repaint();
}

void Widget::on_pushButtonEnableTabDB1_clicked()
{
    ui->tabWidget->setTabEnabled(1, true);
    ui->frame_4->setEnabled(false);
    LODToUseDB1 = ui->comboBoxLODDB1->currentText().toInt();
    analyzer1.LODToBeAnalyzed = LODToUseDB1;
    if (!ui->frame_5->isEnabled())
    {        double maxLat, minLat, maxLon, minLon, maxHeight, minHeight;
        maxLat = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxLat;
        minLat = analyzer1.LODList[analyzer1.LODToBeAnalyzed].minLat;
        maxLon = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxLon;
        minLon = analyzer1.LODList[analyzer1.LODToBeAnalyzed].minLon;
        maxHeight = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxHeight;
        minHeight = analyzer1.LODList[analyzer1.LODToBeAnalyzed].minHeight;
        if (maxLat < analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLat)
            maxLat = analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLat;
        if (maxLon < analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLon)
            maxLon = analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLon;
        if (maxHeight < analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxHeight)
            maxHeight = analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxHeight;
        if (minLat > analyzer2.LODList[analyzer1.LODToBeAnalyzed].minLat)
            minLat = analyzer2.LODList[analyzer1.LODToBeAnalyzed].minLat;
        if (minLon > analyzer2.LODList[analyzer1.LODToBeAnalyzed].minLon)
            minLon = analyzer2.LODList[analyzer1.LODToBeAnalyzed].minLon;
        if (minHeight > analyzer2.LODList[analyzer1.LODToBeAnalyzed].minHeight)
            minHeight = analyzer2.LODList[analyzer1.LODToBeAnalyzed].minHeight;

        analyzer1.CreateBoundingBoxes(maxLat, minLat, maxLon, minLon,
                                        maxHeight, minHeight);
        ui->checkBoxBoundingBox1->setEnabled(true);
        analyzer2.CreateBoundingBoxes(maxLat, minLat, maxLon, minLon,
                                        maxHeight, minHeight);
        ui->checkBoxBoundingBox2->setEnabled(true);
    }
}

void Widget::on_pushButtonEnableTabDB2_clicked()
{
    ui->tabWidget->setTabEnabled(2, true);
    ui->tabWidget->setTabEnabled(3, true);
    ui->frame_5->setEnabled(false);
    LODToUseDB2 = ui->comboBoxLODDB2->currentText().toInt();
    analyzer2.LODToBeAnalyzed = LODToUseDB2;
    if (!ui->frame_4->isEnabled())
    {        double maxLat, minLat, maxLon, minLon, maxHeight, minHeight;

$$\text{maxLat} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxLat}$$
$$\text{minLat} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].minLat}$$
$$\text{maxLon} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxLon}$$
$$\text{minLon} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].minLon}$$
$$\text{maxHeight} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxHeight}$$
$$\text{minHeight} = \text{analyzer1.LODList[analyzer1.LODToBeAnalyzed].minHeight}$$

if (\text{maxLat} < \text{analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLat})
\text{maxLat} = \text{analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLat}
\text{if (maxLon < analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLon)}
\text{maxLon} = \text{analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxLon}
\text{if (maxHeight < analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxHeight)}
\text{maxHeight} = \text{analyzer2.LODList[analyzer1.LODToBeAnalyzed].maxHeight}

\text{analyzer1.CreateBoundingBoxes(maxLat, minLat, maxLon, minLon, maxHeight, minHeight);}$$
\text{ui->checkBoxBoundingBox1->setEnabled(true);}$$
\text{analyzer2.CreateBoundingBoxes(maxLat, minLat, maxLon, minLon, maxHeight, minHeight);}$$
\text{ui->checkBoxBoundingBox2->setEnabled(true);}$$

\text{void Widget::on_tabWidget_currentChanged(int index)}$$
\text{if((index != 1) && (index != 2))}
\text{if(!ui->widgetVisualization_1->terrainVertices.isEmpty())}
\text{ui->widgetVisualization_1->terrainVertices.clear();}$$
\text{if(!ui->widgetVisualization_1->terrainNormals.isEmpty())}
\text{ui->widgetVisualization_1->terrainNormals.clear();}$$
\text{if(!ui->widgetVisualization_1->terrainTextureCoordinates.isEmpty())}
\text{ui->widgetVisualization_1->terrainTextureCoordinates.clear();}$$

\text{if(!ui->widgetVisualization_2->terrainVertices.isEmpty())}
\text{ui->widgetVisualization_2->terrainVertices.clear();}$$
\text{if(!ui->widgetVisualization_2->terrainNormals.isEmpty())}
\text{ui->widgetVisualization_2->terrainNormals.clear();}$$
\text{if(!ui->widgetVisualization_2->terrainTextureCoordinates.isEmpty())}
\text{ui->widgetVisualization_2->terrainTextureCoordinates.clear();}$$
\text{if(index == 1)}$$
\text{if(isFirstTimeRedenringDB1)}$$
\text{QVector3D LODCenter;
for (int i = 0; i < analyzer1.LODList[LODToUseDB1].polygons.size(); i++)
{
    oyPolygon currentPoly;
    currentPoly = analyzer1.LODList[LODToUseDB1].polygons[i];
    for (int j = 0; j < currentPoly.vertices.size(); j++)
    {
        ui->widgetVisualization_1->terrainVertices.push_back(currentPoly.vertices[j]);
        ui->widgetVisualization_1->terrainNormals.push_back(currentPoly.normal);
        switch (j)
        {
            case 0:
            ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(0, 0));
            break;
            case 1:
            ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(1, 0));
            break;
            case 2:
            ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(1, 1));
            break;
        }
    }
    LODCenter.setX(analyzer1.LODList[LODToUseDB1].centerX);
    LODCenter.setY(analyzer1.LODList[LODToUseDB1].centerY);
    LODCenter.setZ(analyzer1.LODList[LODToUseDB1].centerZ);
    ui->widgetVisualization_1->objectCenter = LODCenter;
    ui->widgetVisualization_1->cameraPosition = LODCenter;
    ui->widgetVisualization_1->cameraPosition.setZ(analyzer1.LODList[LODToUseDB1].maxZ + 600.0);
    ui->widgetVisualization_1->cameraForwardDirection = QVector3D(0, 1, 0);
    ui->widgetVisualization_1->cameraUpDirection = QVector3D(0, 0, 1);
    ui->widgetVisualization_1->cameraLookAt = ui->widgetVisualization_1->cameraPosition + ui->widgetVisualization_1->cameraForwardDirection;
    ui->widgetVisualization_1->cameraRightDirection = QVector3D::crossProduct(ui->widgetVisualization_1->cameraForwardDirection, ui->widgetVisualization_1->cameraUpDirection);
    ui->widgetVisualization_1->cameraRightDirection.normalize();
    //ui->widgetVisualization_1->setTerrainBuffer();
    ui->widgetVisualization_1->lightPosition.setX(LODCenter.x());
    ui->widgetVisualization_1->lightPosition.setY(LODCenter.y());
    ui->widgetVisualization_1->lightPosition.setZ(analyzer1.LODList[LODToUseDB1].maxZ + 5000);
    ui->widgetVisualization_1->enableDrawTerrain = true;
    ui->widgetVisualization_1->repaint();
    isFirstTimeRedenringDB1 = false;
connect(ui->widgetVisualization_1, SIGNAL(cameraUpdated()), this,
SLOT(HandleWidgetVisualization1CameraUpdated()));
}
else {
    ui->widgetVisualization_2->terrainVertices.clear();
    ui->widgetVisualization_2->terrainNormals.clear();
    ui->widgetVisualization_2->terrainTextureCoordinates.clear();
    for(int i = 0; i < analyzer1.LODList[LODToUseDB1].polygons.size(); i++)
    {
        oyPolygon currentPoly;
        currentPoly = analyzer1.LODList[LODToUseDB1].polygons[i];
        for(int j = 0; j < currentPoly.vertices.size(); j++)
        {
            ui->widgetVisualization_1->terrainVertices.push_back(currentPoly.vertices[j]);
            ui->widgetVisualization_1->terrainNormals.push_back(currentPoly.Normal);
            switch(j)
            {
            case 0:
                ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(0,0));
                break;
            case 1:
                ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(1,0));
                break;
            case 2:
                ui->widgetVisualization_1->terrainTextureCoordinates.push_back(QVector2D(1,1));
                break;
            }
        }
    }
    if (index == 2)
    {
        if (isFirstTimeRedenringDB2)
        {
            QVector3D LODCenter;
            for(int i = 0; i < analyzer2.LODList[LODToUseDB2].polygons.size(); i++)
            {
                oyPolygon currentPoly;
                currentPoly = analyzer2.LODList[LODToUseDB2].polygons[i];
                for(int j = 0; j < currentPoly.vertices.size(); j++)
                {
                    ui->widgetVisualization_2->terrainVertices.push_back(currentPoly.vertices[j]);
                    ui->widgetVisualization_2->terrainNormals.push_back(currentPoly.Normal);
                    switch(j)
```cpp
{  
    case 0:  
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(0, 0));  
        break;
    case 1:  
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(1, 0));  
        break;
    case 2:  
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(1, 1));  
        break;
}
LODCenter.setX(analyzer2.LODList[LODToUseDB2].centerX);
LODCenter.setY(analyzer2.LODList[LODToUseDB2].centerY);
LODCenter.setZ(analyzer2.LODList[LODToUseDB2].centerZ);
ui->widgetVisualization_2->objectCenter = LODCenter;
ui->widgetVisualization_2->cameraPosition = LODCenter;
ui->widgetVisualization_2->cameraPosition.setZ(analyzer1.LODList[LODToUseDB2].maxZ + 600.0);
ui->widgetVisualization_2->cameraForwardDirection = QVector3D(0, 1, 0);
ui->widgetVisualization_2->cameraUpDirection = QVector3D(0, 0, 1);
ui->widget Visualization_2->cameraLookAt = ui->widgetVisualization_2->cameraPosition;
ui->widgetVisualization_2->cameraRightDirection = QVector3D::crossProduct(ui->widgetVisualization_2->cameraForwardDirection, ui->widgetVisualization_2->cameraUpDirection);
ui->widgetVisualization_2->cameraRightDirection.normalize();
ui->widgetVisualization_2->setTerrainBuffer();
ui->widgetVisualization_2->step = 14;
ui->widgetVisualization_2->lightPosition.setX(LODCenter.x());
ui->widgetVisualization_2->lightPosition.setY(LODCenter.y());
ui->widgetVisualization_2->lightPosition.setZ(analyzer2.LODList[LODToUseDB2].maxZ + 5000);
ui->widgetVisualization_2->enableDrawTerrain = true;
ui->widgetVisualization_2->repaint();
isFirstTimeRedenringDB2 = false;
connect(ui->widgetVisualization_2, SIGNAL(cameraUpdated()), this, SLOT(HandleWidgetVisualization2CameraUpdated()));
} else {
    ui->widgetVisualization_1->terrainVertices.clear();
    ui->widgetVisualization_1->terrainNormals.clear();
    ui->widgetVisualization_1->terrainTextureCoordinates.clear();
    for(int i = 0; i < analyzer2.LODList[LODToUseDB2].polygons.size(); i++)
    {
        oYPolygon currentPoly;
        currentPoly = analyzer2.LODList[LODToUseDB2].polygons[i];
```
for(int j = 0; j < currentPoly.vertices.size(); j++)
{
    ui->widgetVisualization_2->terrainVertices.push_back(currentPoly.vertices[j]);
    ui->widgetVisualization_2->terrainNormals.push_back(currentPoly.Normal);
    switch(j)
    {
    case 0:
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(0,0));
        break;
    case 1:
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(1,0));
        break;
    case 2:
        ui->widgetVisualization_2->terrainTextureCoordinates.push_back(QVector2D(1,1));
        break;
    }
}


void Widget::on_pushButtonCreateBlocks_clicked()
{
    ui->pushButtonCreateBlocks->setEnabled(false);
    ui->spinBoxLines->setEnabled(false);
    ui->spinBoxColumns->setEnabled(false);
    ui->frame->repaint();

    analyzer1.CreateBlocks(ui->spinBoxLines->value(), ui->spinBoxColumns->value());
    ui->checkBoxDrawBlockBorders1->setEnabled(true);
    ui->widgetVisualization_1->repaint();

    analyzer2.CreateBlocks(analyzer1.terrainBlockList);
    ui->checkBoxDrawBlockBorders2->setEnabled(true);
    ui->widgetVisualization_2->repaint();

    msgBoxInfoWarn.setText("The blocks were created.");
    msgBoxInfoWarn.setIcon(QMessageBox::Information);
    msgBoxInfoWarn.exec();
    ui->frame_2->setEnabled(true);
}

void Widget::on_pushButtonSetEyepoints_clicked()
{
    ui->pushButtonSetEyepoints->setEnabled(false);
    ui->comboBoxEyePointsNumber->setEnabled(false);
    ui->frame_7->setVisible(true);
    ui->frame_9->setVisible(true);
ui->frame_2->repaint();

analyzer1.FillSkyEyePointsList(ui->comboBoxEyePointsNumber-
>currentText().toInt());

TDB1CalcThread1 = new ThreadHeavyCalculations(this, &analyzer1);
TDB1CalcThread1->eyepointsPerBlock = ui->comboBoxEyePointsNumber-
>currentText().toInt();
TDB1CalcThread1->initialBlock = 0;
TDB1CalcThread1->blockOffset = analyzer1.terrainBlockList.size() / 2;
TDB1CalcThread1->operationMode = 0;
TDB1CalcThread1->createTDBLocalCopy = false;
connect(TDB1CalcThread1, SIGNAL(resultEyepointProjectionsReady()), this,
SLOT(HandleThreadCalculationsEyepointProjectionsReady()));
connect(TDB1CalcThread1, SIGNAL(resultLOSReady()), this,
SLOT(HandleThreadCalculationsLOSPrepare());
connect(TDB1CalcThread1, SIGNAL(updateThreadCalculationProgress(int)),
ui->progressBar_2, SLOT(setValue(int))));

TDB1CalcThread2 = new ThreadHeavyCalculations(this, &analyzer1);
TDB1CalcThread2->eyepointsPerBlock = ui->comboBoxEyePointsNumber-
>currentText().toInt();
TDB1CalcThread2->initialBlock = analyzer1.terrainBlockList.size() / 2;
TDB1CalcThread2->blockOffset = analyzer1.terrainBlockList.size() -
(analyzer1.terrainBlockList.size() / 2);
TDB1CalcThread2->operationMode = 0;
TDB1CalcThread2->createTDBLocalCopy = true;
connect(TDB1CalcThread2, SIGNAL(resultEyepointProjectionsReady()), this,
SLOT(HandleThreadCalculationsEyepointProjectionsReady()));
connect(TDB1CalcThread2, SIGNAL(resultLOSReady()), this,
SLOT(HandleThreadCalculationsLOSPrepare());
connect(TDB1CalcThread2, SIGNAL(updateThreadCalculationProgress(int)),
ui->progressBar_3, SLOT(setValue(int))));

TDB2CalcThread1 = new ThreadHeavyCalculations(this, &analyzer2);
TDB2CalcThread1->blockList = analyzer1.terrainBlockList;
TDB2CalcThread1->initialBlock = 0;
TDB2CalcThread1->blockOffset = analyzer2.terrainBlockList.size() / 2;
TDB2CalcThread1->operationMode = 1;
TDB2CalcThread1->createTDBLocalCopy = false;
connect(TDB2CalcThread1, SIGNAL(resultEyepointProjectionsReady()), this,
SLOT(HandleThreadCalculationsEyepointProjectionsReady()));
connect(TDB2CalcThread1, SIGNAL(resultLOSReady()), this,
SLOT(HandleThreadCalculationsLOSPrepare());
connect(TDB2CalcThread1, SIGNAL(updateThreadCalculationProgress(int)),
ui->progressBar_4, SLOT(setValue(int))));

TDB2CalcThread2 = new ThreadHeavyCalculations(this, &analyzer2);
TDB2CalcThread2->blockList = analyzer1.terrainBlockList;
TDB2CalcThread2->initialBlock = analyzer2.terrainBlockList.size() / 2;
TDB2CalcThread2->blockOffset = analyzer2.terrainBlockList.size() -
(analyzer2.terrainBlockList.size() / 2);
TDB2CalcThread2->operationMode = 1;
TDB2CalcThread2->createTDBLocalCopy = true;
connect(TDB2CalcThread2, SIGNAL(resultEyepointProjectionsReady()), this, SLOT(HandleThreadCalculationsEyepointProjectionsReady()));
connect(TDB2CalcThread2, SIGNAL(resultLOSReady()), this, SLOT(HandleThreadCalculationsLOSReady()));
connect(TDB2CalcThread2, SIGNAL(updateThreadCalculationProgress(int)), ui->progressBar_5, SLOT(setValue(int)));

isTDB1CalcThread1Ready = false;
isTDB1CalcThread2Ready = false;
isTDB2CalcThread1Ready = false;
isTDB2CalcThread2Ready = false;

startSetEyepointsTDB1 = QDateTime::currentDateTime();
qDebug("\nTDB1 - probe terrain started at:");
qDebug() << startSetEyepointsTDB1;
TDB1CalcThread1->start();
TDB1CalcThread2->start();

startSetEyepointsTDB2 = QDateTime::currentDateTime();
qDebug("\nTDB2 - probe terrain started at:");
qDebug() << startSetEyepointsTDB2;
TDB2CalcThread1->start();
TDB2CalcThread2->start();

void Widget::on_checkBoxDrawBlockBorders1_clicked()
{
if (ui->checkBoxDrawBlockBorders1->isChecked())
{
for(int i = 0; i < analyzer1.terrainBlockList.size(); i++)
{
if (analyzer1.terrainBlockList[i].cornersXYZ.size() == 8)
{
ui->widgetVisualization_1->
bordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[0])
;
ui->widgetVisualization_1->
bordersColors.push_back(QVector3D(0, 0, 1))
;
ui->widgetVisualization_1->
bordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[1])
;
ui->widgetVisualization_1->
bordersColors.push_back(QVector3D(0, 0, 1))
;
ui->widgetVisualization_1->
bordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[2])
;
ui->widgetVisualization_1->
bordersColors.push_back(QVector3D(0, 0, 1));
}
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[2]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[3]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[3]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[0]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[4]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[5]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[5]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[6]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[6]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1->blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[7]);
ui->widgetVisualization_1->blocksBordersColors.push_back(QVector3D(0, 0, 1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[7])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[4])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[0])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[1])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[2])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[3])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[4])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[5])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[6])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_1-
>blocksBordersVertices.push_back(analyzer1.terrainBlockList[i].cornersXYZ[7])
>blocksBordersColors.push_back(QVector3D(0,0,1));

}
else
{
    msgBoxError.setText("This terrain block has less than 8 vertices.");
    msgBoxError.setIcon(QMessageBox::Critical);
    msgBoxError.exec();
}
ui->widgetVisualization_1->enableDrawBlockBorders = true;
ui->widgetVisualization_1->repaint();
else
{
    ui->widgetVisualization_1->enableDrawBlockBorders = false;
    ui->widgetVisualization_1->repaint();
    ui->widgetVisualization_1->blocksBordersVertices.clear();
    ui->widgetVisualization_1->blocksBordersColors.clear();
}
}

void Widget::on_checkBoxDrawBlockBorders2_clicked()
{
    if (ui->checkBoxDrawBlockBorders2->isChecked())
    {
        for(int i = 0; i < analyzer2.terrainBlockList.size(); i++)
        {
            if (analyzer1.terrainBlockList[i].cornersXYZ.size() == 8)
            {
                ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[0]);
                ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
                ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[1]);
                ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
                ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[1]);
                ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
                ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[2]);
                ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
                ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[2]);
                ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
            }
        }
    }
}
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[3])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[3])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[0])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[4])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[5])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[5])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[6])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[6])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[7])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2-
>blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[7])
>blocksBordersColors.push_back(QVector3D(0,0,1));
ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[4]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[0]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[4]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[1]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[2]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[5]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[6]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
    ui->widgetVisualization_2->blocksBordersVertices.push_back(analyzer2.terrainBlockList[i].cornersXYZ[7]);
    ui->widgetVisualization_2->blocksBordersColors.push_back(QVector3D(0,0,1));
}
else
{
    msgBoxError.setText("This terrain block has less than 8 vertices.");
    msgBoxError.setIcon(QMessageBox::Critical);
```cpp
msgBoxError.exec();
}
ui->widgetVisualization_2->enableDrawBlockBorders = true;
ui->widgetVisualization_2->repaint();
}
else
{
    ui->widgetVisualization_2->enableDrawBlockBorders = false;
    ui->widgetVisualization_2->repaint();
    ui->widgetVisualization_2->blocksBordersVertices.clear();
    ui->widgetVisualization_2->blocksBordersColors.clear();
}
}

void Widget::on_checkBoxDrawEyepoints1_clicked()
{
    if (ui->checkBoxDrawEyepoints1->isChecked())
    {
        for (int i = 0; i < analyzer1.terrainBlockList.size(); i++)
        {
            for (int j = 0; j < analyzer1.terrainBlockList[i].eyePointSkyXYZList.size(); j++)
            {
                ui->widgetVisualization_1->eyepointSkyVertices.push_back(analyzer1.terrainBlockList[i].eyePointSkyXYZList[j]);
                ui->widgetVisualization_1->eyepointSkyColors.push_back(QVector3D(1, 0.65, 0));
            }
        }
        ui->widgetVisualization_1->enableDrawSkyEyePoints = true;
        ui->widgetVisualization_1->repaint();
    }
    else
    {
        ui->widgetVisualization_1->enableDrawSkyEyePoints = false;
        ui->widgetVisualization_1->repaint();
        ui->widgetVisualization_1->eyepointSkyVertices.clear();
        ui->widgetVisualization_1->eyepointSkyColors.clear();
    }
}

void Widget::on_checkBoxDrawEyepoints2_clicked()
{
    if (ui->checkBoxDrawEyepoints2->isChecked())
    {
        for (int i = 0; i < analyzer2.terrainBlockList.size(); i++)
        {
            for (int j = 0; j < analyzer2.terrainBlockList[i].eyePointSkyXYZList.size(); j++)
            {
                ui->widgetVisualization_2->eyepointSkyVertices.push_back(analyzer2.terrainBlockList[i].eyePointSkyXYZList[j]);
            }
        }
    }
}
```
ui->widgetVisualization_2->eyepointSkyColors.push_back(QVector3D(1, 0.65, 0));
}
ui->widgetVisualization_2->enableDrawSkyEyePoints = true;
ui->widgetVisualization_2->repaint();
}
else{
  ui->widgetVisualization_2->enableDrawSkyEyePoints = false;
ui->widgetVisualization_2->repaint();
ui->widgetVisualization_2->eyepointSkyVertices.clear();
ui->widgetVisualization_2->eyepointSkyColors.clear();
}
}
void Widget::on_checkBoxDrawGroundEyepoints1_clicked()
{
  if (ui->checkBoxDrawGroundEyepoints1->isChecked())
  {
    for(int i = 0; i < analyzer1.terrainBlockList.size(); i++)
    {
      for(int j = 0; j < analyzer1.terrainBlockList[i].eyePointSkyXYZList.size(); j++)
      {
        ui->widgetVisualization_1->eyepointGroundVertices.push_back(analyzer1.terrainBlockList[i].eyePointGroundXYZList[j]);
        ui->widgetVisualization_1->eyepointGroundColors.push_back(QVector3D(0.9, 0, 1));
      }
    }
    ui->widgetVisualization_1->enableDrawGroundEyePoints = true;
    ui->widgetVisualization_1->repaint();
  }
  else{
    ui->widgetVisualization_1->enableDrawGroundEyePoints = false;
    ui->widgetVisualization_1->repaint();
    ui->widgetVisualization_1->eyepointGroundVertices.clear();
    ui->widgetVisualization_1->eyepointGroundColors.clear();
  }
}
void Widget::on_checkBoxDrawGroundEyepoints2_clicked()
{
  if (ui->checkBoxDrawGroundEyepoints2->isChecked())
  {
    for(int i = 0; i < analyzer2.terrainBlockList.size(); i++)
    {
      for(int j = 0; j < analyzer2.terrainBlockList[i].eyePointSkyXYZList.size(); j++)
      {
void Widget::on_pushButtonSetupLOSTestAtt_clicked()
{
    double startingHeight, heightInc, pitchInc;
    int heightIncCount, azimuthIncCount, pitchIncCount;
    startingHeight = ui->doubleSpinBoxStartingHeight->value();
    heightInc = ui->doubleSpinBoxHeightIncrement->value();
    heightIncCount = ui->spinBoxHeightIncCount->value();
    azimuthIncCount = ui->spinBoxAzimuthCount->value();
    pitchInc = ui->doubleSpinBoxPitchtIncrement->value();
    pitchIncCount = ui->spinBoxPitchCount->value();

    ui->doubleSpinBoxStartingHeight->setEnabled(false);
    ui->doubleSpinBoxHeightIncrement->setEnabled(false);
    ui->spinBoxHeightIncCount->setEnabled(false);
    ui->spinBoxAzimuthCount->setEnabled(false);
    ui->doubleSpinBoxPitchtIncrement->setEnabled(false);
    ui->spinBoxPitchCount->setEnabled(false);
    ui->frame_3->setEnabled(true);
    analyzer1.CreateDirVecList(startingHeight, heightInc, heightIncCount,
                                azimuthIncCount, pitchInc, pitchIncCount);
    ui->checkBoxDrawLOSDirVec1->setEnabled(true);
    analyzer2.CreateDirVecList(startingHeight, heightInc, heightIncCount,
                                azimuthIncCount, pitchInc, pitchIncCount);
    ui->checkBoxDrawLOSDirVec2->setEnabled(true);
    msgBoxInfoWarn.setText("Directional vectors were set for each eye-point in the map.");
    msgBoxInfoWarn.setIcon(QMessageBox::Information);
    msgBoxInfoWarn.exec();
    ui->frame_6->setEnabled(true);
}

void Widget::on_doubleSpinBoxPitchtIncrement_valueChanged(double arg1)
if ((ui->spinBoxPitchCount->value() * arg1) >= 90)
{
    msgBoxInfoWarn.setText("(Pitch increment) * (# of pitch increments) must not exceed 90 degrees.");
    msgBoxInfoWarn.setIcon(QMessageBox::Warning);
    msgBoxInfoWarn.exec();
    ui->doubleSpinBoxPitchtIncrement->stepDown();
}
}

void Widget::on_spinBoxPitchCount_valueChanged(int arg1)
{
    if ((ui->doubleSpinBoxPitchtIncrement->value() * arg1) >= 90)
    {
        msgBoxInfoWarn.setText("(Pitch increment) * (# of pitch increments) must not exceed 90 degrees.");
        msgBoxInfoWarn.setIcon(QMessageBox::Warning);
        msgBoxInfoWarn.exec();
        ui->spinBoxPitchCount->stepDown();
    }
}

void Widget::on_checkBoxDrawLOSDirVec1_clicked()
{
    if (ui->checkBoxDrawLOSDirVec1->isChecked())
    {
        for (int i = 0; i < analyzer1.terrainBlockList.size(); i++)
        {
            QVector3D eyePos, dir;
            for (int j = 0; j < analyzer1.terrainBlockList[i].LOSDatasetList.size(); j++)
            {
                eyePos = analyzer1.terrainBlockList[i].LOSDatasetList[j].eyePosXYZ;
                for (int k = 0; k < analyzer1.terrainBlockList[i].LOSDatasetList[j].dirXYZList.size(); k++)
                {
                    dir = analyzer1.terrainBlockList[i].LOSDatasetList[j].dirXYZList[k];
                    ui->widgetVisualization_1->dirVecLOSVertices.push_back(eyePos);
                    ui->widgetVisualization_1->dirVecLOSColors.push_back(QVector3D(1, 0, 0));
                    ui->widgetVisualization_1->dirVecLOSVertices.push_back(eyePos + 15 * dir);
                    ui->widgetVisualization_1->dirVecLOSColors.push_back(QVector3D(0, 0, 1));
                }
            }
            ui->widgetVisualization_1->enableDrawDirVecLOS = true;
            ui->widgetVisualization_1->repaint();
        }
    }
}
{   
    ui->widgetVisualization_1->enableDrawDirVecLOS = false;
    ui->widgetVisualization_1->repaint();
    ui->widgetVisualization_1->dirVecLOSVertices.clear();
    ui->widgetVisualization_1->dirVecLOSColors.clear();
}
}

void Widget::on_checkBoxDrawLOSDirVec2_clicked()
{
    if (ui->checkBoxDrawLOSDirVec2->isChecked())
    {
        for (int i = 0; i < analyzer2.terrainBlockList.size(); i++)
        {
            QVector3D eyePos, dir;
            for (int j = 0; j < analyzer2.terrainBlockList[i].LOSDataSetList.size(); j++)
            {
                eyePos = analyzer2.terrainBlockList[i].LOSDataSetList[j].eyePosXYZ;
                for (int k = 0; k < analyzer2.terrainBlockList[i].LOSDataSetList[j].dirXYZList.size(); k++)
                {
                    dir = analyzer2.terrainBlockList[i].LOSDataSetList[j].dirXYZList[k];
                    ui->widgetVisualization_2->dirVecLOSVertices.push_back(eyePos);
                    ui->widgetVisualization_2->dirVecLOSColors.push_back(QVector3D(1,0,0));
                    ui->widgetVisualization_2->dirVecLOSVertices.push_back(eyePos + 15 * dir);
                    ui->widgetVisualization_2->dirVecLOSColors.push_back(QVector3D(0,0,1));
                }
            }
            ui->widgetVisualization_2->enableDrawDirVecLOS = true;
            ui->widgetVisualization_2->repaint();
        }
    }
    else
    {
        ui->widgetVisualization_2->enableDrawDirVecLOS = false;
        ui->widgetVisualization_2->repaint();
        ui->widgetVisualization_2->dirVecLOSVertices.clear();
        ui->widgetVisualization_2->dirVecLOSColors.clear();
    }
}

void Widget::on_checkBoxDrawLOSrays1_clicked()
{
    ui->widgetVisualization_1->enableDrawEyepointsOnly = false;
    if (ui->checkBoxDrawLOSrays1->isChecked())
    {
        if (ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
        {
            // Code for drawing top uncorrelated rays
        }
    }
}
{ 
    ui->checkBoxDrawTopUncorrelatedRays1->setChecked(false);
    ClearLOS RaysVisWidget1();
}

if (ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
{
    ui->checkBoxDrawTopUncorrelatedRays2->setChecked(false);
    ClearLOS RaysVisWidget2();
}

for (int i = 0; i < compareTDBObj->comparData. LOSRayDiffUnityOrderedList.size(); i++)
{
    QOSRayDiffUnity currentDiffUnity;
    QVector3D eyePos, intersectPt;
    currentDiffUnity = compareTDBObj->comparData. LOSRayDiffUnityOrderedList[i];
    eyePos = compareTDBObj->comparData. LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.eyePosXYZ;
    intersectPt = compareTDBObj->comparData. LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.intersectPtXYZ;

    ui->widgetVisualization_1->LOS RaysVertices.push_back(eyePos);
    ui->widgetVisualization_1->LOS RaysColors.push_back(QVector3D(0, 1, 0.27));
    ui->widgetVisualization_1->LOS IntersectionVertices.push_back(intersectPt);
    ui->widgetVisualization_1->LOS IntersectionColors.push_back(QVector3D(1, 0, 0));
}

for (int i = 0; i < analyzer1. terrainBlockList.size(); i++)
{
    for (int k = 0; k < analyzer1. terrainBlockList[i].problematicRayList.size(); k++)
    {
        QVector3D eyePos, rayDir, auxVec;
        eyePos = analyzer1. terrainBlockList[i].problematicRayList[k].eyePosXYZ;
        rayDir = analyzer1. terrainBlockList[i].problematicRayList[k].rayDirXYZ;
        auxVec = eyePos + 200 * rayDir;
        ui->widgetVisualization_1->problematic RaysVertices.push_back(eyePos);
        ui->widgetVisualization_1->problematic RaysColors.push_back(QVector3D(0.443, 0.203, 0.756));
        ui->widgetVisualization_1->problematic RaysVertices.push_back(auxVec);
    }
ui->widgetVisualization_1->problematicRaysColors.push_back(QVector3D(0.443, 0.203, 0.756));

}
ui->widgetVisualization_1->enableDrawLOSrays = true;
if(ui->widgetVisualization_1->problematicRaysVertices.size() > 0)
{
    ui->widgetVisualization_1->enableDrawProblematicRays = true;
}
ui->widgetVisualization_1->repaint();
else
{
    ClearLOSRaysVisWidget1();
}

void Widget::on_checkBoxDrawLOSrays2_clicked()
{
    ui->widgetVisualization_2->enableDrawEyepointsOnly = false;
    if (ui->checkBoxDrawLOSrays2->isChecked())
    {
        if(ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
        {
            ui->checkBoxDrawTopUncorrelatedRays1->setChecked(false);
            ClearLOSRaysVisWidget1();
        }
        if(ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
        {
            ui->checkBoxDrawTopUncorrelatedRays2->setChecked(false);
            ClearLOSRaysVisWidget2();
        }
        for (int i = 0; i < compareTDBObj->comparData.LOSRayDiffUnityOrderedList.size(); i++)
        {
            oyLOSRayDiffUnity currentDiffUnity;
            QVector3D eyePos, intersectPt;
            currentDiffUnity = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB2RayXYZ.eyePosXYZ;
            intersectPt = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB2RayXYZ.intersectPtXYZ;
            ui->widgetVisualization_2->LOSraysVertices.push_back(eyePos);
            ui->widgetVisualization_2->LOSraysColors.push_back(QVector3D(0, 1, 0.27));
            ui->widgetVisualization_2->LOSraysVertices.push_back(intersectPt);
            ui->widgetVisualization_2->LOSraysColors.push_back(QVector3D(0, 1, 0.27));
            ui->widgetVisualization_2->actualEyepointsVertices.push_back(eyePos);
        }
    }
ui->widgetVisualization_2->actualEyepointsColors.push_back(QVector3D(0.219, 0.49, 0.843));
ui->widgetVisualization_2->LOSIntersectionVertices.push_back(intersectPt);
ui->widgetVisualization_2->LOSIntersectionColors.push_back(QVector3D(1, 0, 0));
}
for(int i = 0; i < analyzer2.terrainBlockList.size(); i++)
{
    for(int k = 0; k < analyzer2.terrainBlockList[i].problematicRayList.size(); k++)
    {
        QVector3D eyePos, rayDir, auxVec;
        eyePos = analyzer2.terrainBlockList[i].problematicRayList[k].eyePosXYZ;
        rayDir = analyzer2.terrainBlockList[i].problematicRayList[k].rayDirXYZ;
        auxVec = eyePos + 200 * rayDir;
        ui->widgetVisualization_2->problematicRaysVertices.push_back(eyePos);
        ui->widgetVisualization_2->problematicRaysColors.push_back(QVector3D(0.443, 0.203, 0.756));
        ui->widgetVisualization_2->problematicRaysVertices.push_back(auxVec);
        ui->widgetVisualization_2->problematicRaysColors.push_back(QVector3D(0.443, 0.203, 0.756));
    }
}
ui->widgetVisualization_2->enableDrawLOS Rays = true;
if(ui->widgetVisualization_2->problematicRaysVertices.size() > 0)
{
    ui->widgetVisualization_2->enableDrawProblematicRays = true;
}
ui->widgetVisualization_2->repaint();
else
{
    ClearLOS RaysVisWidget2();
}
}

void Widget::on_pushButtonStartLOSTests_clicked()
{
    ui->pushButtonStartLOSTests->setEnabled(false);
    ui->pushButtonClearSettings->setEnabled(false);
    ui->progressBar_2->reset();
    ui->progressBar_3->reset();
    ui->progressBar_4->reset();
    ui->progressBar_5->reset();
    ui->frame_7->setVisible(true);
    ui->frame_9->setVisible(true);
    ui->frame_6->repaint();
    TDB1CalcThread1->operationMode = 2;
TDB1CalcThread2->operationMode = 2;
TDB2CalcThread1->operationMode = 2;
TDB2CalcThread2->operationMode = 2;

isTDB1CalcThread1Ready = false;
isTDB1CalcThread2Ready = false;
isTDB2CalcThread1Ready = false;
isTDB2CalcThread2Ready = false;

startLOSTDB1 = QDateTime::currentDateTime();
qDebug("\nTDB1 - LOS tests started at: ");
qDebug() << startLOSTDB1;
TDB1CalcThread1->start();
TDB1CalcThread2->start();

startLOSTDB2 = QDateTime::currentDateTime();
qDebug("\nTDB2 - LOS tests started at: ");
qDebug() << startLOSTDB2;
TDB2CalcThread1->start();
TDB2CalcThread2->start();

void Widget::on_pushButtonClearSettings_clicked()
{
    analyzer1.terrainBlockList.clear();
analyzer2.terrainBlockList.clear();
    if(compareTDBObj->initialized)
        delete compareTDBObj;

    // clear block borders
    ui->spinBoxLines->setEnabled(true);
    ui->spinBoxColumns->setEnabled(true);
    ui->pushButtonCreateBlocks->setEnabled(true);
    ui->frame->setEnabled(true);
    ui->frame->repaint();

    ui->widgetVisualization_1->enableDrawBlockBorders = false;
    ui->checkBoxDrawBlockBorders1->setEnabled(false);
    ui->checkBoxDrawBlockBorders1->setChecked(false);
    ui->widgetVisualization_1->blocksBordersVertices.clear();
    ui->widgetVisualization_1->blocksBordersColors.clear();

    ui->widgetVisualization_2->enableDrawBlockBorders = false;
    ui->checkBoxDrawBlockBorders2->setEnabled(false);
    ui->checkBoxDrawBlockBorders2->setChecked(false);
    ui->widgetVisualization_2->blocksBordersVertices.clear();
    ui->widgetVisualization_2->blocksBordersColors.clear();

    // clear eyepoints
    ui->pushButtonSetEyepoints->setEnabled(true);
    ui->comboBoxEyePointsNumber->setEnabled(true);
    ui->frame_2->setEnabled(false);
    ui->frame_2->repaint();

    ui->widgetVisualization_1->enableDrawSkyEyePoints = false;
ui->widgetVisualization_1->enableDrawGroundEyePoints = false;
ui->checkBoxDrawEyepoints1->setEnabled(false);
ui->checkBoxDrawGroundEyePoints1->setChecked(false);
ui->checkBoxDrawGroundEyePoints1->setEnabled(false);
ui->checkBoxDrawGroundEyePoints1->setChecked(false);
ui->widgetVisualization_1->eyePointGroundVertices.clear();
ui->widgetVisualization_1->eyePointGroundColors.clear();
ui->widgetVisualization_1->eyePointSkyVertices.clear();
ui->widgetVisualization_1->eyePointSkyColors.clear();

ui->widgetVisualization_2->enableDrawSkyEyePoints = false;
ui->widgetVisualization_2->enableDrawGroundEyePoints = false;
ui->checkBoxDrawEyepoints2->setEnabled(false);
ui->checkBoxDrawGroundEyePoints2->setChecked(false);
ui->checkBoxDrawGroundEyePoints2->setEnabled(false);
ui->checkBoxDrawGroundEyePoints2->setChecked(false);
ui->widgetVisualization_2->eyePointGroundVertices.clear();
ui->widgetVisualization_2->eyePointGroundColors.clear();
ui->widgetVisualization_2->eyePointSkyVertices.clear();
ui->widgetVisualization_2->eyePointSkyColors.clear();

// clear eye point settings and direction vectors
ui->doubleSpinBoxStartingHeight->setEnabled(true);
ui->doubleSpinBoxHeightIncrement->setEnabled(true);
ui->spinBoxHeightIncCount->setEnabled(true);
ui->spinBoxAzimuthCount->setEnabled(true);
ui->doubleSpinBoxPitchIncrement->setEnabled(true);
ui->spinBoxPitchCount->setEnabled(true);
ui->pushButtonSetupLOSTestAtt->setEnabled(true);
ui->frame_3->setEnabled(false);
ui->frame_3->repaint();

ui->widgetVisualization_1->enableDrawDirVecLOS = false;
ui->checkBoxDrawLOSDirVec1->setChecked(false);
ui->checkBoxDrawLOSDirVec1->setEnabled(false);
ui->widgetVisualization_1->dirVecLOSVertices.clear();
ui->widgetVisualization_1->dirVecLOSColors.clear();

ui->widgetVisualization_2->enableDrawDirVecLOS = false;
ui->checkBoxDrawLOSDirVec2->setChecked(false);
ui->checkBoxDrawLOSDirVec2->setEnabled(false);
ui->widgetVisualization_2->dirVecLOSVertices.clear();
ui->widgetVisualization_2->dirVecLOSColors.clear();

// clear LOS rays
ui->pushButtonStartLOSTests->setEnabled(true);
ui->pushButtonSaveLOSTestsResults->setEnabled(false);
ui->frame_6->setEnabled(false);

ui->widgetVisualization_1->enableDrawLOSrays = false;
ui->checkBoxDrawLOSrays1->setChecked(false);
ui->checkBoxDrawLOSrays1->setEnabled(false);
ui->widgetVisualization_1->LOSraysVertices.clear();
ui->widgetVisualization_1->LOSraysColors.clear();
ui->widgetVisualization_1->LOSIntersectionVertices.clear();
ui->widgetVisualization_1->LOSIntersectionColors.clear();
ui->widgetVisualization_1->actualEyepointsVertices.clear();
ui->widgetVisualization_1->actualEyepointsColors.clear();

ui->widgetVisualization_2->enableDrawLOSRays = false;
ui->checkBoxDrawLOSRays2->setChecked(false);
ui->checkBoxDrawLOSRays2->setEnabled(false);
ui->widgetVisualization_2->LOSRaysVertices.clear();
ui->widgetVisualization_2->LOSRaysColors.clear();
ui->widgetVisualization_2->LOSIntersectionVertices.clear();
ui->widgetVisualization_2->LOSIntersectionColors.clear();
ui->widgetVisualization_2->actualEyepointsVertices.clear();
ui->widgetVisualization_2->actualEyepointsColors.clear();

ui->widgetVisualization_1->postPositionLOS.clear();
ui->widgetVisualization_1->postColorLOS.clear();
ui->widgetVisualization_1->postHeightLOS.clear();
ui->widgetVisualization_1->enableDrawPostsLOS = false;
ui->checkBoxDrawLOSAbsDiffPosts1->setEnabled(false);
ui->checkBoxDrawLOSAbsDiffPosts1->setChecked(false);
ui->widgetVisualization_1->postPositionElev.clear();
ui->widgetVisualization_1->postColorElev.clear();
ui->widgetVisualization_1->postHeightElev.clear();
ui->widgetVisualization_1->enableDrawPostsElev = false;
ui->checkBoxDrawElevAbsDiffPosts1->setEnabled(false);
ui->checkBoxDrawElevAbsDiffPosts1->setChecked(false);

ui->widgetVisualization_2->postPositionLOS.clear();
ui->widgetVisualization_2->postColorLOS.clear();
ui->widgetVisualization_2->postHeightLOS.clear();
ui->widgetVisualization_2->enableDrawPostsLOS = false;
ui->checkBoxDrawLOSAbsDiffPosts2->setEnabled(false);
ui->checkBoxDrawLOSAbsDiffPosts2->setChecked(false);
ui->widgetVisualization_2->postPositionElev.clear();
ui->widgetVisualization_2->postColorElev.clear();
ui->widgetVisualization_2->postHeightElev.clear();
ui->widgetVisualization_2->enableDrawPostsElev = false;
ui->checkBoxDrawElevAbsDiffPosts2->setEnabled(false);
ui->checkBoxDrawElevAbsDiffPosts2->setChecked(false);

ui->frame_10->setEnabled(false);
ui->frame_11->setEnabled(false);
ui->checkBoxDrawTopUncorrelatedRays1->setChecked(false);
ui->checkBoxDrawTopUncorrelatedRays2->setChecked(false);
ui->spinBoxTopUncorrelatedRays1->setValue(0);
ui->spinBoxTopUncorrelatedRays2->setValue(0);

ui->widgetVisualization_1->repaint();
ui->widgetVisualization_2->repaint();

delete TDB1CalcThread1;
delete TDB1CalcThread2;
delete TDB2CalcThread1;
delete TDB2CalcThread2;
void Widget::on_checkBoxBoundingBox1_clicked()
{
    if (ui->checkBoxBoundingBox1->isChecked())
    {
        for (int i = 0; i < analyzer1.LODList[analyzer1.LODToBeAnalyzed].boundingBoxInternal.size(); i++)
        {
            for (int j = 0; j < analyzer1.LODList[analyzer1.LODToBeAnalyzed].boundingBoxInternal[i].vertices.size(); j++)
            {
                ui->widgetVisualization_1->boundingBoxVertices.push_back(analyzer1.LODList[analyzer1.LODToBeAnalyzed].boundingBoxInternal[i].vertices[j]);
                ui->widgetVisualization_1->boundingBoxColors.push_back(QVector4D(0.5, 0.5, 0.5, 0.5));
            }
        }
    }
    else
    {
        ui->widgetVisualization_1->enableBoundingBox = false;
        ui->widgetVisualization_1->repaint();
    }
}

void Widget::on_checkBoxBoundingBox2_clicked()
{ if (ui->checkBoxBoundingBox2->isChecked())
{
    for(int i = 0; i < analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxInternal.size(); i++)
    {
        for(int j = 0; j < analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxInternal[i].vertices.size(); j++)
        {
            ui->widgetVisualization_2->boundingBoxVertices.push_back(analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxInternal[i].vertices[j]);
            ui->widgetVisualization_2->boundingBoxColors.push_back(QVector4D(0.5,0.5,0.5,0.5));
        }
    }
    for(int i = 0; i < analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxExternal.size(); i++)
    {
        for(int j = 0; j < analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxExternal[i].vertices.size(); j++)
        {
            ui->widgetVisualization_2->boundingBoxVertices.push_back(analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxExternal[i].vertices[j]);
            ui->widgetVisualization_2->boundingBoxColors.push_back(QVector4D(0.5,0.5,0.5,0.5));
        }
    }
    for(int i = 0; i < analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxEdges.size(); i++)
    {
        ui->widgetVisualization_2->boundingBoxEdgesVertices.push_back(analyzer2.LODList[analyzer2.LODToBeAnalyzed].boundingBoxEdges[i]);
        ui->widgetVisualization_2->boundingBoxEdgesColors.push_back(QVector3D(0,0,0));
    }
    ui->widgetVisualization_2->enableBoundingBox = true;
    ui->widgetVisualization_2->repaint();
}
else
{
    ui->widgetVisualization_2->enableBoundingBox = false;
    ui->widgetVisualization_2->repaint();
    ui->widgetVisualization_2->boundingBoxEdgesVertices.clear();
    ui->widgetVisualization_2->boundingBoxEdgesColors.clear();
}
}

void Widget::on_pushButtonSaveLOSTestsResults_clicked()
{
    ui->pushButtonSaveLOSTestsResults->setEnabled(false);
ui->frame_6->repaint();

if (compareTDBObj->printResults())
{
    ui->pushButtonClose->setEnabled(true);
    QMessageBox::setText("The results were saved successfully.\n");
    QMessageBox::setIcon(QMessageBox::Information);
    QMessageBox::exec();
}
else
{
    QMessageBox::setText("The file with the LOS results could not be saved!");
    QMessageBox::setIcon(QMessageBox::Warning);
    QMessageBox::exec();
    ui->pushButtonSaveLOSTestsResults->setEnabled(true);
}

void Widget::HandleWidgetVisualization1CameraUpdated()
{
    camPosLLE = analyzer1.convertXYZtoLLE(ui->widgetVisualization_1->cameraPosition);
    camLookAtLLE = analyzer1.convertXYZtoLLE(ui->widgetVisualization_1->cameraLookAt);
    if (ui->tabWidget->isTabEnabled(2))
    {
        ui->widgetVisualization_2->cameraPosition = analyzer2.convertLLEtoXYZ(camPosLLE);
        ui->widgetVisualization_2->cameraUpDirection = QVector3D(0, 0, 1);
        ui->widgetVisualization_2->cameraUpDirection.normalize();
        ui->widgetVisualization_2->cameraLookAt = analyzer2.convertLLEtoXYZ(camLookAtLLE);
        ui->widgetVisualization_2->cameraForwardDirection = ui->
        widgetVisualization_2->cameraLookAt - ui->widgetVisualization_2->
        cameraPosition;
        ui->widgetVisualization_2->cameraForwardDirection.normalize();
        ui->widgetVisualization_2->cameraRightDirection =
        QVector3D::crossProduct(ui->widgetVisualization_2->cameraForwardDirection,
        ui->widgetVisualization_2->cameraUpDirection);
        ui->widgetVisualization_2->cameraRightDirection.normalize();
        ui->widgetVisualization_2->repaint();
    }
}

void Widget::HandleWidgetVisualization2CameraUpdated()
{
    camPosLLE = analyzer2.convertXYZtoLLE(ui->widgetVisualization_2->
    cameraPosition);
    camLookAtLLE = analyzer2.convertXYZtoLLE(ui->widgetVisualization_2->
    cameraLookAt);
    if (ui->tabWidget->isTabEnabled(1))
    {

ui->widgetVisualization_1->cameraPosition = analyzer1.convertLLEtoXYZ(camPosLLE);
ui->widgetVisualization_1->cameraUpDirection = QVector3D(0,0,1);
ui->widgetVisualization_1->cameraUpDirection.normalize();
ui->widgetVisualization_1->cameraLookAt = analyzer1.convertLLEtoXYZ(camLookAtLLE);
ui->widgetVisualization_1->cameraForwardDirection = ui->widgetVisualization_1->cameraLookAt - ui->widgetVisualization_1->cameraPosition;
ui->widgetVisualization_1->cameraForwardDirection.normalize();
ui->widgetVisualization_1->cameraRightDirection = QVector3D::crossProduct(ui->widgetVisualization_1->cameraForwardDirection, ui->widgetVisualization_1->cameraUpDirection);
ui->widgetVisualization_1->cameraRightDirection.normalize();
ui->widgetVisualization_1->repaint();
}

void Widget::HandleThreadCalculationsEyepointProjectionsReady()
{
  if (QObject::sender() == TDB1CalcThread1)
  {
    isTDB1CalcThread1Ready = true;
  }
  if (QObject::sender() == TDB1CalcThread2)
  {
    isTDB1CalcThread2Ready = true;
  }
  if (QObject::sender() == TDB2CalcThread1)
  {
    isTDB2CalcThread1Ready = true;
  }
  if (QObject::sender() == TDB2CalcThread2)
  {
    isTDB2CalcThread2Ready = true;
  }
  if (isTDB1CalcThread1Ready && isTDB1CalcThread2Ready)
  {
    if (!ui->checkBoxDrawEyepoints1->isEnabled()) && (!ui->checkBoxDrawGroundEyepoints1->isEnabled())
    {
      endSetEyepointsTDB1 = QDateTime::currentDateTime();
      int probeCount = analyzer1.terrainBlockList.size() * analyzer1.terrainBlockList[0].eyePointSkyXYZList.size();
      qDebug("nTDB1 - probe terrain finished at:");
      qDebug() << endSetEyepointsTDB1;
      qDebug("Total of polygons in the TDB1 (LOD %d): %d",
             analyzer1.LODToBeAnalyzed,
             analyzer1.LODList[analyzer1.LODToBeAnalyzed].polygons.size());
      qDebug("Time spent probing the TDB1: %d min %d sec",
             startSetEyepointsTDB1.secsTo(endSetEyepointsTDB1)/60,
             startSetEyepointsTDB1.secsTo(endSetEyepointsTDB1)%60,
             startSetEyepointsTDB1.msecsTo(endSetEyepointsTDB1));
  }
}
qDebug("Number of terrain probes: %d", probeCount);
qDebug("Probes per second: %f", ((double)(probeCount) / ((double)
startSetEyepointsTDB1.msecsTo(endSetEyepointsTDB1)) * 1000.0));
qDebug("Seconds per probe: %f", ((double)(startSetEyepointsTDB1.msecsTo(endSetEyepointsTDB1)) / ((double)
probeCount * 1000.0)));
ui->frame_7->setVisible(false);
ui->widgetVisualization_1->repaint();
ui->checkBoxDrawEyepoints1->setEnabled(true);
ui->checkBoxDrawGroundEyepoints1->setEnabled(true);
}
}
if (isTDB2CalcThread1Ready && isTDB2CalcThread2Ready)
{
    if (!((ui->checkBoxDrawEyepoints2->isEnabled()) && (ui->
        checkBoxDrawGroundEyepoints2->isEnabled())))
    {
        endSetEyepointsTDB2 = QDateTime::currentDateTime();
        int probeCount = analyzer2.terrainBlockList.size() *
analyzer2.terrainBlockList[0].eyePointSkyXYZList.size();
        qDebug("\nTDB2 - probe terrain finished at:");
        qDebug() << endSetEyepointsTDB2;
        qDebug("Total of polygons in the TDB2 (LOD %d): %d",
analyzer2.LODToBeAnalyzed,
analyzer2.LODList[analyzer2.LODToBeAnalyzed].polygons.size());
        qDebug("Time spent probing the TDB2: %d min %d.%d sec",
startSetEyepointsTDB2.secsTo(endSetEyepointsTDB2)/60,
startSetEyepointsTDB2.secsTo(endSetEyepointsTDB2)%60,
startSetEyepointsTDB2.msecsTo(endSetEyepointsTDB2));
        qDebug("Number of terrain probes: %d", probeCount);
        qDebug("Probes per second: %f", ((double)(probeCount) / ((double)
startSetEyepointsTDB2.msecsTo(endSetEyepointsTDB2)) * 1000.0));
        qDebug("Seconds per probe: %f", ((double)(startSetEyepointsTDB2.msecsTo(endSetEyepointsTDB2)) / ((double)
probeCount * 1000.0)));
        ui->frame_9->setVisible(false);
        ui->widgetVisualization_2->repaint();
        ui->checkBoxDrawEyepoints2->setEnabled(true);
        ui->checkBoxDrawGroundEyepoints2->setEnabled(true);
    }
}
if (isTDB1CalcThread1Ready && isTDB1CalcThread2Ready &&
isTDB2CalcThread1Ready && isTDB2CalcThread2Ready)
{
    if (endSetEyepointsTDB2 > endSetEyepointsTDB1)
    {
        qDebug("\nTotal time spent probing the databases: %d min %d.%d
sec", startSetEyepointsTDB1.secsTo(endSetEyepointsTDB2)/60,
startSetEyepointsTDB1.secsTo(endSetEyepointsTDB2)%60,
startSetEyepointsTDB1.msecsTo(endSetEyepointsTDB2));
    }
    else
    {

```cpp
qDebug("\nTotal time spent probing the databases: %d min %d.%d sec", startSetEyepointsTDB1.secsTo(endSetEyepointsTDB1)/60,
startSetEyepointsTDB1.secsTo(endSetEyepointsTDB1)%60,
startSetEyepointsTDB1.msecsTo(endSetEyepointsTDB1));
}
msgBoxInfoWarn.setText("The latitude and longitude of the eye-points in each block were set.");
msgBoxInfoWarn.setIcon(QMessageBox::Information);
msgBoxInfoWarn.exec();
ui->frame_3->setEnabled(true);
}
void Widget::HandleThreadCalculationsLOSReady()
{
    if (QObject::sender() == TDB1CalcThread1)
    {
        isTDB1CalcThread1Ready = true;
    }
    if (QObject::sender() == TDB1CalcThread2)
    {
        isTDB1CalcThread2Ready = true;
    }
    if (QObject::sender() == TDB2CalcThread1)
    {
        isTDB2CalcThread1Ready = true;
    }
    if (QObject::sender() == TDB2CalcThread2)
    {
        isTDB2CalcThread2Ready = true;
    }
    if (isTDB1CalcThread1Ready && isTDB1CalcThread2Ready)
    {
        if (!ui->checkBoxDrawLOSRays1->isEnabled())
        {
            endLOSTDB1 = QDateTime::currentDateTime();
            qDebug("\nTDB1 - LOS tests finished at: ");
            qDebug() << endLOSTDB1;
            int LOSTestsPerLatLongCount, eyepointGroundProjectionsCount,
            totalLOSTests;
            LOSTestsPerLatLongCount = (ui->spinBoxHeightIncCount->value() + 1) * (ui->spinBoxAzimuthCount->value()) * (2 * (ui->spinBoxPitchCount->value() + 1));
            eyepointGroundProjectionsCount = ui->spinBoxLines->value() * ui->spinBoxColumns->value() * ui->comboBoxEyePointsNumber->currentText().toInt();
            totalLOSTests = LOSTestsPerLatLongCount * eyepointGroundProjectionsCount;
            qDebug("Time spent doing LOS tests: %d min %d.%d sec", startLOSTDB1.secsTo(endLOSTDB1)/60, startLOSTDB1.secsTo(endLOSTDB1)%60,
            startLOSTDB1.msecsTo(endLOSTDB1));
            qDebug("Number of LOS tests: %d", totalLOSTests);
```
qDebug(QString("Total of polygons in the TDB1 (LOD %d): %d"),
analyzer1.LODToBeAnalyzed,
analyzer1.LODList[analyzer1.LODToBeAnalyzed].polygons.size());
qDebug(QString("LOS tests per second: %f", ((double)(totalLOSTests) / ((double)(startLOSTDB1).msecsTo(endLOSTDB1))) * 1000.0));
qDebug(QString("Seconds per LOS test: %f", ((double)(startLOSTDB1).msecsTo(endLOSTDB1)) / ((double)(totalLOSTests * 1000.0))));
ui->widgetVisualization_1->repaint();
ui->checkBoxDrawLOSRays1->setEnabled(true);
ui->frame_7->setVisible(false);
}
}
if (isTDB2CalcThread1Ready && isTDB2CalcThread2Ready)
{
  if (!ui->checkBoxDrawLOSRays2->isEnabled())
  {
    endLOSTDB2 = QDateTime::currentDateTime();
    qDebug(QString("\nTDB2 - LOS tests finished at:"));
    int LOSTestsPerLatLongCount,
eyepointGroundProjectionsCount;
    startLOSTDB2 = endLOSTDB2;
    int totalLOSTests;
    totalLOSTests = LOSTestsPerLatLongCount * eyepointGroundProjectionsCount;
    qDebug(QString("Time spent doing LOS tests: %d min %d.%d sec", startLOSTDB2.secsTo(endLOSTDB2)/60, startLOSTDB2.secsTo(endLOSTDB2)%60, startLOSTDB1.msecsTo(endLOSTDB2));
    qDebug(QString("Number of LOS tests: %d", totalLOSTests));
    qDebug(QString("Total of polygons in the TDB2 (LOD %d): %d",alyzer2.LODToBeAnalyzed,
analyzer2.LODList[analyzer2.LODToBeAnalyzed].polygons.size());
qDebug(QString("LOS tests per second: %f", ((double)(totalLOSTests) / ((double)(startLOSTDB2).msecsTo(endLOSTDB2))) * 1000.0));
qDebug(QString("Seconds per LOS test: %f", ((double)(startLOSTDB2).msecsTo(endLOSTDB2)) / ((double)(totalLOSTests * 1000.0))));
ui->widgetVisualization_2->repaint();
ui->checkBoxDrawLOSRays2->setEnabled(true);
ui->frame_9->setVisible(false);
}
}
if (isTDB1CalcThread1Ready && isTDB1CalcThread2Ready && isTDB2CalcThread1Ready && isTDB2CalcThread2Ready)
{
  if (endLOSTDB2 > endLOSTDB1)
  {

```cpp
qDebug("Total time spent calculating LOS: %d min %d.%d sec", startLOSTDB1 secsTo (endLOSTDB2) / 60, startLOSTDB1 secsTo (endLOSTDB2) % 60, startLOSTDB1 msecsTo (endLOSTDB2));
}
else
{
    qDebug("Total time spent calculating LOS: %d min %d.%d sec", startLOSTDB1 secsTo (endLOSTDB1) / 60, startLOSTDB1 secsTo (endLOSTDB1) % 60, startLOSTDB1 msecsTo (endLOSTDB1));
    compareTDBObj = new TDBComparison (this, &analyzer1, &analyzer2);
    compareTDBObj->blockColumnCount = ui->spinBoxColumns->value();
    compareTDBObj->blockRowCount = ui->spinBoxLines->value();
    compareTDBObj->eyePointsOrthoProjPerBlock = ui->comboBoxEyePointsNumber->currentText().toInt();
    compareTDBObj->eyePointsPerOrthoProj = ui->spinBoxHeightIncCount->value() + 1;
    compareTDBObj->rayAzimuthVarCount = ui->spinBoxAzimuthCount->value();
    compareTDBObj->rayPitchVarCount = 2 * (ui->spinBoxPitchCount->value()) + 1;
    compareTDBObj->rearrangeData();
    ui->checkBoxDrawLOSAbsDiffPosts1->setEnabled (true);
    ui->checkBoxDrawLOSAbsDiffPosts2->setEnabled (true);
    ui->checkBoxDrawElevAbsDiffPosts1->setEnabled (true);
    ui->checkBoxDrawElevAbsDiffPosts2->setEnabled (true);
    ui->frame_10->setEnabled (true);
    ui->frame_11->setEnabled (true);
    ui->spinBoxTopUncorrelatedRays1->setMinimum (0);
    ui->spinBoxTopUncorrelatedRays1->setMaximum (compareTDBObj->comparData.LOSRayDiffUnityOrderedList.size());
    ui->spinBoxTopUncorrelatedRays2->setMinimum (0);
    ui->spinBoxTopUncorrelatedRays2->setMaximum (compareTDBObj->comparData.LOSRayDiffUnityOrderedList.size());
    msgBoxInfoWarn.setText ("The LOS tests are concluded.");
    msgBoxInfoWarn.setIcon (QMessageBox::Information);
    msgBoxInfoWarn.exec();
    ui->pushButtonClearSettings->setEnabled (true);
    ui->pushButtonSaveLOSTestsResults->setEnabled (true);
    ui->frame_6->repaint();
}
}
void Widget::on_checkBoxDrawOriginAxes1_clicked()
{
    if (ui->checkBoxDrawOriginAxes1->isChecked())
    {
        widgetVisualization_1->enableDrawOriginAxes = true;
        ui->widgetVisualization_1->repaint();
    }
    else
    {
```
ui->widgetVisualization_1->enableDrawOriginAxes = false;
ui->widgetVisualization_1->repaint();
}

void Widget::on_checkBoxDrawOriginAxes2_clicked()
{
    if (ui->checkBoxDrawOriginAxes2->isChecked())
    {
        ui->widgetVisualization_2->enableDrawOriginAxes = true;
        ui->widgetVisualization_2->repaint();
    }
    else
    {
        ui->widgetVisualization_2->enableDrawOriginAxes = false;
        ui->widgetVisualization_2->repaint();
    }
}

void Widget::on_checkBoxDrawLOSAbsDiffPosts1_clicked()
{
    if (ui->checkBoxDrawLOSAbsDiffPosts1->isChecked())
    {
        double DBmaxXRange, DBmaxYRange, DBmaxZRange, DBmaxRange;
        double maxBlockLOSRayAbsDiffAvg, minBlockLOSRayAbsDiffAvg,
        rangeBlockLOSRayAbsDiffAvg;
        DBmaxXRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxX -
        analyzer1.LODList[analyzer1.LODToBeAnalyzed].minX;
        DBmaxYRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxY -
        analyzer1.LODList[analyzer1.LODToBeAnalyzed].minY;
        DBmaxZRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxZ -
        analyzer1.LODList[analyzer1.LODToBeAnalyzed].minZ;
        DBmaxRange = sqrt(pow(DBmaxXRange, 2) + pow(DBmaxYRange, 2) +
        pow(DBmaxZRange, 2));

        for(int i = 0; i < compareTDBObject-
        >comparData.blockAnalysisData.size(); i++)
        {
            double centerX = (analyzer1.terrainBlockList[i].maxX +
            analyzer1.terrainBlockList[i].minX)/2;
            double centerY = (analyzer1.terrainBlockList[i].maxY +
            analyzer1.terrainBlockList[i].minY)/2;
            double centerZ =
            analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxZ;
            double currentLOSRayAbsDiffAvg = compareTDBObject-
            >comparData.blockAnalysisData[i].blockLOSRayAbsDiffAvg;
            QVector3D postPos(centerX, centerY, centerZ);
            ui->widgetVisualization_1->postPositionLOS.push_back(postPos);
            if (i == 0)
            {
                maxBlockLOSRayAbsDiffAvg = currentLOSRayAbsDiffAvg;
                minBlockLOSRayAbsDiffAvg = currentLOSRayAbsDiffAvg;
            }
maxBlockLOSRayAbsDiffAvg = (maxBlockLOSRayAbsDiffAvg < currentLOSRayAbsDiffAvg) ? currentLOSRayAbsDiffAvg : maxBlockLOSRayAbsDiffAvg;
minBlockLOSRayAbsDiffAvg = (minBlockLOSRayAbsDiffAvg > currentLOSRayAbsDiffAvg) ? currentLOSRayAbsDiffAvg : minBlockLOSRayAbsDiffAvg;
}

rangeBlockLOSRayAbsDiffAvg = maxBlockLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg;
for (int i = 0; i < compareTDBObj->comparData.blockAnalysisData.size(); i++)
{
    double currentLOSRayAbsDiffAvg = compareTDBObj->comparData.blockAnalysisData[i].blockLOSRayAbsDiffAvg;
    ui->widgetVisualization_1->postHeightLOS.push_back((DBmaxRange/1000.0) * currentLOSRayAbsDiffAvg);
    double redIntensity, greenIntensity;
    redIntensity = (currentLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg) / rangeBlockLOSRayAbsDiffAvg;
    greenIntensity = abs(1 - ((currentLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg) / rangeBlockLOSRayAbsDiffAvg));
    ui->widgetVisualization_1->postColorLOS.push_back(QVector3D(redIntensity, greenIntensity, 0));
}
ui->widgetVisualization_1->enableDrawPostsLOS = true;
ui->widgetVisualization_1->repaint();
else
{
    ui->widgetVisualization_1->enableDrawPostsLOS = false;
    ui->widgetVisualization_1->repaint();
    ui->widgetVisualization_1->postPositionLOS.clear();
    ui->widgetVisualization_1->postColorLOS.clear();
    ui->widgetVisualization_1->postHeightLOS.clear();
}

void Widget::on_checkBoxDrawLOSAbsDiffPosts2_clicked()
{
    if (ui->checkBoxDrawLOSAbsDiffPosts2->isChecked())
    {
        double DBmaxXRange, DBmaxYRange, DBmaxZRange, DBmaxRange;
        double maxBlockLOSRayAbsDiffAvg, minBlockLOSRayAbsDiffAvg,
        rangeBlockLOSRayAbsDiffAvg;
        DBmaxXRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxX - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minX;
        DBmaxYRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxY - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minY;
        DBmaxZRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxZ - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minZ;
        DBmaxRange = sqrt(pow(DBmaxXRange, 2) + pow(DBmaxYRange, 2) + pow(DBmaxZRange, 2));
    }
}
for(int i = 0; i < compareTDBObj->comparData.blockAnalysisData.size(); i++)
{
    double centerX = (analyzer2.terrainBlockList[i].maxX + analyzer2.terrainBlockList[i].minX) / 2;
    double centerY = (analyzer2.terrainBlockList[i].maxY + analyzer2.terrainBlockList[i].minY) / 2;
    double centerZ = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxZ;
    double currentLOSRayAbsDiffAvg = compareTDBObj->comparData.blockAnalysisData[i].blockLOSRayAbsDiffAvg;
    QVector3D postPos(centerX, centerY, centerZ);
    ui->widgetVisualization_2->postPositionLOS.push_back(postPos);

    if (i == 0)
    {
        maxBlockLOSRayAbsDiffAvg = currentLOSRayAbsDiffAvg;
        minBlockLOSRayAbsDiffAvg = currentLOSRayAbsDiffAvg;
    } else
    {
        maxBlockLOSRayAbsDiffAvg = (maxBlockLOSRayAbsDiffAvg < currentLOSRayAbsDiffAvg) ? currentLOSRayAbsDiffAvg : maxBlockLOSRayAbsDiffAvg;
        minBlockLOSRayAbsDiffAvg = (minBlockLOSRayAbsDiffAvg > currentLOSRayAbsDiffAvg) ? currentLOSRayAbsDiffAvg : minBlockLOSRayAbsDiffAvg;
    }

    rangeBlockLOSRayAbsDiffAvg = maxBlockLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg;
    for(int i = 0; i < compareTDBObj->comparData.blockAnalysisData.size(); i++)
    {
        double currentLOSRayAbsDiffAvg = compareTDBObj->comparData.blockAnalysisData[i].blockLOSRayAbsDiffAvg;
        QVector3D postPos((DBmaxRange/1000.0) * currentLOSRayAbsDiffAvg);
        ui->widgetVisualization_2->postHeightLOS.push_back(postPos);
        redIntensity, greenIntensity;
        redIntensity = ((currentLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg) / rangeBlockLOSRayAbsDiffAvg);
        greenIntensity = abs(1 - ((currentLOSRayAbsDiffAvg - minBlockLOSRayAbsDiffAvg) / rangeBlockLOSRayAbsDiffAvg));
        ui->widgetVisualization_2->postColorLOS.push_back(QVector3D(redIntensity, greenIntensity, 0));

        ui->widgetVisualization_2->enableDrawBaseSideLenghtLOS = (analyzer2.terrainBlockList[0].maxX - analyzer2.terrainBlockList[0].minX) / 15.0;

        ui->widgetVisualization_2->enableDrawPostsLOS = true;
    }
else
{
    ui->widgetVisualization_2->enableDrawPostsLOS = false;
    ui->widgetVisualization_2->repaint();
    ui->widgetVisualization_2->postPositionLOS.clear();
    ui->widgetVisualization_2->postColorLOS.clear();
}
void Widget::on_checkBoxDrawElevAbsDiffPosts1_clicked()
{
    if (ui->checkBoxDrawElevAbsDiffPosts1->isChecked())
    {
        double DBmaxXRange, DBmaxYRange, DBmaxZRange, DBmaxRange;
        double maxBlockElevAbsDiffAvg, minBlockElevAbsDiffAvg,
            rangeBlockElevAbsDiffAvg;
        DBmaxXRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxX -
            analyzer1.LODList[analyzer1.LODToBeAnalyzed].minX;
        DBmaxYRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxY -
            analyzer1.LODList[analyzer1.LODToBeAnalyzed].minY;
        DBmaxZRange = analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxZ -
            analyzer1.LODList[analyzer1.LODToBeAnalyzed].minZ;
        DBmaxRange = sqrt(pow(DBmaxXRange, 2) + pow(DBmaxYRange, 2) +
            pow(DBmaxZRange, 2));

        for (int i = 0; i < compareTDBObj-
            >comparData.blockAnalysisData.size(); i++)
            {
            double centerX = (analyzer1.terrainBlockList[i].maxX +
                analyzer1.terrainBlockList[i].minX)/2;
            double centerY = (analyzer1.terrainBlockList[i].maxY +
                analyzer1.terrainBlockList[i].minY)/2;
            double centerZ =
                analyzer1.LODList[analyzer1.LODToBeAnalyzed].maxZ;
            double currentElevAbsDiffAvg = compareTDBObj-
                >comparData.blockAnalysisData[i].blockElevAbsDiffAvg;
            QVector3D postPos(centerX, centerY, centerZ);
            ui->widgetVisualization_1->postPositionElev.push_back(postPos);
            if (i == 0)
            {
                maxBlockElevAbsDiffAvg = currentElevAbsDiffAvg;
                minBlockElevAbsDiffAvg = currentElevAbsDiffAvg;
            }
            maxBlockElevAbsDiffAvg = (maxBlockElevAbsDiffAvg <
                currentElevAbsDiffAvg) ? currentElevAbsDiffAvg : maxBlockElevAbsDiffAvg;
            minBlockElevAbsDiffAvg = (minBlockElevAbsDiffAvg >
                currentElevAbsDiffAvg) ? currentElevAbsDiffAvg : minBlockElevAbsDiffAvg;
            }
            rangeBlockElevAbsDiffAvg = maxBlockElevAbsDiffAvg -
                minBlockElevAbsDiffAvg;
            for (int i = 0; i < compareTDBObj-
                >comparData.blockAnalysisData.size(); i++)
                {
                double currentElevAbsDiffAvg = compareTDBObj-
                    >comparData.blockAnalysisData[i].blockElevAbsDiffAvg;
                ui->widgetVisualization_1-
                    >postHeightElev.push_back((DBmaxRange/1000.0) * currentElevAbsDiffAvg);
                double redIntensity, greenIntensity;
                redIntensity = (currentElevAbsDiffAvg - minBlockElevAbsDiffAvg) /
                    rangeBlockElevAbsDiffAvg;
            ui->widgetVisualization_2->postHeightLOS.clear();
        }
greenIntensity = abs(1 - ((currentElevAbsDiffAvg - minBlockElevAbsDiffAvg) / rangeBlockElevAbsDiffAvg));
ui->widgetVisualization_1->postColorElev.push_back(QVector3D(redIntensity, greenIntensity, 0));
ui->widgetVisualization_1->enableDrawPostsElev = true;
ui->widgetVisualization_1->repaint();
}
else
{
ui->widgetVisualization_1->enableDrawPostsElev = false;
ui->widgetVisualization_1->repaint();
ui->widgetVisualization_1->postPositionElev->clear();
ui->widgetVisualization_1->postColorElev->clear();
ui->widgetVisualization_1->postHeightElev->clear();
}
}

void Widget::on_checkBoxDrawElevAbsDiffPosts2_clicked()
{
    if (ui->checkBoxDrawElevAbsDiffPosts2->isChecked())
    {
        double DBmaxXRange, DBmaxYRange, DBmaxZRange, DBmaxRange;
        double maxBlockElevAbsDiffAvg, minBlockElevAbsDiffAvg, rangeBlockElevAbsDiffAvg;
        DBmaxXRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxX - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minX;
        DBmaxYRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxY - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minY;
        DBmaxZRange = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxZ - analyzer2.LODList[analyzer2.LODToBeAnalyzed].minZ;
        DBmaxRange = sqrt(pow(DBmaxXRange, 2) + pow(DBmaxYRange, 2) + pow(DBmaxZRange, 2));
        for(int i = 0; i < compareTDBObj->comparData.blockAnalysisData.size(); i++)
        {
            double centerX = (analyzer2.terrainBlockList[i].maxX + analyzer2.terrainBlockList[i].minX)/2;
            double centerY = (analyzer2.terrainBlockList[i].maxY + analyzer2.terrainBlockList[i].minY)/2;
            double centerZ = analyzer2.LODList[analyzer2.LODToBeAnalyzed].maxZ;
            double currentElevAbsDiffAvg = compareTDBObj->
                >comparData.blockAnalysisData[i].blockElevAbsDiffAvg;
            QVector3D postPos(centerX, centerY, centerZ);
            ui->widgetVisualization_2->postPositionElev->push_back(postPos);
            if (i == 0)
            {
                maxBlockElevAbsDiffAvg = currentElevAbsDiffAvg;
                minBlockElevAbsDiffAvg = currentElevAbsDiffAvg;
            }
        }
    }
}

maxBlockElevAbsDiffAvg = (maxBlockElevAbsDiffAvg < currentElevAbsDiffAvg) ? currentElevAbsDiffAvg : maxBlockElevAbsDiffAvg;
minBlockElevAbsDiffAvg = (minBlockElevAbsDiffAvg > currentElevAbsDiffAvg) ? currentElevAbsDiffAvg : minBlockElevAbsDiffAvg;

rangeBlockElevAbsDiffAvg = maxBlockElevAbsDiffAvg - minBlockElevAbsDiffAvg;

for(int i = 0; i < compareTDBObj->comparData.blockAnalysisData.size(); i++)
{
  double currentElevAbsDiffAvg = compareTDBObj->comparData.blockAnalysisData[i].blockElevAbsDiffAvg;
  ui->widgetVisualization_2->postHeightElev.push_back((DBmaxRange/1000.0) * currentElevAbsDiffAvg);
  double redIntensity, greenIntensity;
  redIntensity = (currentElevAbsDiffAvg - minBlockElevAbsDiffAvg) / rangeBlockElevAbsDiffAvg;
  greenIntensity = abs(1 - ((currentElevAbsDiffAvg - minBlockElevAbsDiffAvg) / rangeBlockElevAbsDiffAvg));
  ui->widgetVisualization_2->postColorElev.push_back(QVector3D(redIntensity, greenIntensity, 0));
  ui->widgetVisualization_2->enableDrawPostsElev = true;
  ui->widgetVisualization_2->repaint();
} else
{
  ui->widgetVisualization_2->enableDrawPostsElev = false;
  ui->widgetVisualization_2->repaint();
  ui->widgetVisualization_2->postPositionElev.clear();
  ui->widgetVisualization_2->postColorElev.clear();
  ui->widgetVisualization_2->postHeightElev.clear();
}

void Widget::on_checkBoxDrawTopUncorrelatedRays1_clicked()
{
  if(ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
  {
    if(ui->checkBoxDrawLOSrays1->isChecked())
    { ui->checkBoxDrawLOSrays1->setChecked(false);
      ClearLOSraysVisWidget1();
    }
    if(ui->checkBoxDrawLOSrays2->isChecked())
    { ui->checkBoxDrawLOSrays2->setChecked(false);
      ClearLOSraysVisWidget2();
    }
    if(!ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
    { ui->checkBoxDrawTopUncorrelatedRays2->setChecked(false);

```
{ ui->checkBoxDrawTopUncorrelatedRays2->setChecked(true);
on_checkBoxDrawTopUncorrelatedRays2_clicked();
}

on_checkBoxDrawTopUncorrEyepointsOnly1_clicked();
for (int i = 0; i < ui->spinBoxTopUncorrelatedRays1->value(); i++)
{
    oyLOSRayDiffUnity currentDiffUnity;
    QVector3D eyePos, intersectPt;
    currentDiffUnity = compareTDBObj-
>compData.LOSRayDiffUnityOrderedList[i];
    eyePos = compareTDBObj-
>compData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.eyePosXYZ;
    intersectPt = compareTDBObj-
>compData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.intersectPtXYZ;
    ui->widgetVisualization_1->LOS RaysVertices.push_back(eyePos);
    ui->widgetVisualization_1-
>LOS RaysColors.push_back(QVector3D(0.5, 0, 0.65));
    ui->widgetVisualization_1-
>LOS RaysVertices.push_back(intersectPt);
    ui->widgetVisualization_1-
>LOS RaysColors.push_back(QVector3D(0.5, 0, 0.65));
    ui->widgetVisualization_1-
>actualEyepointsVertices.push_back(eyePos);
    ui->widgetVisualization_1-
>actualEyepointsColors.push_back(QVector3D(0.219, 0.49, 0.843));
    ui->widgetVisualization_1-
>LOS IntersectionVertices.push_back(intersectPt);
    ui->widgetVisualization_1-
>LOS IntersectionColors.push_back(QVector3D(1, 0, 0));
    ui->widgetVisualization_1->enableDrawLOS Rays = true;
    ui->widgetVisualization_1->repaint();
}
else
{
    if(ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
    {
        ui->checkBoxDrawTopUncorrelatedRays2->setChecked(false);
        on_checkBoxDrawTopUncorrelatedRays2_clicked();
    }
    ClearLOS RaysVisWidget1();
}

void Widget::on_checkBoxDrawTopUncorrelatedRays2_clicked()
{
    if(ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
    {
        if(ui->checkBoxDrawLOS Rays1->isChecked())
        {
            ui->checkBoxDrawLOS Rays1->setChecked(false);
            ClearLOS RaysVisWidget1();
        }
    }
if (ui->checkBoxDrawLOS Rays2->isChecked())
{
    ui->checkBoxDrawLOS Rays2->setChecked(false);
    ClearLOS RaysVisWidget2();
}
if (!ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
{
    ui->checkBoxDrawTopUncorrelatedRays1->setChecked(true);
    on_checkBoxDrawTopUncorrelatedRays1_clicked();
}
on_checkBoxDrawTopUncorrEyepointsOnly2_clicked();
for (int i = 0; i < ui->spinBoxTopUncorrelatedRays2->value(); i++)
{
    OyLOS RayDiffUnity currentDiffUnity;
    QVector3D eyePos, intersectPt;
    currentDiffUnity = compareTDBObj->comparData. LOS RayDiffUnityOrderedList[i];
    eyePos = compareTDBObj->comparData. LOS RayDiffUnityOrderedList[i].TDB2RayXYZ. eyePosXYZ;
    intersectPt = compareTDBObj->comparData. LOS RayDiffUnityOrderedList[i].TDB2RayXYZ. intersecPtXYZ;
    ui->widgetVisualization_2->LOS RaysVertices.push_back(eyePos);
    ui->widgetVisualization_2->LOS RaysColors.push_back(QVector3D(0.5, 0, 0.65));
    ui->widgetVisualization_2->LOS intersectionVertices.push_back(intersectPt);
    ui->widgetVisualization_2->LOS intersectionColors.push_back(QVector3D(1.0, 0, 0));
    }
ui->widgetVisualization_2->enableDrawLOS Rays = true;
ui->widgetVisualization_2->repaint();
}
else
{
    if (ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
    {
        ui->checkBoxDrawTopUncorrelatedRays1->setChecked(false);
        on_checkBoxDrawTopUncorrelatedRays1_clicked();
    }
    ClearLOS RaysVisWidget2();
}

void Widget::on_spinBoxTopUncorrelatedRays1_valueChanged(int arg1)
{
void Widget::on_spinBoxTopUncorrelatedRays2_valueChanged(int arg1)
{
    ui->spinBoxTopUncorrelatedRays1->setValue(arg1);
    if (ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
    {
        DrawTopUncorrRaysVisWidgets1();
        DrawTopUncorrRaysVisWidgets2();
    }
}

void Widget::on_spinBoxTopUncorrelatedRays1_valueChanged(int arg1)
{
    ui->spinBoxTopUncorrelatedRays2->setValue(arg1);
    if (ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
    {
        DrawTopUncorrRaysVisWidgets1();
        DrawTopUncorrRaysVisWidgets2();
    }
}

void Widget::DrawTopUncorrRaysVisWidgets1()
{
    ClearLOSRaysVisWidget1();
    if (compareTDBObj)
    {
        for (int i = 0; i < ui->spinBoxTopUncorrelatedRays1->value(); i++)
        {
            oYLOSRayDiffUnity currentDiffUnity;
            QVector3D eyePos, intersectPt;
            currentDiffUnity = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.eyeposXYZ;
            eyePos = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.eyeposXYZ;
            intersectPt = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.intersectptXYZ;
            ui->widgetVisualization_1->LOSraysVertices.push_back(eyePos);
            ui->widgetVisualization_1->LOSraysColors.push_back(QVector3D(0.5, 0, 0.65));
            ui->widgetVisualization_1->LOSraysVertices.push_back(intersectPt);
            ui->widgetVisualization_1->LOSraysColors.push_back(QVector3D(0.5, 0, 0.65));
            ui->widgetVisualization_1->actualEyepointsVertices.push_back(eyePos);
            ui->widgetVisualization_1->actualEyepointsColors.push_back(QVector3D(0.219, 0.49, 0.843));
            ui->widgetVisualization_1->LOSIntersectionVertices.push_back(intersectPt);
            ui->widgetVisualization_1->LOSIntersectionColors.push_back(QVector3D(1, 0, 0));
        }
        ui->widgetVisualization_1->enableDrawLOSrays = true;
        ui->widgetVisualization_1->repaint();
    }
    else
    {
        // Handle the case when compareTDBObj is false...
    }
}
```cpp
void Widget::DrawTopUncorrRaysVisWidgets2()
{
    ClearLOSRaysVisWidget2();
    if (compareTDBObj)
    {
        for (int i = 0; i < ui->spinBoxTopUncorrelatedRays1->value(); i++)
        {
            oyLOSRayDiffUnity currentDiffUnity;
            QVector3D eyePos, intersectPt;
            currentDiffUnity = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB2RayXYZ.eyeposXYZ;
            eyePos = compareTDBObj->comparData.LOSRayDiffUnityOrderedList[i].TDB2RayXYZ.intersectPtXYZ;
            ui->widgetVisualization_2->LOSRaysVertices.push_back(eyePos);
            ui->widgetVisualization_2->LOSColors.push_back(QVector3D(0.5, 0, 0.65));
            ui->widgetVisualization_2->actualEyepointsVertices.push_back(eyePos);
            ui->widgetVisualization_2->actualEyepointsColors.push_back(QVector3D(0.219, 0.49, 0.843));
            ui->widgetVisualization_2->LOSIntersectionVertices.push_back(intersectPt);
            ui->widgetVisualization_2->LOSIntersectionColors.push_back(QVector3D(1, 0, 0));
        }
        ui->widgetVisualization_2->enableDrawLOSRays = true;
    } else
    {
        qDebug("compareTDBObj was deleted.");
        ui->widgetVisualization_2->enableDrawLOSRays = false;
        ui->widgetVisualization_2->repaint();
    }
}

void Widget::ClearLOSRaysVisWidget1()
{
    ui->widgetVisualization_1->enableDrawLOSRays = false;
    ui->widgetVisualization_1->enableDrawProblematicRays = false;
    ui->widgetVisualization_1->repaint();
    ui->widgetVisualization_1->LOSVertices.clear();
}
```
void Widget::ClearLOSRaysVisWidget2()
{
    ui->widgetVisualization_2->enableDrawLOSRays = false;
    ui->widgetVisualization_2->enableDrawProblematicRays = false;
    ui->widgetVisualization_2->repaint();
    ui->widgetVisualization_2->LOSIntersectionVertices.clear();
    ui->widgetVisualization_2->LOSIntersectionColors.clear();
    ui->widgetVisualization_2->problematicRaysVertices.clear();
    ui->widgetVisualization_2->problematicRaysColors.clear();
}

void Widget::on_checkBoxDrawTopUncorrEyepointsOnly1_clicked()
{
    if(ui->checkBoxDrawTopUncorrelatedRays1->isChecked())
    {
        if(ui->checkBoxDrawTopUncorrEyepointsOnly1->isChecked())
        {
            ui->widgetVisualization_1->enableDrawEyepointsOnly = true;
        }
        else
        {
            ui->widgetVisualization_1->enableDrawEyepointsOnly = false;
        }
        ui->widgetVisualization_1->repaint();
    }
}

void Widget::on_checkBoxDrawTopUncorrEyepointsOnly2_clicked()
{
    if(ui->checkBoxDrawTopUncorrelatedRays2->isChecked())
    {
        if(ui->checkBoxDrawTopUncorrEyepointsOnly2->isChecked())
        {
            ui->widgetVisualization_2->enableDrawEyepointsOnly = true;
        }
        else
        {
            ui->widgetVisualization_2->enableDrawEyepointsOnly = false;
        }
        ui->widgetVisualization_2->repaint();
    }
}
threadtraversescenegraph.h

#ifndef THREADSAVETRAVERSESCENEGRAH_H
#define THREADSAVETRAVERSESCENEGRAH_H
#include <QThread>
#include <QFile>
#include <QTextStream>
#include <QStringList>
#include <QDateTime>
#include <mgapiall.h>
#include <math.h>
#define MAX_LOD 20

struct Vertex{
    double coord[3];
    double lat, lon, height;
    QString nodeName;
    Vertex& operator =(const Vertex& o)
    {
        coord[0] = o.coord[0];
        coord[1] = o.coord[1];
        coord[2] = o.coord[2];
        lat = o.lat;
        lon = o.lon;
        height = o.height;
        nodeName = o.nodeName;
        return *this;
    }
};
struct PolygonInfoLOD {
    double normal_i;
    double normal_j;
    double normal_k;
    int vertexCount;
    QList<Vertex> vertices;
    QString nodeName;
    QString fileName;
    int LOD;
    PolygonInfoLOD& operator =(const PolygonInfoLOD& o)
    {
        normal_i = o.normal_i;
        normal_j = o.normal_j;
        normal_k = o.normal_k;
        vertexCount = o.vertexCount;
        nodeName = o.nodeName;
        fileName = o.fileName;
        LOD = o.LOD;
        vertices = o.vertices;
        return *this;
    }
};
class ThreadTraverseSceneGraph : public QThread
{
    Q_OBJECT
public:
    ThreadTraverseSceneGraph(QObject *Parent, QString completeFileName);
    ~ThreadTraverseSceneGraph();

    void run();

    void ListInstances (mgrec *node);
    void Traverse (mgrec *node, int LOD);
    void quickTraverse (mgrec *node, int LOD);

    QString fullFileName, filePath, fileName, errorString;
    QFile traverseLogFile;
    QTextStream traverseStream;

    bool onlyTriangles;
    bool saveTraverLog;
    int DataBaseOrderNumber;

    bool isOpenFlightFirstCall, isOpenFlightLastCall;

    mgrec *db; //presagis
    mgprojection dbProjection; //presagis

    QList<PolygonInfoLOD> polygonInfoList[MAX_LOD];
    int extRefCount, totalNodeCount, traversedNodeCount, dbMaxLOD;
    int currentPercent, oldPercent;
    int PolygonCountLOD[MAX_LOD];
    double XMeanLOD[MAX_LOD];
    double YMeanLOD[MAX_LOD];
    double ZMeanLOD[MAX_LOD];
    double XNormalizedMeanLOD[MAX_LOD];
    double YNormalizedMeanLOD[MAX_LOD];
    double ZNormalizedMeanLOD[MAX_LOD];
    double XStdDevLOD[MAX_LOD];
    double YStdDevLOD[MAX_LOD];
    double ZStdDevLOD[MAX_LOD];
    double XNormalizedStdDevLOD[MAX_LOD];
    double YNormalizedStdDevLOD[MAX_LOD];
    double ZNormalizedStdDevLOD[MAX_LOD];
    bool valuesMinMaxIntialized[MAX_LOD];
    double maxX[MAX_LOD], minX[MAX_LOD], centerX[MAX_LOD];
    double maxY[MAX_LOD], minY[MAX_LOD], centerY[MAX_LOD];
    double maxZ[MAX_LOD], minZ[MAX_LOD], centerZ[MAX_LOD];
    double maxLat[MAX_LOD], maxLon[MAX_LOD], maxHeight[MAX_LOD];
    double minLat[MAX_LOD], minLon[MAX_LOD], minHeight[MAX_LOD];
}
private:

signals:
  void resultReady();
  void error(QString error);
  void infoReady(QString information);
  void updateProgress(int value);
  void standardOut(QString out);
};

#undef // THREADSAVETRAVERSESCENEGRAPH_H

#include "threadtraversescenegraph.h"

ThreadTraverseSceneGraph::ThreadTraverseSceneGraph(QObject *Parent, QString completeFileName)
  :QThread(Parent)
{
  this->fullFileName = completeFileName;
  QStringList auxStringList;
  auxStringList = fullFileName.split("/", QString::KeepEmptyParts, Qt::CaseSensitive);
  fileName = auxStringList[auxStringList.size()-1];
  this->filePath = ""
  for (int i = 0; i < (auxStringList.size()-1); i++)
  {
    filePath.append(auxStringList[i]);
    filePath.append("/");
  }

  this->errorString = ""
  extRefCount = 0;
  currentPercent = 0;
  oldPercent = 0;
  totalNodeCount = 0;
  traversedNodeCount = 0;
  dbMaxLOD = -1;
  for(int i = 0; i < MAX_LOD; i++) // initialize vectors
  {
    PolygonCountLOD[i] = 0;
    XMeanLOD[i] = 0;
    YMeanLOD[i] = 0;
    ZMeanLOD[i] = 0;
    XStdDevLOD[i] = 0;
    YStdDevLOD[i] = 0;
    ZStdDevLOD[i] = 0;
    valuesMinMaxIntialized[i] = false;
  }
onlyTriangles = true;
saveTraverLog = true;
dataBaseOrderNumber = 0;
}

ThreadTraverseSceneGraph::~ThreadTraverseSceneGraph()
{
}

void ThreadTraverseSceneGraph::run()
{
    QString auxString;
    int argc = 1;
    char *argv;
    argv = (char*) malloc(1024*sizeof(char));

    if ((sizeof(fullFileName.toStdString().c_str()) < 1024) &&
        (sizeof(fullFileName.toStdString().c_str()) > 0))
    {
        strcpy_s(argv, 1024, fullFileName.toStdString().c_str());
    }
    else
    {
        errorString = "Error getting file name."
        emit error(errorString);
        return;
    }

    if (this->isOpenFlightFirstCall)
    {
        mgInit (&argc, &argv);
    }

    if (! (db = mgOpenDb (argv)))
    {
        char msgbuf [1024];
        mgGetLastError (msgbuf, 1024);
        mgExit ();
        errorString = msgbuf;
        emit error(errorString);
    }
    else
    {
        emit infoReady("OpenFlight database opened successfully.");
        dbProjection = mgNewProjection(db);
        auxString = "File name: ";
        auxString.append(argv);
        free(argv);
        emit standardOut(auxString);
        if (saveTraverLog == true)
        {
            traverseLogFile.setFileName("traverse_log.txt");
            if (!traverseLogFile.open(QIODevice::WriteOnly | QIODevice::Text))
            {
                errorString = "Error creating traverse log file."
            }
        }
    }
}
emit error(errorString);
}
else
{
  traverseStream.setDevice(&traverseLogFile);
  traverseStream << "Summary of the file " << fullFileName << 
  "\n";

  QDateTime now;
  int currentLOD = 0;
  mgMostDetail(db);
  do
  {
    auxString = "Performing quick scan. LOD = ";
    auxString.append(QString::number(currentLOD));
    auxString.append(". Current time: ");
    now = QDateTime::currentDateTime();
    auxString.append(now.toString("MMM dd, yyyy. hh:mm:ss.zzz.");
  }
  while (mgLessDetail(db));
  auxString = "Quick scan finished. Current time: ";
  auxString.append(now.toString("MMM dd, yyyy. hh:mm:ss.zzz.");
  emit standardOut(auxString);
  quickTraverse(db, currentLOD);
  currentLOD++;}
} while (mgLessDetail(db));
auxString = "Max LOD found: ";
auxString += QString::number(dbMaxLOD);
auxString = "Number of external references: ";
auxString.append(QString::number(extRefCount));
auxString += QString::number(dbMaxLOD);
auxString = "Number of external references: ";
auxString.append(QString::number(extRefCount));
if (onlyTriangles)
{
    emit standardOut("All the polygons in this database are originales.");
}
else
{
emit standardOut("Some polygons in this database are not originales.");
}
}
else
{
    QDateTime now;
    int currentLOD = 0;
    mgMostDetail(db);
    do
    {
        auxString = "Performing quick scan. LOD = ";
        auxString.append(QString::number(currentLOD));
        auxString.append(". Current time: ");
        now = QDateTime::currentDateTime();
        auxString.append(now.toString("MMM dd, yyyy. hh:mm:ss.zzz."));
        emit standardOut(auxString);
        quickTraverse(db, currentLOD);
        currentLOD++;
    } while (mgLessDetail(db));
    auxString += "Quick scan finished. Current time: ";
    now = QDateTime::currentDateTime();
    auxString.append(now.toString("MMM dd, yyyy. hh:mm:ss.zzz."));
    emit standardOut(auxString);
    auxString = "Max LOD found: ";
    auxString += QString::number(dbMaxLOD);
    emit standardOut(auxString);
    mgMostDetail(db);
    currentLOD = 0;
    do
    {
        Traverse(db, currentLOD);
        currentLOD++;
    } while (mgLessDetail(db));
    traverseLogFile.close();
    auxString = "Traverse log file saved.");
    emit standardOut(auxString);
    auxString = "Number of external references: ");
    auxString.append(QString::number(extRefCount));
    emit standardOut(auxString);
    if (onlyTriangles)
    {
        emit standardOut("All the polygons in this database are originales.");
    }
}
else
{
    emit standardOut("Some polygons in this database are not
triangles.");
}
}
}
mgCloseDb(db);
}
for (int count = 0; count < MAX_LOD; count++)
{
    PolygonCountLOD[count] = polygonInfolList[count].size();
    for (int n = 0; n < PolygonCountLOD[count]; n++)
    {
        XMeanLOD[count] += polygonInfolList[count][n].normal_i;
        YMeanLOD[count] += polygonInfolList[count][n].normal_j;
        ZMeanLOD[count] += polygonInfolList[count][n].normal_k;
    }
}
double normalMeanModule;
for (int count = 0; count < MAX_LOD; count++)
{
    XMeanLOD[count] = XMeanLOD[count] / ((double)PolygonCountLOD[count]);
    YMeanLOD[count] = YMeanLOD[count] / ((double)PolygonCountLOD[count]);
    ZMeanLOD[count] = ZMeanLOD[count] / ((double)PolygonCountLOD[count]);
    normalMeanModule = sqrt(pow(XMeanLOD[count], 2) + pow(YMeanLOD[count], 2) + pow(ZMeanLOD[count], 2));
    XNormalizedMeanLOD[count] = XMeanLOD[count] / normalMeanModule;
    YNormalizedMeanLOD[count] = YMeanLOD[count] / normalMeanModule;
    ZNormalizedMeanLOD[count] = ZMeanLOD[count] / normalMeanModule;
}
for (int count = 0; count < MAX_LOD; count++)
{
    for (int n = 0; n < PolygonCountLOD[count]; n++)
    {
        XStdDevLOD[count] += pow((XMeanLOD[count] -
           polygonInfolList[count][n].normal_i), 2);
        YStdDevLOD[count] += pow((YMeanLOD[count] -
           polygonInfolList[count][n].normal_j), 2);
        ZStdDevLOD[count] += pow((ZMeanLOD[count] -
           polygonInfolList[count][n].normal_k), 2);
    }
}
double normalStdDevModule;
for (int count = 0; count < MAX_LOD; count++)
{
    XStdDevLOD[count] = sqrt(XStdDevLOD[count] / ((double)PolygonCountLOD[count]));
    YStdDevLOD[count] = sqrt(YStdDevLOD[count] / ((double)PolygonCountLOD[count]));
    ZStdDevLOD[count] = sqrt(ZStdDevLOD[count] / ((double)PolygonCountLOD[count]));
}
normalStdDevModule = sqrt(pow(XStdDevLOD[count], 2) +
pow(YStdDevLOD[count], 2) + pow(ZStdDevLOD[count], 2));
XNormalizedStdDevLOD[count] = XStdDevLOD[count] / normalStdDevModule;
YNormalizedStdDevLOD[count] = YStdDevLOD[count] / normalStdDevModule;
ZNormalizedStdDevLOD[count] = ZStdDevLOD[count] / normalStdDevModule;
}
for(int count = 0; count < MAX_LOD; count++)
{
    if (PolygonCountLOD[count] != 0)
    {
        auxString = "\nNumber of polygons at LOD ";
        auxString.append(QString::number(count));
        auxString.append(":
");
        auxString.append(QString::number(PolygonCountLOD[count]));
        emit standardOut(auxString);
        auxString = "Mean (or average) of the normal vectors of the polygons in
the database at LOD ";
        auxString.append(QString::number(count));
        auxString.append(":
");
        auxString.append(QString::number(XMeanLOD[count]));
        auxString.append(",
");
        auxString.append(QString::number(YMeanLOD[count]));
        auxString.append(",
");
        auxString.append(QString::number(ZMeanLOD[count]));
        auxString.append("\n");
        auxString.append("length of the mean vector = ");
        auxString.append(QString::number(sqrt(pow(XMeanLOD[count], 2) +
pow(YMeanLOD[count], 2) + pow(ZMeanLOD[count], 2)))));
        emit standardOut(auxString);
        auxString = "Standard deviation of the normal vectors of the polygons in
the database at LOD ";
        auxString.append(QString::number(count));
        auxString.append(":
");
        auxString.append(QString::number(XStdDevLOD[count]));
        auxString.append(",
");
        auxString.append(QString::number(YStdDevLOD[count]));
        auxString.append(",
");
        auxString.append(QString::number(ZStdDevLOD[count]));
        auxString.append("\n");
        auxString.append("length of the std. dev. vector = ");
        auxString.append(QString::number(sqrt(pow(XStdDevLOD[count], 2) +
pow(YStdDevLOD[count], 2) + pow(ZStdDevLOD[count], 2)))));
        emit standardOut(auxString);
        auxString = "Boundaries:
";
        auxString.append("\nmin X = ");
        auxString.append(QString::number(minX[count], 'f', 6));
        auxString.append("\nmax X = ");
        auxString.append(QString::number(maxX[count], 'f', 6));
        auxString.append("\nmin Y = ");
        auxString.append(QString::number(minY[count], 'f', 6));
        auxString.append("\nmax Y = ");
        auxString.append(QString::number(maxY[count], 'f', 6));
        auxString.append(QString::number(maxZ[count], 'f', 6));
auxString += "\nmin Z = \n";
auxString.append(QString::number(minZ[count], 'f', 6));
auxString += "\nmax Z = \n";
auxString.append(QString::number(maxZ[count], 'f', 6));
emit standardOut(auxString);

centerX[count] = (minX[count] + maxX[count]) / 2;
centerY[count] = (minY[count] + maxY[count]) / 2;
centerZ[count] = (minZ[count] + maxZ[count]) / 2;

auxString = "Min latitude = ";
auxString += QString::number(minLat[count], 'f', 6);
auxString += " degrees.\nMin longitude = ";
auxString += QString::number(minLon[count], 'f', 6);
auxString += " degrees.\nMin elevation = ";
auxString += QString::number(minHeight[count], 'f', 6);
auxString += " meters."
emit standardOut(auxString);

auxString = "Max latitude = ";
auxString += QString::number(maxLat[count], 'f', 6);
auxString += " degrees.\nMax longitude = ";
auxString += QString::number(maxLon[count], 'f', 6);
auxString += " degrees.\nMax elevation = ";
auxString += QString::number(maxHeight[count], 'f', 6);
auxString += " meters.";
emit standardOut(auxString);

if (isOpenFlightLastCall)
{
    mgExit();
}
auxString = "The terrain geometry is loaded into computer memory. You can visualize it on the tab \"Visualize Database \#\"; 
auxString += QString::number(dataBaseOrderNumber);
auxString += "\n"
emit infoReady(auxString);
emit resultReady();

void ThreadTraverseSceneGraph::Traverse (mgrec *node, int LOD)
{
    // traverse down to the vertex level,
    // going into external references
    int nodeType;
    char *nodeName;
    mgrec *thisnode = node;
    mgrec *newDb;

    traversedNodeCount++;

    oldPercent = currentPercent;
    currentPercent = ((double) traversedNodeCount) / ((double) totalNodeCount) * 100;

    // traverse down to the vertex level,
    // going into external references
    int nodeType;
    char *nodeName;
    mgrec *thisnode = node;
    mgrec *newDb;

    traversedNodeCount++;

    oldPercent = currentPercent;
    currentPercent = ((double) traversedNodeCount) / ((double) totalNodeCount) * 100;

    // traverse down to the vertex level,
if((currentPercent - oldPercent) == 1)
{
    emit updateProgress(currentPercent);
}

while (thisnode)
{
    nodeType = mgGetCode (thisnode);
    nodeName = mgGetName(thisnode);
    if(saveTraverLog == true)
    {
        traverseStream << "\nNode name: " << nodeName << "\n";
        traverseStream << "Node type: " << nodeType << "\n";
    }
    // instanced node -- only traverse one instance
    if (mgIsInstance (thisnode) && mgIsFirstInstance (thisnode))
    {
        if(saveTraverLog == true)
        {
            traverseStream << "The node ""<< nodeName <<"\" is an instace node. Listing all the instances of its referenced node.\n";
        }
        ListInstances (mgGetReference(thisnode));
    }
    switch (nodeType)
    {
    case eFltVertex:
    {
        if(saveTraverLog == true)
        {
            traverseStream << "The node ""<< nodeName <<"\" is the Vertex node. Morph vertices will be ignored.\n";
        }
        break;
    }
    case eFltHeader:
    {
        if (mgGetChild (thisnode))
        {
            char * fileName;
            QString pathToFile = "";
            fileName = mgRec2Filename(thisnode);
            pathToFile.append(fileName);
            if(saveTraverLog == true)
            {
                traverseStream << "The node ""<< nodeName <<"\" is a Header node.\n";
                traverseStream << "File name: ""<< pathToFile <<"\". Getting next child.\n";
            }
            Traverse (mgGetChild (thisnode), LOD);
            mgFree (fileName);
        }
        break;
    }
case eFltPolygon:
{
    double i, j, k, module, x, y, z;
    PolygonInfoLOD currentPolygon;
    Vertex currentVertex;
    mgrec * vertexNode;
    
    if (mgGetPolyNormal(thisnode, &i, &j, &k))
    {
        if (saveTraverLog == true)
        {
            traverseStream << "The node " << nodeName << " is a Polygon node. The normal vector of " << nodeName << " is: ";
            traverseStream << "\ni = " << i;
            traverseStream << "\nj = " << j;
            traverseStream << "\nk = " << k;
        }
        module = sqrt(i*i + j*j + k*k);
        if (saveTraverLog == true)
        {
            traverseStream << "\nmodule of the normal vector = " << module << "\n";
        }
        currentPolygon.normal_i = i;
        currentPolygon.normal_j = j;
        currentPolygon.normal_k = k;
        currentPolygon.LOD = LOD;
        char * fileName, * vertexNodeName;
        fileName = mgRec2Filename(thisnode);
        currentPolygon.fileName = fileName;
        currentPolygon.nodeName = nodeName;
        currentPolygon.vertexCount = 0;
        vertexNode = mgGetChild(thisnode);
        
        while (vertexNode)
        {
            if (mgGetCode(vertexNode) == eFltVertex)
            {
                vertexNodeName = mgGetName(vertexNode);
                currentVertex.nodeName = vertexNodeName;
                if (mgGetVtxCoord (vertexNode, &x, &y, &z))
                {
                    currentVertex.coord[0] = x;
                    currentVertex.coord[1] = y;
                    currentVertex.coord[2] = z;
                    mgcoord3d xyz;
                    mgprojcoord lle;
                    xyz = mgMakeCoord3d (x, y, z);
                    if (mgProjectionConvertXYZtoLLE(dbProjection, &xyz, &lle))
                    {
                        currentVertex.lat = lle.lat;
                        currentVertex.lon = lle.lon;
                    }
                }
            }
        }
    }
}
currentVertex.height = l1e.height;
}
else
{
    emit error("Could not convert XYZ to LLE.");
}
if (!valuesMinMaxIntialized[LOD])
{
    maxX[LOD] = currentVertex.coord[0];
    minX[LOD] = currentVertex.coord[0];
    maxY[LOD] = currentVertex.coord[1];
    minY[LOD] = currentVertex.coord[1];
    maxZ[LOD] = currentVertex.coord[2];
    minZ[LOD] = currentVertex.coord[2];
    maxLat[LOD] = currentVertex.lat;
    minLat[LOD] = currentVertex.lat;
    maxLon[LOD] = currentVertex.lon;
    minLon[LOD] = currentVertex.lon;
    maxHeight[LOD] = currentVertex.height;
    minHeight[LOD] = currentVertex.height;
    valuesMinMaxIntialized[LOD] = true;
}
if (currentVertex.coord[0] < minX[LOD])
    minX[LOD] = currentVertex.coord[0];
if (currentVertex.coord[0] > maxX[LOD])
    maxX[LOD] = currentVertex.coord[0];
if (currentVertex.coord[1] < minY[LOD])
    minY[LOD] = currentVertex.coord[1];
if (currentVertex.coord[1] > maxY[LOD])
    maxY[LOD] = currentVertex.coord[1];
if (currentVertex.coord[2] < minZ[LOD])
    minZ[LOD] = currentVertex.coord[2];
if (currentVertex.coord[2] > maxZ[LOD])
    maxZ[LOD] = currentVertex.coord[2];
if (currentVertex.lat < minLat[LOD])
    minLat[LOD] = currentVertex.lat;
if (currentVertex.lat > maxLat[LOD])
    maxLat[LOD] = currentVertex.lat;
if (currentVertex.lon < minLon[LOD])
    minLon[LOD] = currentVertex.lon;
if (currentVertex.lon > maxLon[LOD])
    maxLon[LOD] = currentVertex.lon;
if (currentVertex.height < minHeight[LOD])
    minHeight[LOD] = currentVertex.height;
if (currentVertex.height > maxHeight[LOD])
    maxHeight[LOD] = currentVertex.height;

    currentPolygon.vertexCount++;}
currentPolygon.vertices.push_back(currentVertex);
vertexNode = mgGetNext(vertexNode);
if (currentPolygon.vertices.size() != 3)
{
    onlyTriangles = false;
}
polygonInfolList[LOD].append(currentPolygon);
}
if (mgGetChild (thisnode))
{
    if (saveTraverLog == true)
    {
        traverseStream << "The \""<< nodeName <<\" node has a child. Getting next child of \""<< nodeName <<\".\n";
        Traverse (mgGetChild (thisnode), LOD);
    }
    break;
} // subfaces
if (mgGetNestedChild (thisnode))
{
    if (saveTraverLog == true)
    {
        traverseStream << "The node \""<< nodeName <<\" has a nested child (subface). Getting next nested child of \""<< nodeName <<\".\n";
        Traverse (mgGetNestedChild (thisnode), LOD);
    }
    break;
}
case eFltLod:
{
    if (mgGetChild (thisnode))
    {
        if (mgIsFlagOn(thisnode))
        {
            if (saveTraverLog == true)
            {
                traverseStream << "The node \""<< nodeName <<\" is a LOD node. Its flag is On. Getting next child of \""<< nodeName <<\".\n";
            }
            Traverse (mgGetChild (thisnode), LOD);
        }
        else
        {
            if (saveTraverLog == true)
            {
                traverseStream << "The node \""<< nodeName <<\" is a LOD node. Its flag is Off. Its children will be ignored.\n";
            }
        }
    }
    break;
}
case eFltGroup:
    {
        if (mgGetChild (thisnode))
            {
                if(saveTraverLog == true)
                    {
                        traverseStream << "The node ""<< nodeName "" is a Group node. Getting next child of ""<< nodeName "":"n;"
                    }
                Traverse (mgGetChild (thisnode), LOD);
            }
        break;
    }
case eFltXref:
    {
        char filename[1024];
        QString pathToFile = this->filePath;
        mgGetAttBuf (thisnode, fltXrefFilename, filename);
        pathToFile.append(filename);
        if(saveTraverLog == true)
            {
                traverseStream << "The node ""<< nodeName "" is a External Reference node.""n;"
            }
        if (!newDb = mgOpenDb (pathToFile.toStdString().c_str()))
            {
                if(saveTraverLog == true)
                    {
                        traverseStream << "Error opening external the reference file ""<< pathToFile "":n;"
                    }
            }
        else
            {
                if(saveTraverLog == true)
                    {
                        traverseStream << "Opening external reference file ""<< pathToFile "":n;"
                    }
                extRefCount++;
                int currentLOD = 0;
                mgMostDetail(newDb);
                do
                    {
                    if(saveTraverLog == true)
                        {
                            traverseStream << "##############################################################################

Current LOD = " << currentLOD << " 

";``
```
traverseStream <<
"########################################################################
#########################
    \n"
;
    Traverse (newDb, currentLOD);
    currentLOD++;
} while (mgLessDetail(newDb));
mgCloseDb(newDb);
} break;
} default:
{
    if (saveTraverLog == true)
    {
        traverseStream << "The node " << nodeName << " is neither a Vertex, nor a Polygon, nor a LOD, nor a Group, nor an External Reference.\n";
        // children
        if (mgGetChild (thisnode))
        {
            if (saveTraverLog == true)
            {
                traverseStream << "The node " << nodeName << " has a child. Getting next child of " << nodeName << ".\n"
            }
            Traverse (mgGetChild (thisnode), LOD);
        }
    }
    // traverse right
    if (! (mgGetNext (thisnode)))
    {
        if (saveTraverLog == true)
        {
            traverseStream << "The node " << nodeName << " doesn't have more siblings.\n"
        }
        thisnode = MG_NULL;
    } else
    {
        if (saveTraverLog == true)
        {
            traverseStream << "Getting next sibling of " << nodeName << ".\n"
        }
        thisnode = mgGetNext (thisnode);
    }
    mgFree(nodeName);
}

void ThreadTraversalSceneGraph::ListInstances (mgrec *node)
mrec *thisnode = node;
if(saveTraverLog == true)
{
    traverseStream << "listing instances of " << mgGetName(node) << ".\n";
}
thisnode = mgGetFirstInstance (node);
while (thisnode)
{
    if(saveTraverLog == true)
    {
        traverseStream << "\tinstance of " << mgGetName(node) << ".\n";
    }
    thisnode = mgGetNextInstance (thisnode);
}

void ThreadTraverseSceneGraph::quickTraverse (mrec *node, int LOD)
{
    // traverse down to the vertex level,
    // going into external references
    int nodeType;
    mrec *thisnode = node;
    mrec *newDb;
totalNodeCount++;
    if (LOD > dbMaxLOD)
        dbMaxLOD = LOD;
    while (thisnode)
    {
        nodeType = mgGetCode (thisnode);
        switch (nodeType)
        {
            case eFltVertex:
            {
                break;
            }
            case eFltHeader:
            {
                if (mgGetChild (thisnode))
                {
                    quickTraverse (mgGetChild (thisnode), LOD);
                }
                break;
            }
            case eFltPolygon:
            {
                if (mgGetChild (thisnode))
                {
                    quickTraverse (mgGetChild (thisnode), LOD);
                }
                // subfaces
                if (mgGetNestedChild (thisnode))
                {
                    quickTraverse (mgGetNestedChild (thisnode), LOD);
                }
            }
        }
    }
}

165
break;
}

case eFltLod:
{
    if (mgGetChild (thisnode))
    {
        if (mgIsFlagOn(thisnode))
        {
            quickTraverse (mgGetChild (thisnode), LOD);
        }
        break;
    }
}

case eFltGroup:
{
    if (mgGetChild (thisnode))
    {
        quickTraverse (mgGetChild (thisnode), LOD);
        break;
    }
}

case eFltXref:
{
    char filename[1024];
    QString pathToFile = this->filePath;
    mgGetAttBuf (thisnode, fltXrefFilename, filename);
    pathToFile.append(filename);

    if (!(newDb = mgOpenDb (pathToFile.toStdString().c_str()))))
    {
        QString auxStr;
        auxStr = "Error opening external the reference file \"
        auxStr.append(pathToFile);
        auxStr.append("\" in quick traverse.\n")
        emit standardOut(auxStr);
    }
    else
    {
        int currentLOD = 0;
        mgMostDetail(newDb);
        do
        {
            quickTraverse (newDb, currentLOD);
            currentLOD++;
        } while (mgLessDetail(newDb));
        mgCloseDb(newDb);
        break;
    }
}

default:
{
    // children
    if (mgGetChild (thisnode))
    {

quickTraverse (mgGetChild (thisnode), LOD);
}
}
// traverse right
if (! (mgGetNext (thisnode)))
{
    thisnode = MG_NULL;
}
else
{
    thisnode = mgGetNext (thisnode);
}
}

threadheavycalculations.h

#ifndef THREADHEAVYCALCULATIONS_H
#define THREADHEAVYCALCULATIONS_H
#include <QThread>
#include <QtCore>
#include "tdbanalyzer.h"

class ThreadHeavyCalculations : public QThread
{
    Q_OBJECT
public:
    ThreadHeavyCalculations(QObject *Parent, TDBAnalyzer *tdbToAnalyse);
    ~ThreadHeavyCalculations();

    void run();

    TDBAnalyzer * heavyCalculationTDBAnalyzer;
    int eyePointsPerBlock;
    QVector<oyTerrainBlock> blockList;
    int initialBlock, blockOffset;

    // operationMode == 0 for probe terrain based on number of sky eye-points per block
    // operationMode == 1 for probe terrain based on block list
    // operationMode == 2 for LOS calculation
    unsigned int operationMode;
    int percentComplete;
    bool createTDBLocalCopy;

public slots:
    void receiveProgress(int value);
    void receiveError(QString errorThreadCalc);
    void receiveInfo(QString info);
signals:
    void resultLOSReady();
    void resultEyepointProjectionsReady();
    void errorThreadCalc(QString errorThreadCalc);
    void infoThreadCalc(QString errorThreadCalc);
    void updateThreadCalculationProgress(int value);
};
#endif // THREADHEAVYCALCULATIONS_H

threadheavycalculations.cpp

#include "threadheavycalculations.h"

ThreadHeavyCalculations::ThreadHeavyCalculations(QObject *Parent, TDBAnalyzer *tdbToAnalyse) : QThread(Parent)
{
    heavyCalculationTDBAnalyzer = tdbToAnalyse;
    createTDBLocalCopy = false;
}

ThreadHeavyCalculations::~ThreadHeavyCalculations()
{
}

void ThreadHeavyCalculations::run()
{
    if (createTDBLocalCopy)
    {
        TDBAnalyzer *TDBLocalCopy;
        TDBLocalCopy = new TDBAnalyzer();
        TDBLocalCopy->LODCount = heavyCalculationTDBAnalyzer->LODCount;
        TDBLocalCopy->LODToBeAnalyzed = heavyCalculationTDBAnalyzer->LODToBeAnalyzed;
        TDBLocalCopy->LODList = heavyCalculationTDBAnalyzer->LODList;
        TDBLocalCopy->terrainBlockList = heavyCalculationTDBAnalyzer->terrainBlockList;
        TDBLocalCopy->fileName = heavyCalculationTDBAnalyzer->fileName;
        TDBLocalCopy->dbProjection = heavyCalculationTDBAnalyzer->dbProjection;
        connect(TDBLocalCopy, SIGNAL(updateProgress(int)), this, SLOT(receiveProgress(int)));
        connect(TDBLocalCopy, SIGNAL(error(QString)), this, SLOT(receiveError(QString)));
        connect(TDBLocalCopy, SIGNAL(infoReady(QString)), this, SLOT(receiveInfo(QString)));

        // operationMode == 0 for probe terrain based on number of sky eye-points per block
        if (operationMode == 0)
        {
            // code...
        }
    }
}
TDBLocalCopy->FillEyePointsList(eyepointsPerBlock, initialBlock, blockOffset);
    emit resultEyepointProjectionsReady();
} // operationMode == 1 for probe terrain based on block list
if (operationMode == 1)
{
    TDBLocalCopy->FillEyePointsList(blockList, initialBlock, blockOffset);
    emit resultEyepointProjectionsReady();
} // operationMode == 2 for LOS calculation
if (operationMode == 2)
{
    TDBLocalCopy->CalculateLOS(initialBlock, blockOffset);
    emit resultLOSReady();
} for(int i = initialBlock; i < (initialBlock + blockOffset); i++)
{    heavyCalculationTDBAnalyzer->terrainBlockList[i] = TDBLocalCopy->terrainBlockList[i];
    delete TDBLocalCopy;
} else
{
    connect(heavyCalculationTDBAnalyzer, SIGNAL(updateProgress(int)),
            this, SLOT(receiveProgress(int)));
    // operationMode == 0 for probe terrain based on number of sky eye-points per block
    if (operationMode == 0)
    {
        heavyCalculationTDBAnalyzer->FillEyePointsList(eyepointsPerBlock, initialBlock, blockOffset);
        emit resultEyepointProjectionsReady();
    } // operationMode == 1 for probe terrain based on block list
    if (operationMode == 1)
    {
        heavyCalculationTDBAnalyzer->FillEyePointsList(blockList, initialBlock, blockOffset);
        emit resultEyepointProjectionsReady();
    } // operationMode == 2 for LOS calculation
    if (operationMode == 2)
    {
        heavyCalculationTDBAnalyzer->CalculateLOS(initialBlock, blockOffset);
        emit resultLOSReady();
    }
}

void ThreadHeavyCalculations::receiveProgress(int value)
emit updateThreadCalculationProgress(value);
}

void ThreadHeavyCalculations::receiveError(QString error)
{
    emit errorThreadCalc(error);
}

void ThreadHeavyCalculations::receiveInfo(QString info)
{
    emit infoThreadCalc(info);
}

tdbcomparison.h

#ifndef TDBCOMPARISON_H
#define TDBCOMPARISON_H
#define INVALID_LAT_LON_VALUE -400.0

#include <QObject>
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
#include <math.h>
#include "util.h"
#include "tdbanalyzer.h"

class TDBComparison : public QObject
{
    Q_OBJECT
public:
    TDBComparison(QObject *parent = 0, TDBAnalyzer *database1 = 0, TDBAnalyzer *database2 = 0);
    ~TDBComparison();
    oyTDBComparisonData comparData;
    TDBAnalyzer *TDB1, *TDB2;
    int blockRowCount, blockColumnCount;
    int eyePointsOrthoProjPerBlock;
    int eyePointsPerOrthoProj;
    int rayAzimuthVarCount;
    int rayPitchVarCount;
    bool initialized;

    void rearrangeData();
    void calculateTDBStatistics(oyTDBComparisonData *TDBCompData);
    void calculateGrndPosStatistics(oyGrndPosAnalysisData *grndPos);
    void calculateTerrainBlockStatistics(oyTerrainBlockAnalysisData *terrainBlock);

    bool printResults();

signals:

170
void error(QString);

public slots:

};
#endif // TDBCOMPARISON_H

tdbcomparison.cpp

#include "tdbcomparison.h"

TDBComparison::TDBComparison(QObject *parent, TDBAnalyzer *database1, TDBAnalyzer *database2) :
    QObject(parent)
{
    TDB1 = database1;
    TDB2 = database2;
    initialized = true;
}

TDBComparison::~TDBComparison()
{
    comparData.blockAnalysisData.clear();
    comparData.elevDiffList.clear();
    comparData.grndPosAalysisData.clear();
    comparData.LOSRayDiffList.clear();
    TDB1 = NULL;
    TDB2 = NULL;
    initialized = false;
}

void TDBComparison::rearrangeData()
{
    int LOSRayDiffListIndex = 0;
    int ElevDiffListIndex = 0;
    if (TDB1->terrainBlockList.size() != TDB2->terrainBlockList.size())
    {
        emit error("The length of terrain block list of the TDB1 is different than
                   the length of terrain block list of the TDB2.");
        return;
    }
    for(int i = 0; i < TDB1->terrainBlockList.size(); i++)
    {
        oyTerrainBlockAnalysisData currentBlock;
        if (TDB1->terrainBlockList[i].eyePointSkyXYZList.size() != TDB2-
            >terrainBlockList[i].eyePointSkyXYZList.size())
        {
            emit error("The total of virtual eye-points doesn't match in both
                        blocks (XYZ). ") ;
            return;
        }
    }
if (TDB1->terrainBlockList[i].eyePointSkyLLEList.size() != TDB2->terrainBlockList[i].eyePointSkyLLEList.size())
{
    emit error("The total of virtual eye-points doesn't match in both blocks (LLE)." namely blockIndex); return;
}
if (TDB1->terrainBlockList[i].eyePointGroundXYZList.size() != TDB2->terrainBlockList[i].eyePointGroundXYZList.size())
{
    emit error("The total of eye-point orthogonal projections doesn't match in both blocks (XYZ)." namely blockIndex); return;
}
if (TDB1->terrainBlockList[i].eyePointGroundLLEList.size() != TDB2->terrainBlockList[i].eyePointGroundLLEList.size())
{
    emit error("The total of eye-point orthogonal projections doesn't match in both blocks (LLE)." namely blockIndex); return;
}
else
{
    for(int j = 0; j < TDB1->terrainBlockList[i].eyePointGroundLLEList.size(); j++)
    {
        double elevDiff = TDB1->terrainBlockList[i].eyePointGroundLLEList[j].z() - TDB2->terrainBlockList[i].eyePointGroundLLEList[j].z();
        comparData.elevDiffList.push_back(elevDiff);
        currentBlock.blockElevDiffIndexList.push_back(ElevDiffListIndex);
        ElevDiffListIndex++;
    }
}
if (TDB1->terrainBlockList[i].LOSDatasetList.size() != TDB2->terrainBlockList[i].LOSDatasetList.size())
{
    emit error("The total of LOS tests doesn't match in both blocks."); return;
}
else
{
    oyGrndPosAnalysisData currentGroundPos;
    double lat = INVALID_LAT_LON_VALUE, lon = INVALID_LAT_LON_VALUE;
    double previousLat = INVALID_LAT_LON_VALUE, previousLon = INVALID_LAT_LON_VALUE;
    for(int j = 0; j < TDB1->terrainBlockList[i].LOSDatasetList.size(); j++)
    {
        if(TDB1->terrainBlockList[i].LOSDatasetList[j].dirXYZList.size() != TDB2->terrainBlockList[i].LOSDatasetList[j].dirXYZList.size())
        {
            // Code block
        }
    }
}
emit error("The length of the direction list is different at this eye-point for each TDB.");
    return;
}
if(TDB1->terrainBlockList[i].LOSDatasetList[j].upDirXYZList.size() != TDB2->terrainBlockList[i].LOSDatasetList[j].upDirXYZList.size())
{
    emit error("The length of the up-direction list is different at this eye-point for each TDB.");
    return;
}
if(TDB1->terrainBlockList[i].LOSDatasetList[j].intersectPntXYZList.size() != TDB2->terrainBlockList[i].LOSDatasetList[j].intersectPntXYZList.size())
{
    emit error("The length of the intersection point list is different at this eye-point for each TDB.");
    return;
}
if(TDB1->terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList.size() != TDB2->terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList.size())
{
    emit error("The length of the array \"isRayIntersectingBoundingBox\" is different at this eye-point for each TDB.");
    return;
}
if(TDB1->terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList.size() != TDB2->terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList.size())
{
    emit error("The length of the distance to intersection point list is different at this eye-point for each TDB.");
    return;
}
else{
    previousLat = lat;
    previousLon = lon;
    QVector3D currentLLE = TDB1->convertXYZtoLLE(TDB1->terrainBlockList[i].LOSDatasetList[j].eyePosXYZ);
    lat = currentLLE.x();
    lon = currentLLE.y();
    if ((previousLat != lat) || (previousLon != lon))
    {
        if ((previousLat != INVALID_LAT_LON_VALUE) && (previousLon != INVALID_LAT_LON_VALUE))
        {
            currentGroundPos.lat = previousLat;
            currentGroundPos.lon = previousLon;
            comparData.grndPosAnalysisData.push_back(currentGroundPos);
        }
    }
}
currentGroundPos.grndPosLOSRayDiffIndexList.clear();
    }
}
for(int k = 0; k < TDB1-terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList.size(); k++)
{
    QCoreApplication::processEvents();
    double LOSDiff = TDB1-terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList[k] - TDB2-terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList[k];
    comparData.LOSRayDiffList.push_back(LOSDiff);
}
 currentsBlock.blockLOSRayDiffIndexList.push_back(LOSRayDiffListIndex);
currentGroundPos.grndPosLOSRayDiffIndexList.push_back(LOSRayDiffListIndex);
LOSRayDiffListIndex++;

    oyLOSRayDiffUnity currentLOSRayDiffUnity;
    currentLOSRayDiffUnity.TDB1RayXYZ.eyePosXYZ = TDB1-terrainBlockList[i].LOSDatasetList[j].eyePosXYZ;
    currentLOSRayDiffUnity.TDB1RayXYZ.intersectPtXYZ = TDB1-terrainBlockList[i].LOSDatasetList[j].intersectPntXYZList[k];
    currentLOSRayDiffUnity.TDB1RayXYZ.isRayIntersectingBBList = TDB1-terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList[k];
    currentLOSRayDiffUnity.TDB2RayXYZ.eyePosXYZ = TDB2-terrainBlockList[i].LOSDatasetList[j].eyePosXYZ;
    currentLOSRayDiffUnity.TDB2RayXYZ.intersectPtXYZ = TDB2-terrainBlockList[i].LOSDatasetList[j].intersectPntXYZList[k];
    currentLOSRayDiffUnity.TDB2RayXYZ.isRayIntersectingBBList = TDB2-terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList[k];

    if((currentLOSRayDiffUnity.TDB1RayXYZ.isRayIntersectingBB && currentLOSRayDiffUnity.TDB2RayXYZ.isRayIntersectingBB) || (!currentLOSRayDiffUnity.TDB1RayXYZ.isRayIntersectingBB && !currentLOSRayDiffUnity.TDB2RayXYZ.isRayIntersectingBB))
    {
        currentLOSRayDiffUnity.BBIntersectingCorresp = true;
    }
else
    {
        currentLOSRayDiffUnity.BBIntersectingCorresp = false;
    }

    currentLOSRayDiffUnity.absDiffLOS = abs(LOSDiff);
    currentLOSRayDiffUnity.diffLOS = LOSDiff;
    currentLOSRayDiffUnity.AGLHeight = TDB1-terrainBlockList[i].LOSDatasetList[j].actualHeightAGL;
QVector3D currentEyepointPosLLE,
currentEyepointOrthoProjection;
currentEyepointPosLLE = TDB1->convertXYZtoLLE(currentLOSRayDiffUnity.TDB1RayXYZ.eyePosXYZ);
currentEyepointOrthoProjection = currentEyepointPosLLE;

currentEyepointOrthoProjection.setZ(currentEyepointPosLLE.z() - currentLOSRayDiffUnity.AGLHeight);
currentLOSRayDiffUnity.groundProjLLE = currentEyepointOrthoProjection;

currentLOSRayDiffUnity.azimuth = TDB1->terrainBlockList[i].LOSDatasetList[j].azimuthAngleList[k];
currentLOSRayDiffUnity.pitch = TDB1->terrainBlockList[i].LOSDatasetList[j].pitchAngleList[k];
bool currentLOSRayDiffUnityInserted = false;
//
*****************************************************************************
***** //
currentLOSRayDiffUnityInserted = false;
for(int l = 0; l < comparData.LOSRayDiffUnityOrderedList.size(); l++)
{
    if(currentLOSRayDiffUnity.absDiffLOS < comparData.LOSRayDiffUnityOrderedList[l].absDiffLOS)
    {
        continue;
    }
    else
    {
        comparData.LOSRayDiffUnityOrderedList.insert(l, currentLOSRayDiffUnity);
        currentLOSRayDiffUnityInserted = true;
        break;
    }
}
if(!currentLOSRayDiffUnityInserted)
{
comparData.LOSRayDiffUnityOrderedList.push_back(currentLOSRayDiffUnity);
}
}
if(j == TDB1->terrainBlockList[i].LOSDatasetList.size() - 1)
{
    currentGroundPos.lat = lat;
currentGroundPos.lon = lon;
comparData.grndPosAnalysisData.push_back(currentGroundPos);
currentGroundPos.grndPosLOSRayDiffIndexList.clear();
}
}
comparData.blockAnalysisData.push_back(currentBlock);
}
}
calculateTDBStatistics(&comparData);
for(int i = 0; i < comparData.blockAnalysisData.size(); i++)
{
calculateTerrainBlockStatistics(&comparData.blockAnalysisData[i]);
}
for(int i = 0; i < comparData.grndPosAnalysisData.size(); i++)
{
calculateGrndPosStatistics(&comparData.grndPosAnalysisData[i]);
}
}

void TDBComparison::calculateTDBStatistics(oyTDBComparisonData *TDBCompData)
{
    // Calculate LOS ray length diff average, min and max.
    // Also calculate LOS ray length absolute diff average min and max.
    double LOSDiffAvg = 0, LOSAbsDiffAvg = 0;
    double LOSDiffMin = 0, LOSAbsDiffMin = 0;
    double LOSDiffMax = 0, LOSAbsDiffMax = 0;
    for(int i = 0; i < TDBCompData->LOSRayDiffList.size(); i++)
    {
        double currentDiffValue = TDBCompData->LOSRayDiffList[i];
        LOSDiffAvg += currentDiffValue;
        LOSAbsDiffAvg += abs(currentDiffValue);
        if(i == 0)
        {
            LOSDiffMin = currentDiffValue;
            LOSDiffMax = currentDiffValue;
            LOSAbsDiffMin = abs(currentDiffValue);
            LOSAbsDiffMax = abs(currentDiffValue);
        }
        else
        {
            LOSDiffMin = (LOSDiffMin > currentDiffValue)? currentDiffValue : LOSDiffMin;
            LOSDiffMax = (LOSDiffMax < currentDiffValue)? currentDiffValue : LOSDiffMax;
            LOSAbsDiffMin = (LOSAbsDiffMin > abs(currentDiffValue))?(abs(currentDiffValue) : LOSAbsDiffMin;
            LOSAbsDiffMax = (LOSAbsDiffMax < abs(currentDiffValue))? abs(currentDiffValue) : LOSAbsDiffMax;
        }
    }
    LOSDiffAvg = LOSDiffAvg / TDBCompData->LOSRayDiffList.size();
    LOSAbsDiffAvg = LOSAbsDiffAvg / TDBCompData->LOSRayDiffList.size();
    // Calculate LOS ray length diff std dev and std error.
    // Also calculate LOS ray length absolute diff std dev and std error.
    double LOSDiffStdDev = 0, LOSAbsDiffStdDev = 0;
    for(int i = 0; i < TDBCompData->LOSRayDiffList.size(); i++)
    {
        double currentDiffValue = TDBCompData->LOSRayDiffList[i];
        LOSDiffStdDev += pow(LOSDiffAvg - currentDiffValue, 2);
    }
}
\[
\text{LOSAbsDiffStdDev} += \text{pow}(\text{LOSAbsDiffAvg} - \text{abs}(\text{currentDiffValue}), 2);
\]

\[
\text{LOSDiffStdDev} = \sqrt{\text{LOSDiffStdDev} / (\text{TDBCompData-}\text{LOSRayDiffList.size()} - 1)};
\]

\[
\text{LOSAbsDiffStdDev} = \sqrt{\text{LOSAbsDiffStdDev} / (\text{TDBCompData-}\text{LOSRayDiffList.size()} - 1)};
\]

\[
\text{LOSDiffStdErr} = \text{LOSDiffStdDev} / \sqrt{(\text{double})\text{TDBCompData-}\text{LOSRayDiffList.size()}};
\]

\[
\text{LOSAbsDiffStdErr} = \text{LOSAbsDiffStdDev} / \sqrt{(\text{double})\text{TDBCompData-}\text{LOSRayDiffList.size()}};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffMax} = \text{LOSDiffMax};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffMin} = \text{LOSDiffMin};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffAvg} = \text{LOSDiffAvg};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffStdDev} = \text{LOSDiffStdDev};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffStdErr} = \text{LOSDiffStdErr};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayAbsDiffMax} = \text{LOSAbsDiffMax};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayAbsDiffMin} = \text{LOSAbsDiffMin};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayAbsDiffAvg} = \text{LOSAbsDiffAvg};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayAbsDiffStdDev} = \text{LOSAbsDiffStdDev};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayAbsDiffStdErr} = \text{LOSAbsDiffStdErr};
\]

\[
\text{TDBCompData}\rightarrow\text{overallLOSRayDiffSampleSize} = \text{TDBCompData-}\text{LOSRayDiffList.size()};
\]

// Calculate elevation diff average, min and max.
// Also calculate elevation absolute diff average min and max.
\[
\text{elevDiffAvg} = 0, \text{elevAbsDiffAvg} = 0;
\]
\[
\text{elevDiffMin} = 0, \text{elevAbsDiffMin} = 0;
\]
\[
\text{elevDiffMax} = 0, \text{elevAbsDiffMax} = 0;
\]

for(int i = 0; i < TDBCompData->elevDiffList.size(); i++)
{
  double currentDiffValue = TDBCompData->elevDiffList[i];
  elevDiffAvg += currentDiffValue;
  elevAbsDiffAvg += abs(currentDiffValue);
  if(i == 0)
  {
    elevDiffMin = currentDiffValue;
    elevDiffMax = currentDiffValue;
    elevAbsDiffMin = abs(currentDiffValue);
    elevAbsDiffMax = abs(currentDiffValue);
  }
  else
  {
    elevDiffMin = (elevDiffMin > currentDiffValue) ? currentDiffValue : elevDiffMin;
    elevDiffMax = (elevDiffMax < currentDiffValue) ? currentDiffValue : elevDiffMax;
    elevAbsDiffMin = (elevAbsDiffMin > abs(currentDiffValue)) ? abs(currentDiffValue) : elevAbsDiffMin;
    elevAbsDiffMax = (elevAbsDiffMax < abs(currentDiffValue)) ? abs(currentDiffValue) : elevAbsDiffMax;
  }
}

elevDiffAvg = elevDiffAvg / TDBCompData->elevDiffList.size();
elevAbsDiffAvg = elevAbsDiffAvg / TDBCompData->elevDiffList.size();

// Calculate elevation diff std dev and std error.
// Also calculate elevation absolute diff std dev and std error.
double elevDiffStdDev = 0, elevAbsDiffStdDev = 0;
for(int i = 0; i < TDBCompData->elevDiffList.size(); i++)
{
    double currentDiffValue = TDBCompData->elevDiffList[i];
    elevDiffStdDev += pow(elevDiffAvg - currentDiffValue, 2);
    elevAbsDiffStdDev += pow(elevAbsDiffAvg - abs(currentDiffValue), 2);
}

elevDiffStdDev = sqrt(elevDiffStdDev / (TDBCompData->elevDiffList.size() - 1));
elevAbsDiffStdDev = sqrt(elevAbsDiffStdDev / (TDBCompData->elevDiffList.size() - 1));
double elevDiffStdErr = 0, elevAbsDiffStdErr = 0;
elevDiffStdErr = elevDiffStdDev / sqrt((double)TDBCompData->elevDiffList.size());
elevAbsDiffStdErr = elevAbsDiffStdDev / sqrt((double)TDBCompData->elevDiffList.size());

TDBCompData->overallElevationDiffMax = elevDiffMax;
TDBCompData->overallElevationDiffMin = elevDiffMin;
TDBCompData->overallElevationDiffAvg = elevDiffAvg;
TDBCompData->overallElevationDiffStdDev = elevDiffStdDev;
TDBCompData->overallElevationDiffStdErr = elevDiffStdErr;
TDBCompData->overallElevationAbsDiffMax = elevAbsDiffMax;
TDBCompData->overallElevationAbsDiffMin = elevAbsDiffMin;
TDBCompData->overallElevationAbsDiffAvg = elevAbsDiffAvg;
TDBCompData->overallElevationAbsDiffStdDev = elevAbsDiffStdDev;
TDBCompData->overallElevationAbsDiffStdErr = elevAbsDiffStdErr;
TDBCompData->overallElevationDiffSampleSize = TDBCompData->elevDiffList.size();
}

void TDBComparison::calculateTerrainBlockStatistics(oyTerrainBlockAnalysisData *terrainBlock)
{
    // Calculate LOS ray length diff average, min and max.
    // Also calculate LOS ray length absolute diff average min and max.
double LOSDiffAvg = 0, LOSAbsDiffAvg = 0;
double LOSDiffMin = 0, LOSAbsDiffMin = 0;
double LOSDiffMax = 0, LOSAbsDiffMax = 0;
for(int i = 0; i < terrainBlock->blockLOSRayDiffIndexList.size(); i++)
{
    double currentDiffValue = comparData.LOSRayDiffList[terrainBlock->blockLOSRayDiffIndexList[i]];
    LOSDiffAvg += currentDiffValue;
    LOSAbsDiffAvg += abs(currentDiffValue);
    if(i == 0)
    {
        LOSDiffMin = currentDiffValue;
        LOSAbsDiffMin = abs(currentDiffValue);
    }
LOSAbsDiffMax = abs(currentDiffValue);
}
else {
    LOSDiffMin = (LOSDiffMin > currentDiffValue) ? currentDiffValue : LOSDiffMin;
    LOSDiffMax = (LOSDiffMax < currentDiffValue) ? currentDiffValue : LOSDiffMax;
    LOSAbsDiffMin = (LOSAbsDiffMin > abs(currentDiffValue)) ? abs(currentDiffValue) : LOSAbsDiffMin;
    LOSAbsDiffMax = (LOSAbsDiffMax < abs(currentDiffValue)) ? abs(currentDiffValue) : LOSAbsDiffMax;
}
}
LOSDiffAvg = LOSDiffAvg / terrainBlock->blockLOSRayDiffIndexList.size();
LOSAbsDiffAvg = LOSAbsDiffAvg / terrainBlock->blockLOSRayDiffIndexList.size();

// Calculate LOS ray length diff std dev and std error.
// Also calculate LOS ray length absolute diff std dev and std error.
double LOSDiffStdDev = 0, LOSAbsDiffStdDev = 0;
for (int i = 0; i < terrainBlock->blockLOSRayDiffIndexList.size(); i++) {
    double currentDiffValue = compaData.LOSRayDiffList[terrainBlock->blockLOSRayDiffIndexList[i]];
    LOSDiffStdDev += pow(LOSDiffAvg - currentDiffValue, 2);
    LOSAbsDiffStdDev += pow(LOSAbsDiffAvg - abs(currentDiffValue), 2);
}
LOSDiffStdDev = sqrt(LOSDiffStdDev / (terrainBlock->blockLOSRayDiffIndexList.size() - 1));
LOSAbsDiffStdDev = sqrt(LOSAbsDiffStdDev / (terrainBlock->blockLOSRayDiffIndexList.size() - 1));
double LOSDiffStdErr = 0, LOSAbsDiffStdErr = 0;
LOSDiffStdErr = LOSDiffStdDev / sqrt((double) terrainBlock->blockLOSRayDiffIndexList.size());
LOSAbsDiffStdErr = LOSAbsDiffStdDev / sqrt((double) terrainBlock->blockLOSRayDiffIndexList.size());

terrainBlock->blockLOSRayDiffMax = LOSDiffMax;
terrainBlock->blockLOSRayDiffMin = LOSDiffMin;
terrainBlock->blockLOSRayDiffAvg = LOSDiffAvg;
terrainBlock->blockLOSRayDiffStdDev = LOSDiffStdDev;
terrainBlock->blockLOSRayDiffStdErr = LOSDiffStdErr;
terrainBlock->blockLOSRayAbsDiffMax = LOSAbsDiffMax;
terrainBlock->blockLOSRayAbsDiffMin = LOSAbsDiffMin;
terrainBlock->blockLOSRayAbsDiffAvg = LOSAbsDiffAvg;
terrainBlock->blockLOSRayAbsDiffStdDev = LOSAbsDiffStdDev;
terrainBlock->blockLOSRayAbsDiffStdErr = LOSAbsDiffStdErr;
terrainBlock->blockLOSRayDiffSampleSize = terrainBlock->blockLOSRayDiffIndexList.size();

// Calculate elevation diff average, min and max.
// Also calculate elevation absolute diff average min and max.
double elevDiffAvg = 0, elevAbsDiffAvg = 0;
double elevDiffMin = 0, elevAbsDiffMin = 0;
double elevDiffMax = 0, elevAbsDiffMax = 0;
for(int i = 0; i < terrainBlock->blockElevDiffIndexList.size(); i++)
{
    double currentDiffValue = comparData.elevDiffList[terrainBlock->
    blockElevDiffIndexList[i]];  
    elevDiffAvg += currentDiffValue;
    elevAbsDiffAvg += abs(currentDiffValue);
    if (i == 0)
    {
        elevDiffMin = currentDiffValue;
        elevDiffMax = currentDiffValue;
        elevAbsDiffMin = abs(currentDiffValue);
        elevAbsDiffMax = abs(currentDiffValue);
    }
    else
    {
        elevDiffMin = (elevDiffMin > currentDiffValue) ? currentDiffValue : elevDiffMin;
        elevDiffMax = (elevDiffMax < currentDiffValue) ? currentDiffValue : elevDiffMax;
        elevAbsDiffMin = (elevAbsDiffMin > abs(currentDiffValue)) ? abs(currentDiffValue) : elevAbsDiffMin;
        elevAbsDiffMax = (elevAbsDiffMax < abs(currentDiffValue)) ? abs(currentDiffValue) : elevAbsDiffMax;
    }
    
    elevDiffAvg = elevDiffAvg / terrainBlock->blockElevDiffIndexList.size();
    elevAbsDiffAvg = elevAbsDiffAvg / terrainBlock->blockElevDiffIndexList.size();
}

// Calculate elevation diff std dev and std error.
// Also calculate elevation absolute diff std dev and std error.
double elevDiffStdDev = 0, elevAbsDiffStdDev = 0;
for(int i = 0; i < terrainBlock->blockElevDiffIndexList.size(); i++)
{
    double currentDiffValue = comparData.elevDiffList[terrainBlock->
    blockElevDiffIndexList[i]];  
    elevDiffStdDev += pow(elevDiffAvg - currentDiffValue, 2);
    elevAbsDiffStdDev += pow(elevAbsDiffAvg - abs(currentDiffValue), 2);
}
elevDiffStdDev = sqrt(elevDiffStdDev / (terrainBlock->
    blockElevDiffIndexList.size() - 1));
elevAbsDiffStdDev = sqrt(elevAbsDiffStdDev / (terrainBlock->
    blockElevDiffIndexList.size() - 1));

double elevDiffStdErr = 0, elevAbsDiffStdErr = 0;
elevDiffStdErr = elevDiffStdDev / sqrt((double)terrainBlock->
    blockElevDiffIndexList.size());
elevAbsDiffStdErr = elevAbsDiffStdDev / sqrt((double)terrainBlock->
    blockElevDiffIndexList.size());

terrainBlock->blockElevDiffMax = elevDiffMax;
terrainBlock->blockElevDiffMin = elevDiffMin;
terrainBlock->blockElevDiffAvg = elevDiffAvg;
terrainBlock->blockElevDiffStdDev = elevDiffStdDev;
terrainBlock->blockElevDiffStdErr = elevDiffStdErr;
terrainBlock->blockElevAbsDiffMax = elevAbsDiffMax;
terrainBlock->blockElevAbsDiffMin = elevAbsDiffMin;
terrainBlock->blockElevAbsDiffAvg = elevAbsDiffAvg;
terrainBlock->blockElevAbsDiffStdDev = elevAbsDiffStdDev;
terrainBlock->blockElevAbsDiffStdErr = elevAbsDiffStdErr;
terrainBlock->blockElevDiffSampleSize = terrainBlock->blockElevDiffIndexList.size();
}

void TDBComparison::calculateGrndPosStatistics(oyGrndPosAnalysisData *grndPos)
{
    double LOSDiffAvg = 0, LOSAbsDiffAvg = 0;
    double LOSDiffMin = 0, LOSAbsDiffMin = 0;
    double LOSDiffMax = 0, LOSAbsDiffMax = 0;
    for(int i = 0; i < grndPos->grndPosLOSRayDiffIndexList.size(); i++)
    {
        double currentDiffValue = comparData.LOSRayDiffList[grndPos->grndPosLOSRayDiffIndexList[i]];
        LOSDiffAvg += currentDiffValue;
        LOSAbsDiffAvg += abs(currentDiffValue);
        if(i == 0)
        {
            LOSDiffMin = currentDiffValue;
            LOSDiffMax = currentDiffValue;
            LOSAbsDiffMin = abs(currentDiffValue);
            LOSAbsDiffMax = abs(currentDiffValue);
        }
        else
        {
            LOSDiffMin = (LOSDiffMin > currentDiffValue)? currentDiffValue : LOSDiffMin;
            LOSDiffMax = (LOSDiffMax < currentDiffValue)? currentDiffValue : LOSDiffMax;
            LOSAbsDiffMin = (LOSAbsDiffMin > abs(currentDiffValue))? abs(currentDiffValue) : LOSAbsDiffMin;
            LOSAbsDiffMax = (LOSAbsDiffMax < abs(currentDiffValue))? abs(currentDiffValue) : LOSAbsDiffMax;
        }
    }
    LOSDiffAvg = LOSDiffAvg / grndPos->grndPosLOSRayDiffIndexList.size();
    LOSAbsDiffAvg = LOSAbsDiffAvg / grndPos->grndPosLOSRayDiffIndexList.size();

    // Calculate LOS ray length diff std dev and std error.
    // Also calculate LOS ray length absolute diff std dev and std error.
double LOSDiffStdDev = 0, LOSAbsDiffStdDev = 0;
    for(int i = 0; i < grndPos->grndPosLOSRayDiffIndexList.size(); i++)
    {
        double currentDiffValue = comparData.LOSRayDiffList[grndPos->grndPosLOSRayDiffIndexList[i]];
        LOSDiffStdDev += pow(LOSDiffAvg - currentDiffValue, 2);
        LOSAbsDiffStdDev += pow(LOSAbsDiffAvg - abs(currentDiffValue), 2);
    }
LOSDiffStdDev = \sqrt{\frac{\text{LOSDiffStdDev}}{\text{grndPos-}\overline{\text{grndPosLOSRayDiffIndexList.size()}} - 1}};
LOSAbsDiffStdDev = \sqrt{\frac{\text{LOSAbsDiffStdDev}}{\text{grndPos-}\overline{\text{grndPosLOSRayDiffIndexList.size()}} - 1}};

double LOSDiffStdErr = 0, LOSAbsDiffStdErr = 0;
LOSDiffStdErr = LOSDiffStdDev / sqrt((double) grndPos-
>\text{grndPosLOSRayDiffIndexList.size()});
LOSAbsDiffStdErr = LOSAbsDiffStdDev / sqrt((double) grndPos-
>\text{grndPosLOSRayDiffIndexList.size()});

grndPos->grndPosRayDiffMax = LOSDiffMax;
grndPos->grndPosRayDiffMin = LOSDiffMin;
grndPos->grndPosRayDiffAvg = LOSDiffAvg;
grndPos->grndPosLOSRayDiffStdDev = LOSDiffStdDev;
grndPos->grndPosLOSRayDiffStdErr = LOSDiffStdErr;
grndPos->grndPosRayAbsDiffMax = LOSAbsDiffMax;
grndPos->grndPosRayAbsDiffMin = LOSAbsDiffMin;
grndPos->grndPosRayAbsDiffAvg = LOSAbsDiffAvg;
grndPos->grndPosLOSRayAbsDiffStdDev = LOSAbsDiffStdDev;
grndPos->grndPosLOSRayAbsDiffStdErr = LOSAbsDiffStdErr;
grndPos->grndPosLOSRayDiffSampleSize = grndPos-
>\text{grndPosLOSRayDiffIndexList.size()};

bool TDBComparison::printResults()
{
    QString resultsFilePath, suggestedFileName;
    QDateTime now = QDateTime::currentDateTime();
    QFile LOSTestsResultsFile;
    QTextStream LOSTestsResultsStream;
    suggestedFileName = "LOS_RESULTS_";
suggestedFileName.append(now.toString("MMM.dd.yyyy.h.mm.ap"));
suggestedFileName.replace( ".", "_" );
suggestedFileName.append( "\n" );
resultsFilePath = suggestedFileName;
LOSTestsResultsFile.setFileName(resultsFilePath);
if (!LOSTestsResultsFile.open(QIODevice::WriteOnly | QIODevice::Text))
    return false;
else

    LOSTestsResultsStream.setDevice(&LOSTestsResultsFile);
    LOSTestsResultsStream << "This file contains the results of the analysis of the following databases:\n";
    LOSTestsResultsStream << TDB1->fileName << "\n";
    LOSTestsResultsStream << TDB2->fileName << "\n";
    LOSTestsResultsStream << "LOD selected for DB#1: " << TDB1->LODToBeAnalyzed << "\n";
    LOSTestsResultsStream << "LOD selected for DB#2: " << TDB2->LODToBeAnalyzed << "\n";
    LOSTestsResultsStream << "Number of blocks: " << blockRowCount << "x" << blockColumnCount << " = " << comparData.blockAnalysisData.size() << "\n";
}
LOSTestsResultsStream << "Number of eye point ground projections per block: " << eyePointsOrthoProjPerBlock << "\n";
LOSTestsResultsStream << "Number of height increments per eye point ground projection: " << eyePointsPerOrthoProj << "\n";
LOSTestsResultsStream << "Number of azimuths per eye point: " << rayAzimuthsVarCount << "\n";
LOSTestsResultsStream << "Number of pitch variations per azimuth: " << rayPitchVarsVarCount << "\n";
int totalLOSTestsPerDB;
totalLOSTestsPerDB = comparData.LOSRayDiffList.size();
LOSTestsResultsStream << "Total LOS tests per DB: " << totalLOSTestsPerDB << "\n"
LOSTestsResultsStream << "Total LOS tests per block: " << comparData.blockAnalysisData[0].blockLOSRayDiffIndexList.size() << "\n\n";

LOSTestsResultsStream << "############################################################################
###########################
############################################################################
comparData.totalLOSTestsPerDB = 0;
for(int i = 0; i < TDB1->terrainBlockList.size(); i++)
{
    LOSTestsResultsStream << "\n---------- Block_ " << TDB1->terrainBlockList[i].lineIndex << " _ " << TDB1->terrainBlockList[i].columnIndex << " ----------------\n";
    LOSTestsResultsStream << "Lat\Lon\tDB#1 Elev\tDB#2 Elev\tElev Diff\tAbsolute Elev Diff\n";
    for(int j = 0 ; j < TDB1->terrainBlockList[i].eyePointGroundLLEList.size(); j++)
    {
        if(TDB1->terrainBlockList[i].eyePointGroundLLEList[j].x() == TDB2->terrainBlockList[i].eyePointGroundLLEList[j].x())
        {
            if(TDB1->terrainBlockList[i].eyePointGroundLLEList[j].y() == TDB2->terrainBlockList[i].eyePointGroundLLEList[j].y())
            {
                LOSTestsResultsStream << QString::number(TDB1->terrainBlockList[i].eyePointGroundLLEList[j].x(), 'f', 6) << "\t";
                LOSTestsResultsStream << QString::number(abs(TDB1->terrainBlockList[i].eyePointGroundLLEList[j].y()) - TDB2->terrainBlockList[i].eyePointGroundLLEList[j].y()), 'f', 6) << "\t";
                LOSTestsResultsStream << QString::number(abs(TDB1->terrainBlockList[i].eyePointGroundLLEList[j].z()) - TDB2->terrainBlockList[i].eyePointGroundLLEList[j].z()), 'f', 6) << "\t";
                LOSTestsResultsStream << QString::number(comparData.elevDiffList[comparData.blockAnalysisData[i].blockElevDiffIndexList[j]], 'f', 6) << "\t";
                LOSTestsResultsStream << QString::number(comparData.terrainBlockList[comparData.blockAnalysisData[i].blockElevDiffIndexList[j]], 'f', 6) << "\n";
            }
        }
    }
}

183
else
{
    LOSTestsResultsStream << "***Error: insufficient data for comparison: there is no correspondence in latitude and longitude.***";
}
}
LOSTestsResultsStream << "\nMin elev diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Max elev diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Average elev diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Dev elev diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Err elev diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevDiffStdErr, 'f', 6) << "\n";
LOSTestsResultsStream << "Min elev absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevAbsDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Max elev absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevAbsDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Average elev absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevAbsDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Dev elev absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevAbsDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Err elev absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockElevAbsDiffStdErr, 'f', 6) << "\n";
L0STestsResultsStream << "\n\nOverall Min elev diff:\t" <<
QString::number(comparData.overallElevationDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Max elev diff:\t" <<
QString::number(comparData.overallElevationDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Average elev diff:\t" <<
QString::number(comparData.overallElevationDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Dev elev diff:\t" <<
QString::number(comparData.overallElevationDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Err elev diff:\t" <<
QString::number(comparData.overallElevationDiffStdErr, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Min elev absolute diff:\t" <<
QString::number(comparData.overallElevationAbsDiffMin, 'f', 6) << "\n";


LOSTestsResultsStream << "Overall Max elev absolute diff:\n" << 
QString::number(comparData.overallElevationAbsDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Average elev absolute diff:\n" << 
QString::number(comparData.overallElevationAbsDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Dev elev absolute diff:\n" << 
QString::number(comparData.overallElevationAbsDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Err elev absolute diff:\n" << 
QString::number(comparData.overallElevationAbsDiffStdErr, 'f', 6) << "\n";

for(int i = 0; i < TDB1->terrainBlockList.size(); i++)
{
    int LOSRayIndexIter = 0;
    LOSTestsResultsStream << "\n\n" 
    TDB1->terrainBlockList[i].lineIndex << "_" << TDB1-> 
    terrainBlockList[i].columnIndex << "_" 
    LOSTestsResultsStream << "DB#1 Block roughness:\n" << 
    QString::number(TDB1->terrainBlockList[i].blockNormalVectorStdDev.length(), 'f', 6) << "\n";
    LOSTestsResultsStream << "DB#2 Block roughness:\n" << 
    QString::number(TDB2->terrainBlockList[i].blockNormalVectorStdDev.length(), 'f', 6) << "\n";

    LOSTestsResultsStream << "Lat\tLon\tHeight AGL\tAzimuth angle\tPitch angle\tDB#1 Ray Lenght\tDB#2 Ray Lenght Diff\tRay Lenght Abs Diff\t"
    LOSTestsResultsStream << "Hit BB(DB#1)?:\tHit BB(DB#2)?:\tBB Hit Corresp.\n";
    for(int j = 0 ; j < TDB1-> 
    terrainBlockList[i].LOSDataSetList.size(); j++)
    {
        for(int k = 0; k < TDB1-
    terrainBlockList[i].LOSDataSetList[j].dirXYZList.size(); k++)
        {
            QCoreApplication::processEvents();
            QVector3D TDB1EyePosLLE, TDB2EyePosLLE;
            TDB1EyePosLLE = TDB1->convertXYZtoXYZ(TDB1-
            terrainBlockList[i].LOSDataSetList[j].eyePosXYZ);
            TDB2EyePosLLE = TDB2->convertXYZtoXYZ(TDB2-
            terrainBlockList[i].LOSDataSetList[j].eyePosXYZ);
            if((TDB1EyePosLLE.x() == TDB2EyePosLLE.x()) && 
            (TDB1EyePosLLE.y() == TDB2EyePosLLE.y()))
            {
                if (TDB1-
            terrainBlockList[i].LOSDataSetList[j].actualHeightAGL == TDB2-
            terrainBlockList[i].LOSDataSetList[j].actualHeightAGL)
if (TDB1->terrainBlockList[i].LOSDatasetList[j].azimuthAngleList[k] == TDB2->terrainBlockList[i].LOSDatasetList[j].azimuthAngleList[k]) && (TDB1->terrainBlockList[i].LOSDatasetList[j].pitchAngleList[k] == TDB2->terrainBlockList[i].LOSDatasetList[j].pitchAngleList[k])
{
    LOSTestsResultsStream << QStrin::number(TDB1EyePosLLE.x(), 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB1EyePosLLE.y(), 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB1->terrainBlockList[i].LOSDatasetList[j].actualHeightAGL, 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB1->terrainBlockList[i].LOSDatasetList[j].azimuthAngleList[k], 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB1->terrainBlockList[i].LOSDatasetList[j].pitchAngleList[k], 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB1->terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList[k], 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(TDB2->terrainBlockList[i].LOSDatasetList[j].eyeToIntersectPtDistList[k], 'f', 6) << "\t";
    LOSTestsResultsStream << QStrin::number(abs(comparData.LOSRayDiffList[comparData.blockAnalysisData[i].blockLOSRayDiffIndexList[LOSRayIndexIter]]), 'f', 6) << "\t";
    bool isRayinTDB1IntersectingBB, corresp;
    isRayinTDB1IntersectingBB = TDB1->terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList[k];
    isRayinTDB2IntersectingBB = TDB2->terrainBlockList[i].LOSDatasetList[j].isRayIntersectingBBList[k];
    corresp = ((isRayinTDB1IntersectingBB && !isRayinTDB2IntersectingBB) || (!isRayinTDB1IntersectingBB && isRayinTDB2IntersectingBB))?
        true : false;
    LOSTestsResultsStream << isRayinTDB1IntersectingBB << "\t";
    LOSTestsResultsStream << isRayinTDB2IntersectingBB << "\t";
    LOSTestsResultsStream << corresp << "\n";
    LOSRayIndexIter++;
}
else
{
LOSTestsResultsStream << "***Error: insufficient data for comparison: there is no correspondence in azimuth and pitch angles.***";
}
else
{
    qDebug("Error: TDB1 > terrainBlockList[%d].LOSDatasetList[%d].actualHeightAGL = %f", i, j, TDB1 > terrainBlockList[i].LOSDatasetList[j].actualHeightAGL);
    qDebug("However, TDB2 > terrainBlockList[%d].LOSDatasetList[%d].actualHeightAGL = %f", i, j, TDB2 > terrainBlockList[i].LOSDatasetList[j].actualHeightAGL);
    LOSTestsResultsStream << "***Error: insufficient data for comparison: there is no correspondence AGL height.***";
}
else
{
    qDebug("Error: TDB1EyePosLLE = (%f, %f, %f); TDB1EyePosLLE = (%f, %f, %f)", TDB1EyePosLLE.x(), TDB1EyePosLLE.y(), TDB1EyePosLLE.z(), TDB2EyePosLLE.x(), TDB2EyePosLLE.y(), TDB2EyePosLLE.z());
    LOSTestsResultsStream << "***Error: insufficient data for comparison: there is no correspondence in latitude and longitude.***";
}
}

LOSTestsResultsStream << "\nMin LOS ray length diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Max LOS ray length diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Average LOS ray length diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Dev LOS ray length diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Err LOS ray length diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayDiffStdErr, 'f', 6) << "\n";
LOSTestsResultsStream << "Min LOS ray length absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayAbsDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Max LOS ray length absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOSRayAbsDiffMax, 'f', 6) << "\n";

187
LOSTestsResultsStream << "Average LOS ray length absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOS Ray Abs Diff Avg, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Dev LOS ray length absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOS Ray Abs Diff Std Dev, 'f', 6) << "\n";
LOSTestsResultsStream << "Std Err LOS ray length absolute diff in this block:\t" <<
QString::number(comparData.blockAnalysisData[i].blockLOS Ray Abs Diff Std Err, 'f', 6) << "\n";
}
LOSTestsResultsStream << "\n\n---------------------------------------
--- Overall test (LOS test list ordered by absolute ray length difference) --
---------------------------------------\n"
LOSTestsResultsStream << "Lat\tLon\tH\n\tRich\tDB#1 Ray Lenght\tDB#2 Ray Lenght\tRay Lenght Diff\tRay Lenght Abs Diff\tt\tHit BB(DB#1)?\tHit BB(DB#2)?\tBB Hit Corresp.\n";
for(int i = 0; i < comparData.LOS Ray Diff Unity Ordered List.size(); i++)
{
    QCoreApplication::processEvents();
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].groundProjLLE.x(), 'f', 6) << "\t";
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].groundProjLLE.y(), 'f', 6) << "\t";
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].AGLHeight, 'f', 6) << "\t";
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].azimuth, 'f', 6) << "\t";
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].pitch, 'f', 6) << "\t";
    QVector3D auxVector;
    auxVector =
comparData.LOS Ray Diff Unity Ordered List[i].TDB1RayXYZ.intersecPtXYZ -
comparData.LOS Ray Diff Unity Ordered List[i].TDB1RayXYZ.eyePosXYZ;
    LOSTestsResultsStream << QString::number(auxVector.length(), 'f', 6) << "\t";
    auxVector =
comparData.LOS Ray Diff Unity Ordered List[i].TDB2RayXYZ.intersecPtXYZ -
comparData.LOS Ray Diff Unity Ordered List[i].TDB2RayXYZ.eyePosXYZ;
    LOSTestsResultsStream << QString::number(auxVector.length(), 'f', 6) << "\t";
    LOSTestsResultsStream <<
QString::number(comparData.LOS Ray Diff Unity Ordered List[i].diff LOS, 'f', 6) << "\t";
"}
LOSTestsResultsStream << QString::number(comparData.LOSRayDiffUnityOrderedList[i].absDiffLOS, 'f', 6) << "\t";
LOSTestsResultsStream << comparData.LOSRayDiffUnityOrderedList[i].TDB1RayXYZ.isRayIntersectingBB << "\t";
LOSTestsResultsStream << comparData.LOSRayDiffUnityOrderedList[i].TDB2RayXYZ.isRayIntersectingBB << "\t";
LOSTestsResultsStream << comparData.LOSRayDiffUnityOrderedList[i].BBIntersectingCorresp << "\n";
}
LOSTestsResultsStream << "\n\nOverall Min LOS ray length diff:\n" << QString::number(comparData.overallLOSRayDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Max LOS ray length diff:\n" << QString::number(comparData.overallLOSRayDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Average LOS ray length diff:\n" << QString::number(comparData.overallLOSRayDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Dev LOS ray length diff:\n" << QString::number(comparData.overallLOSRayDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Err LOS ray length diff:\n" << QString::number(comparData.overallLOSRayDiffStdErr, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Min LOS ray length absolute diff:\n" << QString::number(comparData.overallLOSRayAbsDiffMin, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Max LOS ray length absolute diff:\n" << QString::number(comparData.overallLOSRayAbsDiffMax, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Average LOS ray length absolute diff:\n" << QString::number(comparData.overallLOSRayAbsDiffAvg, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Dev LOS ray length absolute diff:\n" << QString::number(comparData.overallLOSRayAbsDiffStdDev, 'f', 6) << "\n";
LOSTestsResultsStream << "Overall Std Err LOS ray length absolute diff:\n" << QString::number(comparData.overallLOSRayAbsDiffStdErr, 'f', 6) << "\n";
LOSTestsResultsStreamFile.close();
return true;
}

tdbanalyzer.h

#ifndef TDBANALYZER_H
#define TDBANALYZER_H
#define PI 3.14159265358979323846

#include <QObject>
#include <QtCore>
#include <QVector3D>
#include <QMatrix4x4>

189
```cpp
#include <math.h>
#include <mgapiall.h>
#include "util.h"

class TDBAnalyzer : public QObject
{
    Q_OBJECT
public:
    TDBAnalyzer(QObject *parent = 0);
    ~TDBAnalyzer();

    int LODCount;
    int LODToBeAnalyzed;
    QVector<oyLOD> LODList;
    QVector<oyTerrainBlock> terrainBlockList;
    QString fileName;
    mgprojection dbProjection; //presagis

    void CreateBoundingBoxes(double maxLat, double minLat, double maxLon, double minLon, double maxHeight, double minHeight);
    void CreateBlocks(int lines, int columns);
    void CreateBlocks(QVector<oyTerrainBlock> blockList);
    void FillSkyEyePointsList(int eyePointsPerBlock);
    void FillEyePointsList(int eyePointsPerBlock, int initialBlock, int offset);
    void FillEyePointsList(QVector<oyTerrainBlock> blockList, int initialBlock, int offset);
    QVector3D ProbeTerrainLLE(double lat, double lon, double height);
    bool LineTriangleIntersection(QVector3D eyePos, QVector3D dir, QVector3D A, QVector3D B, QVector3D C, QVector3D &intersection);
    void CreateDirVecList(double startingHeight, double heightInc, int heightIncCount, double azimuthIncCount, double pitchInc, int pitchIncCount);
    void CalculateLOS(int initialBlock, int offset);
    QVector3D convertXYZtoLLE(QVector3D vecXYZ);
    QVector3D convertLLEtoXYZ(QVector3D vecLLE);
    void compareDatabases(TDBAnalyzer * TDBforComparison);

signals:
    void error(QString error);
    void infoReady(QString information);
    void updateProgress(int value);
public slots:
};
#endif // TDBANALYZER_H
```

tbanalyzer.cpp

```cpp
#include "tdbanalyzer.h"

TDBAnalyzer::TDBAnalyzer(QObject *parent) :
    QObject(parent)
```
TDBAnalyzer::~TDBAnalyzer()
{
}

void TDBAnalyzer::CreateBlocks(int lines, int columns)
{
    double latRange, latIncrement, lonRange, lonIncrement;
    latRange = LODList[LODToBeAnalyzed].maxLat -
               LODList[LODToBeAnalyzed].minLat;
    lonRange = LODList[LODToBeAnalyzed].maxLon -
               LODList[LODToBeAnalyzed].minLon;
    latIncrement = latRange / lines;
    lonIncrement = lonRange / columns;

    for(int i = 0; i < lines; i++)
    {
        for(int j = 0; j < columns; j++)
        {
            oyTerrainBlock currentBlock;
            currentBlock.LOD = LODToBeAnalyzed;
            currentBlock.lineIndex = i;
            currentBlock.columnIndex = j;

            currentBlock.westBoundDegrees = LODList[LODToBeAnalyzed].minLon +
                                             lonIncrement * j;
            currentBlock.eastBoundDegrees = LODList[LODToBeAnalyzed].minLon +
                                             lonIncrement * (j + 1);
            currentBlock.southBoundDegrees = LODList[LODToBeAnalyzed].minLat +
                                              latIncrement * i;
            currentBlock.northBoundDegrees = LODList[LODToBeAnalyzed].minLat +
                                              latIncrement * (i + 1);
            currentBlock.centerLat = (currentBlock.northBoundDegrees +
                                       currentBlock.southBoundDegrees)/2;
            currentBlock.centerLon = (currentBlock.westBoundDegrees +
                                      currentBlock.eastBoundDegrees)/2;

            QVector3D corner;
            corner =
            convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                         currentBlock.westBoundDegrees,
                                         LODList[LODToBeAnalyzed].maxHeight));
            currentBlock.cornersXYZ.push_back(corner);
            corner =
            convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                         currentBlock.westBoundDegrees,
                                         LODList[LODToBeAnalyzed].maxHeight));
            currentBlock.cornersXYZ.push_back(corner);
            corner =
            convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                         currentBlock.eastBoundDegrees,
                                         LODList[LODToBeAnalyzed].maxHeight));
            currentBlock.cornersXYZ.push_back(corner);
            corner =
            convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                         currentBlock.eastBoundDegrees,
                                         LODList[LODToBeAnalyzed].maxHeight));
            currentBlock.cornersXYZ.push_back(corner);
        }
    }
}
corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees, currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
currentBlock.cornersXYZ.push_back(corner);
corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees, currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
currentBlock.cornersXYZ.push_back(corner);
corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees, currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
currentBlock.cornersXYZ.push_back(corner);
corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees, currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
currentBlock.cornersXYZ.push_back(corner);
for(int k = 0; k < currentBlock.cornersXYZ.size(); k++)
{
    if (k == 0)
    {
        currentBlock.maxX = currentBlock.cornersXYZ[0].x();
        currentBlock.minX = currentBlock.cornersXYZ[0].x();
        currentBlock.maxY = currentBlock.cornersXYZ[0].y();
        currentBlock.minY = currentBlock.cornersXYZ[0].y();
        currentBlock.maxZ = currentBlock.cornersXYZ[0].z();
        currentBlock.minZ = currentBlock.cornersXYZ[0].z();
    }
    else
    {
        currentBlock.maxX = (currentBlock.maxX <
        currentBlock.cornersXYZ[k].x()) ? currentBlock.cornersXYZ[k].x() :
        currentBlock.maxX;
        currentBlock.maxY = (currentBlock.maxY <
        currentBlock.cornersXYZ[k].y()) ? currentBlock.cornersXYZ[k].y() :
        currentBlock.maxY;
        currentBlock.maxZ = (currentBlock.maxZ <
        currentBlock.cornersXYZ[k].z()) ? currentBlock.cornersXYZ[k].z() :
        currentBlock.maxZ;
        currentBlock.minX = (currentBlock.minX >
        currentBlock.cornersXYZ[k].x()) ? currentBlock.cornersXYZ[k].x() :
        currentBlock.minX;
        currentBlock.minY = (currentBlock.minY >
        currentBlock.cornersXYZ[k].y()) ? currentBlock.cornersXYZ[k].y() :
        currentBlock.minY;
        currentBlock.minZ = (currentBlock.minZ >
        currentBlock.cornersXYZ[k].z()) ? currentBlock.cornersXYZ[k].z() :
        currentBlock.minZ;
    }
}
QVector3D polyNormalAverage(0, 0, 0);
for(int k = 0; k < LODList[LODToBeAnalyzed].polygons.size(); k++)
{
    // verify if the polygon center of mass is inside the block
if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.x() < currentBlock.maxX) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.x() > currentBlock.minX))
{
    if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.y() < currentBlock.maxY) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.y() > currentBlock.minY))
    {
        if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.z() < currentBlock.maxZ) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.z() > currentBlock.minZ))
        {
            currentBlock.polyIndexList.push_back(k);
        }
    }
}
if (currentBlock.polyIndexList.size() > 0)
{
    polyNormalAverage = polyNormalAverage / currentBlock.polyIndexList.size();
    QVector3D polyNormalStdDev;
    for(int k = 0; k < currentBlock.polyIndexList.size(); k++)
    {
        QPolygon currentPoly = LODList[LODToBeAnalyzed].polygons[currentBlock.polyIndexList[k]];
        polyNormalStdDev.setX(pow(currentPoly.Normal.x() - currentBlock.blockNormalVectorAverage.x(), 2));
        polyNormalStdDev.setY(pow(currentPoly.Normal.y() - currentBlock.blockNormalVectorAverage.y(), 2));
        polyNormalStdDev.setZ(pow(currentPoly.Normal.z() - currentBlock.blockNormalVectorAverage.z(), 2));
    }
    currentBlock.blockNormalVectorStdDev = polyNormalStdDev;
    terrainBlockList.push_back(currentBlock);
}
else
{
    emit error("This terrain block does not contain any polygon!");
    qDebug("This terrain block does not contain any polygon!");
}
void TDBAnalyzer::CreateBlocks(QVector<oyTerrainBlock> blockList)
{
    terrainBlockList.clear();
    for(int i = 0; i < blockList.size(); i++)
    {
        oyTerrainBlock currentBlock;
        currentBlock.lineIndex = blockList[i].lineIndex;
        currentBlock.columnIndex = blockList[i].columnIndex;
        currentBlock.northBoundDegrees = blockList[i].northBoundDegrees;
        currentBlock.southBoundDegrees = blockList[i].southBoundDegrees;
        currentBlock.westBoundDegrees = blockList[i].westBoundDegrees;
        currentBlock.eastBoundDegrees = blockList[i].eastBoundDegrees;

        QVector3D corner;
        corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                            currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].maxHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                            currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].maxHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                            currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].maxHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                            currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].maxHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                            currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                            currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.northBoundDegrees,
                                            currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                            currentBlock.eastBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
        currentBlock.cornersXYZ.push_back(corner);
        corner = convertLLEtoXYZ(QVector3D(currentBlock.southBoundDegrees,
                                            currentBlock.westBoundDegrees, LODList[LODToBeAnalyzed].minHeight));
        currentBlock.cornersXYZ.push_back(corner);
        for(int k = 0; k < currentBlock.cornersXYZ.size(); k++)
        {
            if (k == 0)
            {
                currentBlock.maxX = currentBlock.cornersXYZ[0].x();
                currentBlock.minX = currentBlock.cornersXYZ[0].x();
                currentBlock.maxY = currentBlock.cornersXYZ[0].y();
                currentBlock.minY = currentBlock.cornersXYZ[0].y();
                currentBlock.maxZ = currentBlock.cornersXYZ[0].z();
                currentBlock.minZ = currentBlock.cornersXYZ[0].z();
            }
            else
            {
            }
        }
    }
}
currentBlock.maxX = (currentBlock.maxX < currentBlock.cornersXYZ[k].x()) ? currentBlock.cornersXYZ[k].x() : currentBlock.maxX;
currentBlock.maxY = (currentBlock.maxY < currentBlock.cornersXYZ[k].y()) ? currentBlock.cornersXYZ[k].y() : currentBlock.maxY;
currentBlock.maxZ = (currentBlock.maxZ < currentBlock.cornersXYZ[k].z()) ? currentBlock.cornersXYZ[k].z() : currentBlock.maxZ;
currentBlock.minX = (currentBlock.minX > currentBlock.cornersXYZ[k].x()) ? currentBlock.cornersXYZ[k].x() : currentBlock.minX;
currentBlock.minY = (currentBlock.minY > currentBlock.cornersXYZ[k].y()) ? currentBlock.cornersXYZ[k].y() : currentBlock.minY;
currentBlock.minZ = (currentBlock.minZ > currentBlock.cornersXYZ[k].z()) ? currentBlock.cornersXYZ[k].z() : currentBlock.minZ;
}
}
QVector3D polyNormalAverage(0, 0, 0);
for(int k = 0; k < LODList[LODToBeAnalyzed].polygons.size(); k++)
{
    // verify if the polygon center of mass is inside the block
    if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.x() < currentBlock.maxX) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.x() > currentBlock.minX))
    {
        if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.y() < currentBlock.maxY) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.y() > currentBlock.minY))
        {
            if ((LODList[LODToBeAnalyzed].polygons[k].Centroid.z() < currentBlock.maxZ) && (LODList[LODToBeAnalyzed].polygons[k].Centroid.z() > currentBlock.minZ))
            {
                polyNormalAverage += LODList[LODToBeAnalyzed].polygons[k].Normal;
            }
        }
    }
}
if (currentBlock.polyIndexList.size() > 0)
{
    polyNormalAverage = polyNormalAverage / currentBlock.polyIndexList.size();
    currentBlock.blockNormalVectorAverage = polyNormalAverage;
}
QVector3D polyNormalStdDev(0, 0, 0);
for(int k = 0; k < currentBlock.polyIndexList.size(); k++)
{
    curPoly currentPoly;
currentPoly = LODList[LODToBeAnalyzed].polygons[currentBlock.polyIndexList[k]];
    polyNormalStdDev.setX(polyNormalStdDev.x() +
    pow(currentPoly.Normal.x() - currentBlock.blockNormalVectorAverage.x(), 2));
    polyNormalStdDev.setY(polyNormalStdDev.y() +
    pow(currentPoly.Normal.y() - currentBlock.blockNormalVectorAverage.y(), 2));
    polyNormalStdDev.setZ(polyNormalStdDev.z() +
    pow(currentPoly.Normal.z() - currentBlock.blockNormalVectorAverage.z(), 2));
}    polyNormalStdDev.setX(sqrt(polyNormalStdDev.x() /
    currentBlock.polyIndexList.size()));
    polyNormalStdDev.setY(sqrt(polyNormalStdDev.y() /
    currentBlock.polyIndexList.size()));
    polyNormalStdDev.setZ(sqrt(polyNormalStdDev.z() /
    currentBlock.polyIndexList.size()));
    currentBlock.blockNormalVectorStdDev = polyNormalStdDev;
    currentBlock.OD = LODToBeAnalyzed;
    terrainBlockList.push_back(currentBlock);
}    emit error("This terrain block does not contain any polygon!");
    qDebug("This terrain block does not contain any polygon!");
}

void TDBAnalyzer::FillSkyEyePointsList(int eyePointsPerBlock) {
    double totalEyepoints;
    totalEyepoints = terrainBlockList.size() * eyePointsPerBlock;
    for(int i = 0; i < terrainBlockList.size(); i++) {
        double latRange, latIncrement, lonRange, lonIncrement;
        latRange = terrainBlockList[i].northBoundDegrees -
        terrainBlockList[i].southBoundDegrees;
        lonRange = terrainBlockList[i].eastBoundDegrees -
        terrainBlockList[i].westBoundDegrees;
        latIncrement = latRange / (sqrt((double)eyePointsPerBlock) + 1);
        lonIncrement = lonRange / (sqrt((double)eyePointsPerBlock) + 1);
        for (int j = 1; j < (sqrt((double)eyePointsPerBlock) + 1); j++) {// longitude iterator
            for (int k = 1; k < (sqrt((double)eyePointsPerBlock) + 1); k++) {// latitude iterator
                QVector3D eyeSkyLLE(0, 0, 0), eyeSkyXYZ(0, 0, 0);
                eyeSkyLLE.setX(terrainBlockList[i].southBoundDegrees + k *
                latIncrement);
                eyeSkyLLE.setY(terrainBlockList[i].westBoundDegrees + j *
                lonIncrement);
                eyeSkyLLE.setZ(LODList[LODToBeAnalyzed].maxHeight + 20);
                eyeSkyXYZ = convertLLEtoXYZ(QVector3D(eyeSkyLLE.x(),
                eyeSkyLLE.y(), eyeSkyLLE.z()));
            }
        }
    }
}
terrainBlockList[i].eyePointSkyLLEList.push_back(eyeSkyLLE);
terrainBlockList[i].eyePointSkyXYZList.push_back(eyeSkyXYZ);
} else {
    terrainBlockList[i].EyePointOrthoProjectionCount =
        terrainBlockList[i].eyePointSkyLLEList.size();
}
}

void TDBAnalyzer::FillEyePointsList(int eyePointsPerBlock, int initialBlock, int offset) {
    double totalEyepoints, currentEyepointNumber = 0;
    int currentPercent = 0, oldPercent;
    totalEyepoints = offset * eyePointsPerBlock;

    if ((initialBlock < 0) || (initialBlock > terrainBlockList.size()) ||
        (initialBlock + offset) > terrainBlockList.size()) {
        emit error("Cannot access specified terrain block in the block array: the array range has been exceeded.");
        qDebug("Cannot access specified terrain block in the block array: the array range has been exceeded.");
        return;
    }
    for(int i = initialBlock; i < (initialBlock + offset); i++) {
        for (int j = 0; j < terrainBlockList[i].eyePointSkyLLEList.size(); j++) {
            QVector3D eyeGroundLLE(0, 0, 0), eyeGroundXYZ(0, 0, 0);
            eyeGroundLLE = ProbeTerrainLLE(terrainBlockList[i].eyePointSkyLLEList[j].x(),
                              terrainBlockList[i].eyePointSkyLLEList[j].y(),
                              terrainBlockList[i].eyePointSkyLLEList[j].z());
            eyeGroundXYZ = convertLLEtoXYZ(eyeGroundLLE);
            terrainBlockList[i].eyePointGroundLLEList.push_back(eyeGroundLLE);
            terrainBlockList[i].eyePointGroundXYZList.push_back(eyeGroundXYZ);

            currentEyepointNumber++;
            oldPercent = currentPercent;
            currentPercent = (currentEyepointNumber/totalEyepoints) * 100;
            if((currentPercent - oldPercent) == 1) {
                emit updateProgress(currentPercent);
            }
        }
    }
}

void TDBAnalyzer::FillEyePointsList(QVector<oyTerrainBlock> blockList, int initialBlock, int offset)
{  
double totalEyepoints, currentEyepointNumber = 0;
int currentPercent = 0, oldPercent;
totalEyepoints = offset * blockList[0].eyePointSkyLLEList.size();

if ((initialBlock < 0) || (initialBlock > terrainBlockList.size()) ||  
    (initialBlock + offset) > terrainBlockList.size())
{
    emit error("Cannot access specified terrain block in the block array: the array range has been exceeded.");
    qDebug("Cannot access specified terrain block in the block array: the array range has been exceeded.");
    return;
}
if (terrainBlockList.size() == blockList.size())
{
    for(int i = initialBlock; i < (initialBlock + offset); i++)
    {
        for(int j = 0; j < blockList[i].eyePointSkyLLEList.size(); j++)
        {
            QVector3D currentEyepointSkyLLE, currentEyepointSkyXYZ, eyeGroundLLE, eyeGroundXYZ;
            currentEyepointSkyLLE = blockList[i].eyePointSkyLLEList[j];
            currentEyepointSkyLLE.setZ(LODList[LODToBeAnalyzed].maxHeight + 20);
            currentEyepointSkyXYZ = convertLLEtoXYZ(currentEyepointSkyLLE);
            terrainBlockList[i].eyePointSkyLLEList.push_back(currentEyepointSkyLLE);
            terrainBlockList[i].eyePointSkyXYZList.push_back(currentEyepointSkyXYZ);
            eyeGroundLLE = ProbeTerrainLLE(currentEyepointSkyLLE.x(), currentEyepointSkyLLE.y(), currentEyepointSkyLLE.z());
            eyeGroundXYZ = convertLLEtoXYZ(eyeGroundLLE);
            terrainBlockList[i].eyePointGroundLLEList.push_back(eyeGroundLLE);
            terrainBlockList[i].eyePointGroundXYZList.push_back(eyeGroundXYZ);
            currentEyepointNumber++;
            oldPercent = currentPercent;
            currentPercent = (currentEyepointNumber/totalEyepoints) * 100;
            if((currentPercent - oldPercent) == 1)
            {
                emit updateProgress(currentPercent);
            }
        }
    }
}
else
{  

emit error("Error: The size of the block list of the TDB#1 is different than the size of the terrain block list of the TDB#2.");
qDebug("Error: The size of the block list of the TDB#1 is different than the size of the terrain block list of the TDB#2.");
}

QVector3D TDBAnalyzer::ProbeTerrainLLE(double lat, double lon, double height) {
    QVector3D terrainProblePointLLE, terrainProblePointXYZ, floatingEyePointXYZ, downwardDirection, cameraUpDirection, cameraLookAt;
    QVector3D auxVector;
    QMatrix4x4 vMatrix, pMatrix, vpMatrix;
    QVector4D testFrustrum;
    QVector<int> testedPolyIndices;
    double frustrumSize = 200;
    double heightRange;
    bool intersectionFound = false, testThisPoly = false;

    heightRange = LODList[LODToBeAnalyzed].maxHeight - LODList[LODToBeAnalyzed].minHeight + 25;

    floatingEyePointXYZ = convertLLEtoXYZ(QVector3D(lat, lon, height));
    auxVector = convertLLEtoXYZ(QVector3D(lat, lon, height - 10));
    downwardDirection = auxVector - floatingEyePointXYZ;
    downwardDirection.normalize();
    cameraLookAt = floatingEyePointXYZ + 10 * downwardDirection;
    auxVector = convertLLEtoXYZ(QVector3D(lat + 0.01, lon, height));
    cameraUpDirection = auxVector - floatingEyePointXYZ;
    cameraUpDirection.normalize();

    while (!intersectionFound)
    {
        vMatrix.setToIdentity();
        pMatrix.setToIdentity();
        vMatrix.lookAt(floatingEyePointXYZ, cameraLookAt, cameraUpDirection);
        pMatrix.ortho(-frustrumSize, frustrumSize, -frustrumSize, frustrumSize, 1, heightRange);
        vpMatrix = pMatrix * vMatrix;

        for (int i = 0; i < LODList[LODToBeAnalyzed].polygons.size(); i++)
        {
            if (testedPolyIndices.contains(i))
            {
                continue; // this polygon has been tested before
            }

            testThisPoly = false;
            testFrustrum.setX(LODList[LODToBeAnalyzed].polygons[i].Centroid.x());
            testFrustrum.setY(LODList[LODToBeAnalyzed].polygons[i].Centroid.y());
testFrustrum.setZ(LODList[LODToBeAnalyzed].polygons[i].Centroid.z());
testFrustrum.setW(1);
testFrustrum = vpMatrix * testFrustrum;
testFrustrum = testFrustrum / testFrustrum.w();

    if ((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) &&
        (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
    {
        testThisPoly = true;
    }
    else
    {
        for (int j = 0; j < LODList[LODToBeAnalyzed].polygons[i].vertices.size(); j++)
        {
            testFrustrum.setX(LODList[LODToBeAnalyzed].polygons[i].vertices[j].x());
            testFrustrum.setY(LODList[LODToBeAnalyzed].polygons[i].vertices[j].y());
            testFrustrum.setZ(LODList[LODToBeAnalyzed].polygons[i].vertices[j].z());
            testFrustrum.setW(1);
            testFrustrum = vpMatrix * testFrustrum;
            if ((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) &&
                (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
            {
                testThisPoly = true;
                break;
            }
        }
    }
    if (testThisPoly)
    {
        QVector3D A, B, C;
        if (LODList[LODToBeAnalyzed].polygons[i].vertices.size() != 3)
        {
            emit error("This polygon is not a triangle");
            qDebug("This polygon is not a triangle");
        }
        else
        {
            A = LODList[LODToBeAnalyzed].polygons[i].vertices[0];
            B = LODList[LODToBeAnalyzed].polygons[i].vertices[1];
            C = LODList[LODToBeAnalyzed].polygons[i].vertices[2];
            intersectionFound = LineTriangleIntersection(floatingEyePointXYZ, downwardDirection, A, B, C, terrainProblePointXYZ);
            if(intersectionFound)
            {
                terrainProblePointLLE = convertXYZtoLLE(terrainProblePointXYZ);
                break;
            }
if (!intersectionFound)
{
    // Increase frustrum size to test more polygons.
    frustrumSize += 200;
}
return terrainProblePointLLE;

bool TDBAnalyzer::LineTriangleIntersection(QVector3D eyePos, QVector3D dir, QVector3D A, QVector3D B, QVector3D C, QVector3D &intersection)
{
    QVector3D A_minus_B, A_minus_C, A_minus_P;
    QVector3D intersectPt, A_minus_intPt, B_minus_intPt, C_minus_intPt;
    double numerator, denominator, t;
    double theta1, theta2, theta3, cosine, sumAngles;
    double two_times_Pi = 4 * asin(1.0);
    
    A_minus_B = A - B;
    A_minus_C = A - C;
    A_minus_P = A - eyePos;
    
    numerator = A_minus_P.x() * A_minus_B.y() * A_minus_C.z();
    numerator += A_minus_B.x() * A_minus_C.y() * A_minus_P.z();
    numerator += A_minus_P.y() * A_minus_B.z() * A_minus_C.x();
    numerator -= A_minus_C.x() * A_minus_B.y() * A_minus_P.z();
    numerator -= A_minus_B.x() * A_minus_P.y() * A_minus_C.z();
    numerator -= A_minus_P.x() * A_minus_C.y() * A_minus_B.z();
    
    denominator = dir.x() * A_minus_B.y() * A_minus_C.z();
    denominator += A_minus_B.x() * A_minus_C.y() * dir.z();
    denominator += dir.y() * A_minus_B.z() * A_minus_C.x();
    denominator -= A_minus_C.x() * A_minus_B.y() * dir.z();
    denominator -= A_minus_B.x() * dir.y() * A_minus_C.z();
    denominator -= dir.x() * A_minus_C.y() * A_minus_B.z();
    
    if (denominator != 0)
    {
        t = numerator / denominator;
        if (t < 0)
        {
            // the line intersected a polygon behind the eye-point
            return false;
        }
        
        intersectPt = eyePos + t * dir;
        A_minus_intPt = A - intersectPt;
        if (A_minus_intPt.length() < 0.0001)
        {
            intersection = A;
            qDebug("Intersection point is A\n");
        }
    }
}
return true;
}
B_minus_intPt = B - intersectPt;
if (B_minus_intPt.length() < 0.0001)
{
    intersection = B;
    qDebug("Intersection point is B\n");
    return true;
}
C_minus_intPt = C - intersectPt;
if (C_minus_intPt.length() < 0.0001)
{
    intersection = C;
    qDebug("Intersection point is C\n");
    return true;
}
cosine = QVector3D::dotProduct(A_minus_intPt, B_minus_intPt) / 
(A_minus_intPt.length() * B_minus_intPt.length());
theta1 = acos(cosine);
cosine = QVector3D::dotProduct(A_minus_intPt, C_minus_intPt) / 
(A_minus_intPt.length() * C_minus_intPt.length());
theta2 = acos(cosine);
cosine = QVector3D::dotProduct(B_minus_intPt, C_minus_intPt) / 
(B_minus_intPt.length() * C_minus_intPt.Length());
theta3 = acos(cosine);
sumAngles = theta1 + theta2 + theta3;
if(abs(two_times_Pi - sumAngles) < 0.0001)
{
    intersection = intersectPt;
    return true;
}
else
{
    // The point is not inside the triangle
    return false;
}
}
else
{
    // Denominator is zero
    if(abs(numerator) < 0.00001)
    {
        qDebug("Numerator and the denominator are zero!");
        qDebug("The line might belong to the plane defined by the
triangle!");
    }
    return false;
}
}
void TDBAnalyzer::CreateDirVecList(double startingHeight, double heightInc, 
int heightIncCount, int azimuthIncCount, double pitchInc, int pitchIncCount) 
{
    double azimuthInc;
    azimuthInc = 360.0 / ((double) azimuthIncCount);
for (int i = 0; i < terrainBlockList.size(); i++)
{
    for (int j = 0; j < terrainBlockList[i].eyePointGroundLLEList.size(); j++)
    {
        oyLOSTestDataset newLOSdataset;
        for (int k = 0; k <= heightIncCount; k++)
        {
            QVector3D upwardDir, forwardDir, rightDir;
            QVector3D auxVector1, auxVector2, auxVector3, auxVector4;
            QMatrix4x4 rotMatrix;

            newLOSdataset.dirXYZList.clear();
            newLOSdataset.actualHeightAGL = startingHeight + k * heightInc;

            newLOSdataset.eyePosXYZ =
            convertLEtoXYZ(QVector3D(terrainBlockList[i].eyePointGroundLLEList[j].x(),
                                      terrainBlockList[i].eyePointGroundLLEList[j].y(),
                                      terrainBlockList[i].eyePointGroundLLEList[j].z() +
                                      newLOSdataset.actualHeightAGL));

            auxVector1 =
            convertLEtoXYZ(QVector3D(terrainBlockList[i].eyePointGroundLLEList[j].x() +
                                      0.01, terrainBlockList[i].eyePointGroundLLEList[j].y(),
                                      terrainBlockList[i].eyePointGroundLLEList[j].z() +
                                      newLOSdataset.actualHeightAGL));
            upwardDir = auxVector1 - newLOSdataset.eyePosXYZ;
            upwardDir.normalize();

            auxVector1 =
            convertLEtoXYZ(QVector3D(terrainBlockList[i].eyePointGroundLLEList[j].x() +
                                      0.01, terrainBlockList[i].eyePointGroundLLEList[j].y(),
                                      terrainBlockList[i].eyePointGroundLLEList[j].z() +
                                      newLOSdataset.actualHeightAGL));
            forwardDir = auxVector1 - newLOSdataset.eyePosXYZ;
            forwardDir.normalize();

            rightDir = QVector3D::crossProduct(forwardDir, upwardDir);
            rightDir.normalize();

            for (int l = 0; l < azimuthIncCount; l++)
            {
                rotMatrix.setToIdentity();
                rotMatrix.rotate(l * azimuthInc, upwardDir);
                //auxVector1 is the forward vector with new azimuth
                auxVector1 = rotMatrix * forwardDir;
                auxVector1.normalize();

                auxVector2 = QVector3D::crossProduct(auxVector1,
                                                    upwardDir);
                auxVector2.normalize();
                for (int m = 0; m <= pitchIncCount; m++)
                {
                    //auxVector2 is the rightDir vector with respect to
                    auxVector2 =
                
                //auxVector2 is the rightDir vector with respect to
                auxVector2 =
                
                //auxVector2 is the rightDir vector with respect to
                auxVector2 =
            
        //auxVector2 is the rightDir vector with respect to
        auxVector2 =
        auxVector2.normalize();
        for (int m = 0; m <= pitchIncCount; m++)
        {
rotMatrix.setToIdentity();
rotMatrix.rotate(m * pitchInc, auxVector2);
// auxVector3 is auxVector1 with new pitch angle
auxVector3 = rotMatrix * auxVector1;
auxVector3.normalize();
// auxVector4 is the upwardDir with respect to
auxVector3
auxVector4 = QVector3D::crossProduct(auxVector2, auxVector3);
auxVector4.normalize();
newLOSdataset.dirXYZList.push_back(auxVector3);
newLOSdataset.upDirXYZList.push_back(auxVector4);
newLOSdataset.azimuthAngleList.push_back(1 * azimuthInc);
newLOSdataset.pitchAngleList.push_back(m * pitchInc);
if (m != 0)
{
    rotMatrix.setToIdentity();
    rotMatrix.rotate(-1 * m * pitchInc, auxVector2);
    auxVector3 = rotMatrix * auxVector1;
    auxVector3.normalize();
    auxVector4 = QVector3D::crossProduct(auxVector2, auxVector3);
    auxVector4.normalize();
    newLOSdataset.dirXYZList.push_back(auxVector3);
    newLOSdataset.upDirXYZList.push_back(auxVector4);
    newLOSdataset.azimuthAngleList.push_back(1 * azimuthInc);
    newLOSdataset.pitchAngleList.push_back(-1 * m * pitchInc);
}
} } } } 
terrainBlockList[i].LOSDataSetList.append(newLOSDataSet);
}
} 
void TDBAnalyzer::CalculateLOS(int initialBlock, int offset) 
{
    double totalLOSTests, currentLOSTestNumber = 0;
    //totalLOSTests = terrainBlockList.size() * terrainBlockList[0].LOSDataSetList.size() * terrainBlockList[0].LOSDataSetList[0].dirXYZList.size();
    totalLOSTests = offset * terrainBlockList[0].LOSDataSetList[0].dirXYZList.size();
    double rangeX, rangeY, rangeZ, maxRange;
    rangeX = LODList[LODToBeAnalyzed].maxX - LODList[LODToBeAnalyzed].minX;
    rangeY = LODList[LODToBeAnalyzed].maxY - LODList[LODToBeAnalyzed].minY;
    rangeZ = LODList[LODToBeAnalyzed].maxZ - LODList[LODToBeAnalyzed].minZ;
    maxRange = sqrt(pow(rangeX, 2) + pow(rangeY, 2) + pow(rangeZ, 2));

    if ((initialBlock < 0) || (initialBlock > terrainBlockList.size()) ||
        (initialBlock + offset) > terrainBlockList.size()))
{ emit error("Cannot access specified terrain block in the block array: the array range was exceeded."); qDebug("Cannot access specified terrain block in the block array: the array range was exceeded."); return; }

for(int i = initialBlock; i < (offset + initialBlock); i++)
{
    for(int j = 0; j < terrainBlockList[i].LOSDatasetList.size(); j++)
    {
        QVector3D eyePos;
        eyePos = terrainBlockList[i].LOSDatasetList[j].eyePosXYZ;

        for (int k = 0; k < terrainBlockList[i].LOSDatasetList[j].dirXYZList.size(); k++)
        {
            QMatrix4x4 vMatrix, pMatrix, vpMatrix;
            QVector4D testFrustrum;
            QVector3D A, B, C, Centroid, dirVector, upDirVector;
            QVector3D intersectVec, currentDistVec, currentIntersectVec;
            double candidateDist, minorDist = maxRange + 1;
            bool foundAtLeastOneIntersect = false;

            vMatrix.setToIdentity();
            pMatrix.setToIdentity();
            dirVector = terrainBlockList[i].LOSDatasetList[j].dirXYZList[k];
            upDirVector = terrainBlockList[i].LOSDatasetList[j].upDirXYZList[k];

            vMatrix.lookAt(eyePos - 1.1 * dirVector, eyePos + 5 * dirVector, upDirVector);
            pMatrix.ortho(-maxRange/20, maxRange/20, -maxRange/20, maxRange/20, 1, maxRange);
            // setup the projection view matrix
            vpMatrix = pMatrix * vMatrix;
            for (int l = 0; l < LODList[LODToBeAnalyzed].polygons.size(); l++)
            {
                bool testThisPoly = true;

                // check if the triangle is back facing the eyepoint
                if (QVector3D::dotProduct(dirVector, LODList[LODToBeAnalyzed].polygons[l].Normal) > 0)
                {
                    testThisPoly = false;
                }
                if (testThisPoly)
                {
                    if (LODList[LODToBeAnalyzed].polygons[l].vertices.size() == 3)
                    {
A = LODList[LODToBeAnalyzed].polygons[1].vertices[0];
B = LODList[LODToBeAnalyzed].polygons[1].vertices[1];
C = LODList[LODToBeAnalyzed].polygons[1].vertices[2];
Centroid = LODList[LODToBeAnalyzed].polygons[1].Centroid;
else
{
    emit error("This polygon in not a triangle.");
    qDebug("This polygon in not a triangle.");
    return;
}

// eliminate triangles that are farther than the closest intersection point
QVector3D eyeToA_Dist, eyeToB_Dist, eyeToC_Dist;
eyeToA_Dist = A - eyePos;
eyeToB_Dist = B - eyePos;
eyeToC_Dist = C - eyePos;
if((minorDist < eyeToA_Dist.length()) && (minorDist < eyeToB_Dist.length()) && (minorDist < eyeToC_Dist.length()))
{
    testThisPoly = false;
}

if(testThisPoly)
{
    testFrustrum.setX(Centroid.x());
    testFrustrum.setY(Centroid.y());
    testFrustrum.setZ(Centroid.z());
    testFrustrum.setW(1);
    testFrustrum = vpMatrix * testFrustrum;
    testFrustrum = testFrustrum / testFrustrum.w();
    if((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) && (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
    {
        // the centroid is in the frustrum
        // the polygon will be tested
        testThisPoly = true;
    }
    else
    {
        testFrustrum.setX(A.x());
        testFrustrum.setY(A.y());
        testFrustrum.setZ(A.z());
        testFrustrum.setW(1);
        testFrustrum = vpMatrix * testFrustrum;
        testFrustrum = testFrustrum / testFrustrum.w();
    }
}

if ((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) && (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
testFrustrum.w();
    if ((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) && (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
    {
        // B is in the frustrum
        // the polygon will be tested
        testThisPoly = true;
    }
    testFrustrum.setX(C.x());
    testFrustrum.setY(C.y());
    testFrustrum.setZ(C.z());
    testFrustrum.setW(1);
    testFrustrum = vpMatrix * testFrustrum;
    testFrustrum = testFrustrum / testFrustrum.w();
    if ((testFrustrum.x() >= -1) && (testFrustrum.x() <= 1) && (testFrustrum.y() >= -1) && (testFrustrum.y() <= 1))
    {
        // C is in the frustrum
        // the polygon will be tested
        testThisPoly = true;
    }
else
{
    // neither the Centroid, nor A, nor B, nor C are in the frustrum
    // the polygon will not be tested
    // it is more likely that the ray will not intersect this polygon
    testThisPoly = false;
}

if(testThisPoly)
{
    if (LineTriangleIntersection(eyePos, dirVector, A, B, C, currentIntersectVec))
    {

207
ray and one polygon  

to eye-point  

polygons are checked  

// we have found the intersection of between the  
// however, this poly may not be the closest one  
// we should continue our search until all the  

foundAtLeastOneIntersect = true;  
currentDistVec = currentIntersectVec - eyePos;  
candidateDist = currentDistVec.length();  
if( candidateDist < minorDist)  
{  
    minorDist = candidateDist;  
    intersectVec = currentIntersectVec;  
}  
}  
if(l == (LODList[LODToBeAnalyzed].polygons.size() - 1))  
{  
    // we have reached the end of the list of polygons  
    if(foundAtLeastOneIntersect)  
    {  
        terrainBlockList[i].LOSDataSetList[j].intersectPntXYZList.push_back(intersectVec);  
        terrainBlockList[i].LOSDataSetList[j].eyeToIntersectPtDistList.push_back(minorDist);  
        terrainBlockList[i].LOSDataSetList[j].isRayIntersectingBBList.push_back(false);  
    }  
    else  
    {  
        // test collision against bounding volume  
        for(int m = 0; m < LODList[LODToBeAnalyzed].boundingBoxInternal.size(); m++)  
        {  
            A = LODList[LODToBeAnalyzed].boundingBoxInternal[m].vertices[0];  
            B = LODList[LODToBeAnalyzed].boundingBoxInternal[m].vertices[1];  
            C = LODList[LODToBeAnalyzed].boundingBoxInternal[m].vertices[2];  
            if (LineTriangleIntersection(eyePos, dirVector, A, B, C, currentIntersectVec))  
            {  
                // we have found the intersection of  
                between the ray and the bounding box  
                currentDistVec = currentIntersectVec - eyePos;  
                candidateDist = currentDistVec.length();  
                terrainBlockList[i].LOSDataSetList[j].intersectPntXYZList.push_back(currentIntersectVec);  
            }  
        }  
    }  
}  
}  
}  
}  
}  
}  
}  
}  
}  
}  
}  
}
terrainBlockList[i].LOSdatasetList[j].eyeToIntersectPtDistList.push_back(candidateDist);

terrainBlockList[i].LOSdatasetList[j].isRayIntersectingBBList.push_back(true);
    break;
} else 
    if (m == (LODList[LODToBeAnalyzed].boundingBoxInternal.size() - 1))
    {
        emit error("This ray doesn't intersect neither a polygon nor the bounding box");
        qDebug("This ray doesn't intersect neither a polygon nor the bounding box");
        oyProblematicRays currentProblematicRay;
        currentProblematicRay.eyePosXYZ = eyePos;
        currentProblematicRay.rayDirXYZ = dirVector;

        terrainBlockList[i].problematicRayList.push_back(currentProblematicRay);
    }

    currentLOSTestNumber = currentLOSTestNumber + 1;
    emit updateProgress(100 * (currentLOSTestNumber / totalLOSTests));
}
}

void TDBAnalyzer::CreateBoundingBoxes(double maxLat, double minLat, double maxLon, double minLon, double maxHeight, double minHeight)
{
    oyPolygon upperTriangle1, upperTriangle2, bottomTriangle1, bottomTriangle2;
    oyPolygon eastTriangle1, eastTriangle2, westTriangle1, westTriangle2;
    oyPolygon northTriangle1, northTriangle2, southTriangle1, southTriangle2;
    QVector3D corner[8];

    corner[0] = convertLLEtoXYZ(QVector3D(maxLat + 0.001, maxLon + 0.001, maxHeight + 800));
    corner[1] = convertLLEtoXYZ(QVector3D(minLat - 0.001, maxLon + 0.001, maxHeight + 800));
    corner[2] = convertLLEtoXYZ(QVector3D(minLat - 0.001, minLon - 0.001, maxHeight + 800));
    corner[3] = convertLLEtoXYZ(QVector3D(maxLat + 0.001, minLon - 0.001, maxHeight + 800));
    corner[4] = convertLLEtoXYZ(QVector3D(maxLat + 0.001, maxLon + 0.001, minHeight - 800));
corner[5] = convertLLEtoXYZ(QVector3D(minLat - 0.001, maxLon + 0.001, minHeight - 800));
corner[6] = convertLLEtoXYZ(QVector3D(minLat - 0.001, minLon - 0.001, minHeight - 800));
corner[7] = convertLLEtoXYZ(QVector3D(maxLat + 0.001, minLon - 0.001, minHeight - 800));

upperTriangle1.vertices.push_back(corner[0]);
upperTriangle1.vertices.push_back(corner[1]);
upperTriangle1.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(upperTriangle1);

upperTriangle2.vertices.push_back(corner[2]);
upperTriangle2.vertices.push_back(corner[3]);
upperTriangle2.vertices.push_back(corner[0]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(upperTriangle2);

bottomTriangle1.vertices.push_back(corner[4]);
bottomTriangle1.vertices.push_back(corner[5]);
bottomTriangle1.vertices.push_back(corner[6]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(bottomTriangle1);

bottomTriangle2.vertices.push_back(corner[6]);
bottomTriangle2.vertices.push_back(corner[7]);
bottomTriangle2.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(bottomTriangle2);

eastTriangle1.vertices.push_back(corner[1]);
eastTriangle1.vertices.push_back(corner[0]);
eastTriangle1.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(eastTriangle1);

eastTriangle2.vertices.push_back(corner[4]);
eastTriangle2.vertices.push_back(corner[5]);
eastTriangle2.vertices.push_back(corner[1]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(eastTriangle2);

westTriangle1.vertices.push_back(corner[2]);
westTriangle1.vertices.push_back(corner[6]);
westTriangle1.vertices.push_back(corner[7]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(westTriangle1);

westTriangle2.vertices.push_back(corner[7]);
westTriangle2.vertices.push_back(corner[3]);
westTriangle2.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(westTriangle2);

northTriangle1.vertices.push_back(corner[3]);
northTriangle1.vertices.push_back(corner[7]);
northTriangle1.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(northTriangle1);

northTriangle2.vertices.push_back(corner[4]);
northTriangle2.vertices.push_back(corner[0]);
northTriangle2.vertices.push_back(corner[3]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(northTriangle2);

southTriangle1.vertices.push_back(corner[2]);
southTriangle1.vertices.push_back(corner[1]);
southTriangle1.vertices.push_back(corner[5]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(southTriangle1);

southTriangle2.vertices.push_back(corner[5]);
southTriangle2.vertices.push_back(corner[6]);
southTriangle2.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxInternal.push_back(southTriangle2);

upperTriangle1.vertices.clear();
upperTriangle2.vertices.clear();
bottomTriangle1.vertices.clear();
bottomTriangle2.vertices.clear();
westTriangle1.vertices.clear();
westTriangle2.vertices.clear();
eastTriangle1.vertices.clear();
eastTriangle2.vertices.clear();
northTriangle1.vertices.clear();
northTriangle2.vertices.clear();
southTriangle1.vertices.clear();
southTriangle2.vertices.clear();

upperTriangle1.vertices.push_back(corner[1]);
upperTriangle1.vertices.push_back(corner[0]);
upperTriangle1.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(upperTriangle1);

upperTriangle2.vertices.push_back(corner[3]);
upperTriangle2.vertices.push_back(corner[2]);
upperTriangle2.vertices.push_back(corner[0]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(upperTriangle2);

bottomTriangle1.vertices.push_back(corner[5]);
bottomTriangle1.vertices.push_back(corner[4]);
bottomTriangle1.vertices.push_back(corner[6]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(bottomTriangle1);

bottomTriangle2.vertices.push_back(corner[7]);
bottomTriangle2.vertices.push_back(corner[6]);
bottomTriangle2.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(bottomTriangle2);

eastTriangle1.vertices.push_back(corner[0]);
eastTriangle1.vertices.push_back(corner[1]);
eastTriangle1.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(eastTriangle1);

eastTriangle2.vertices.push_back(corner[5]);
eastTriangle2.vertices.push_back(corner[4]);
eastTriangle2.vertices.push_back(corner[1]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(eastTriangle2);
westTriangle1.vertices.push_back(corner[6]);
wstTriangle1.vertices.push_back(corner[2]);
wstTriangle1.vertices.push_back(corner[7]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(westTriangle1);

westTriangle2.vertices.push_back(corner[3]);
wstTriangle2.vertices.push_back(corner[7]);
wstTriangle2.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(westTriangle2);

northTriangle1.vertices.push_back(corner[7]);
northTriangle1.vertices.push_back(corner[3]);
northTriangle1.vertices.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(northTriangle1);

northTriangle2.vertices.push_back(corner[0]);
northTriangle2.vertices.push_back(corner[4]);
northTriangle2.vertices.push_back(corner[3]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(northTriangle2);

southTriangle1.vertices.push_back(corner[1]);
southTriangle1.vertices.push_back(corner[2]);
southTriangle1.vertices.push_back(corner[5]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(southTriangle1);

southTriangle2.vertices.push_back(corner[6]);
southTriangle2.vertices.push_back(corner[5]);
southTriangle2.vertices.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxExternal.push_back(southTriangle2);

LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[0]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[1]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[1]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[3]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[3]);
//
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[5]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[5]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[6]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[6]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[7]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[7]);
//
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[0]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[4]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[1]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[5]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[2]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[6]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[3]);
LODList[LODToBeAnalyzed].boundingBoxEdges.push_back(corner[7]);
}

QVector3D TDBAnalyzer::convertXYZtoLLE(QVector3D vecXYZ)
{
    QVector3D vecLLE;
    mgcoord3d xyzToConvert;
    mgprojcoord lleConverted;
    xyzToConvert = mgMakeCoord3d(vecXYZ.x(), vecXYZ.y(), vecXYZ.z());
    if(!mgProjectionConvertXYZtoLLE(dbProjection, &xyzToConvert,
        &lleConverted))
    {
        emit error("Error converting from LLE to XYZ");
        qDebug("Error converting from LLE to XYZ");
    }
    vecLLE.setX(lleConverted.lat);
    vecLLE.setY(lleConverted.lon);
    vecLLE.setZ(lleConverted.height);
    return vecLLE;
}

QVector3D TDBAnalyzer::convertLLEtoXYZ(QVector3D vecLLE)
{
    QVector3D vecXYZ;
    mgprojcoord lleToConvert;
    mgcoord3d xyzConverted;

    lleToConvert.lat = vecLLE.x();
    lleToConvert.lon = vecLLE.y();
    lleToConvert.height = vecLLE.z();
    if(!mgProjectionConvertLLEtoXYZ(dbProjection, &lleToConvert,
        &xyzConverted))
    {
        emit error("Error converting from LLE to XYZ");
        qDebug("Error converting from LLE to XYZ");
    }
    vecXYZ.setX(xyzConverted.x);
    vecXYZ.setY(xyzConverted.y);
    vecXYZ.setZ(xyzConverted.z);
    return vecXYZ;
}
myopenglwidget.h

/**************************************************************************
**
** Copyright (C) 2012 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
** * Redistributions of source code must retain the above copyright
** notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
** notice, this list of conditions and the following disclaimer in
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
** of its contributors may be used to endorse or promote products derived
** from this software without specific prior written permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
**************************************************************************

#ifndef MYOPENGLWIDGET_H
#define MYOPENGLWIDGET_H

#include <QGLWidget>
#include <QGLShaderProgram>
#include <QGLBuffer>
#include <QColor>

#include <QMouseEvent>
#include <QWheelEvent>

#include <QTimer>

#endif
class myOpenGLWidget : public QGLWidget
{
    Q_OBJECT
public:
    myOpenGLWidget(QWidget *parent = 0);
    ~myOpenGLWidget();

    QSize sizeHint() const;

    QVector<QVector3D> terrainVertices;
    QVector<QVector3D> terrainColors;
    QVector<QVector2D> terrainTextureCoordinates;
    QVector<QVector3D> terrainNormals;
    QGLBuffer terrainBuffer;
    int terrainNumberOfVertices;
    bool isTerrainBufferSet;

    QVector<QVector3D> blocksBordersVertices;
    QVector<QVector3D> blocksBordersColors;

    QVector<QVector3D> eyepointSkyVertices;
    QVector<QVector3D> eyepointSkyColors;

    QVector<QVector3D> eyepointGroundVertices;
    QVector<QVector3D> eyepointGroundColors;

    QVector<QVector3D> dirVecLOSVertices;
    QVector<QVector3D> dirVecLOSColors;

    QVector<QVector3D> LOSRaysVertices;
    QVector<QVector3D> LOSRaysColors;

    QVector<QVector3D> LOSIntersectionVertices;
    QVector<QVector3D> LOSIntersectionColors;

    QVector<QVector3D> actualEyepointsVertices;
    QVector<QVector3D> actualEyepointsColors;

    QVector<QVector3D> boundingBoxVertices;
    QVector<QVector4D> boundingBoxColors;

    QVector<QVector3D> boundingBoxEdgesVertices;
    QVector<QVector3D> boundingBoxEdgesColors;

    QVector<QVector3D> problematicRaysVertices;
    QVector<QVector3D> problematicRaysColors;

    QVector<QVector3D> postPositionLOS;
    QVector<QVector3D> postColorLOS;
    QVector<double> postHeightLOS;
    double postBaseSideLenghtLOS;

    QVector<QVector3D> postPositionElev;
    QVector<QVector3D> postColorElev;
    QVector<double> postHeightElev;
}
double postBaseSideLengthElev;
QVector3D objectCenter;
QVector3D lightPosition;

bool enableDrawOriginAxes;
bool enableDrawTerrain;
bool enableDrawBlockBorders;
bool enableDrawSkyEyePoints;
bool enableDrawGroundEyePoints;
bool enableDrawDirVecLOS;
bool enableDrawLOSrays;
bool enableBoundingBox;
bool enableDrawProblematicRays;
bool enableDrawPostsLOS;
bool enableDrawPostsElev;
bool enableDrawEyePointsOnly;

QVector3D cameraPosition, cameraForwardDirection;
QVector3D cameraUpDirection, cameraRightDirection, cameraLookAt;
QTimer *shiftTimer;
bool isShiftKeyPressed, isAutoRepeatEnabled;
double step;

GLuint grassTexture, cubeTexture;

protected:
void initializeGL();
void resizeGL(int width, int height);
void paintGL();

void mousePressEvent(QMouseEvent *event);
void mouseMoveEvent(QMouseEvent *event);
void mouseReleaseEvent(QMouseEvent *event);
void wheelEvent(QWheelEvent *event);
void keyPressEvent(QKeyEvent *event);
void keyReleaseEvent(QKeyEvent *event);

private:
QMatrix4x4 pMatrix;
QGLShaderProgram shaderProgram;
QGLShaderProgram lightingShaderProgram;
QVector<QVector3D> cubeVertices;
QVector<QVector3D> cubeColors;
QVector<QVector3D> cubeNormals;
QVector<QVector3D> xyzCoordSysVertices;
QVector<QVector3D> xyzCoordSysColors;
QVector<QVector2D> cubeTextureCoordinates;

QPoint lastMousePosition;

void drawCube(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawOriginXYZ(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
```cpp
void drawTerrain(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawBlocksBorders(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawSkyEyePoints(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawDirVecLOS(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawLOSIntersection(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawActualEyepoints(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawProblematicRays(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix);
void drawPost(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix, QVector3D Color);
signals:
    void cameraUpdated();
public slots:
    void handleShiftPressedEvent();
    void setTerrainBuffer();
    void destroyTerrainBuffer();
};
#endif // MYOPENGLWIDGET_H

myopenglwidget.cpp
```
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

**

******************************************************************************

```
#include "myopenglwidget.h"

#ifdef WIN32
    #include <GL/glext.h>
    PFNGLACTIVETEXTUREPROC pGlActiveTexture = NULL;
    #define glActiveTexture pGlActiveTexture
#endif //WIN32

myOpenGLWidget::myOpenGLWidget(QWidget *parent) :
QLWidget(QGLFormat(QGL::SampleBuffers | QGL::DoubleBuffer/* Additional format options*/), parent)
{
    cameraPosition.setX(5);
    cameraPosition.setY(5);
    cameraPosition.setZ(5);
    cameraLookAt.setX(0);
    cameraLookAt.setY(0);
    cameraLookAt.setZ(0);
    cameraForwardDirection = cameraLookAt - cameraPosition;
    cameraForwardDirection.normalize();
    cameraUpDirection.setX(0);
    cameraUpDirection.setY(0);
    cameraUpDirection.setZ(1);
    cameraRightDirection = QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
    cameraRightDirection.normalize();
    this->setCursor(Qt::OpenHandCursor);

    isShiftKeyPressed = false;
    shiftTimer = new QTimer(this);
    connect(shiftTimer, SIGNAL(timeout()), this, SLOT(handleShiftPressedEvent()));
    shiftTimer->start(50);
    isAutoRepeatEnabled = false;

    QMatrix4x4 lightTransformation;
    lightTransformation.translate(0,0,10000);
    lightPosition = QVector3D(0,0,0);
    lightPosition = lightTransformation * lightPosition;
    step = 0.1;

    enableDrawOriginAxes = false;
    enableDrawTerrain = false;
```
objectCenter.setX(0);
objectCenter.setY(0);
objectCenter.setZ(0);
isTerrainBufferSet = false;
enableDrawBlockBorders = false;
enableDrawSkyEyePoints = false;
enableDrawGroundEyePoints = false;
enableDrawDirVecLOS = false;
enableDrawLOSRays = false;
enableBoundingBox = false;
enableDrawProblematicRays = false;
enableDrawPostsLOS = false;
enableDrawPostsElev = false;
enableDrawEyepointsOnly = false;
}

myOpenGLWidget::~myOpenGLWidget()
{
}

QSize myOpenGLWidget::sizeHint() const
{
    return QSize(640, 480);
}

void myOpenGLWidget::initializeGL()
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_MULTISAMPLE);
    //glEnable(GL_POINT_SMOOTH);
    //glEnable(GL_LINE_SMOOTH);
    //glEnable(GL_POLYGON_SMOOTH);

#ifdef WIN32
    glActiveTexture = (PFNGLACTIVETEXTUREPROC) wglGetProcAddress((LPCSTR) "glActiveTexture");
#endif
    qglClearColor(QColor(124, 217, 240, 1));
    shaderProgram.addShaderFromSourceFile(QGLShader::Vertex,
        ":/vertexShader.vsh");
    shaderProgram.addShaderFromSourceFile(QGLShader::Fragment,
        ":/fragmentShader.fsh");
    shaderProgram.link();
    lightingShaderProgram.addShaderFromSourceFile(QGLShader::Vertex,
        ":/lightingVertexShader.vsh");
    lightingShaderProgram.addShaderFromSourceFile(QGLShader::Fragment,
        ":/lightingFragmentShader.fsh");
    lightingShaderProgram.link();

    cubeVertices << QVector3D(-0.5, -0.5, 1.0) << QVector3D(0.5, -0.5, 1.0) << QVector3D(0.5, 0.5, 1.0) // Front
        << QVector3D(-0.5, 0.5, 1.0) << QVector3D(-0.5, -0.5, 1.0)
```cpp
// Bottom
QVector3D(0.5, -0.5, 0.0) // Back
QVector3D(-0.5, 0.5, 0.0) // Left
QVector3D(-1.0, 1.0, 0.0) // Front
cubeColors << QVector3D(1, 0, 0) << QVector3D(0, 1, 1) << QVector3D(1, 0, 0); // Back
cubeNormals << QVector3D(0, 0, 1) << QVector3D(0, 0, 1); // Front
```
void myOpenGLWidget::resizeGL(int width, int height)
{
    if (height == 0) {
        height = 1;
    }

    pMatrix.setToIdentity();
pMatrix.perspective(60.0, (float) width / (float) height, 0.5, 100000);
    glViewport(0, 0, width, height);
}

QVector3D cubeTextureCoordinates << QVector2D(0, 0) << QVector2D(1, 0) <<
QVector2D(1, 1) // Front << QVector2D(1, 1) << QVector2D(0, 1) <<
QVector2D(0, 0) // Back << QVector2D(0, 0) << QVector2D(1, 0) <<
QVector2D(1, 1) // Left << QVector2D(1, 1) << QVector2D(0, 1) <<
QVector2D(0, 0) // Right << QVector2D(0, 0) << QVector2D(0, 1) <<
QVector2D(0, 1) // Top << QVector2D(0, 0) << QVector2D(1, 0) <<
QVector2D(0, 0) // Bottom << QVector2D(0, 0) << QVector2D(1, 0) <<
QVector2D(0, 1);

xyzCoordSysVertices << QVector3D(0, 0, 0) << QVector3D(10000, 0, 0) // x
    << QVector3D(0, 0, 0) << QVector3D(0, 10000, 0) // y
    << QVector3D(0, 0, 0) << QVector3D(0, 0, 10000); // z
xyzCoordSysColors << QVector3D(1, 0, 0) << QVector3D(1, 0, 0) // x
    << QVector3D(0, 1, 0) << QVector3D(0, 1, 0) // y
    << QVector3D(0, 0, 1) << QVector3D(0, 0, 1); // z

grassTexture = bindTexture(QPixmap(":/grass-texture.jpg"));

}
```cpp
void myOpenGLWidget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    QMatrix4x4 mMatrix;
    QMatrix4x4 vMatrix;
    vMatrix.lookAt(cameraPosition, cameraLookAt, cameraUpDirection);
    emit cameraUpdated();
    if (enableDrawTerrain)
    {
        drawTerrain(mMatrix, vMatrix);
    }
    if (enableDrawBlockBorders)
    {
        drawBlockBorders(mMatrix, vMatrix);
    }
    if (enableDrawSkyEyePoints)
    {
        drawSkyEyePoints(mMatrix, vMatrix);
    }
    if (enableDrawGroundEyePoints)
    {
        drawSkyGroundPoints(mMatrix, vMatrix);
    }
    if (enableDrawDirVecLOS)
    {
        drawDirVecLOS(mMatrix, vMatrix);
    }
    if (enableDrawLOSrays)
    {
        if (enableDrawEyePointsOnly)
        {
            drawActualEyePoints(mMatrix, vMatrix);
        }
        else
        {
            drawActualEyePoints(mMatrix, vMatrix);
            drawLOSrays(mMatrix, vMatrix);
            drawLOSIntersection(mMatrix, vMatrix);
        }
    }
    if (enableBoundingBox)
    {
        drawBoundingBox(mMatrix, vMatrix);
        drawBoundingBoxEdges(mMatrix, vMatrix);
    }
    if (enableDrawProblematicRays)
    {
        drawProblematicRays(mMatrix, vMatrix);
    }
    if (enableDrawPostsLOS)
    {
```
{ if((postPositionLOS.size() == postHeightLOS.size()) &&
   (postHeightLOS.size() == postColorLOS.size()))
{
    for(int i = 0; i < postPositionLOS.size(); i++)
    {
      mMatrix.setToIdentity();
      mMatrix.translate(postPositionLOS[i]);
      mMatrix.scale(postBaseSideLenghtLOS, postBaseSideLenghtLOS, postHeightLOS[i]);
      drawPost(mMatrix, vMatrix, postColorLOS[i]);
    }
  }
}
if (enableDrawPostsElev)
{
  if((postPositionElev.size() == postHeightElev.size()) &&
     (postHeightElev.size() == postColorElev.size()))
  {
    for(int i = 0; i < postPositionElev.size(); i++)
    {
      mMatrix.setToIdentity();
      mMatrix.translate(postPositionElev[i]);
      mMatrix.scale(postBaseSideLenghtElev, postBaseSideLenghtElev, postHeightElev[i]);
      drawPost(mMatrix, vMatrix, postColorElev[i]);
    }
  }
}
if (enableDrawOriginAxes)
{
  drawCube(mMatrix, vMatrix);
  drawOriginXYZ(mMatrix, vMatrix);
}

void myOpenGLWidget::mousePressEvent(QMouseEvent *event)
{
  this->setFocus();
  lastMousePosition = event->pos();
  this->setCursor(Qt::ClosedHandCursor);
  event->accept();
}

void myOpenGLWidget::mouseMoveEvent(QMouseEvent *event)
{
  QMatrix4x4 cameraTransformation;
  int deltaX = event->x() - lastMousePosition.x();
  int deltaY = event->y() - lastMousePosition.y();
  lastMousePosition = event->pos();
  if (event->buttons() & Qt::LeftButton)
  {
    cameraTransformation.rotate(((double)deltaY)/10.0, cameraRightDirection);
  }
}
cameraForwardDirection = cameraTransformation *
cameraForwardDirection;
cameraForwardDirection.normalize();
cameraLookAt = 2500 * cameraForwardDirection + cameraPosition;
cameraRightDirection =
QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
cameraRightDirection.normalize();

if (event->buttons() & Qt::RightButton)
{
    cameraPosition = cameraPosition -
((step*deltaX/3.25)*cameraRightDirection);
cameraLookAt = cameraLookAt -
((step*deltaX/3.25)*cameraRightDirection);
cameraForwardDirection = cameraLookAt - cameraPosition;
cameraForwardDirection.normalize();
cameraRightDirection =
QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
cameraRightDirection.normalize();

    cameraPosition = cameraPosition + QVector3D(0,0,step*deltaY/5.0);
cameraLookAt = cameraLookAt + QVector3D(0,0,step*deltaY/5.0);
cameraForwardDirection = cameraLookAt - cameraPosition;
cameraForwardDirection.normalize();
cameraRightDirection =
QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
cameraRightDirection.normalize();
}
updateGL();
event->accept();
}

void myOpenGLWidget::mousePressEvent(QMouseEvent *event)
{
    this->setCursor(Qt::OpenHandCursor);
event->accept();
}

void myOpenGLWidget::wheelEvent(QWheelEvent *event)
{
    int delta = event->delta();
    if (event->orientation() == Qt::Vertical)
    {
        cameraPosition = cameraPosition +
((step*delta/10.0)*cameraForwardDirection);
cameraLookAt = cameraLookAt + 
((step*delta/10.0)*cameraForwardDirection);
} 
cameraForwardDirection = cameraLookAt - cameraPosition; 
cameraForwardDirection.normalize();
cameraRightDirection = QVector3D::crossProduct(cameraForwardDirection, 
cameraUpDirection); 
cameraRightDirection.normalize();
updateGL();
event->accept();
}

void myOpenGLWidget::keyPressEvent(QKeyEvent *event)
{

QMatrix4x4 cameraTransformation;
if(event->isAutoRepeat())
{
    isAutoRepeatEnabled = true;
    step *= 2;
}
switch(event->key())
{
    case Qt::Key_A:
        cameraPosition = cameraPosition - (step*cameraRightDirection); 
cameraLookAt = cameraLookAt - (step*cameraRightDirection);
        break;
    case Qt::Key_W:
        cameraPosition = cameraPosition + (step*cameraForwardDirection); 
cameraLookAt = cameraLookAt + (step*cameraForwardDirection);
        break;
    case Qt::Key_S:
        cameraPosition = cameraPosition - (step*cameraForwardDirection); 
cameraLookAt = cameraLookAt - (step*cameraForwardDirection);
        break;
    case Qt::Key_D:
        cameraPosition = cameraPosition + (step*cameraRightDirection); 
cameraLookAt = cameraLookAt + (step*cameraRightDirection);
        break;
    case Qt::Key_Space:
        cameraPosition = cameraPosition + QVector3D(0,0,step); 
cameraLookAt = cameraLookAt + QVector3D(0,0,step);
        isShiftKeyPressed = false;
        break;
    case Qt::Key_Shift:
        isShiftKeyPressed = true;
        break;
    case Qt::Key_Up:
        cameraTransformation.rotate(1.5, cameraRightDirection); 
cameraForwardDirection = cameraTransformation * 
cameraForwardDirection;
        cameraLookAt = 2500 * cameraForwardDirection + cameraPosition;
        break;
    case Qt::Key_Down:
        cameraTransformation.rotate(-1.5, cameraRightDirection);
cameraForwardDirection = cameraTransformation *
cameraForwardDirection;
cameraLookAt = 2500 * cameraForwardDirection + cameraPosition;
break;
case Qt::Key_Left:
cameraTransformation.rotate(1.5, cameraUpDirection);
cameraForwardDirection = cameraTransformation *
cameraForwardDirection;
cameraLookAt = 2500 * cameraForwardDirection + cameraPosition;
break;
case Qt::Key_Right:
cameraTransformation.rotate(-1.5, cameraUpDirection);
cameraForwardDirection = cameraTransformation *
cameraForwardDirection;
cameraLookAt = 2500 * cameraForwardDirection + cameraPosition;
break;
case Qt::Key_Equal:
step *= 1.5;
break;
case Qt::Key_Minus:
step *= 0.75;
break;
}
cameraForwardDirection = cameraLookAt - cameraPosition;
cameraForwardDirection.normalize();
cameraRightDirection = QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
cameraRightDirection.normalize();
updateGL();
event->accept();
}
void myOpenGLWidget::keyReleaseEvent(QKeyEvent *event)
{
if (isAutoRepeatEnabled)
{
    step = step / 2;
    isAutoRepeatEnabled = false;
}
switch(event->key())
{
case Qt::Key_Shift:
    isShiftKeyPressed = false;
break;
}
void myOpenGLWidget::drawCube(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", cubeVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", cubeColors.constData());
    shaderProgram.enableAttributeArray("color");
    glDrawArrays(GL_TRIANGLES, 0, cubeVertices.size());
shaderProgram.disableAttributeArray("vertex");
shaderProgram.disableAttributeArray("color");
shaderProgram.release();
}

void myOpenGLWidget::drawOriginXYZ(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    xyzCoordSysVertices.constData();
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("vertex", xyzCoordSysVertices.constData());
    shaderProgram.enableAttributeArray("color");
    glLineWidth(3);
    glDrawArrays(GL_LINES, 0, xyzCoordSysVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.release();
}

void myOpenGLWidget::drawTerrain(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    mMatrix.setToIdentity();
    QMatrix4x4 mvMatrix;
    mvMatrix = vMatrix * mMatrix;
    QMatrix3x3 normalMatrix;
    normalMatrix = mvMatrix.normalMatrix();
    lightingShaderProgram.bind();
    lightingShaderProgram.setUniformValue("mvpMatrix", pMatrix * mvMatrix);
    lightingShaderProgram.setUniformValue("mvMatrix", mvMatrix);
    lightingShaderProgram.setUniformValue("normalMatrix", normalMatrix);
    lightingShaderProgram.setUniformValue("lightPosition", vMatrix * lightPosition);
    lightingShaderProgram.setUniformValue("ambientColor", QColor(96, 96, 96));
    lightingShaderProgram.setUniformValue("diffuseColor", QColor(160, 160, 160));
    lightingShaderProgram.setUniformValue("specularColor", QColor(128, 128, 128));
    lightingShaderProgram.setUniformValue("ambientReflection", (GLfloat) 1.0);
    lightingShaderProgram.setUniformValue("diffuseReflection", (GLfloat) 1.0);
    lightingShaderProgram.setUniformValue("specularReflection", (GLfloat) 0.0);
    lightingShaderProgram.setUniformValue("shininess", (GLfloat) 100.0);
    if (isTerrainBufferSet)
    {
        if (terrainBuffer.bind())
        {
            int offset = 0;
            lightingShaderProgram.setAttributeBuffer("vertex", GL_FLOAT, offset, 3, 0);
            lightingShaderProgram.enableAttributeArray("vertex");
        }
    }
offset += terrainNumberOfVertices * 3 * sizeof(GLfloat);
lghtingShaderProgram.setAttributeBuffer("normal", GL_FLOAT, offset, 3, 0);
lghtingShaderProgram.enableAttributeArray("normal");
offset += terrainNumberOfVertices * 3 * sizeof(GLfloat);
lghtingShaderProgram.setAttributeBuffer("textureCoordinate", GL_FLOAT, offset, 2, 0);
lghtingShaderProgram.enableAttributeArray("textureCoordinate");
terrainBuffer.release();
lghtingShaderProgram.setUniformValue("texture", 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, grassTexture);
glActiveTexture(0);
glDrawArrays(GL_TRIANGLES, 0, terrainNumberOfVertices);
}
else {
  qDebug("Binding frame buffer failed during painting.");
}
else {
lghtingShaderProgram.setAttributeBuffer("vertex", terrainVertices.constData());
lghtingShaderProgram.enableAttributeArray("vertex");
lghtingShaderProgram.setAttributeArray("normal", terrainNormals.constData());
lghtingShaderProgram.enableAttributeArray("normal");
lghtingShaderProgram.setAttributeArray("textureCoordinate", terrainTextureCoordinates.constData());
lghtingShaderProgram.enableAttributeArray("textureCoordinate");
lghtingShaderProgram.setUniformValue("texture", 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, grassTexture);
activeTexture(0);
DrawArrays(GL_TRIANGLES, 0, terrainVertices.size());
lghtingShaderProgram.disableAttributeArray("vertex");
lghtingShaderProgram.disableAttributeArray("normal");
lghtingShaderProgram.disableAttributeArray("textureCoordinate");
lghtingShaderProgram.release();
}

void myOpenGLWidget::handleShiftPressedEvent()
{
  if (isShiftKeyPressed == true)
  {
    cameraPosition = cameraPosition + QVector3D(0,0,-step);
    cameraLookAt = cameraLookAt + QVector3D(0,0,-step);
    cameraForwardDirection = cameraLookAt - cameraPosition;
    cameraForwardDirection.normalize();
    cameraRightDirection = QVector3D::crossProduct(cameraForwardDirection, cameraUpDirection);
    cameraRightDirection.normalize();
    updateGL();
  }
void myOpenGLWidget::setTerrainBuffer()
{
    terrainNumberOfVertices = terrainVertices.size();
    if(!terrainBuffer.isCreated())
    {
        if(terrainBuffer.create())
        {
            if(terrainBuffer.bind())
            {
                terrainBuffer.allocate(terrainNumberOfVertices * (3+3+2) * sizeof(GLfloat));

                int offset = 0;
                terrainBuffer.write(offset, terrainVertices.constData(),
                                    terrainNumberOfVertices * 3 * sizeof(GLfloat));
                offset += terrainNumberOfVertices * 3 * sizeof(GLfloat);
                terrainBuffer.write(offset, terrainNormals.constData(),
                                     terrainNumberOfVertices * 3 * sizeof(GLfloat));
                offset += terrainNumberOfVertices * 3 * sizeof(GLfloat);
                terrainBuffer.write(offset,
                                     terrainTextureCoordinates.constData(), terrainNumberOfVertices * 2 *
                                     sizeof(GLfloat));

                terrainBuffer.release();
                terrainVertices.clear();
                terrainNormals.clear();
                terrainTextureCoordinates.clear();
                isTerrainBufferSet = true;
            } else
                { qDebug("Binding frame buffer failed during frame buffer creation.");
                }
            } else
                { qDebug("Frame buffer not created.");
                }
        } else
            { qDebug("Frame buffer not created because it already existed.");
            }
    } else
        { qDebug("Frame buffer not created because it already existed.");
        }
}

void myOpenGLWidget::destroyTerrainBuffer()
{
    if (terrainBuffer.isCreated())
    {
        terrainBuffer.destroy();
    } else
        { qDebug("Frame buffer not created because it already existed.");
        }
}
```cpp
qDebug("Frame buffer destroyed.");

isTerrainBufferSet = false;

void myOpenGLWidget::drawBlocksBorders(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", 
                                blocksBordersVertices.constData());
    shaderProgram.setAttributeArray("vertex", 
                                blocksBordersColors.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.enableAttributeArray("color");
    glLineWidth(3);
    glDrawArrays(GL_LINES, 0, blocksBordersVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawSkyEyePoints(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", 
                                eyepointSkyVertices.constData());
    shaderProgram.setAttributeArray("color", eyepointSkyColors.constData());
    glPointSize(5);
    glDrawArrays(GL_POINTS, 0, eyepointSkyVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.release();
}

void myOpenGLWidget::drawSkyGroundPoints(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", 
                                eyepointGroundVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", eyepointGroundColors.constData());
    shaderProgram.enableAttributeArray("color");
    glPointSize(7);
    glDrawArrays(GL_POINTS, 0, eyepointGroundVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.release();
}
```
```cpp
void myOpenGLWidget::drawDirVecLOS(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", dirVecLOSVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", dirVecLOSColors.constData());
    shaderProgram.enableAttributeArray("color");
    glLineWidth(2);
    glDrawArrays(GL_LINES, 0, dirVecLOSVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawLOSRays(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", LOSRaysVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", LOSRaysColors.constData());
    shaderProgram.enableAttributeArray("color");
    glLineWidth(2);
    glDrawArrays(GL_LINES, 0, LOSRaysVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawLOSIntersection(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", LOSIntersectionVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", LOSIntersectionColors.constData());
    shaderProgram.enableAttributeArray("color");
    glPointSize(7);
    glDrawArrays(GL_POINTS, 0, LOSIntersectionVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawActualEyepoints(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
```
shaderProgram.setAttributeArray("vertex", actualEyepointsVertices.constData());
shaderProgram.enableAttributeArray("vertex");
shaderProgram.setAttributeArray("color", actualEyepointsColors.constData());
shaderProgram.enableAttributeArray("color");
glPointSize(7);
glDrawArrays(GL_POINTS, 0, actualEyepointsVertices.size());
shaderProgram.disableAttributeArray("vertex");
shaderProgram.disableAttributeArray("color");
shaderProgram.release();
}

void myOpenGLWidget::drawBoundingBox(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", boundingBoxVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", boundingBoxColors.constData());
    shaderProgram.enableAttributeArray("color");
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glDrawArrays(GL_TRIANGLES, 0, boundingBoxVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawBoundingBoxEdges(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setAttributeArray("vertex", boundingBoxEdgesVertices.constData());
    shaderProgram.enableAttributeArray("vertex");
    shaderProgram.setAttributeArray("color", boundingBoxEdgesColors.constData());
    shaderProgram.enableAttributeArray("color");
    glLineWidth(2);
    glDrawArrays(GL_LINES, 0, boundingBoxEdgesVertices.size());
    shaderProgram.disableAttributeArray("vertex");
    shaderProgram.disableAttributeArray("color");
    shaderProgram.release();
}

void myOpenGLWidget::drawProblematicRays(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix)
{
    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
shaderProgram.setAttributeArray("vertex", problematicRaysVertices.constData());
shaderProgram.enableAttributeArray("vertex");
shaderProgram.setAttributeArray("color", problematicRaysColors.constData());
shaderProgram.enableAttributeArray("color");
gLineWidth(2);
DrawArrays(GL_LINES, 0, problematicRaysVertices.size());
shaderProgram.disableAttributeArray("vertex");
shaderProgram.disableAttributeArray("color");
shaderProgram.release();

}  

void myOpenGLWidget::drawPost(QMatrix4x4 mMatrix, QMatrix4x4 vMatrix, QVector3D Color)
{
    QMatrix4x4 mvMatrix;
    mvMatrix = vMatrix * mMatrix;
    QMatrix3x3 normalMatrix;
    normalMatrix = mvMatrix.normalMatrix();
    lightingShaderProgram.bind();
    lightingShaderProgram.setUniformValue("mvpMatrix", pMatrix * mvMatrix);
    lightingShaderProgram.setUniformValue("mvMatrix", mvMatrix);
    lightingShaderProgram.setUniformValue("normalMatrix", normalMatrix);
    lightingShaderProgram.setUniformValue("lightPosition", vMatrix * lightPosition);
    lightingShaderProgram.setUniformValue("ambientColor", QColor(96, 96, 96));
    lightingShaderProgram.setUniformValue("diffuseColor", QColor(160, 160, 160));
    lightingShaderProgram.setUniformValue("specularColor", QColor(255, 255, 255));
    lightingShaderProgram.setUniformValue("ambientReflection", (GLfloat) 1.0);
    lightingShaderProgram.setUniformValue("diffuseReflection", (GLfloat) 1.0);
    lightingShaderProgram.setUniformValue("specularReflection", (GLfloat) 0.8);
    lightingShaderProgram.setUniformValue("shininess", (GLfloat) 100.0);
    lightingShaderProgram.setAttributeArray("vertex", cubeVertices.constData());
    lightingShaderProgram.enableAttributeArray("vertex");
    lightingShaderProgram.setAttributeArray("normal", cubeNormals.constData());
    lightingShaderProgram.enableAttributeArray("normal");
    lightingShaderProgram.setAttributeArray("textureCoordinate", cubeTextureCoordinates.constData());
    lightingShaderProgram.enableAttributeArray("textureCoordinate");
    lightingShaderProgram.setUniformValue("texture", 0);

    int Red, Green, Blue;
    Red = Color.x() * 255;
    Green = Color.y() * 255;
    Blue = Color.z() * 255;
QPixmap pixmapColor(64, 64);
pixmapColor.fill(QColor(Red, Green, Blue, 255));
cubeTexture = bindTexture(QPixmap(pixmapColor));

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, cubeTexture);
glActiveTexture(0);
glDrawArrays(GL_TRIANGLES, 0, cubeVertices.size());

lightingShaderProgram.disableAttributeArray("vertex");
lightingShaderProgram.disableAttributeArray("normal");
lightingShaderProgram.disableAttributeArray("textureCoordinate");
lightingShaderProgram.release();

```
widget.ui

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>Widget</class>
  <widget class="QWidget" name="Widget">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>1156</width>
        <height>715</height>
      </rect>
    </property>
    <property name="minimumSize">
      <size>
        <width>1156</width>
        <height>715</height>
      </size>
    </property>
    <property name="maximumSize">
      <size>
        <width>1156</width>
        <height>732</height>
      </size>
    </property>
    <property name="windowTitle">
      <string>Visual Terrain Database Correlation Assessment Tool</string>
    </property>
    <widget class="QTabWidget" name="tabWidget">
      <property name="geometry">
        <rect>
          <x>10</x>
          <y>5</y>
          <width>1137</width>
          <height>680</height>
        </rect>
      </property>

```

234
<property name="tabPosition">
  <enum>QTabWidget::North</enum>
</property>

<property name="currentIndex">
  <number>0</number>
</property>

<widget class="QWidget" name="tab">
  <attribute name="title">
    <string>Load Databases</string>
  </attribute>
  <widget class="QPushButton" name="pushButtonOpenDB_1">
    <property name="geometry">
      <rect>
        <x>15</x>
        <y>10</y>
        <width>150</width>
        <height>23</height>
      </rect>
    </property>
    <property name="text">
      <string>Open Terrain Database #1</string>
    </property>
  </widget>
  <widget class="QTextEdit" name="textEdit">
    <property name="geometry">
      <rect>
        <x>15</x>
        <y>40</y>
        <width>545</width>
        <height>525</height>
      </rect>
    </property>
    <property name="readOnly">
      <bool>true</bool>
    </property>
  </widget>
  <widget class="QProgressBar" name="progressBar">
    <property name="geometry">
      <rect>
        <x>15</x>
        <y>620</y>
        <width>350</width>
        <height>23</height>
      </rect>
    </property>
    <property name="value">
      <number>0</number>
    </property>
    <property name="textVisible">
      <bool>true</bool>
    </property>
    <property name="invertedAppearance">
      <bool>false</bool>
    </property>
  </widget>
</widget>
<widget class="QPushButton" name="pushButtonSaveGeometry">
<property name="geometry">
<rect>
<x>945</x>
<y>615</y>
<width>180</width>
<height>30</height>
</rect>
</property>
<property name="font">
<font>
<pointsize>10</pointsize>
<weight>75</weight>
<bold>true</bold>
</font>
</property>
<property name="autoFillBackground">
<bool>false</bool>
</property>
<property name="text">
<string>Save Terrain Geometry</string>
</property>
</widget>
<widget class="QPushButton" name="pushButtonOpenDB_2">
<property name="geometry">
<rect>
<x>575</x>
<y>10</y>
<width>150</width>
<height>23</height>
</rect>
</property>
<property name="text">
<string>Open Terrain Database #2</string>
</property>
</widget>
<widget class="QCheckBox" name="checkBoxSaveTraverseLog">
<property name="geometry">
<rect>
<x>960</x>
<y>12</y>
<width>155</width>
<height>17</height>
</rect>
</property>
<property name="text">
<string>Save database traverse log</string>
</property>
</widget>
<widget class="QFrame" name="frame_4">
<property name="geometry">
<rect>
<x>15</x>
<y>570</y>
<width>350</width>
</rect>
</property>
</widget>
<property name="frameShape">
<enum>QFrame::StyledPanel</enum>
</property>

<property name="frameShadow">
<enum>QFrame::Raised</enum>
</property>

<widget class="QLabel" name="label_13">
<property name="geometry">
<rect>
<x>5</x>
<y>15</y>
<width>95</width>
<height>13</height>
</rect>
</property>
<property name="text">
<string>Database #2 LOD:</string>
</property>
</widget>

<widget class="QComboBox" name="comboBoxLODDB2">
<property name="geometry">
<rect>
<x>100</x>
<y>10</y>
<width>45</width>
<height>22</height>
</rect>
</property>
</widget>

<widget class="QPushButton" name="pushButtonEnableTabDB2">
<property name="geometry">
<rect>
<x>150</x>
<y>10</y>
<width>190</width>
<height>23</height>
</rect>
</property>
<property name="text">
<string>Enable Tab "Visualize Database #2"</string>
</property>
</widget>

<widget class="QTextEdit" name="textEdit_2">
<property name="geometry">
<rect>
<x>575</x>
<y>40</y>
<width>545</width>
<height>525</height>
</rect>
</property>
<property name="readOnly">
<bool>true</bool>
</property>
</widget>
<widget class="QProgressBar" name="progressBar_6">
  <property name="geometry">
    <rect>
      <x>575</x>
      <y>620</y>
      <width>350</width>
      <height>23</height>
    </rect>
  </property>
  <property name="value">
    <number>0</number>
  </property>
</widget>

<widget class="QWidget" name="tab_2">
  <attribute name="title">
    <string>Visualize Database #1</string>
  </attribute>
</widget>

<widget class="myOpenGLWidget" name="widgetVisualization_1">
  <property name="geometry">
    <rect>
      <x>20</x>
      <y>10</y>
      <width>890</width>
      <height>630</height>
    </rect>
  </property>
</widget>

<widget class="QCheckBox" name="checkBoxDrawBlockBorders1">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>80</y>
      <width>100</width>
      <height>17</height>
    </rect>
  </property>
  <property name="text">
    <string>Blocks's borders</string>
  </property>
</widget>

<widget class="QCheckBox" name="checkBoxDrawEyepoints1">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>110</y>
      <width>165</width>
      <height>17</height>
    </rect>
  </property>
  <property name="text">
    <string>Eye-points &quot;floating in the air&quot;</string>
  </property>
</widget>
Top global uncorrelated rays

Number of rays:

Show eye-points only
<rect>
  <x>20</x>
  <y>10</y>
  <width>890</width>
  <height>630</height>
</rect>
</widget>

<widget class="QCheckBox" name="checkBoxDrawBlockBorders2">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>80</y>
      <width>100</width>
      <height>17</height>
    </rect>
  </property>
  <property name="text">
    <string>Blocks's borders</string>
  </property>
</widget>

<widget class="QCheckBox" name="checkBoxDrawEyepoints2">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>110</y>
      <width>165</width>
      <height>17</height>
    </rect>
  </property>
  <property name="text">
    <string>Eye-points "floating in the air"</string>
  </property>
</widget>

<widget class="QCheckBox" name="checkBoxDrawGroundEyepoints2">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>140</y>
      <width>190</width>
      <height>17</height>
    </rect>
  </property>
  <property name="text">
    <string>Eye-points orthogonal projections</string>
  </property>
</widget>

<widget class="QCheckBox" name="checkBoxDrawLOSDirVec2">
  <property name="geometry">
    <rect>
      <x>930</x>
      <y>170</y>
      <width>135</width>
      <height>17</height>
    </rect>
  </property>
</widget>
Number of rays:

Show eye-points only
<string>Number of Lines</string>
</property>
</widget>
<widget class="QSpinBox" name="spinBoxColumns">
<property name="geometry">
<rect>
<x>120</x>
<y>70</y>
<width>42</width>
<height>22</height>
</rect>
</property>
<property name="alignment">
<set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
</property>
<property name="value">
<number>5</number>
</property>
</widget>
<widget class="QSpinBox" name="spinBoxLines">
<property name="geometry">
<rect>
<x>120</x>
<y>40</y>
<width>42</width>
<height>22</height>
</rect>
</property>
<property name="alignment">
<set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
</property>
<property name="value">
<number>5</number>
</property>
</widget>
<widget class="QLabel" name="label_3">
<property name="geometry">
<rect>
<x>10</x>
<y>70</y>
<width>100</width>
<height>22</height>
</rect>
</property>
<property name="text">
<string>Number of Columns</string>
</property>
</widget>
<widget class="QLabel" name="label">
<property name="geometry">
<rect>
<x>45</x>
<y>10</y>
<width>90</width>
<height>26</height>
</rect>
</property>
</widget>
<property name="font">
<font>
<pointsize>12</pointsize>
<weight>75</weight>
<bold>true</bold>
</font>
</property>
<property name="text">
<string>Block Size</string>
</property>
</widget>
<widget class="QPushButton" name="pushButtonCreateBlocks">
<property name="geometry">
<rect>
<x>45</x>
<y>105</y>
<width>90</width>
<height>23</height>
</rect>
</property>
<property name="text">
<string>Create Blocks</string>
</property>
</widget>
<widget class="QFrame" name="frame_2">
<property name="geometry">
<rect>
<x>220</x>
<y>10</y>
<width>210</width>
<height>140</height>
</rect>
</property>
<property name="frameShape">
<enum>QFrame::StyledPanel</enum>
</property>
<property name="frameShadow">
<enum>QFrame::Plain</enum>
</property>
<widget class="QLabel" name="label_4">
<property name="geometry">
<rect>
<x>15</x>
<y>10</y>
<width>180</width>
<height>60</height>
</rect>
</property>
<property name="font">
<font>
<pointsize>12</pointsize>
<weight>75</weight>
</font>
</property>
<bold>true</bold>
</font>
</property>
<property name="text">
<string>Number of Eye-Points Starting Positions per Block</string>
</property>
<property name="alignment">
<set>Qt::AlignCenter</set>
</property>
<widget class="QComboBox" name="comboBoxEyePointsNumber">
<property name="geometry">
<rect>
<x>25</x>
<y>100</y>
<width>45</width>
<height>22</height>
</rect>
</property>
</widget>
<widget class="QPushButton" name="pushButtonSetEyepoints">
<property name="geometry">
<rect>
<x>85</x>
<y>100</y>
<width>111</width>
<height>23</height>
</rect>
</property>
<property name="text">
<string>Set Starting Positions</string>
</property>
</widget>
<widget class="QFrame" name="frame_3">
<property name="geometry">
<rect>
<x>20</x>
<y>160</y>
<width>410</width>
<height>185</height>
</rect>
</property>
<property name="frameShape">
<enum>QFrame::StyledPanel</enum>
</property>
<property name="frameShadow">
<enum>QFrame::Plain</enum>
</property>
</widget>
<widget class="QLabel" name="label_5">
<property name="geometry">
<rect>
<x>50</x>
</rect>
</property>
Line of Sight Test Attributes Setup

Starting height (AGL):

Height increment:
<string># of height increments:</string>
</property>
</widget>
<widget class="QDoubleSpinBox" name="doubleSpinBoxStartingHeight">
<property name="geometry">
<rect>
<x>135</x>
<y>50</y>
<width>60</width>
<height>22</height>
</rect>
</property>
<property name="alignment">
<set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
</property>
<property name="minimum">
<double>0.250000000000000</double>
</property>
<property name="maximum">
<double>200.000000000000000</double>
</property>
<property name="singleStep">
<double>0.250000000000000</double>
</property>
<property name="value">
<double>15.000000000000000</double>
</property>
</widget>
<widget class="QDoubleSpinBox" name="doubleSpinBoxHeightIncrement">
<property name="geometry">
<rect>
<x>135</x>
<y>80</y>
<width>60</width>
<height>22</height>
</rect>
</property>
<property name="alignment">
<set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
</property>
<property name="maximum">
<double>100.000000000000000</double>
</property>
<property name="singleStep">
<double>0.250000000000000</double>
</property>
<property name="value">
<double>20.000000000000000</double>
</property>
</widget>
<widget class="QSpinBox" name="spinBoxHeightIncCount">
<property name="geometry">
<rect>
<x>135</x>
<y>110</y>
</rect>
</property>
<property name="accelerated">
  <bool>false</bool>
</property>
<property name="minimum">
  <number>2</number>
</property>
<property name="maximum">
  <number>16</number>
</property>
<property name="value">
  <number>8</number>
</property>
</widget>
<widget class="QSpinBox" name="spinBoxPitchCount">
  <property name="geometry">
    <rect>
      <x>335</x>
      <y>110</y>
      <width>60</width>
      <height>22</height>
    </rect>
  </property>
  <property name="alignment">
    <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>
  </property>
  <property name="readOnly">
    <bool>false</bool>
  </property>
  <property name="maximum">
    <number>10</number>
  </property>
  <property name="value">
    <number>2</number>
  </property>
</widget>
<widget class="QPushButton" name="pushButtonSetupLOSTestAtt">
  <property name="geometry">
    <rect>
      <x>120</x>
      <y>145</y>
      <width>170</width>
      <height>23</height>
    </rect>
  </property>
  <property name="text">
    <string>Setup LOS Test Attributes</string>
  </property>
</widget>
<widget class="QDoubleSpinBox" name="doubleSpinBoxPitchCountIncrement">
  <property name="geometry">
    <rect>
      <x>335</x>
      <y>80</y>
      <width>60</width>
      <height>22</height>
    </rect>
  </property>
Start Line of Sight Tests

Start LOS Tests

Reset all the settings

Save LOS Tests Results
<x>20</x>
<y>85</y>
<width>60</width>
<height>16</height>
</rect>
</property>
<property name="text">
<string>Thread #2</string>
</property>
</widget>
<widget class="QLabel" name="label_16">
<property name="geometry">
<rect>
<x>20</x>
<y>35</y>
<width>61</width>
<height>16</height>
</rect>
</property>
<property name="text">
<string>Thread #1</string>
</property>
</widget>
<widget class="QLabel" name="label_17">
<property name="geometry">
<rect>
<x>20</x>
<y>5</y>
<width>160</width>
<height>20</height>
</rect>
</property>
<property name="font">
<font>
<br>10/pointsize>
<br>75/weight>
<br>true/bold>
</font>
</property>
<property name="text">
<string>Terrain Database #1</string>
</property>
</widget>
</widget>
<widget class="QFrame" name="frame_9">
<property name="geometry">
<rect>
<x>230</x>
<y>505</y>
<width>200</width>
<height>140</height>
</rect>
</property>
<property name="frameShape">
<enum>QFrame::StyledPanel</enum>
<property name="frameShadow">
<enum>QFrame::Plain</enum>
</property>

<widget class="QProgressBar" name="progressBar_4">
<property name="geometry">
<rect>
<x>15</x>
<y>55</y>
<width>175</width>
<height>23</height>
</rect>
</property>
<property name="value">
<number>0</number>
</property>
</widget>

<widget class="QProgressBar" name="progressBar_5">
<property name="geometry">
<rect>
<x>15</x>
<y>105</y>
<width>175</width>
<height>23</height>
</rect>
</property>
<property name="value">
<number>0</number>
</property>
<property name="orientation">
<enum>Qt::Horizontal</enum>
</property>
</widget>

<widget class="QLabel" name="label_18">
<property name="geometry">
<rect>
<x>20</x>
<y>85</y>
<width>60</width>
<height>16</height>
</rect>
</property>
<property name="text">
<string>Thread #2</string>
</property>
</widget>

<widget class="QLabel" name="label_19">
<property name="geometry">
<rect>
<x>20</x>
<y>35</y>
<width>60</width>
<height>16</height>
</rect>
</property>
</widget>
<property name="text">
    <string>Thread #1</string>
</property>

<widget class="QLabel" name="label_20">
    <property name="geometry">
        <rect>
            <x>20</x>
            <y>5</y>
            <width>160</width>
            <height>20</height>
        </rect>
    </property>
    <property name="font">
        <font>
            <pointsize>10</pointsize>
            <weight>75</weight>
            <bold>true</bold>
        </font>
    </property>
    <property name="text">
        <string>Terrain Database #2</string>
    </property>
</widget>

<widget class="QPushButton" name="pushButtonClose">
    <property name="geometry">
        <rect>
            <x>1044</x>
            <y>687</y>
            <width>100</width>
            <height>23</height>
        </rect>
    </property>
    <property name="text">
        <string>Exit</string>
    </property>
</widget>

<layoutdefault spacing="6" margin="11"/>
<customwidgets>
    <customwidget>
        <class>myOpenGLWidget</class>
        <extends>QWidget</extends>
        <header location="global">myopenglwidget.h</header>
        <container>1</container>
    </customwidget>
</customwidgets>
<resources/>
<connections>
    <connection>
        <sender>pushButtonClose</sender>
        <signal>clicked()</signal>
    </connection>
</connections>
<receiver>Widget</receiver>
<slot>close()</slot>
<hints>
  <hint type="sourcelabel">
    <x>690</x>
    <y>564</y>
  </hint>
  <hint type="destinationlabel">
    <x>551</x>
    <y>560</y>
  </hint>
</hints>
</connection>
</connections>
</ui>

resources.qrc

<RCC>
  <qresource prefix="/">
    <file>fragmentShader.fsh</file>
    <file>vertexShader.vsh</file>
    <file>grass-texture.jpg</file>
    <file>lightingFragmentShader.fsh</file>
    <file>lightingVertexShader.vsh</file>
  </qresource>
</RCC>

vertexShader.vsh

#version 130

/************************************************************************
****
** Copyright (C) 2012 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
** notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
** notice, this list of conditions and the following disclaimer in
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
** of its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

**

** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
** **********************************************************************************

uniform mat4 mvpMatrix;

in vec4 vertex;
in vec4 color;

out vec4 varyingColor;

void main()
{
  varyingColor = color;
  gl_Position = mvpMatrix * vertex;
}

fragmentShader.fsh

#version 130

/**********************************************************************************
**
** Copyright (C) 2012 Digia Plc and/or its subsidiary(-ies).
** Contact: http://www.qt-project.org/legal
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
** notice, this list of conditions and the following disclaimer.
**
** $QT_END_LICENSE$
**
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
** of its contributors may be used to endorse or promote products derived
** from this software without specific prior written permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
****************************************************************************/

/*! [0]
  in vec4 varyingColor;
  out vec4 fragColor;
  
  void main(void)
  {
    fragColor = varyingColor;
  }
  */

lightingVertexShader.vsh
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
** of its contributors may be used to endorse or promote products derived
** from this software without specific prior written permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
**
** $QT_END_LICENSE$
**
**************************************************************************/

//! [0]
uniform mat4 mvpMatrix;
uniform mat4 mvMatrix;
uniform mat3 normalMatrix;
uniform vec3 lightPosition;
in vec4 vertex;
in vec3 normal;
in vec2 textureCoordinate;

out vec3 varyingNormal;
out vec3 varyingLightDirection;
out vec3 varyingViewerDirection;
out vec2 varyingTextureCoordinate;

void main(void)
{
  vec4 eyeVertex = mvMatrix * vertex;
  eyeVertex /= eyeVertex.w;
  varyingNormal = normalMatrix * normal;
  varyingLightDirection = lightPosition - eyeVertex.xyz;
  varyingViewerDirection = -eyeVertex.xyz;
  varyingTextureCoordinate = textureCoordinate;
  gl_Position = mvpMatrix * vertex;
}
//! [0]
lightingFragmentShader.vsh

#version 130

uniform vec4 ambientColor;
uniform vec4 diffuseColor;
uniform vec4 specularColor;
uniform float ambientReflection;
uniform float diffuseReflection;
uniform float specularReflection;
uniform float shininess;
uniform sampler2D texture;

in vec3 varyingNormal;

/**
 ** Copyright (C) 2012 Digia Plc and/or its subsidiary(-ies).
 ** Contact: http://www.qt-project.org/legal
 ** $QT_BEGIN_LICENSE:BSD$
 ** You may use this file under the terms of the BSD license as follows:
 ** * Redistributions of source and binary forms, with or without
 ** modification, are permitted provided that the following conditions are
 ** met:
 ** * Redistributions of source code must retain the above copyright
 ** notice, this list of conditions and the following disclaimer.
 ** * Redistributions in binary form must reproduce the above copyright
 ** notice, this list of conditions and the following disclaimer in
 ** the documentation and/or other materials provided with the
 ** distribution.
 ** * Neither the name of Digia Plc and its Subsidiary(-ies) nor the names
 ** of its contributors may be used to endorse or promote products derived
 ** from this software without specific prior written permission.
 **
 ** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 ** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 ** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 ** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 ** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 ** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 ** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 ** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 ** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 ** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 ** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
 **
 ** $QT_END_LICENSE$
 **
***************************************************************************/

//! [0]
in vec3 varyingLightDirection;
in vec3 varyingViewerDirection;
in vec2 varyingTextureCoordinate;

out vec4 fragColor;

void main(void)
{
    vec3 normal = normalize(varyingNormal);
    vec3 lightDirection = normalize(varyingLightDirection);
    vec3 viewerDirection = normalize(varyingViewerDirection);
    vec4 ambientIllumination = ambientReflection * ambientColor;
    vec4 diffuseIllumination = diffuseReflection * max(0.0,
        dot(lightDirection, normal)) * diffuseColor;
    vec4 specularIllumination = specularReflection * pow(max(0.0,
        dot(-reflect(lightDirection, normal), viewerDirection)
            ), shininess) * specularColor;
    fragColor = texture2D(texture,
        varyingTextureCoordinate) * (ambientIllumination +
        diffuseIllumination)
        + specularIllumination;
}
REFERENCES


Qt (version 4.8.6) [Application Framework]. Oslo, Norway: Qt Company.


Terra Vista (version 14) [Computer software]. Montreal, Quebec, Canada: Presagis.


