Technical University Munich
Chair of Geoinformatics

Simulation of explosions in urban space and result analysis based on CityGML-City Models and a cloud-based 3D-Webclient

Master Thesis

vorgelegt von:	Bruno Willenborg
Studienbereich:	Environmental Engineering
Erstgutachter:	Univ.-Prof Dr. rer. nat. Thomas H. Kolbe
Zweitgutachter:	Dipl.-Geogr. Maximilian Sindram
Bearbeitungszeitraum:	1.8.2014 - 1.2.2015

Acknowledgments

I thank the Chair of Geoinformatics at Technical University Munich for giving me the opportunity to write this mater thesis. Especially I want to thank Maximilian Sindram for his professional advise at any time.

I thank in particular Arno Klomfass from Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute (EMI) who readily provided support at any time with the Apollo Blastsimulator.

I would also like to thank the 'Bayrisches Landesamt für Digitalisierung, Breitband und Vermessung' for providing test data.

Contents

List of Figures **v**

List of Tables **vii**

List of Listings **ix**

Abbreviations **xi**

List of Symbols **xiii**

1 Introduction **1**
1.1 Problem, motivation and relevance 2
1.1.1 Motivation 2
1.1.2 Preconditions and Objectives 3
1.1.3 Relevance 4
1.2 Research questions 4
1.3 Structure and content 4

2 Theoretical basis **5**
2.1 CityGML 5
2.1.1 Semantics and modeling aspects 5
2.1.2 Extensibility of CityGML 6
2.1.3 Building model 7
2.1.4 Geometry and topology model 9
2.2 3D City Database for CityGML 11
2.2.1 3D City DB generic objects model 12
2.2.2 3D City DB geometry model 12
2.2.3 PostgreSQL 14
2.2.4 PostGIS 15
2.3 3D City Database Importer/Exporter 21
2.3.1 Extension with the Plugin API 21
2.3.2 Spreadsheet Generator Plugin 24
2.3.3 KML/KOLLADA Plugin 24

2.4 Cloud-based 3D Web Client . . . 26
2.4.1 Coupling of cloud-based Spreadsheets and 3D visualization . . . 27
2.4.2 GUI and capabilities . . . 27
2.5 Apollo Blastsimulator . . . 30
2.5.1 Calculation models: Engineering method and CFD simulation . . . 30
2.5.2 Physical model and optimizations . . . 31
2.5.3 Dynamic mesh adaption . . . 31
2.5.4 Voxel model specifications . . . 33
2.5.5 Input data file description . . . 35
2.5.6 Damage model and simulation results . . . 37

3 Coupling of CityGML city models and the Apollo Blastsimulator 39
3.1 System architecture and workflow . . . 39
3.1.1 Main system components . . . 39
3.1.2 A typical workflow . . . 40
3.1.3 Relational database model . . . 41
3.2 Apollo Blast 3D City DB Importer/Exporter plugin . . . 44
3.2.1 Overview and package structure . . . 44
3.2.2 Plugin API: Usage and implementation . . . 46
3.2.3 Specification of scenario informations . . . 48
3.2.4 Data structure, storage and exchange of scenario definitions . . . 51
3.2.5 Simulation area and voxel grid description . . . 53
3.2.6 Database interaction . . . 56
3.3 Derivation of the Apollo Blastsimulator Voxel model from CityGML . . . 59
3.3.1 Voxel representation and creation with PostGIS . . . 59
3.3.2 3D intersection with PostGIS . . . 62
3.3.3 Multi-threaded intersection of voxels and CityGML geometries . . . 65
3.4 Execution and controlling of a simulation run . . . 68
3.4.1 Computation of the engineering method . . . 68
3.5 Processing and storage of simulation results in CityGML . . . 69
3.5.1 Simulation results processing . . . 69
3.5.2 Storage as generic attribute . . . 70
3.6 Visualization of Simulation results with the cloud-based 3D Web Client . 73
3.6.1 Preparation: Data export and distribution . . . 73
3.6.2 Setting up the 3D Web Client . . . 76

4 Evaluation and performance 79
4.1 Evaluation of blast events with the 3D Web Client . . . 79
4.1.1 Description of the test scenario . . . 79
4.1.2 A practical usage example: Cost estimation for glass breakage with the 3D Web Client . . . 82
4.1.3 Comparison of simulation in free field and with buildings . . . 86
4.2 Influence of the mesh resolution on simulation results . . . 89

4.3 Performance 95
4.3.1 Multi-threaded 3D intersection performance 96

5 Final considerations and outlook **99**
5.1 Conclusion 99
5.2 Outlook 100

Bibliography **103**

Appendix **107**

A EMI Concept paper **107**

B EMI specifications for the voxel model **115**

C CityGML UML diagrams **125**

D 3DCityDB UML diagrams **129**

E XML examples **133**

F Apollo Blastsimulator results import scripts and example **137**

G Apollo Blast Plugin sample log file **139**

H 3DCityDB ImporterExporter WorkerPool UML Charts **141**

List of Figures

2.1 Modularization of CityGML 7
2.2 Simplified UML diagram of the CityGML *Building* model with no attributes displayed 8
2.3 Combined geometries according to GML 9
2.4 Recursive aggregation and topology of CityGML geometries 10
2.5 Simplified modeling of polygon-based GML geometry classes 13
2.6 UML diagram of the OGC Simple Feature Geometry Type hierarchy . . 16
2.7 PolyhedralSurface with consistent orientation observed from the top . . 18
2.8 3D R-Tree visualization with cubic edges 19
2.9 ImporterExporter preferences window 22
2.10 ImporterExporter GUI 23
2.11 KML representation of the TUM campus LoD2 model in GoogleEarth . 26
2.12 Coupling of cloud-hosted spreadsheets and 3D visualization models . . . 29
2.13 Overview of the GUI of the 3D Web Client 29
2.14 Illustration of mesh refinement in 2D for level 1-4 32
2.15 Automatic Mesh Refinement during blast wave propagation 33
2.16 Voxel representation of the TUM campus, $L_{voxel} = 1.625\,\mathrm{m}$ 34
2.17 Apollo Blastsimulator voxel model coordinate references 36

3.1 System architecture and workflow 41
3.2 Relational database schema of the Apollo Blast Plugin 43
3.3 Package structure of the Apollo Blast Plugin 45
3.4 Apollo Blast Plugin Explosive Devices dialog 48
3.5 Apollo Blast Plugin Explosion Source dialog 48
3.6 Apollo Blast Plugin Calculation options dialog 49
3.7 Settings panel of the Apollo Blast Plugin 50
3.8 JAXB binding of XML to Java objects 52
3.9 Simplified UML model of the Explosive Device data structure 52
3.10 Simplified UML-Diagram of the voxel model 54
3.11 Adaption of the max. hazard radius to fit the Apollo Blastsimulator zoning 55
3.12 Query storage in the Apollo Blast Plugin 57
3.13 Voxel with face IDs according to EMI specifications 60
3.14 Worker Pool implementation UML Chart 67

3.15 Voxel grid with highlighted slices 67
3.16 Apollo Blast Plugin Simulation control dialog. 68
3.17 Comparison of aggregations of simulation results 72
3.18 Spreadsheet Generator Plugin with example column definition. 75
3.19 Excerpt of a GoogleSpreadsheet with attributes for the engineering method. 75
3.20 Add *KML/KMZ Layer* dialog. 76
3.21 *Project Management* dialog. 76
3.22 GUI of the 3D Web Client 77

4.1 LoD2 CityGML city model of the test area around TUM 80
4.2 Voxel mesh of the test scenario at TUM with highlighted mesh errors . . 81
4.3 Selection by Attribute. 82
4.4 Stored Viewpoint. 83
4.5 Selection results and aggregation of surface area. 84
4.6 The TUM main building in its GoogleEarth 3D building representation. 85
4.7 Distribution of the peak overpressure (P_{max}) compared for simulation in free field and with buildings. (Part 1) 87
4.8 Distribution of the peak overpressure (P_{max}) compared for simulation in free field and with buildings. (Part 2) 88
4.9 Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 3$, $L_{voxel} = 1.625\,\mathrm{m}$. 91
4.10 Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 4$, $L_{voxel} = 0.8125\,\mathrm{m}$. 92
4.11 Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 5$, $L_{voxel} = 0.406\,25\,\mathrm{m}$. 93
4.12 Additional surfaces endangered of glass breakage from $MaxLevel = 3$ to $MaxLevel = 5$ voxel mesh. 94
4.13 Time consumption of mesh generation for one to four workers, $MaxLevel = 2, 3$. 96
4.14 Time consumption of mesh generation for one to four workers, $MaxLevel = 4, 5$. 97

5.1 BBox selection dialog of the 3DCityDB ImporterExporter. 102

C.1 UML diagram of CityGML's bulding model 126
C.2 UML diagram of CityGML's geometry model 127

D.1 Database schema of the building model 130
D.2 Database schema of the generic objects model 131

H.1 3D CityDB ImporterExporter Plugin API Worker Pool implementation 142

List of Tables

2.1 Generic Attribute data types 12
2.2 Boolean flags determining aggregation types 13
2.3 Dimensionally Extended Nine-Intersection Model (DE-9IM) 18
2.4 Definition of mesh levels. Resolutions are subdivisions by power of 2 . . . 32
2.5 Apollo Blastsimulator hazard classes for buildings and persons. 37

3.1 Mapping of Quality levels in the Apollo Blast Plugin to mesh refinement levels 50
3.2 Intersection tests for Polygon and PolyhedralSurface 64
3.3 Generic attributes for the engineering method. 71
3.4 Generic attributes for the Apollo Blastsimulator results. 73

4.1 Estimated time consumption for the example scenario. 84
4.2 Voxel meshes with $L_{zone} = 6.5\,\mathrm{m}$ for the TUM test area. 95

List of Listings

2.1 WKT representation of a cubic PolyhedralSurface with outward oriented surfaces. 17
2.2 Implicit and explicit bounding box comparison with PostGIS. 20

3.1 Code snippets to plugin API controllers and event dispatcher. 47
3.2 JAXB example. 52
3.3 Example log message of the VoxelGridBounds class. 55
3.4 Minimal Java example of the database concept. 58
3.5 Simplified example of the voxel generation with a Pl/pgSQL function. . 61
3.6 Query for creation and intersection of voxels and city model geometries. 63
3.7 Pl/pgSQL function for the scaled distance. 69

E.1 XML instance document of the explosive device data structure. 133
E.2 XML instance document of the simulation parameters data structure. . . 134
E.3 XML instance example of a CityGML *WallSurface* with generic attributes of the Apollo Blast Plugin. 135

F.1 Shell script for Apollo Blastsimulator results file CSV import preparation. 137
F.2 Batch cmd line for CSV import ot PostgreSQL. 137
F.3 Sample results file of the ApolloBlast Simulator. 138

G.1 Sample ImporterExport log of a simulation run. 139

Abbreviations

Notation	Description
3DCityDB	3D City DB for CityGML
ABS	Apollo Blastsimulator
ADE	Application Domain Extension
AMR	Automatic Mesh Refinement
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BBox	Bounding Box
BRep	Boundary Representation
CFD	Computational Fluid Dynamics
CityGML	City Geography Markup Language
CLI	Command Line Interface
CTE	Common Table Expression
DB	Database
DBMS	Database Management System
DE-9IM	Dimensionally Extended Nine-Intersection Model
EMI	Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute
GIS	Geographical Information System
GiST	Generalized Search Tree
GML3	Geography Markup Language version 3.1.1
GUI	Graphical User Interface
ImporterExporter	3D City Database Importer/Exporter
JAXB	Java Architecture for XML Binding 2.0
JDBC	Java Database Connectivity
LGPL	GNU Lesser General Public License

Notation	Description
LoD	Level of Detail
MRS	Measure Reference System
OGC	Open Geospatial Consortium
PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language
PostGIS	Post Geographical Information System
PostgreSQL	Postgres Structured Query Language
PSC	PostGIS Project Steering Committee
SFS	Simple Feature Specification
SPSHG	Spreadsheet Generator
SQL	Structured Query Language
SRID	Spatial Referencing System Identifier
SRS	Spatial Reference System
TIN	Triangulated Irregular Network
TUB	Technical University Berlin
TUM	Technical University Munich
UML	Unified Modelling Language
UXOs	Unexploded Ordnance
Voxel	Volume Pixel
WKT	Well-Known Text
XML	Extensible Markup Language

List of Symbols

Notation	Description
L_{voxel}	Edge length of a voxel. [m]
L_{zone}	Edge length of a zone of the voxel grid [m]
M_{TNT}	mass of TNT [kg]
n_{voxel}	Number of voxels per zone edge [-]
P_{ex}	Source of the explosion given in x_0, y_0, z_0
R_{max}	Maximum hazard radius [m]
$R_{max,temp}$	Temporary maximum hazard radius [m]
V_{bbox}	Volume of the bounding box [m^3]
$V_{bbox,temp}$	Volume of the temporary bounding box [m^3]

CHAPTER 1

Introduction

Urbanization is one of the biggest trends of our time. While today already more than 50% of the global population live in urban areas, this figure is estimated to grow up to 70% by the year 2050. (Chen et al., 2012) Due to dens population and concentration of infrastructure and business, this space is growingly vulnerable. One of the most threatening scenarios endangering these regions are explosions.

There are various possible sources for such events with different occurrence probability and damage potential. Accidents with fuels like natural gas are relatively seldom and have very low death rates. (Burgherr, 2015) In Germany's urban centers a much bigger threat is Unexploded Ordnance (UXOs) from world war two. According to estimations a total of 135 000 up to 270 000 UXOs remained in German soil after the war. In 2013, the number of undiscovered units was estimated to a total of 100 000. Every year 5500 UXOs are defused. Frequently disarming or transportation is too dangerous and the devices have to be exploded in situ, as recently experienced in Munich. On 28. August 2012 an American 500 pound aerial bomb uncovered during construction works in Schwabing was detonated in place causing the evacuation of 2500 people. Windows were shattered in a wide area and many buildings suffered structural damage despite precautions taken against the blast wave. (Görl, 2013; Weiss, 2012; Wikipedia, 2015a,b)

The worst case are obviously terrorist attacks, since they will never be completely avoidable and cannot be predicted regarding their damage potential. Therefore measures have to be taken to protect humans and structures against blast effects.

To make the required simulation of blast events and the investigation of their effects fast, reliable and comprehensible, the Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute (EMI) in Freiburg, Germany developed a tool based on Computational Fluid Dynamics (CFD) specialized to detonations and blast waves. The tool, named Apollo Blastsimulator (ABS), allows risk assessment regarding people and structures using what-if analysis even for non-CFD experts. It operates on a finite volume mesh of the built environment, which has to be computed prior to the simulation. (Klomfass and Zweigle, 2013)

An attractive data source for the mesh generation are semantic 3D city models based on open standards like the City Geography Markup Language (CityGML). Their models

become more and more technically mature and data are increasingly available. According to Bayrisches Landesamt für Digitalisierung, Breitband und Vermessung, 2015, already about one third of Bavaria is mapped in CityGML to a Level of Detail (LoD) including simple 3D building geometries with more to come in the near future.

To make these data usable for blast simulations with Apollo Blast an interface between the CityGML standard and the simulation tool is required. Developing this interface and embedding it in a work flow involving all steps from scenario definition over data storage and processing to visualization of simulation results, designed for users with expert knowledge in neither Geographical Information Systems (GISs) nor blast simulations is what this research work is about.

1.1 Problem, motivation and relevance

This thesis emerged from a cooperation of the Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute (EMI) in Freiburg, Germany, and the Chair of Geoinformatics of the Technical University Munich (TUM). Both parties contributed one of the two key components of this project. The CityGML standard and resourceful set of tools around it covering topics from data management to visualization are currently developed at TUM.

The Apollo Blastsimulator is a CFD-based simulation tool specialized to blast events. Based on a finite volume model the software computes the physical effects of detonations and facilitates their assessment regarding risks for structures and people. It is developed and maintained by EMI.

1.1.1 Motivation

Generally speaking, this thesis deals with the coupling of a Geographical Information System (GIS) and a physical simulation in a 3D spatial environment. In the last 30 years, both branches have developed to valuable tools in analysis and decision making regarding natural phenomena. However, due to high specialization, both systems have their strengths and limitations. A big motivation for this project lies in the great potential of combining both systems and make them benefit from each other.

Today's GISs are very well adapted for management, querying, visualization and analysis of spatial data. According to Li et al., 2007, one thing they are currently widely lacking is the representation of time, their data models are static. Hence, dynamic processes cannot be modeled and calculated directly in the system. This is where simulation tools like the Apollo Blastsimulator come in. To efficiently deal with the dynamic nature of the problem they were designed for, they rely on data structures adapted for this kind of problem. Usually, the spatial data required for a simulation run, are not available in that specialized representation making expensive data collection or conversion from a commonly used exchange format necessary.

CityGML is a data model for the representation and exchange of virtual 3D city models. It is an open standard issued by the Open Geospatial Consortium (OGC) and therefore well suited for developing an interface to simulation tools. It's well accepted around the

world and offers a huge amount of ready-to-use data. Moreover, in support of the standard a set of extensible open source software for various tasks exists. By the extension of these tools, developers are enabled to engineer relatively cheap solutions for a specific topic and benefit of an established product at the same time. The coupling approach for CityGML and the Apollo Blastsimulator introduced in this study is strongly built on this concept.

1.1.2 Preconditions and Objectives

The most important precondition for this project is full conformity to the CityGML standard. The solution introduced in this thesis shall implement the standard on all levels to remain open to a broad set of possible usage scenarios and benefit from its good acceptance in business and public administration all around the world. As CityGML is an open standard adapted by the OGC consequently the second precondition is to provide an *Open Source Only* solution for the given problem.

In the startup phase of the project a concept paper was delivered from EMI giving detailed information on the preconditions and objectives regarding the Apollo Blastsimulator. The full paper can be viewed in Appendix A, in this section only the key elements are presented. Summarized the general objectives for this project are:

- Development of an approach making CityGML city models available for blast simulations with the Apollo Blastsimulator
- Visual representation of simulation results (hazard areas) within a CityGML city model
- Extension of an existing CityGML viewer with a Graphical User Interface (GUI) allowing the control of the simulation process and configuration of the visualization
- Usage of open source tools and standards only

To achieve the given objectives, four main categories of challenges to the project could be identified. Combined with the demands listed in the EMI paper they read as follows:

- **User interaction**
  Development of a GUI allowing non expert users to define blast scenarios and control their simulation with the Apollo Blastsimulator. The scenario definition includes the selection of an explosion location and explosive device and the configuration of the several simulation model parameters.
- **Data storage and processing**
  The data acquired in the scenario definition should be stored persistently in a format enabling their easy exchange and reuse. Data processing is mainly required to convert the CityGML data into the model the Apollo Blastsimulator requires for simulation. After a simulation run the results need to be transfered back into the city model, where they are persistently stored for further usage.
- **Simulation**
  Based on the scenario definition blast events shall be simulated. Two different calculation approaches need to be implemented. A simplified, empirical solution and a CFD simulation performed with the Apollo Blastsimulator.

- **Visualization and analysis**
  The results of the blast simulation need to be visualized with an existing CityGML viewer. Moreover, it should be possible to perform analytic tasks with the data.

1.1.3 Relevance

As mentioned before, UXOs regularly cause explosions in urban areas. Additionally, global terrorism is an issue of our time we are confronted with in the news on an almost daily basis. According to Trometer et al., 2014 also mainly urban centers are endangered by this threat. Hence, the strategic and conceptual preparation for individual blast scenarios in cities is required. Modeling and simulation have become a vital tool for specialists and decision makers in the field of risk assessment. (Li et al., 2007) To perform such simulations, mechanical models, that are usable by simulation tools are required. CityGML is an attractive data source for this purpose as it is well accepted and offers good data availability. Hence, the development of a method allowing the mechanical model generation, visualization and the assessment of simulation results for this standard is a valuable contribution to enable better predictions and preparation for blast events in cities.

1.2 Research questions

Summarizing the aforementioned preconditions and objectives for this project, four central research questions have been extracted.

1. How can the Apollo Blastsimulator be embedded in workflow based on CityGML city models that enables non expert users to perform blast simulations and asses their results?
2. How can the mechanical model the Apollo Blastsimulator requires for simulations efficiently be derived from the CityGML geometry model?
3. How can simulation results produced by the Apollo Blastsimulator be stored in CityGML city model?
4. How can simulations results stored in a CityGML city model be visualized and analyzed?

1.3 Structure and content

The structure of this thesis pretty straightforward. In Chapter 2, the theoretical basis of this study will be explained. All tools, standards and concepts used in the project will be briefly explained. After that, the coupling approach for CityGML and the Apollo Blastsimulator developed in this study will be discussed. Both conceptual and implementation details are described in this chapter. Chapter 4 will discuss and evaluate the results that can be generated with the implementation. Finally in Chapter 5, a resume on this study and an outlook to what the future might bring for this project will be given.

CHAPTER 2

Theoretical basis

In this chapter the theoretical basis this project is built on will be discussed. Besides details on CityGML and the tools around it, the Apollo Blastsimulator will be described in detail. To get an idea of how the single components explained here are related and how they interact, I want to encourage you to frequently skip forward to Figure 3.1 on page 41. It gives an overview of the workflow these components operate in and helps to set the different parts in context.

2.1 CityGML

One of the two main components worked with in this thesis is the CityGML standard, which is an open data model for the representation and exchange of virtual 3D city and landscape models. The model includes information on geometry, appearance, semantics and topology of its objects. It is based on the Extensible Markup Language (XML), the ISO 19100 standards family and the Geography Markup Language version 3.1.1 (GML3), which it is an application schema of. CityGML is an official OGC standard, in 2012 its latest issue, version 2.0.0, was released. (Kolbe, 2009; Open Geospatial Consortium, 2012)

2.1.1 Semantics and modeling aspects

The CityGML model not only represents the shape and graphical representation of city objects. A big emphasis is put on the objects semantics and their thematic properties, taxonomies, aggregations and interrelation. Objects are decomposed following not their graphical appearance, but logical criteria which can be observed in the real world according to ISO 19109 definition of geographic objects. This enables CityGML to serve as an information model for applications that require more than the visual appearance of cities, like urban planning or environmental simulations. The CityGML standard defines a common understanding of the segmentation of real world objects of a city like for example buildings, transportation systems or vegetation.

Keeping the Standard open and applicable to a multitude of different domains, was the central idea during its creation. More than 70 members from industry, science, and public administrations contributed to the development process with their demands, experiences

and expertise. All modeling suggestions were rated regarding their multi functional use for various application domains. This way a common definition of entities, attributes and their interrelations could be deduced.

Figure 2.1 shows the resulting functional partitioning of CityGML in modules. The horizontal modules (*Core, GML, Appearance* and *Generics*) contain the most general semantics (classes and attributes) applying to all entities of the model. Most important is the *Core* module as it contains the basic components and concepts of the data model. Every thematic extension needs to implement this module. The abstract class `_CityObject` is its root class. It inherits the required attributes from GML3 abstract `_Feature` (`name`, `gml:id`, `description`) and adds `creationDate` and `terminationDate` to allow histories of features. Moreover it allows via `ExternalReferences` the linking of city objects to external data. Entire city models can be aggregated using the *CityModel* class.

The vertical modules represent the semantic modeling of the different application domains of CityGML like *Building* or *Vegetation*. Theses modules are designed in a universal manner to cover the needs of a wide range of applications. In this study mainly the *Building* model was used, which is explained in detail in Section 2.1.3. The other thematic classes the standard proves are shown in Figure 2.1. They offer a rich set of entities allowing the 'out of the box' representation of both urban and rural scenarios.

All CityGML objects can be represented in five different LoDs. Starting with LoD0, objects gain more details regarding both geometry and thematic information up to LoD4. Each object may be available in different LoDs in the same dataset simultaneously. This allows to conduct analysis and visualization in different resolutions. Moreover the LoD concept makes CityGML datasets easily comparable and gives users an idea of the nature of data regarding granularity, complexity and accuracy. (Kolbe, 2009; Open Geospatial Consortium, 2012)

2.1.2 Extensibility of CityGML

Obviously the aforementioned thematic classes do not cover all possible usage scenarios for semantic 3D city models. Practical applications will frequently need to store and exchange additional information. According to Open Geospatial Consortium, 2012 and Kolbe, 2009 CityGML offers two ways of extension. First, the whole city model can be expanded with new 3D objects, so called *GenericObjects*. Existing entities can be enhanced with an arbitrary number of *GenericAttributes*. As shown in Figure 2.1, these options are provided by the *Generics* module. The second extension mechanism is called Application Domain Extension (ADE). Generally using ADEs achieves the same as working with *Generics*, but they have to be defined within an separate XML schema definition and namespace. This allows the validation of CityGML instance documents against an ADE. As shown on Figure 2.1, ADEs are thematic classes just like, for example *Building* for a specific application domain like *Noise Immission*.

Since CityGML does not provide a package for the simulation of blast events, the usage of at least one extension mechanism is required to allow the storage of simulation results in the city model as mentioned in Section 1.2. Details on this will be discussed later in the study.

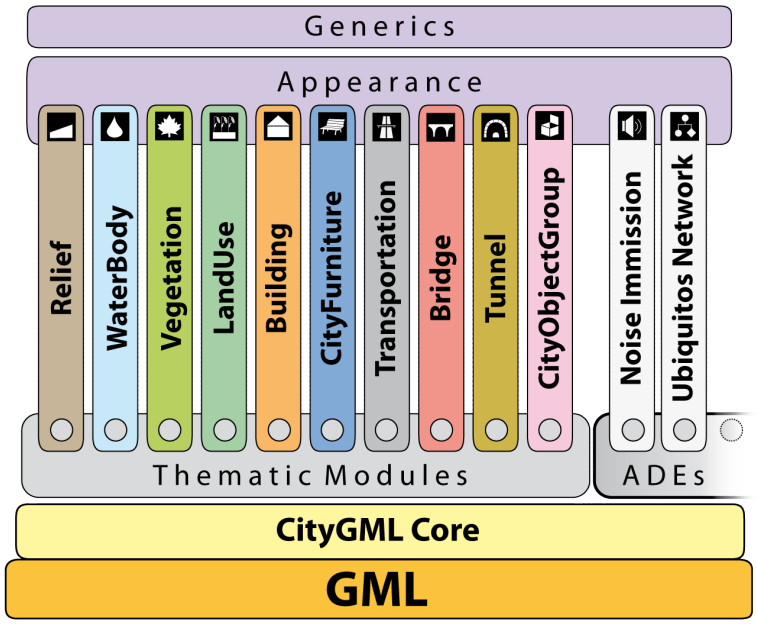


Figure 2.1: Modularization of CityGML. Horizontal modules apply to all the vertical modules representing the different thematic domains. Image from: http://www.virtualcitysystems.de

2.1.3 Building model

The *Building* model is an essential component for blast simulation in urban scenarios. It is one of the most detailed and complex thematic models CityGML provides. Figure 2.2 displays its Unified Modelling Language (UML) diagram in a simplified version for better overview. The full UML chart can be found in Appendix C, Figure C.1.

The pivotal class of the package is the abstract class `_AbstractBuilding` which inherits from `_CityObject` and adds general attributes regarding geometry (stories below/above ground, roof type, measured height) and semantic (class, function, usage). It is specialized to `Building` and the aggregation class `BuildingPart` allowing the construction of hierarchical building complexes of arbitrary depth. Postal information can be attached via the `Address` class.

The geometric and semantic representation of buildings is strongly governed by the LoD concept described in Section 2.1.1. Starting from LoD0 building objects can be refined regarding their geometric shape on the inside and outside, semantics and aggregations with every level. LoD0 supports the representation of buildings by their 3D horizontal footprint and roof surface. In LoD1, a building is represented by a simplified model of its outer shell, the well known block model. As well a `TerrainIntersectionCurve` can be specified. LoD2 introduces semantics for buildings components and allows the modeling of architectural details. With the abstract class `_BoundarySurface` and its derived concrete classes like `WallSurface`, `RoofSurface` or `GroundSurface` a thematic class for all essential components of the outer building shell is provided. Moreover `BuildingInstallations`

are added on LoD2. They allow the modeling of elements of the outer building facade like balconies or chimneys. Starting from LoD3 openings in buildings can be thematically modeled using the abstract class `_Openings` with its subclasses `Window` and `Door`. The highest resolution level, LoD4, allows the representation of the building's interior with the thematic class `Room`. Movable furniture is represented by the class `BuildingFurniture` and is assigned to a `Room`. In contrast immobile interior installations are represented by the class `IntBuildingInstallation`. (Kolbe, 2009; Open Geospatial Consortium, 2012)

In this study only buildings of LoD1 and LoD2 will be used, since only the representation of the outer building shell is required and too many details could well be obstructive for the derivation of a high quality the voxel model described in Section 2.5.4. Though, the features LoD3 and LoD4 offer could be of interest for future development as they allow the assessment of blast effects on the interior of buildings.

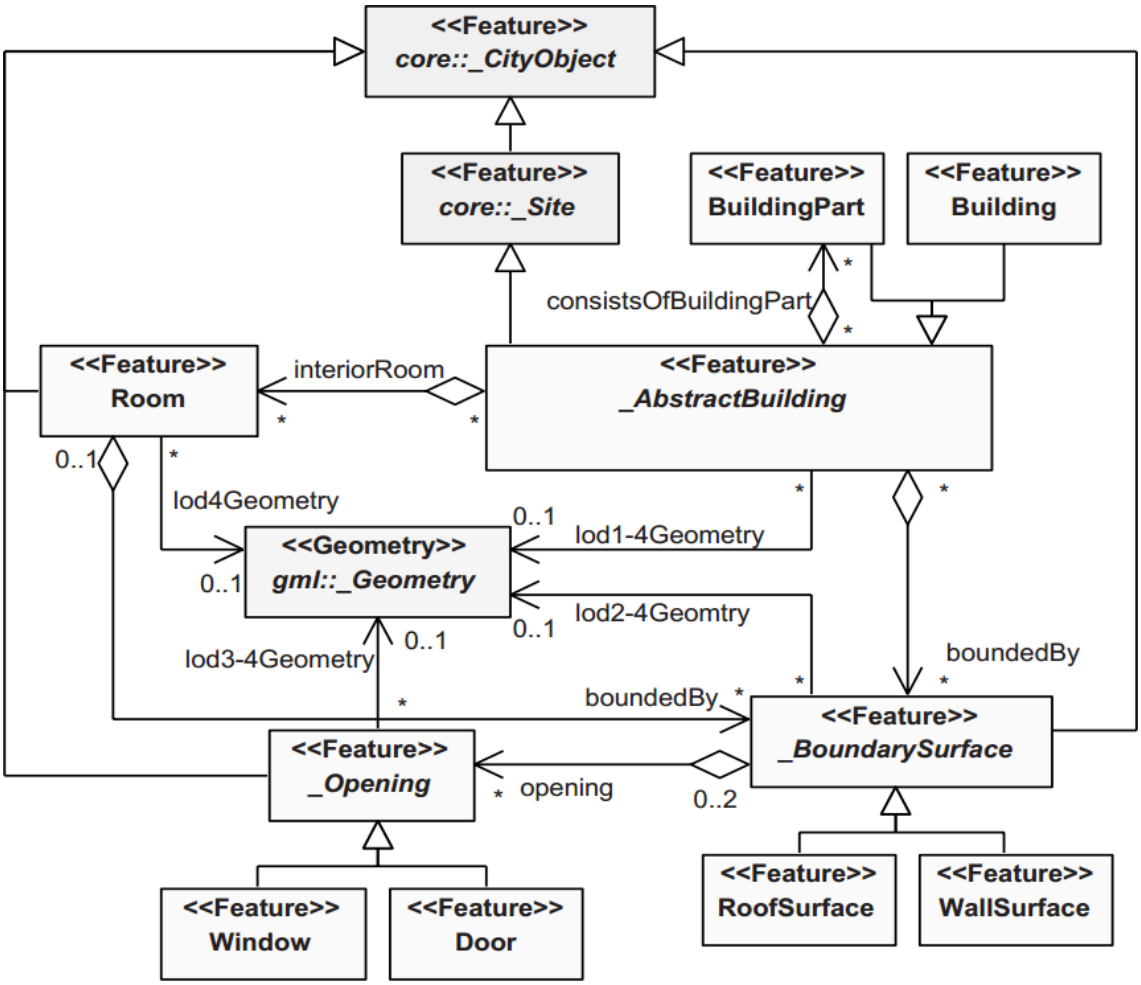


Figure 2.2: Simplified UML diagram of the CityGML *Building* model with no attributes displayed. Image from: Kolbe, 2009

2.1.4 Geometry and topology model

Following Open Geospatial Consortium, 2012 and Kolbe, 2009 the representation of its geometric objects CityGML defines a GML3 profile, which contains a subset of the GML3 geometry model. Figure C.2 from Appendix C gives an overview on the contained geometry classes. GML3, which is based on the 'Spatial Schema' of the ISO 19107 standard (Herring, 2001), represents geometries as *geometric primitives* (e. g. point, linestring, polygon) and combinations of those. Figure 2.3 illustrates the three combination types, aggregation, complex and composite. In geometric aggregates like `MultiPoint` or `MultiSurface` the relation between the single primitive elements is not restricted. In contrast, both *Geometric Complexes* and *Geometric Composites* are topologically structured. Their elements have to be disjoint and non-overlapping. Primitives in complexes are allowed to touch or share (a part of) their boundaries, while primitives in composites have to be connected along their boundaries. For each dimension one aggregate exists (`MultiPoint`, `MultiCurve`, `MultiSurface`, `MultiSolid`), composites can be a `CompositeCurve`, a `CompositeSurface` or a `CompositeSolid`. Volumetric geometries are constructed by a closed surface, following the principles of the well-known Boundary Representation (BRep).

A special type of surface is the `TriangulatedSurface`. It is a topologically structured aggregation of `Triangles`, a so called Triangulated Irregular Network (TIN) which is besides raster, mass points or breakline reliefs used for terrain representation. An indirect reproduction is provided by the subclass `TIN`. Based on `controlPoints`, `stopLines` and `breaklines` the triangulation can be computed when needed.

The aforementioned primitive geometries can be ordered in hierarchical structures of the same dimension using a recursive aggregation schema. Like this, nested object structures of arbitrary depth can be constructed allowing for example the decomposition of a house geometry (`CompositeSolid`) into its body (`CompositeSolid`) and its garage geometry (`Solid`). Figure 2.4 gives an illustration of this example.

Besides the description of geometries, CityGML enables the explicit modeling of topological interrelations of features, which is vital for many applications like volume calculation.

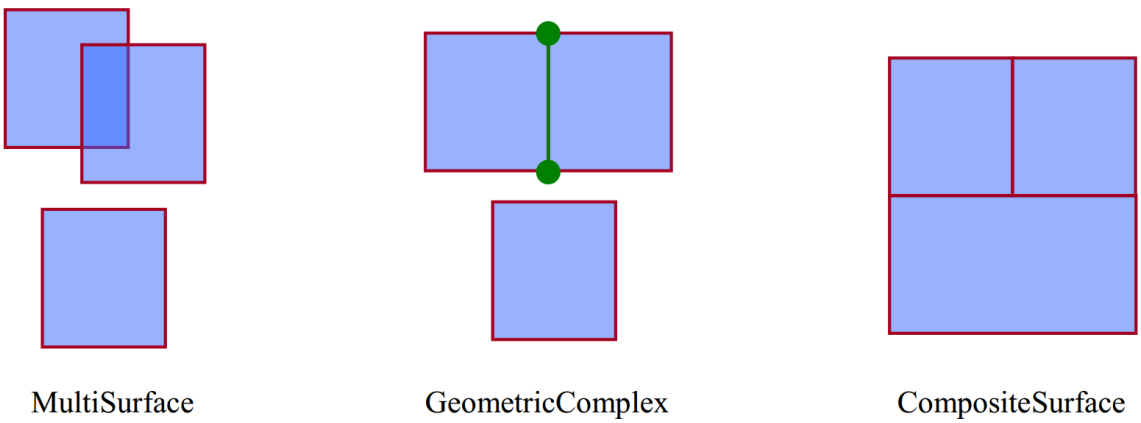


Figure 2.3: Combined geometries according to GML. Image from: Open Geospatial Consortium, 2012

To achieve topological correctness redundant geometries have to be avoided. Figure 2.4 displays the aggregation of two solids. To maintain the explicit topological relation between both of them, their shared geometry is stored only in one of the two solid. The other geometry receives a reference to include the shared surface in its outer shell. As the example in Figure 2.4 shows, this may require the partitioning of geometries, which goes beyond the specifications of BRep, where only visible surfaces are considered. For the sake of explicit topological adjacency and possible deletion without leaving holes in the remaining geometries this is neglected. Topology is realized by using the Extensible Markup Language (XML) concept of *XLinks*. GML3 provides every geometry with a unique identifier (`gml:id`), which can be referenced by other objects using a `xlink:href` attribute.

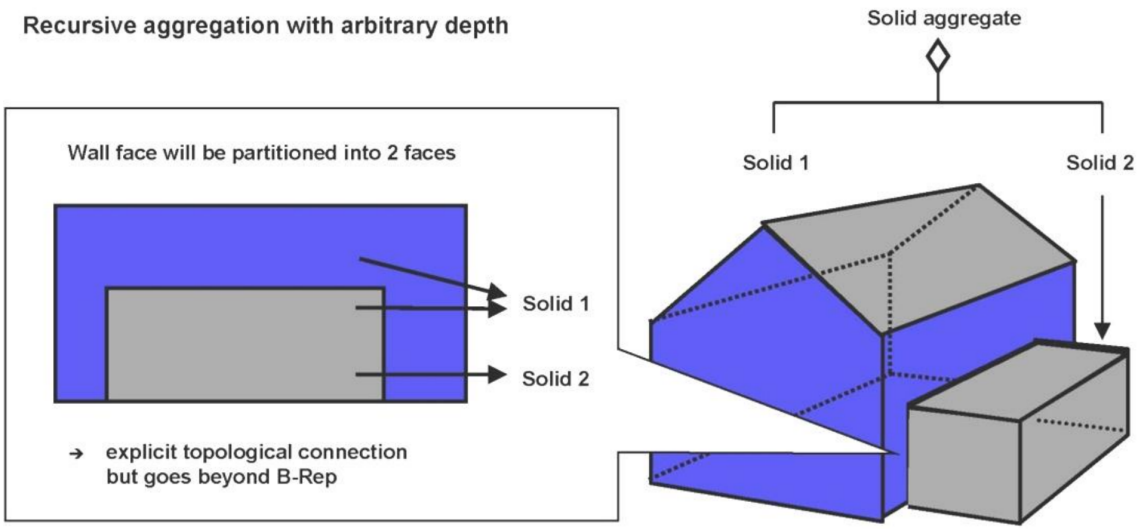


Figure 2.4: Recursive aggregation and topology of CityGML geometries. Image from: Open Geospatial Consortium, 2012

2.2 3D City Database for CityGML

The 3D City DB for CityGML (3DCityDB) is the database solution for management of city models in the CityGML format. Basically the 3DCityDB is a Database (DB) schema for standard object-relational DBs with the ability to handle spatial data. The software is open source and released under the GNU Lesser General Public License (LGPL). Currently there are two versions for different Database Management System (DBMS) available, one for Oracle 10g Spatial and one for Postgres Structured Query Language (PostgreSQL) extended by Post Geographical Information System (PostGIS) for spatial data. While the Oracle software is a commercial product, PostgreSQL is open source. Since one of the main constraints of this thesis is the usage of non-commercial software only, the PostgreSQL version of the 3DCityDB is used in this study. More information on both PostgreSQL and PostGIS can be found in the following sections.

The 3DCityDB offers about the same modeling capabilities like the CityGML standard version 1.0 and 0.4.0. Only some small simplifications were made to improve DB design and performance. For example attributes with multiple occurrence were replaced by a data type (e. g. String) capable of storing arbitrary content, so they could be stored in one column. Moreover some $n : m$ relations were restricted to $1 : n$ to avoid the creation of additional relationship tables. The handling of recursive relations has been simplified as well, since recursive DB queries can be very demanding. To allow the selection of for example all aggregation levels of a geometry in a single query, additional IDs storing the *parent* and the *root* objects ID have been added to the table. An example on this can be observed in Section 2.2.2. Sames applies for the simplification of GML3 geometry types. (Kolbe, Koenig, et al., 2009)

One important restriction of the current version of the 3DCityDB is its lacking support for the ADE extension mechanism. In the context of this thesis the CityGML data model needs to be enhanced for storing blast simulation results. To tackle this issue, the generic city object extension mechanism will be used. It is fully supported by the current version of the 3DCityDB. A detailed discussion of the relational schema follows in Section 2.2.1.

Most relevant for this project are the CityGML building, generic city object and geometry model. The building model has been transfered to the relational schema almost one-to-one and will therefore not be explained in detail. One small change compared to CityGML is the realization of building aggregates. As Figure D.1 from Appendix D shows, the CityGML classes `_AbastactBuilding`, `Building` and `BuildingPart` have been merged into one table. Hence, the aggregation of building parts is realized by the foreign keys `BUILDING_PARENT_ID` and `BUILDING_ROOT_ID`. Furthermore thematic building parts known from Section 2.1.3 as `_BoundarySurface` are now found in the table `THEMATIC_SURFACE`. (Kolbe, Koenig, et al., 2009; Stadler et al., 2009)

Note: The relational schemata of the 3DCityDB found in Appendix D show data types of the Oracle version, yet the table structure is the same in the PostgreSQL/PostGIS release.

2.2.1 3D City DB generic objects model

As mentioned before, generic city objects will be used in this study to adapt the CityGML data model to the specific needs of the application. According to Kolbe, Koenig, et al., 2009, on the 3DCityDB the tables `GENERIC_CITYOBJECT` and `CITYOBJECT_GENERICATTRIB` realize the concept of generics as shown in Figure D.2. `GENERIC_CITYOBJECT` allow the storage of generic city objects. As these are not used in this thesis, they are not further explained here. `CITYOBJECT_GENERICATTRIB` enables the extension of existing city objects. To avoid the creation of an own table for each type of attribute, for each data type a separate column was created. The relevant field for a data set can be identified with the column `datatype` according to Table 2.1. Each city object can have an arbitrary number of generic attributes, which are referenced by the `cityobject_id` column.

Table 2.1: Generic Attribute data types. Adapted from: Kolbe, Koenig, et al., 2009

Data type	Attribute type
1	STRING
2	INTEGER
3	REAL
4	URI
5	DATE
6	BLOP
7	PostGIS GEOMETRY
8	Geometry via surfaces in table `SURFACE_GEOMETRY`

2.2.2 3D City DB geometry model

The geometry model of the 3DCityDB differs significantly from the CityGML specifications. To facilitate high database performance both the geometry representation and the handling of recursions have been simplified.

In the 3DCityDB all spatial properties are constructed out of polygons which are stored each in one row of the table `SURFACE_GEOMETRY` with the PostGIS data type `PolygonZ`. Every entry is identified by an unique `ID` and a `GMLID` to facilitate the concept of xLinks used for CityGML topology. The UML chart in Figure 2.5 illustrates how solids, composites and triangulations are composed out of these polygons by a simplified model of GML3 geometries. For example a *solid* is represented by its outer shell which is a *composite surface* which can be further decomposed to *polygons*. On the database this mechanism is realized by various flags that allow the exact categorization of geometries. Table 2.2 gives an overview on the available types and how they are represented as a database row. `isSolid` distinguished between surfaces and solids, `isComposite` separates aggregates from composites. `MultiSurface` and `MultiSolid` can be distinguished by analyzing their child elements.

Aggregations are realized by the `parent_id` which stores the `ID` of its next higher parent. Like this nested hierarchical structures of arbitrary depth can be created, similar

to the example given in Section 2.1.4 and Figure 2.4. Theses recursive structures are very expensive regarding query time, particularly because the recursion depth is unknown. To cope with this issue another ID has been introduced. The `root_id` stores for each geometry its root element.This allows to avoid recursion for typical high level queries like returning all parts of a building. (Kolbe, Koenig, et al., 2009; Stadler et al., 2009)

Table 2.2: Boolean flags determining aggregation types on the 3DCityDB. Adapted from: Kolbe, Koenig, et al., 2009

	isSolid	isComposite	isTriangulated	Geometry
Polygon, Triangle, Rectangle	☒	☒	☒	`PolygonZ`
MultiSurface	☒	☒	☒	`NULL`
CompositeSurface	☒	☑	☒	`NULL`
TriangulatedeSurface	☒	☒	☑	`NULL`
Solid	☑	☒	☒	`NULL`
MultiSolid	☒	☒	☒	`NULL`
CompositeSolid	☑	☑	☒	`NULL`

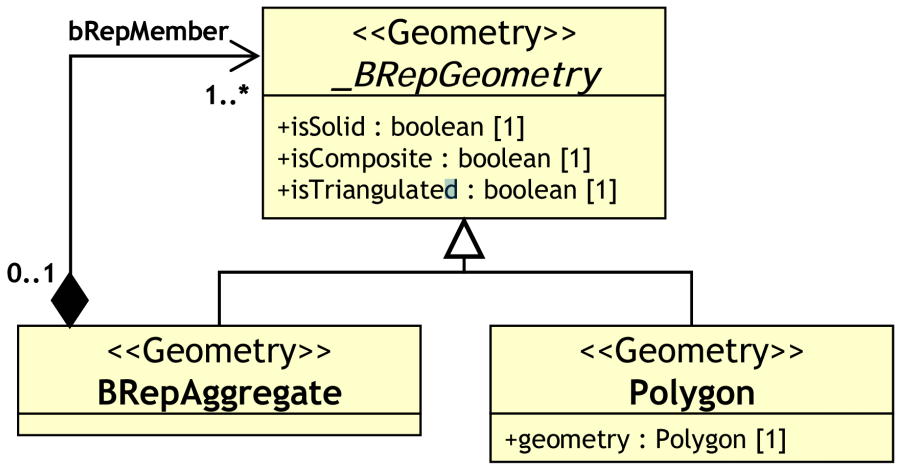


Figure 2.5: Simplified modeling of polygon-based GML geometry classes. Image from: Stadler et al., 2009

2.2.3 PostgreSQL

PostgreSQL is the open source object-relational DBMS the 3DCityDB runs on. After more than 15 years of development PostgreSQL is one the most advanced and widespread free DB systems available. It operates on every major operation system and offers interfaces to most of the popular programming languages of these days. Including its extensive documentation and liberal license[1] policy, PostgreSQL is an ideal tool for open source developers. Moreover, modern DB features like transactions, history management, Multi-Version Concurrency Control (MVCC) and an excellent query planer are provided. The system widely implements the ANSI-SQL:2008 standard, making it easy to get started, since experience from other data base systems can be instantly used. Another important feature are the flexible index structures PostgreSQL offers. The Generalized Search Tree (GiST) indexing system provides a wide range of sorting and searching algorithms like the well-known B-tree or R-tree and is easily extensible. This enables for example the efficient querying of spatial data using PostGIS.

One other aspect of PostgreSQL's highly customizable design with special relevance for this thesis are the so called *Stored Procedures*. These are database functions that allow a client to call a set of instructions in one step. They are stored on the database and extend its functionality. Various common programming languages like C, C++ or Java are supported for this. In this study its own language, *Procedural Language/PostgreSQL Structured Query Language (PL/pgSQL)*, was used. It is based on Structured Query Language (SQL) but adds several features like control structures, that are originally not supported. Moreover it inherits all user defined data types, functions and operators, allowing the usage of all PostGIS functionality. The main benefit in the usage of PL/pgSQL functions is increased performance, especially when client and server run on different machines. Since only the procedure call has to be sent to the server, additional round trips between client and server can be avoided. Intermediate results that are not needed by the client do not have to be transfered reducing network traffic. (The PostgreSQL Global Development Group, 2015a,b)

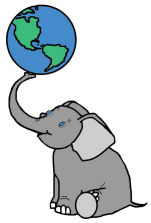
Dealing with spatial data is very costly regarding memory usage and processing power. The amount of data to be handled can be huge even for scenarios of small extent and the underlying data structures are just as complex as the algorithms working on them. Parallel computation is one approach to cope with this issue. PostgreSQL currently offers parallelism both on client and server side. Though, latter is implemented only for background processes that cannot directly be exploited by developers. Currently there is no native concurrent execution of expensive single queries that often appear with spatial tasks. A possible way of server side parallelism are the aforementioned stored procedures, which can perform parallel tasks if implemented. For the client side PostgreSQL offers parallelization based on multiple connections. An application can open several connections to a database and handle these on different threads. Like this it's possible to use multiple

1 PostgreSQL License: http://www.opensource.org/licenses/postgresql

CPUs and I/O channels. Though, each connection is limited to a single query being executed at a time. One of the biggest challenges of this study was to identify tasks that could profit from these preconditions regarding concurrent query execution and find a working implementation to it. (The PostgreSQL Global Development Group, 2011, 2015a)

2.2.4 PostGIS

PostGIS is the spatial database extension used in this project enhancing PostgreSQL to a full featured spatial DBMS. It is released under the GNU General Public License (GPLv2) and developed by contributors from all over the world led by the PostGIS Project Steering Committee (PSC). PostGIS adds spatial data types for geometry, geography, raster and more to PostgreSQL. Moreover a rich set of functions, operators and index enhancements to work with theses types is provided.



PostGIS is one of the key components of this study. All CityGML geometries and the voxel model (Section 2.5.4) are represented with PostGIS data types, all geometry processing is conducted using PostGIS functions, operators and indices. Since its initial release in 2001 it has grown to a huge open source project, for example the current documentation (see PostGIS Project Steering Committee, 2015) covers more than 700 pages. Only a small subset of PostGIS 's features is used in this project, the most important are summed up here. For detailed information please refer to the official homepage[1].

Geometry model

The geometry types PostGIS supports are based on the 'Simple Feature' specification by the OGC (Open Geospatial Consortium, 2011, 2010) and extend its definitions in some cases. Spatial 2D objects can be represented in the Well-Known Text (WKT) from according to the OGC standard. Here are examples of the three basic geometry types `POINT`, `LINESTRING` and `POLYGON`:

```
POINT(1 1)
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
```

The OGC standard requires that spatial objects formats include a Spatial Referencing System Identifier (SRID), but the text representation is not capable of that. To resolve this issue PostGIS introduced an extension to WKT. EWKT is a superset of WKT, therefore every valid WKT is a valid EWKT. Besides the additional SRID support EWKT adds the possibility to represent 3D, 3DM and 4D objects. Below are examples of 3D EWKT representations:

```
POINT(1 1 0) -- XYZ
SRID=32632;POINT(1 1 0) -- XYZ with SRID
```

1 http://postgis.net/

```
POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
```

PostGIS brings more than 40 functions to create spatial database object from all kinds of formats and representation. In this study only conversion from WKT and EWKT was performed:

```
geometry = ST_GeomFromText('POINT(1 1 0)')
geometry = ST_GeomFromEWKT('SRID=32632;POINT(1 1 0)')
```

Besides the aforementioned basic geometries PostGIS implements all other types of OGC's 'Simple Feature' specification. (Open Geospatial Consortium, 2011, 2010) As Figure 2.6 illustrates they are organized in a hierarchical structure. All types inherit a Spatial Reference System (SRS) and optionally a Measure Reference System (MRS) from the pivotal element `GEOMETRY`. The subclasses of `GEOMETRY` differentiate the model into the three geometric primitives (`POINT`, `CURVE`, `SURFACE`) plus the superclass for geometry aggregates `GEOMETRYCOLLECTION`. Below that all types can be decomposed to `POINT`s. For example a `POLYHEDRALSURFACE` consists of one or more `POLYGON`s which are represented by one or more closed `LINESTRING`s, the `LINEARRING`s which are made of `POINT`s. All geometries can be aggregated to either heterogeneous (`GEOMETRYCOLLECTION`) or homogeneous collections (e. g. `MULTIPOINT`).

Of special importance to this study is the `POLYHEDRALSURFACE` as it is being used to model the voxel model (Section 2.5.4). It is a continuous collection of polygons with no gaps or overlaps between the single elements. A LineString of a polygon patch shall at most be the boundary of two adjacent polygons. A PolyhedralSurface is well described by looking at a TIN, which is a special case of a PolyhedralSurface consisting of triangles

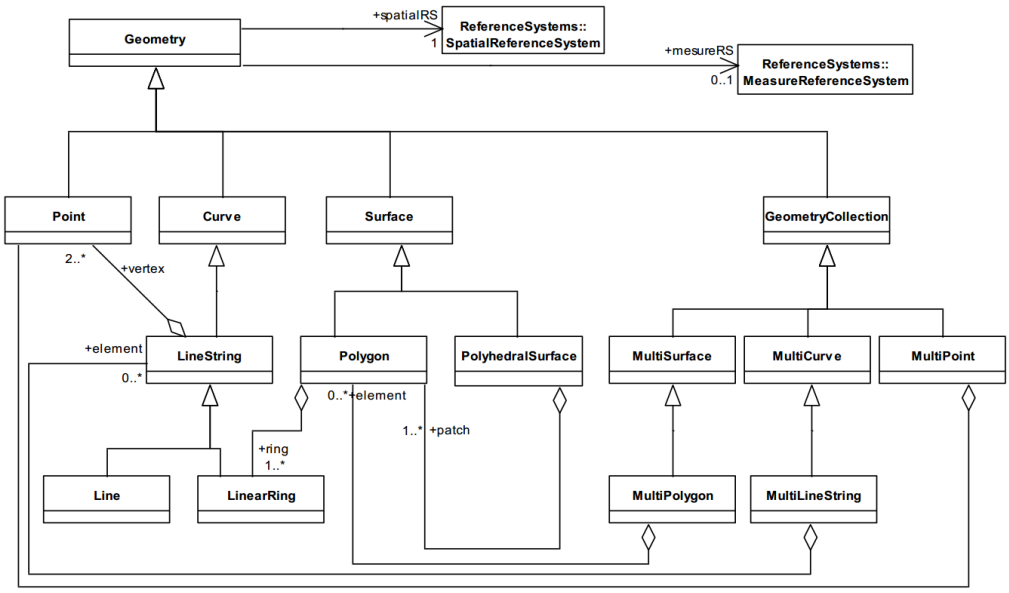


Figure 2.6: UML diagram of the OGC Simple Feature specification SQL Geometry Type hierarchy. Image from: Open Geospatial Consortium, 2010

only. All polygons have the same orientation sine the surface is continuous as Figure 2.7 illustrates.

If each LineString of all polygon patches in a PolyhedralSurface is the boundary of exactly two polygons, the surface is closed resulting in a polyhedron enclosing a solid. One can check for this condition with the function `ST_isClosed()`. *Though it might seem like it in the first place, PostGIS does not recognize a closed PolyhedralSurface as a solid.* Regarding the interaction of geometries discussed later this is important to keep in mind.

The special case of closed PolyhedralSurfaces is used in this study to represent voxel cubes. The WKT representation of a cubic PolyhedralSurface with edge length one is shown in Listing 2.1. All faces are oriented outwards the solid. The ordering of the surfaces in a PolyhedralSurface is arbitrary, however in this example it matches to the ordering specified by EMI for voxels (see Section 2.5.6). (Corti, 2014; Obe et al., 2011; PostGIS Project Steering Committee, 2015)

Intersection tests

A vital operation for this study is to test for the spatial relationship of two geometries. PostGIS implements the Dimensionally Extended Nine-Intersection Model (DE-9IM) of the 'Simple Feature Specification' (Open Geospatial Consortium, 2011, 2010) for those tests. The model is based on the pairwise comparison of the intersection of the *Interior*, *Exterior* and *Boundary* ($I(a)$, $E(a)$, $B(a)$) of two geometries a and b. The boundary of a geometry is the set of geometries describing it of the next lower dimension. For example a `POLYGONs` boundary is the `LINESTRING` that separates its interior and exterior. The interior of a geometry is what is left when the boundary is removed, while the exterior is the surface around the geometry which is not on the boundary or the interior. For each test the *maximum dimension* ($dim(x)$) of the intersection geometry is returned according to Table 2.3. Based on the resulting matrix the spatial relation of the two geometries is exactly described.

To make the work with this concept easier a set of named relationships (disjoint, touches, crosses, within, overlaps) was introduced, each described by a specific pattern matrix of the same structure as in Table 2.3. The possible pattern values are $\{T, F, *, 0, 1, 2\}$. T stands for any dimension (0, 1, 2), F for empty dimension or no intersection. $*$ represents

```
POLYHEDRALSURFACE(
    ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
    ((1 1 1, 1 0 1, 1 0 0, 1 1 0, 1 1 1)),
    ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
    ((1 1 1, 1 1 0, 0 1 0, 0 1 1, 1 1 1)),
    ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
    ((1 1 1, 0 1 1, 0 0 1, 1 0 1, 1 1 1))
)
```

Listings 2.1: WKT representation of a cubic PolyhedralSurface with outward oriented surfaces.

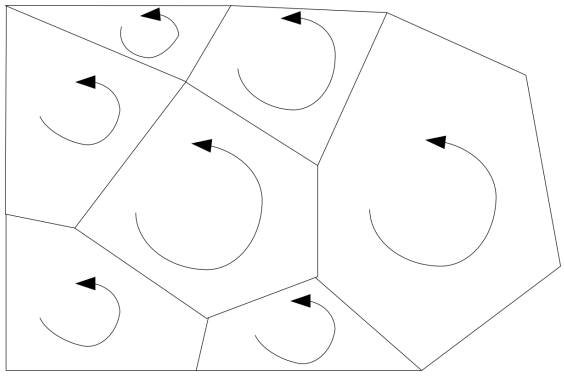


Figure 2.7: PolyhedralSurface with consistent orientation observed from the top. Image from: Open Geospatial Consortium, 2011

Table 2.3: Dimensionally Extended Nine-Intersection Model (DE-9IM). Adapted from: (Open Geospatial Consortium, 2011)

	Interior	Boundary	Exterior
Interior	$dim(I(a) \cap I(b))$	$dim(I(a) \cap B(b))$	$dim\,(I(a) \cap E(b))$
Boundary	$dim(B(a) \cap I(b))$	$dim(B(a) \cap B(b))$	$dim\,(B(a) \cap E(b))$
Exterior	$dim(E(a) \cap I(b))$	$dim(E(a) \cap B(b))$	$dim\,(E(a) \cap E(b))$

an arbitrary value. For example the pattern of the within relationship looks like this:

$$\begin{Bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{Bmatrix}$$

In PostGIS theses (and some more) spatial relations have been translated into a set of operators and functions returning boolean values according to the official documentation. (PostGIS Project Steering Committee, 2015) This makes it easy to query spatial relationships or have them as join criteria using standard SQL. Most relevant for this study is `ST_3DIntersects(geom, geom)`, which checks for 'any intersection' in 3D space. When working with these functions and operators it is important to check the input geometry objects for validity according to the Simple Feature Specification (SFS) as invalid geometries can lead to unexpected results. PostGIS offers three functions for that. `ST_isValid(geom)` returns a boolean value, `ST_isValidDetail(geom)` and `ST_isValidReason(geom)` deliver detailed information on the invalidity and where it occurs in the geometry.

Index usage

Indices make spatial queries for large data sets feasible. They organize data into search tree structures which allows accessing them without sequentially scanning all entires in a database table. PostgreSQL provides a broad range of index implementations for different

fields of application. B-Trees are commonly used for data with one dimensional sort ordering like number or letters. R-Trees divide date into rectangles (2D) or cubes (3D) and are well suited for spatial data as Figure 2.8 illustrates. According to the official documentation (PostGIS Project Steering Committee, 2015), PostGIS uses an R-Tree on top of the GiST index package. There are two separate versions for 2D and 3D that are set up for a geometry column 'col' in the table 'tab' like this:

```
CREATE INDEX [indexname] ON tab USING GIST (col);
CREATE INDEX [indexname] ON tab USING GIST (col gist_geometry_ops_2d);
```

```
CREATE INDEX [indexname] ON tab USING GIST (col gist_geometry_ops_nd);
```

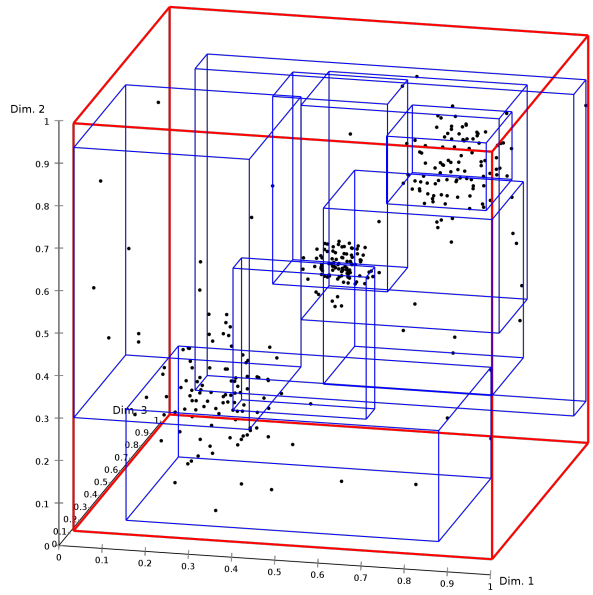


Figure 2.8: 3D R-Tree visualization with cubic edges. Image from: http://commons.wikimedia.org/wiki/File:RTree-Visualization-3D.svg

Usually, once created, indices are silently used by PostgreSQL, but regarding GiST the *QueryPlaner* is not perfectly optimized leading to sequential scans while indices are available. To allow the QueryPlaner to make better choices statistics on the amount and distribution of values have to be gathered. In current releases this can be done by running `VACUUM ANALYZE [TABLENAME] [COLUMNNAME]`. If that won't help index usage can be forced by setting `SET ENABLE_SEQSCAN = FALSE`. This is discouraged though, as it can have a bad influence on other queries. The switch should be re-enabled after use in any case.

When working with spatial indices it is important to keep in mind that only the Bounding Box (BBox)based operators benefit from them. PostGIS provides two BBox data types,

box2d and *box3d* giving the smallest rectangular box enclosing a geometry in 2D/3D space. When indices are applied, first a BBox comparison is conducted to narrow down the number of geometries involved. This is usually done implicitly by PostGIS. Listing 2.2 shows an example of how this works. The top query will actually be executed as the bottom one, because the `ST_Intersects` function involves an automatic BBox comparison, while `_ST_Intersects` is the function for a pure intersection test without BBox comparison. This is the case for most PostGIS functions, for exceptions please take a look at the official documentation. (PostGIS Project Steering Committee, 2015)

Note: Usually all PostGIS functions will implicitly use 2D BBox comparison with the && operator. To make use of 3D indices an explicit 3D BBox comparison with the &&& operator must be forced as shown in Listing 2.2.

```sql
-- Implicit BBox comparison
SELECT *
FROM
    table_a a,
    table_b b
WHERE
    ST_Intersects(a.geom, b.geom);

-- Explicit box comparison
SELECT *
FROM
    table_a a,
    table_b b
WHERE
    a.geom && b.geom AND
    _ST_Intersects(a.geom, b.geom);
```

Listings 2.2: Implicit and explicit bounding box comparison with PostGIS.

2.3 3D City Database Importer/Exporter

The 3D City Database Importer/Exporter (ImporterExporter) is the main data management tool for the 3DCityDB. Most importantly it allows the high-performance import and export of CityGML instance documents of arbitrary size by an easy-to-use GUI. Currently, two different versions of the tool exist, analogue to the 3DCityDB, one for PostgreSQL/PostGIS and for Oracle spatial. Initially the tool was developed at the Institute for Geodesy and Geoinformation Science of Technical University Berlin (TUB). Recently the project moved to TUM where its development continues at the Chair of Geoinformatics. As the other applications used in this project, the ImporterExporter is free software. It's written in Java, licensed under the LGPL and available on the official 3DCityDB homepage.[2]

Besides the graphical front-end the ImporterExporter offers a Command Line Interface (CLI) allowing its integration into batch processes or third party software. Moreover, since version 1.4.0, an Application Programming Interface (API) is supplied, allowing the modular extension of the tool, thus several plugins have emerged since its initial release. They cover tasks like data harmonization or export of objects and attributes to other formats than CityGML. Two of these tools, the *Spreeadsheet Generator* and the *KML/KOLLADA Exporter* are used in this project and will be discussed in detail in the following sections. (Kolbe, Koenig, et al., 2009; Kolbe, Yao, et al., 2013)

The open design and licensing of the ImporterExporter makes it an ideal basis for software development around CityGML. It's fully integrated with the 3DCityDB and allows easy and fast data exchange through its import/export functionalities. With Java one of the most popular and well documented programming languages of today is used. The powerful plugin API provides a rich set of ready-to-use features saving lots of implementation work. Developers can rely on mature and well tested existing software components and profit from the tools acceptance at the same time. Therefore it was an easy choice to design a plugin for present issue.

2.3.1 Extension with the Plugin API

The plugin API of the ImporterExporter is based on the *Java 6 SE Service Provider Specification*, it is seen as a *service definition* while the plugins are *service proivders*. Plugins are self contained and cannot interact with other plugins of the ImporterExporter. The extension mechanism is subdivided into four main categories. Each plugin may implement one more of these modules. (Kolbe, Yao, et al., 2013)

Following C. Nagel, 2011, a *View Extension* allows adding a GUI to a plugin. This way users are enabled to control a plugin's functionalities with well-known GUI elements. As shown on Figure 2.10 in area 1 the ImporterExporter front-end uses a tabbed interface. A view extension adds a new tab here. The *Preference Extension* gives developers the possibility to create a tree structured settings section for a plugin on the preference tab as

2 http://www.3dcitydb.net/

depicted on Figure 2.9. The outsourcing of configuration that is not frequently changed enables users to adapt plugins to their own special needs and helps to keep the main GUI well arranged. Another way of storing plugin specific settings is the *Config Extension*. It facilitates the saving of configurations in the ImporterExporter's main config XML file. All the XML processing is done by the ImporterExporter. Moreover, the file is loaded before a plugin is initialized and saved after it terminates making it ideal to store initial conditions here. The last extension module is the *Menu Extension*. With this mechanism the menu bar of the ImporterExporter (see Figure 2.10 area 1, top left) can be enhanced with a plugin specific menu storing additional controls.

According to Kolbe, Yao, et al., 2013, the API is shipped with every installation of the ImporterExporter. After setup of the main application, it can be found in the "plugin-api" sub folder. Developers need to put the *3dcitydb-impexp-plugin-api.jar* on their classpath to use it. The initialization point of a ImporterExporter plugin, the so called *Provider Class* needs to implement both the *Main Service Interface* (`de.tub.api.plugin.Plugin`) and at least one of the *Extension Interfaces* described before, for example `de.tub.citydb.api.-plugin.extension.view.View-Extension`. After that the following features can be directly accessed by developers:

- Concurrent worker pool facilitating multi-threaded programming
- Global message bus for sharing events around the main application
- Controllers allowing access of core functionality regarding database interaction, logging, views and plugin configuration
- shared GUI components
- Object registry for sharing objects between plugin instances
- Application starter for unobstructed debugging and testing during development

The *WorkerPool* is according to C. Nagel, 2011 an implementation of the popular producer-consumer design pattern for multi-threaded environments. It provides developers with a ready-to-use set of classes for generic types saving the time for an own implementation. Alternatively the Java *ThreadPools* can be used.

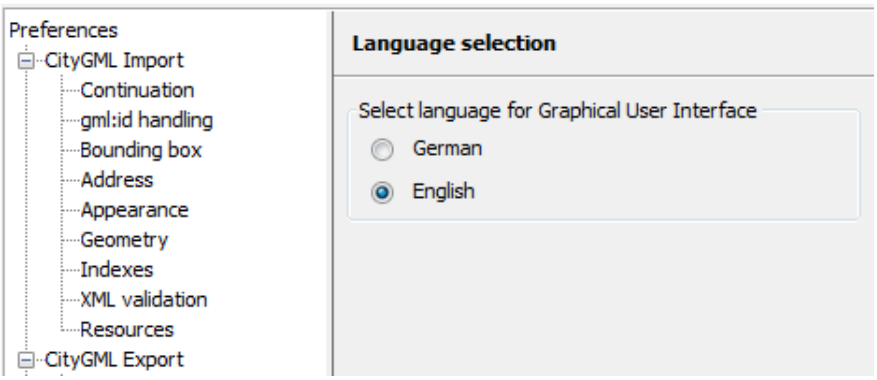


Figure 2.9: ImporterExporter preferences window.

Events are common practice in todays applications for communication between software components. They are based on the observer pattern and help to decouple software modules. The ImporterExporter realizes this concept with the *EventDispatcher* managing *Events* and *EventHandlers*. Any object can sign up to receive events or trigger them itself. The EventDispatcher can both handle synchronous and asynchronous events enabling safe communication between threads.

The *Controllers* the API provides expose core functionality of the ImporterExporter for direct use by developers. The *DatabaseController* gives access to a 3DCityDB instance by providing information and required resources like `java.sql.Connection` objects. A *ViewController* can be used to retrieve reusable GUI components, edit the status text (see Figure 2.10, area 1, bottom left) or clear the console window. User information on ongoing processes, error messages, warnings or debug notices can be shown in the console window shown in area 2 of Figure 2.10 with the *LogController*. All controller singletons are reachable through *ObjectRegistry*. Moreover this allows the registration and lookup of objects by an unique string identifier. Details on the implementation of the plugin API and the usage of its features are discussed later.

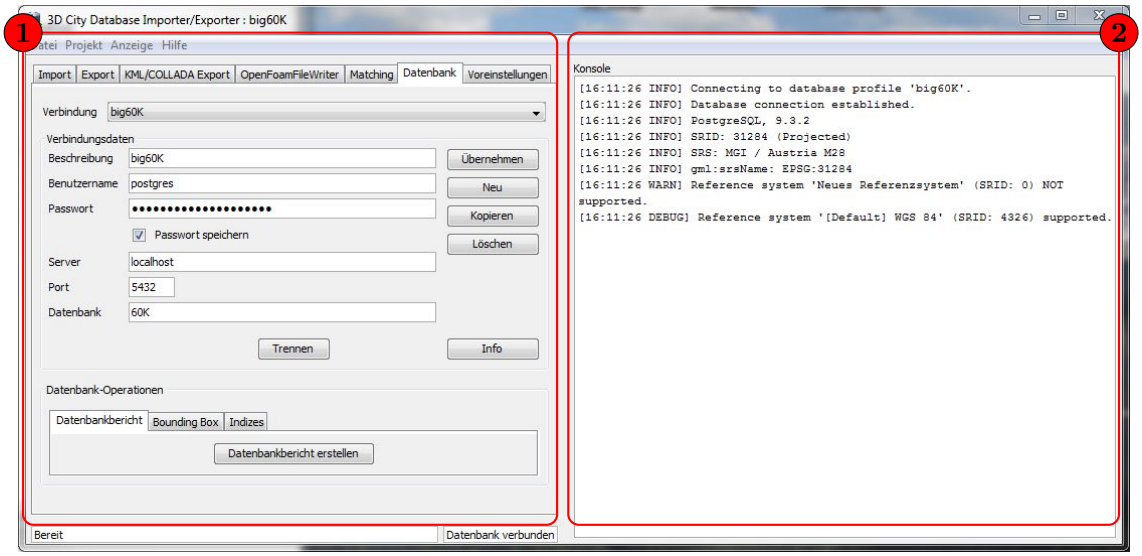


Figure 2.10: ImporterExporter GUI.

2.3.2 Spreadsheet Generator Plugin

The Spreadsheet Generator (SPSHG) plugin extends the ImporterExporter with the capability to export data from a 3DCityDB to a local CSV file or a cloud hosted spreadsheet (currently only GoogleSpreadsheet supported). The plugin was developed at TUB, is licensed under the LGPL and freely available on the official 3DCityDB homepage[1]. The relevance of the SPSHG for this project lies in the coupling approach of online spreadsheets and 3D visualization models described in Section 2.4.1.

The SPSHG comes with an intuitive GUI that easily allows the creation of columns for export and the customization of their content. Output can be filtered by BBox or by CityGML top level thematic class. Generally each row in the output table contains one city object identified by its GMLID. Columns can be named individually and contain either static values, an expression or a combination of both. For expressions all values to the corresponding city object will be returned as comma separated list, if no aggregate function (FIRST, LAST, MIN, MAX, AVG, SUM, COUNT) is used. It is possible to set up conditions for the values to be selected. For example if one wants to return the value for a generic attribute (table `CITYOBJECT_ATTRIB`) named 'Pressure', the expression would look like this:

```
CITYOBJECT\_GENERICATTRIB/REALVAL[ATTRNAME ='PRESSURE']
```

After a set of columns has been created, it can be stored as a template and reused later if needed. The output file can either be exported as a CSV file or directly to GoogleSpreadsheets by entering login credentials of a Google account in the given form. After successfully uploading a file to a cloud service, the *sharing settings* can be changed on the GUI. It's possible to grant/revoke access rights and visibility settings.

Currently only datasets for city objects of the CityGML top level features (see Section 2.1) can be exported. Therefore the export of content belonging to thematic building surfaces was not possible which causes some problem in this study. This issue and a workaround are discussed in detail in. (Naderi et al., 2012)

2.3.3 KML/KOLLADA Plugin

The KML/COLLADA plugin adds export capabilities for KML and COLLADA files for visualization of 3D city models in a broad range of applications like GoogleEarth, ArcGIS or ArcGIS Explorer. The format facilitates the interactive exploration of 3D city model with features like object highlighting or KML information balloons. KML files are used in this study for displaying the city model and the simulation results stored with the cloud-based 3D web client discussed in Section 2.4.

According to Kolbe, Yao, et al., 2013, the KML/COLLADA plugin panel offers similar selection options to the CityGML export. It is possible to select whole areas with a BBox or single objects by their GMLID. For BBox selections tiling is supported. Tiled exports are highly recommended as they offer better performance since parts of the model are loaded/unloaded according to their visibility. The number of tiling rows and columns

1 http://www.3dcitydb.net/

can either be specified manually or be determined automatically based on settings in the preferences tab. Each tile is stored in a separate file with its tile number attached in the file name. A main KML file is generated pointing to the tile files. For GoogleEarth tiles of smaller than 10 MiB are recommended for best performance.

Furthermore the LoD from which the export should be generated can be specified. Two options are available, either explicit specification of a certain level or automatic picking of the highest level available. Higher LoD exports have a big impact on viewing performance as they are more complex and cause bigger export files.

For configuration of what is shown during visualization of the export in the world viewer four options are available. The *Footprint* option produces a pure KML export showing the objects projection on the ground. *Extruded* produces a KML export with the well-known block model based on the objects footprint and measured height. The corresponding CityGML attribute from the feature class building must be filled with a valid value for this.

With the *Geometry* option a pure KML export showing the detailed geometries of ground, wall and roof surfaces is generated. Thematic differentiations are represented by coloring. The preference tab allows the customization of the rendering options. If surfaces are not thematically modeled for LoD1 or LoD2 their coloring is derived from their smallest z-coordinate. Does the surface touches the ground it is considered a wall, if not a roof. Figure 2.11 depicts the test area for this project, the TUM campus and surroundings in Munich in the given display mode in GoogleEarth. The thematic roof surfaces are colored red, walls are gray.

The last option, named *COLLADA*, generates a KML/COLLADA export. Geometries are analogue to the *Geometry* display mode but additionally textures from the CityGML appearance model are supported. The appearance model organizes display styles in themes, which are stored in the city model. They have to be fetched from the database first with the corresponding button on the GUI.

Besides the configuration described here lots of additional settings regarding rendering of thematic surfaces, KML balloons and object altitude are available of the preferences tab of the ImporterExporter. For details on this please refer to the official documentation, see Kolbe, Yao, et al., 2013.

Figure 2.11: KML representation of the TUM campus LoD2 model in GoogleEarth.

2.4 Cloud-based 3D Web Client

The Cloud-based 3D web client is the top (end user) layer of this study. It enables users with limit GIS and blast simulation knowledge to view and analyze a given blast scenario. The web client is currently under development at the Chair of Geoinformatics at TUM and has not yet been released to public. For further information please visit the official website[1].

The basic idea behind the project is to provide an interface between the complex data, methods and tools of semantic 3D city models and easy-to-use text processing and spreadsheet calculations. Users are protected from the complexity of the underlying GIS and simulation system and conduct analysis and decision making in a familiar work environment.

To achieve this a simplification of the semantic city model to the requirements of the given application both spatially and thematically is required. First, only the affected area of the model is selected. Second, unused objects and attributes are removed. Furthermore the structure of the remaining objects is simplified by summarizing their decomposed structure by means of selection and aggregation. The outcome of this process are two files, one containing a 3D visualization model (KML/COLLADA), one a spreadsheet(CSV or

1 http://www.gis.bgu.tum.de/

GoogleSpreadsheet) with the corresponding thematic attributes. Regarding this study, these files are both created and exported by the ImporterExporter as described in Section 2.3.3 and Section 2.3.2. (Yao and Kolbe, 2014; Yao, Sindram, et al., 2014)

2.4.1 Coupling of cloud-based Spreadsheets and 3D visualization

According to Naderi et al., 2012, the general concept of separate 3D visualization data and thematic information and distributing them with a cloud service offers several benefits. Data can be collaboratively edited, maintained and augmented without affecting the original source. This is especially helpful when working with data hosted in an access-restricted environment like public administrations.

The architecture of the approach is simple as Figure 2.12 illustrates. Following Yao, Sindram, et al., 2014 and Yao and Kolbe, 2014, both aforementioned files are uploaded to cloud services and published as HTTP links with the sharing functions of the specific service. The 3D web client receives links to both files allowing him to access them over the web. Visualization is conducted in the client with an *Earth Viewer*. Currently only GoogleEarth is supported, but open source solutions for WebGL based viewers like Cesium are in development. Thematic attributes of the model are loaded from the spreadsheet. In contrast to the visualization model read and write access to these files is possible. Therefore thematic data cannot only be displayed, but edited. When changes are committed they are available to all others working on this share. Consequently multiple users are able to share the work of a project. The 3D web client further supports this by storing all configurations in an online spreadsheet as well. Like this it is possible to share whole work environments including URLs to the base files, loaded additional layers, custom camera views or user comments within a single URL.

As for all collaborative web applications user rights management is required for reasons of security and data consistency. There are two user groups. *Administrators* have full access to all files. They can upload, edit or remove visualization, attribute and configuration files. Moreover they can define access rights for *end users*. This way only trusted users are enabled to edit attributes, configurations or both. Read-only users can still view the model and monitor changes made by others and even better, they can contribute editions as well. To do so they simply need to clone the file the want to modify by creating an online copy of it. After saving their edits to their own file they can share them among other.

2.4.2 GUI and capabilities

The 3D Web Client enables users based on a KML visualization model and thematic information stored in a GoogleSpreadsheet to perform analysis tasks for 3D city models on a GUI running in a standard web browser. An overview of its interface is given in Figure 2.13. This only gives a quick introduction to the client. For a practical usage example with lots of images and explanations please refer to Section 3.6 and Section 4.1.2. Following Yao and Kolbe, 2014 the client's main components are described as follows:

- **3D View:** In the center of the GUI a well-know GoogleEarth window with the classic pan, zoom and rotation functions is shown. In here the 3D city model is visualized based on the input KML layers.

- **Layer tree:** The layer tree on the top left allows managing the loaded KML files. They user can enable or disable whole layers or single tiles of a layer.
- **User Comments and Viewpoint:** With this panel the user is enabled to create, store and exchange viewpoints and comments on a currently focused map section with other users to ease communication in teams.
- **Find Location:** This view offers a search field where the user can lookup either global locations or objects of the city model by their GMLID. The map window is focused to the location after a successful query.
- **Menubar:** At the top of the window a toolbar is shown offering menus for project management, debugging, selection, navigation and drawing. The project management functions allow the exchange of whole web client setups including loaded layers, linked attribute spreadsheets, comments and viewpoints to facilitate multi-user projects.
- **Map Manager:** The map manager gives control over additional layers offered by Google like terrain, 3D buildings or roads.
- **Attribute List:** In the top right of the GUI the attribute list is located. Here the attributes of city objects stored in the spreadsheet are listed. Users with sufficient access rights may edit these properties. With Query by Example (QBE) objects can be selected based on their attribute values.
- **Object Controller:** The Object controller allows operations on selected objects. Items can be highlighted or masked in the 3D view. For object groups simple analytic task like aggregate sum, average, minimum or maximum are available.

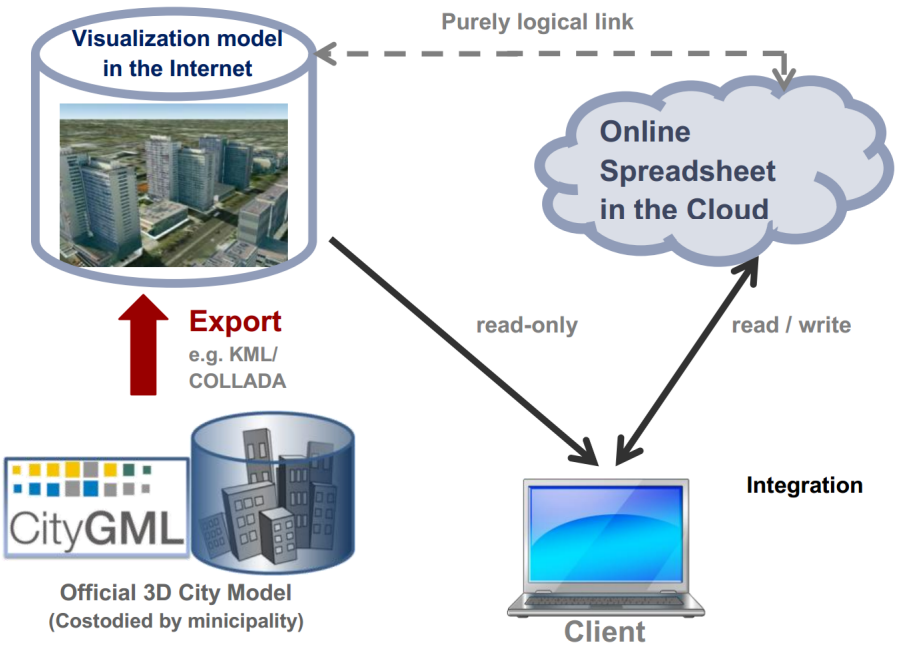


Figure 2.12: Coupling of cloud-hosted spreadsheets and 3D visualization models. Image from: Naderi et al., 2012

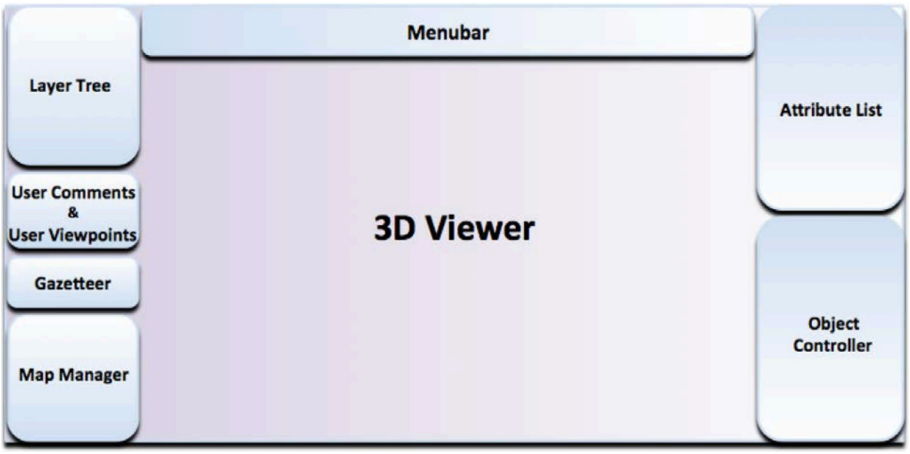


Figure 2.13: Overview of the GUI of the 3D Web Client. Image from: Yao and Kolbe, 2014

2.5 Apollo Blastsimulator

The second column this thesis is built on besides CityGML is the Apollo Blastsimulator. Its contribution to the project is the physical simulation and assessment of blast events based on input data gathered from the city model before. According to Klomfass and Zweigle, 2013 and Klomfass and Herzog, 2010, when analyzing blast effects, mostly two kinds of calculation models are available. First there are strongly simplified approaches based on TNT equivalences and scaled distances. They work very fast but have their restrictions regarding accuracy and the spectrum of permitted applications. Second there are general purpose CFD simulations delivering both accurate results and usability by a broad range of applications for the price of difficult handling and long set-up and computation times. The Apollo Blastsimulator tries to fill the gap between theses two extremes and combines good usability, versatility and computational efficiency. This is achieved by tailoring of the methodological concepts to the application on explosions and blast waves. An important feature in the present context, is the possibility to adjust computation time and accuracy by changing the resolution of the underlying computational mesh. As described in Klomfass and Zweigle, 2013, the Apollo Blastsimulator is therefore well suited for multi-stage analysis typically conducted in risk assessment:

1. Analysis of a large number of scenarios with simple engineering tools and identification of potential critical scenarios
2. Analysis of a reduced number of scenarios with CFD tool using moderate accuracy / moderate computing time settings and confirmation of critical scenarios
3. Hi-fidelity analysis of most critical scenarios with the CFD tool using high accuracy settings

This kind of assessment seems especially appealing in the GIS context of this thesis. The evermore growing amount of spatial data available, allows the selection and processing of large numbers of scenarios as described in step one with low personal effort and high automation. For example, a list with potential explosion locations could be run against a spatial database identifying critical scenarios for further processing.

2.5.1 Calculation models: Engineering method and CFD simulation

The Apollo Blastsimulator is able to operate with two different calculation models. The *free field model* is a so called *engineering method*. It evaluates very quickly and delivers an acceptable degree of accuracy for specific applications. The method is based on the popular concept of *TNT equivalences* which states that for spherical or hemispherical blast waves all blast parameters depend on a *scaled distance*. The built environment is ignored in this approach, therefore no voxel model (see Section 2.5.4) needs to be generated. (Klomfass and Thoma, 2012)

The second calculation model is a CFD simulation based on *finite volumes*. Computational effort for this approach is much higher than for the engineering method, but the

results delivered are much more accurate. The shape of the built environment is taken into account allowing the evaluation of focusing and shadowing effects. The model is explained in detail in the following sections.

2.5.2 Physical model and optimizations

The basic equations for describing blast wave propagation are the conservation laws for transient flow of compressible fluids. For the detonation phase and a subsequent combustion phase chemical reactions have to be included. They are modeled by a time and space dependent chemical energy source. The influence of other physical parameters like viscosity, heat conduction and diffusion are not considered in the physical model since they are negligibly small on the typical time and length scale. The resulting equations are solved with a 3D second order accurate finite volume scheme with explicit time integration. To obtain high efficiency, usability, accuracy and versatility the simulation approach was optimized in several ways.

The physical model was simplified by implementing optimized variations of the solver for the initial detonation phase and the subsequent blast wave propagation. During the propagation phase a common equation of state is used for air and gaseous detonation products while during the initial detonation phase distinct equations for air and detonation products are applied.

Furthermore the Apollo Blastsimulator is capable of automatically setting up the computation based on scenario informations. The scenario objects are mapped on an regular Cartesian grid and can be partially removed allowing object failure during the simulation as well as openings. (Klomfass and Herzog, 2010; Klomfass and Zweigle, 2013)

2.5.3 Dynamic mesh adaption

However, the key feature regarding performance is the dynamic mesh adaption. Since the computational effort is directly proportional to the number of grid cells used in the finite volume approach both memory usage and execution time can heavily be reduced through *Automatic Mesh Refinement (AMR)*. The idea behind this is simply to use an non-uniform computational grid. The general concept can be observed in Figure 2.14. The whole computational domain is first separated into zones of the same size. Each of these zones can be subdivided during the simulation process up to a predefined maximum level for each time step independently from other zones. Table 2.4 gives an overview on the available levels.

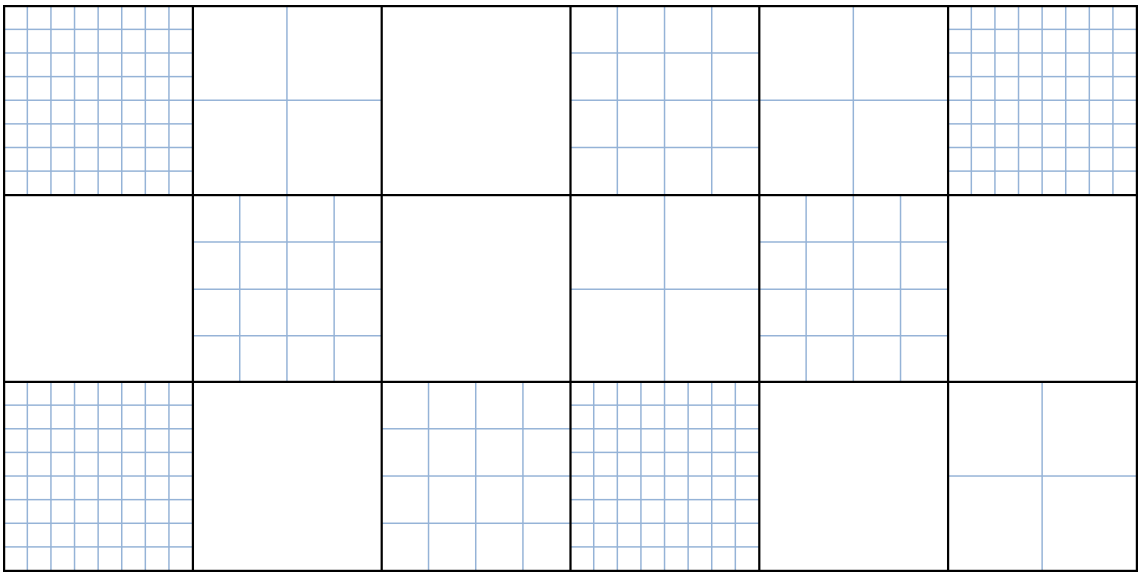


Figure 2.14: Illustration of mesh refinement in 2D for level 1-4, see Table 2.4.

Table 2.4: Definition of mesh levels. Resolutions are subdivisions by power of 2. Adapted from: Klomfass and Zweigle, 2013

Level ($MaxLevel$)	Cells per edge	Cells per zone
1	1	1
2	2	8
3	4	64
4	8	512
5	16	4096
6	32	32 768
7	64	262 144

During calculation the grid is refined depending on the current wave and flow processes. For each computational domain, according to the aforementioned physical model, different re-meshing strategies have been developed to guarantee good performance. An example is given on Figure 2.15. From left to right three time steps of a 50 kg TNT explosion in front of six buildings (blue) are illustrated. The zones can be observed as a regular grid of thin lines. The mesh refinement happens in the cells that appear darker. It follows the blast wave from the initial detonation on its way towards the buildings and between them. The image on the right the reflection of the blast wave from the building front can be witnessed. (Klomfass and Herzog, 2010; Klomfass and Zweigle, 2013)

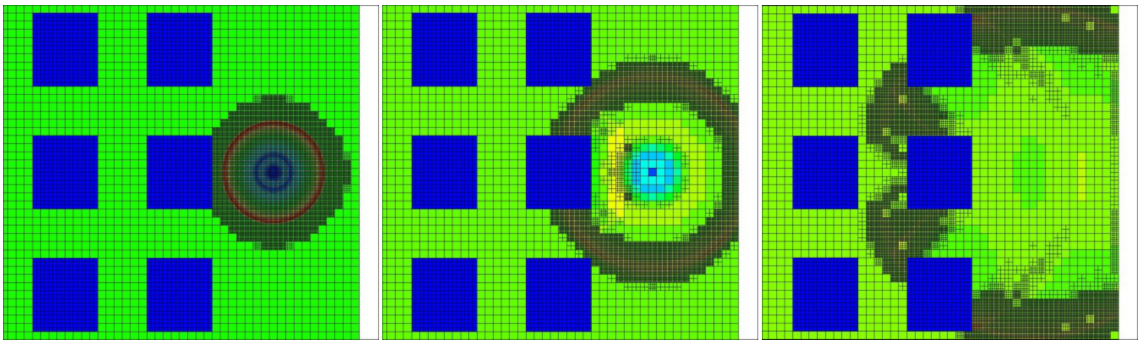


Figure 2.15: Automatic Mesh Refinement during blast wave propagation for three time steps from left to right. Image from: Klomfass and Herzog, 2010

2.5.4 Voxel model specifications

The Apollo Blastsimulator works with a finite volume model. The following section explains the underlying data structure. All information is taken from a specification sheet from EMI and summarized here. For more detailed information and several examples please take a look at Appendix B where the full sheet can be viewed. Theses slides were meant for informational purpose only and are by no means an official documentation.

As mentioned before, the whole computational domain is subdivided into small volumes of the same size. They are also called *Volume Pixels (Voxels)*. All spatial objects in the domain are represented by an aggregation of these Volume Pixels (Voxels). The quality of the approximation of the original objects depends on the size of the Voxels and its orientation relative to the grid. Figure 2.16 illustrates the campus of the TUM in a Voxel representation with an Voxels edge length (L_{voxel}) of about 1.5 m. To increase the quality of the approximation the buildings were rotated to align theirs walls with the gird. Non parallel wall produces ripples as observed on the round building on the left. Especially with coarse grid resolution these bumps can have a negative influence on the simulation quality.

Before a simulation can be executed, a Voxel model has to be generated for the actual scenario. The location of the explosion (P_{ex}) was selected as the *point of origin* (x_0, y_0, z_0) of the computational domain as shown in Figure 2.17 for this project. The real world coordinates of this point are stored in the scenario definition. First the *maximal hazard radius* (R_{max}) according to the *mass of TNT* (M_{TNT}) provided by the user in the scenario definition has to be identified. This relation is derived from the engineering method. It gives the radius in a free field environment, where the overpressure amplitude has decayed to x bar[1], a magnitude that is considered a lower threshold for damaging effects.

$$R_{max} = M_{TNT}{}^{\frac{1}{3}} \cdot 40 \qquad [m] \tag{2.1}$$

Second, the BBox of the calculation domain has to be determined with the resulting R_{max}.

1 Figure not released here.

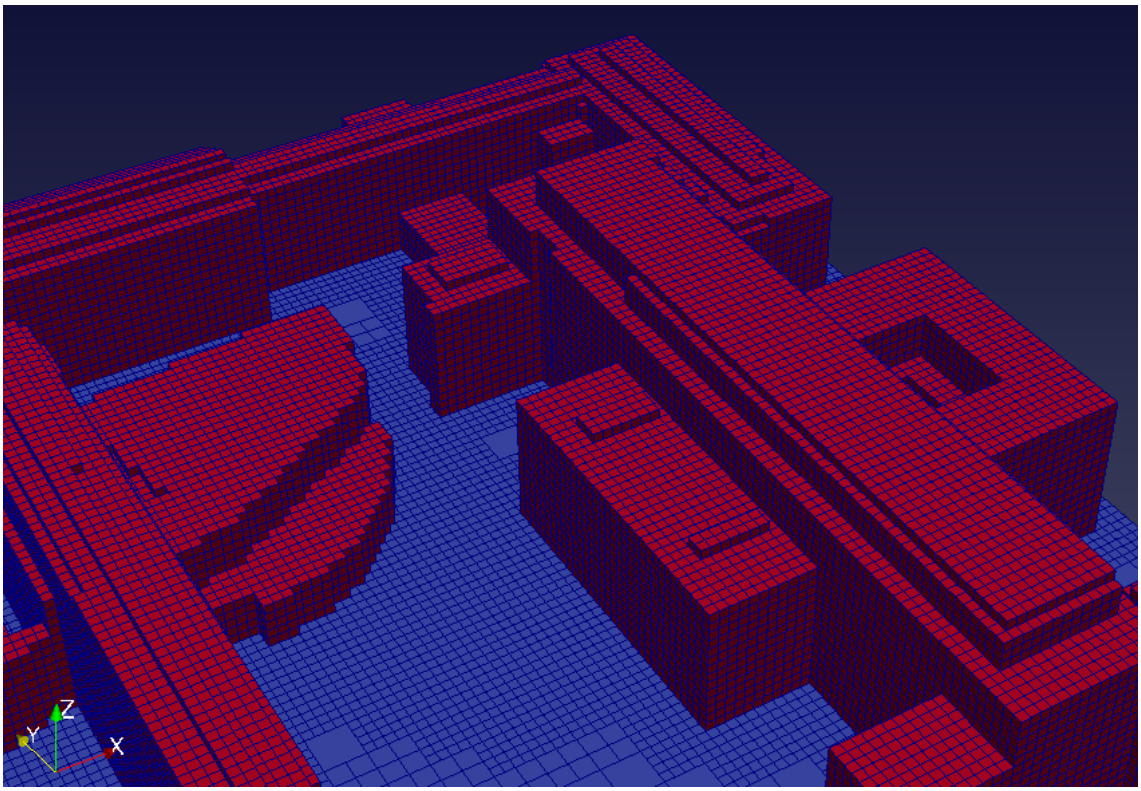


Figure 2.16: Voxel representation of the TUM campus, $L_{voxel} = 1.625\,\mathrm{m}$.

z_{min} is equal to the lowest point of terrain ($z_{min,\,terrain}$) in the BBox.

$$\begin{aligned}
x_{min} &= x_0 - R_{max} \quad [m] \\
y_{min} &= y_0 - R_{max} \quad [m] \\
z_{min} &= z_{min,\,terrain} \quad [m] \\
x_{max} &= x_0 + R_{max} \quad [m] \\
y_{max} &= x_0 + R_{max} \quad [m] \\
z_{max} &= x_0 + R_{max} \quad [m]
\end{aligned} \tag{2.2}$$

The BBox is separated into *zones* of the same volume with an edge length called $RefLength$ or *zone length* (L_{zone}). To allow efficient computation, the number of zones in the calculation domain should be between 10 000 and 100 000 (recommended in the Apollo Blastsimulator manual). This is achieved by determining the *edge length of a zone* (L_{zone}) based on the

volume of the BBox (V_{bbox}) with the following equations:

$$V_{bbox} = (x_{max} - x_{min}) \cdot (y_{max} - y_{min}) \cdot (z_{max} - z_{min}) \qquad [m^3] \tag{2.3}$$

$$L_{zone} = \left[\left(\frac{V_{bbox}}{100\,000} \right)^{\frac{1}{3}}, \left(\frac{V_{bbox}}{10\,000} \right)^{\frac{1}{3}} \right] \qquad [m] \tag{2.4}$$

An exact value for L_{zone} can be determined either by selecting or averaging in the interval given in Equation (2.4). Integral numbers do not influence the performance of the Apollo Blastsimulator, but can be helpful for testing and evaluation.

For the calculation of the *number of Voxels per zone edge* (n_{voxel}) and the *edge length of a Voxel* (L_{voxel}) the maximum mesh refinement level ($MaxLevel$) from the scenario definition is required. With a giving level from Table 2.4 the calculation is performed like this:

$$n_{voxel} = 2^{(MaxLevel-1)} \qquad [-] \tag{2.5}$$

$$L_{voxel} = \frac{L_{zone}}{n_{voxel}} \qquad [m] \tag{2.6}$$

As described before, the user can select the accuracy and the computational effort by the $MaxLevel$. Figure 2.17 illustrates an example with $MaxLevel = 3$ resulting in four Voxels per zone edge and 16 Voxels per zone in total. Both *zones* and *voxels* are organized in a zero based IJK coordinate system with its origin on the explosion location.

At the outer shell of the computational domain boundary conditions are applied. Each outer surface of the BBox ($x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$) gets its individual condition. There are two options, either a closed (wall = 1) or open surface (air = 3). Alternatively to using a layer of voxels as terrain surface, a wall boundary condition can be assigned to the lower side of the domain in case the terrain is flat. It is important that the voxel model's terrain surface is impermeable, so fluid flow (the blast wave) cannot leak out of the computational domain. This can be guaranteed by a continuous layer of voxels, that is only intercepted by a wall boundary condition. Buildings or other structure in the model need to have a closed outer shell of voxels as well, unless openings are modeled intentionally to measure blast effects inside a room for example. An illustrated example on boundary conditions and terrain closure is given in Appendix B.

2.5.5 Input data file description

One of the design goals of the Apollo Blastsimulator was easy usability for a wide range of applications. To achieve this, a simple interface is needed for inbound or outbound communication with other tools. The Apollo Blastsimulator uses well known American Standard Code for Information Interchange (ASCII) files for that purpose. Four text files are currently used for input, output and status information. The files are only structured by line breaks. The input file for a simulation run consists of two parts. The first section, the file header, contains general information on the provided voxel model. Following list summarizes the most important entries:

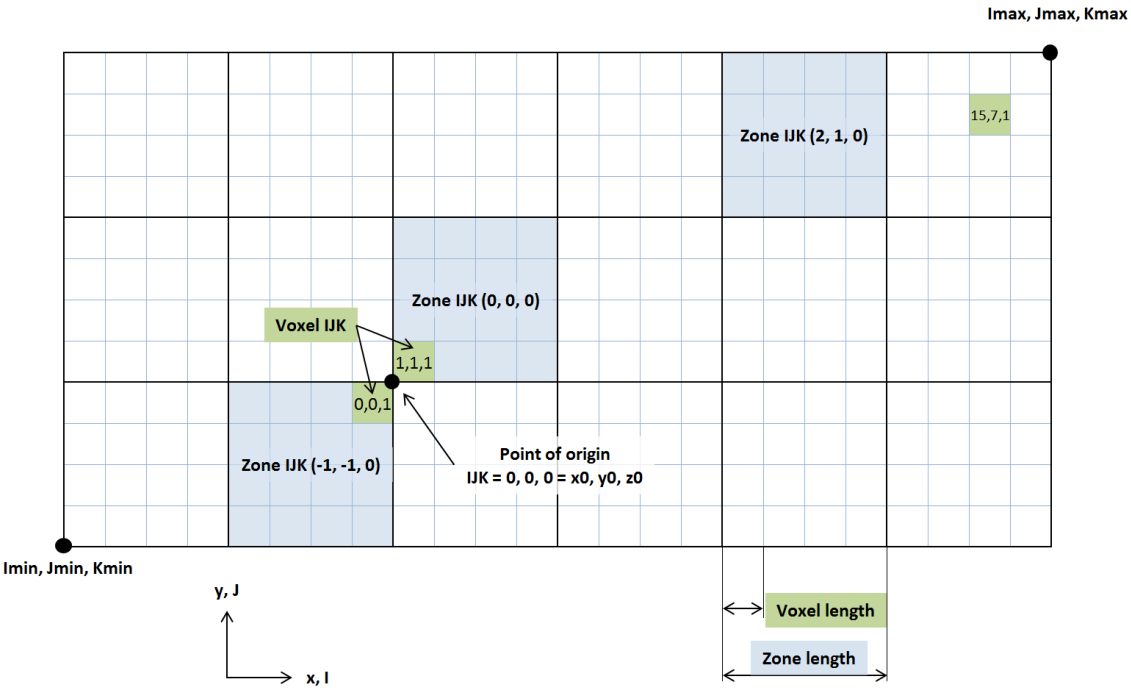


Figure 2.17: Apollo Blastsimulator voxel model coordinate references illustrated for $MaxLevel = 3$, $K_{zone} = 0$, $K_{voxel} = 1$.

- Version number
- MaxLevel
- RefLength (L_{zone})
- Coordinates of the reference point (x_0, y_0, z_0)
- Number of zones
- Number of zone surfaces on the boundary of the simulation area
- Boundary conditions of the calculation domain
- List of object names for aggregation of results

After the file header all voxels that represent a solid structure (e. g. a building wall or a streets) are listed one entry each line with their IJK coordinates (IJK_{voxel}) and an id allowing the back referencing of the voxel to an item of the object list in the header. An ordering of the voxel list is not required but a clustering by zone can be beneficial regarding computation time. A detailed example of the input file can be found in Appendix B. The status file gives information on the current state of the tool during a simulation run. Currently only data on if the current process is active or finished is provided. More detailed information is listed in the log file. The content of the result file is discussed in Section 2.5.6.

2.5.6 Damage model and simulation results

The purpose of the Apollo Blastsimulator in the context of this thesis is risk assessment for blast events in an urban scenario. Therefore, the results delivered by the tool not only contain physical quantities, but *hazard classes* for people and structures as well. These classes are computed based on the two central physical quantities of the blast simulation, the peak overpressure load and the max. overpressure impulse on an object. The underlying damage model is based on so called 'what-if' analysis. For each surface with fluid contact in the model it is determined what would happen if its was for example a concrete or masonry wall or a glass window. In this thesis the following damage categories were analyzed:

- normal, hardened or laminated window glass
- 30 cm masonry or concrete wall
- heavy building damage
- eardrum damage
- lethality threshold
- 50 % lethality

For each category critical values of the physical parameters the simulator calculates are stored, for example the critical max over pressure (P_{crit}) for window glass. These values have been determined experimentally by EMI and could well be abused, so they are not listed here. By comparison of calculated and stored critical values in an assessment tolerance model, one of the hazard classes shown in Table 2.5 is returned for each category. As an example the simplified assessment tolerance model used with the *engineering method* mentioned in Section 2.5.1 is given below. It only uses the max overpressure (P_{load}) and a tolerance value (tol) for evaluation:

$$\begin{aligned} P_{load} < tol \cdot P_{crit} \quad &\Rightarrow \quad dmg = 0 \\ tol \cdot P_{crit} \leq P_{load} \leq (1 + tol) \cdot P_{crit} \quad &\Rightarrow \quad dmg = 0.5 \\ P_{load} > (1 + tol) \cdot P_{crit} \quad &\Rightarrow \quad dmg = 1.0 \end{aligned} \tag{2.7}$$

Past a successful simulation run two files are returned by the Apollo Blastsimulator containing the results, one ASCII formated file and a VTK file.

Table 2.5: Apollo Blastsimulator hazard classes for buildings and persons.

Buildings	Persons	Damage value
no hazard	no lethal hazard	0
slight damage	slight lethal hazard	0.5
Destruction/danger of collapse	lethal hazard	1

Past a successful simulation run a ASCII file containing the damage values recorded on the building surfaces is returned by the Apollo Blastsimulator. This file is structured into three parts. The file header contains information on the processed scenario and the hazard classes as described above:

- Number of hazard classes
- List of names of the hazard classes
- Zone length (L_{zone}), same as on input file
- Coordinates of the explosion source in the domain (P_{ex}) same as on the input file (see Section 2.5.5)

After the file header the list of simulation results follows. For each voxel face with fluid contact one set of results is generated consisting of the *peak overpressure*, *overpressure impulse* and values retrieved from the damage model for all hazard classes listed in the file header. These sets are identified by their voxels IJK coordinates (IJK_{voxel}) and reference ID according to the input file (see Section 2.5.5). Furthermore an ID for the surface of the voxel is given as illustrated in Figure 3.13. For example the face on the x_{min} side of the voxel receives $ID_{face} = 1$.

The last section of the result file lists all voxel faces on the outer shell of the calculation domain having a wall boundary condition. Like this results from areas where the bottom of the model is not a layer of voxels but the border of the domain can be processed. Patches of n times n voxels with the same results can be summarized to one data set here. A detailed description of the output file can be found in Appendix B, an example file can be viewed in Appendix F.

CHAPTER 3

Coupling of CityGML city models and the Apollo Blastsimulator

The following chapter explains the coupling approach for CityGML city models and the Apollo Blastsimulator developed in this study. After giving an overview on the system, conceptual and implementation details are discussed.

3.1 System architecture and workflow

As stated in Section 1.2 one of the central research questions of this study is to find a concept, that enables non expert users to perform blast simulations based on CityGML city models and analyze their results. This section tries to answer that question by introducing the CityGML tool chain and showing how it is used to solve the central challenges of this project as they are: *Simulation*, *User Interaction*, *Data storage and processing* and *Visualization and Analysis*. The main system components are briefly introduced and their interaction is explained following a typical workflow.

3.1.1 Main system components

As mentioned above, the system can be divided into four main jobs as illustrated in Figure 3.1. Top left is the *simulation* area. In this quarter the Apollo Blastsimulator resides. It performs blast simulations based on simple ASCII input files and delivers results the same way.

In the top right the *user interface* is shown. Despite analysis tasks, every user interaction is collected by the ImporterExporter and its plugins. The Apollo Blast Plugin, which was developed in this thesis, allows scenario definition and simulation control, the SpreadSheet Generator and KML/COLLADA plugin are used for generating input files for the visualization back end.

The lower right corner represents the topic of *data storage*. Generally there are two approaches used. First there is a spatially extended relation DBMS, the 3DCityDB. Second, there is file based data storage. The Apollo Blast Plugin stores scenario definitions in two XML files on the local disk, the output files of the SpreadSheet Generator and KML/COLLADA plugins are saved online in a cloud service. Data processing is conducted in both the Apollo Blast Plugin and the database.

The bottom left shows components responsible for *visualization and analysis*. Here the 3D Web Client, which is stored in the cloud as well, is located. It is the user front-end

allowing to view and evaluate the blast effects of a given scenario in a common web browser.

All of the tools involved, despite the 3D Web Client, which has not yet been officially released, are open source and freely available on the internet following the projects guidelines. For more information on the single tools check Chapter 2.

3.1.2 A typical workflow

The typical workflow described here assumes that the user has installed all required software and can access a CityGML city model stored in a 3DCityDB covering the location of the blast event he wants to look into. In support of the following explanations take a look at Figure 3.1 giving an overview on the system architecture. Data streams are shown as arrows connecting the single components.

1. **Scenario definition**
   First step is to define a blast scenario in the front-end of the Apollo Blast Plugin. The users specifies the explosive device, an explosion location and several parameters influencing the computational demand and degree of detail of the scenario. After all settings are configured as desired the process is started.
2. **Data selection**
   Based on the scenario definition the Apollo Blast Plugin computes the area of interest and creates a copy of the city model for the required extent in the 3DCityDB.
3. **Voxel model generation**
   From the extracted part of the city model the voxel model is derived. Details are applied according to the scenario definition. The complete model is save to an ASCII text file as input for the next step.
4. **Blast simulation**
   The generated input file is handed to the Apollo Blastsimulator which performs the physical simulation. During the run status information is processed. After completion the results are written to an output text file.
5. **Result processing**
   The result file is read by the Apollo Blast Plugin and prepared for storage in the city model.
6. **Result storage in the city model**
   The prepared simulation results are stored in the city model using 3DCityDB's. The affected objects are updated in the 3DCityDB.
7. **Export of visualization files**
   Area of interest of the enhanced city model is exported in KML/COLLADA for visualization and as a spreadsheet for thematic information.
8. **Data distribution via cloud services**
   Both files are uploaded to a suitable cloud service and are published to end users via HTTP with user specific access rights.
9. **Visulization and analysis**
   Analysts use the 3D Web Client to visualize the simulation results and evaluate

the effects of the blast event. Optionally assessment results are saved back to a spreadsheet.

10. **Reintegration of assessment results**
    Optionally assessment results are re-imported from the spreadsheet to the city model for permanent storage, distribution or further processing.

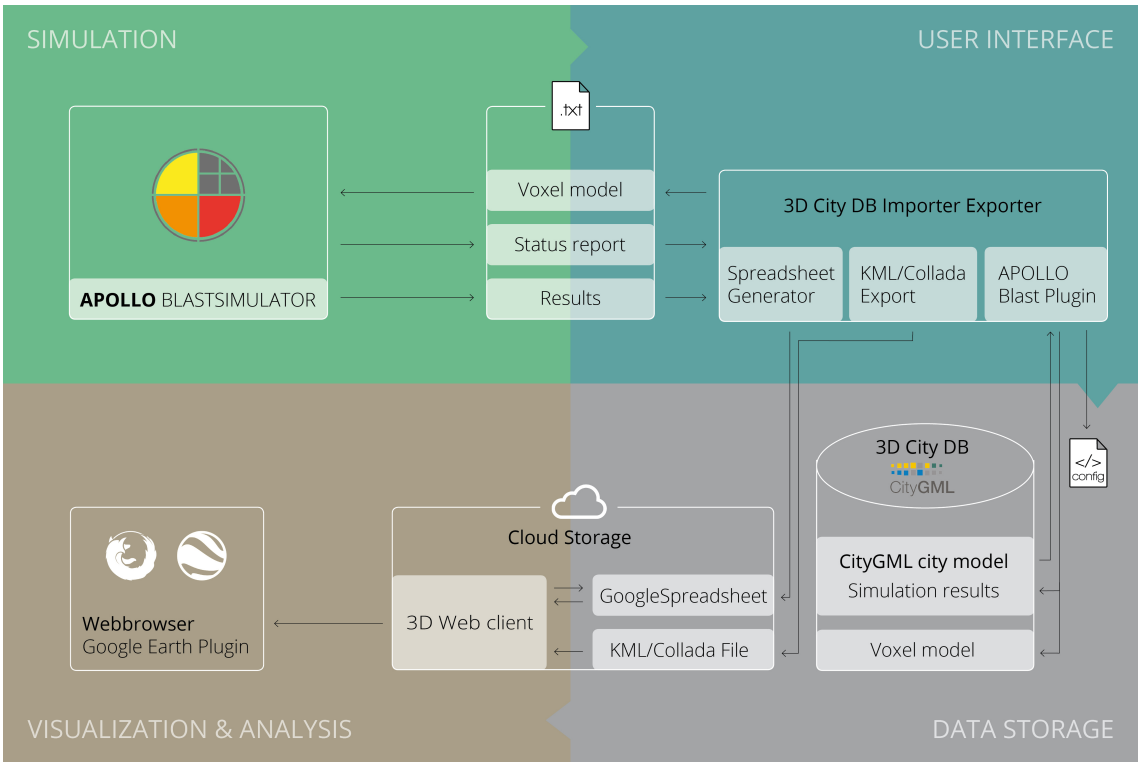


Figure 3.1: System architecture and workflow.

3.1.3 Relational database model

In this application the database instance the 3DCityDB operates on is exploited in several ways. Mainly it facilitates access to the CityGML city model but it is also used for most of the spatial data processing of the plugin. The relational database schema described here provides the table structure for theses operations.

To avoid interference with the 3DCityDB a new schema named `apolloblast` has been introduced storing all tables, sequences and indices. Figure 3.2 gives an overview on the tables and their relations. *Primary keys* are represented by a yellow key symbol, *foreign keys* are indicated by a red arrow. Cardinalities are shown at the end of the relation lines.

Table `ccgeom` is in charge for storing CityGML geometries. In this table a copy of the relevant part of the city model is created, so no harm can be done to the original data. Besides information from the CityGML geometry table (`surface_geometry`) the fields of the corresponding city objects from table `cityobject` are replicated here to allow simple aggregations per top level object. Currently the Apollo Blast Plugin works with LoD2

thematic surfaces. Of course any other CityGML layer could be added here as well. Each entry in `cggeom` is uniquely identified by an integer primary key(`uid`) which is retrieved from the sequence `cggeom_uid`.

The `point` table stores a dump of all points of the geometries from `cggeom`. This is needed for the intersection of CityGML spatial objects and voxel as explained in Section 3.3.2, same applies for the table `intersection_temp`. The relation $(1 : n)$ to `cggeom` is maintained with a foreign key (`uid_cggeom`) referencing the `uid` of `cggeom`.

Voxels are held in the table `voxel`. They are represented by their IJK coordinates (primary key), a `uid` from the sequence `voxel_uid` and their geometry of type `PolyhedralSurfaceZ`. For each voxel $(0 : 6)$ result sets can be stored in the table `results`, one for each surface. Results returned from the Apollo Blastsimulator are described by their IJK coordinates, `face_id` and an array of double values as described in Section 2.5.6. The relation between results and voxels is kept by a foreign key referencing the IJK coordinates. The relation of voxels to CityGML geometries is of cardinality $(n : n)$ as each geometry can be represented by multiple voxels and each voxel may be part of more than one geometry. It is realized with the relational table `intersections` (not shown in Figure 3.2) by the `uids` of both voxels and geometries.

Simulation results computed by the engineering method are placed in the table `engmeth`. For each `ccgeom` zero or one $(0 : 1)$ sets of results may exists. They are linked to CityGML geometries by the corresponding `uid` of a `cggeom`. For each damage category an own column was introduced to simplify aggregations. Following Section 2.5.1 the damage values are calculated depending on the distance to the explosion source (`exsource`). Therefore the columns `dmin`, `dmax` and `davg` contain the minimum, maximum and average distances.

As described in Section 2.2.4 the usage of index structures is vital for reasonable performance especially with spatial queries. Hence GiST indices for 2D and 3D are created for all geometry columns in the schema. Furthermore B-Tree indices are set up for all foreign key references to facilitate fast join operations. Primary keys are natively indexed by PostgreSQL. The whole schema containing all sequences, tables, indices and constraints is stored in a single SQL script and is installed on the database when the plugin requires it. To erase all data the plugin generates, simply the `apolloblast` schema is deleted. If informations (generic attributes) have been written to the CityGML instance on the 3DCityDB they are not affected by the removal of the schema. They can be erased by removing the corresponding attributes from the table `cityobject_genericattrib` by name. Obviously sufficient user rights for table creation and deletion are required.

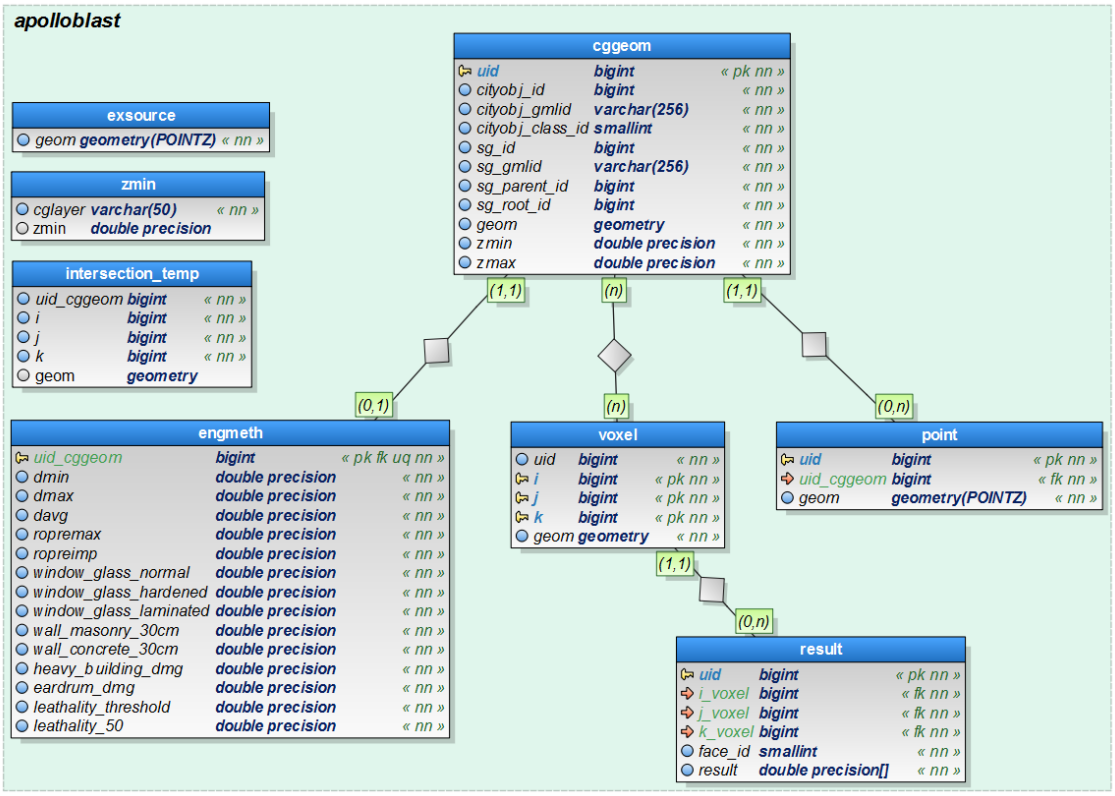


Figure 3.2: Relational database schema of the Apollo Blast Plugin.

3.2 Apollo Blast 3D City DB Importer/Exporter plugin

In this section the design and prototypic implementation of *Apollo Blast Plugin* for the ImporterExporter will be discussed. First some important concepts and how they are implemented will be discussed. After that, we take a look at structure of the entire application and how these concepts work together. The explanations will roughly follow the work flow described before.

3.2.1 Overview and package structure

The Apollo Blast Plugin is structured into packages following a thematic grouping of the classes, the root containing the whole project is `de.tum.citydb.plugins.apolloblast`. Below that packages are ordered alphabetically as the overview diagram on Figure 3.3 shows. The `concurrency` page contains all classes for parallelism. The WorkerPool implementation for parallel database query execution is located here. General management classes for database interaction are found in the `db` package, its child package `sql` stores all SQL queries used in the project. The `engineeringmethod` package holds the classes for the simplified calculation approach discussed in Section 3.4.1. All events used in the Apollo Blast Plugin and the implementation of the Global Message Bus of the ImporterExporter are included in the `event` package. The data structures for the scenario definition are located in the packages `exdev` for the explosive devices and `simpar` for the other simulation parameters. The `plugin` package contains the initialization point of the plugin. Here the class implementing the Main Service Interface of the plugin API located.Classes for IO operations are stored in the package `io` and its sub packages `filewriter` and `filereader`. The `gui` package contains the View Extension of the plugin and all other classes related to the user interface. The last two packages are `voxelgird` and `util`. They hold the data structure for the voxel model and utility classes used all over the application for error handling, logging and formating of numbers and strings.

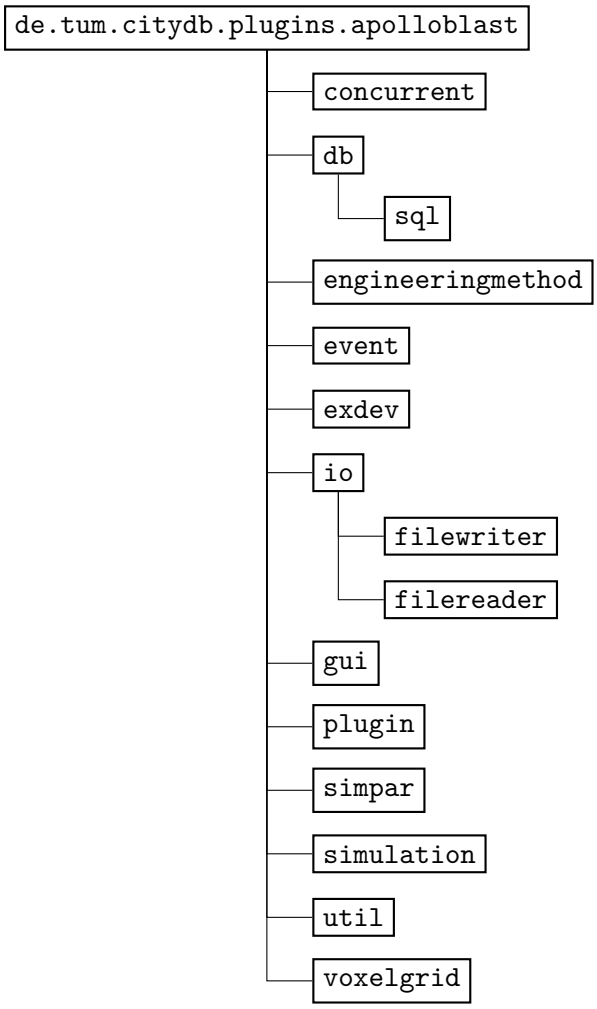


Figure 3.3: Package structure of the Apollo Blast Plugin.

3.2.2 Plugin API: Usage and implementation

As described in Section 2.3.1 an ImporterExporter plugin needs to implement the *Main Service Interface* and at least one of the *Extension Interfaces*. Currently Apollo Blast Plugin only makes use of a *View Extension* to provide a GUI. The required interfaces and classes from the plugin API JAR are:

- *Main Service interface*: `de.tub.citydb.api.plugin.Plugin`
- *View Extension interface*: `de.tub.citydb.api.plugin.extension.view.ViewExtension`
- *View class*: `de.tub.citydb.api.plugin.extension.view.View`

The initialization point of the plugin, the so called *provider class*, both implements the `Plugin` and `ViewExtension` interface. They prescribe methods for initializing the plugin, changing its locale, shutting it down and returning its view instance. This is done in the class `PluginImpl` in the `plugin` package. The View Extension required the implementation of a `View`, which is the initialization point for the GUI panels and prescribes methods for tooltips, icons, returning the panel and setting the tab title of the plugin (currently 'APOLLO Blast'). This is implemented in the class `PluginView` in the `gui` package. The API implementation is rather simple and not further explained here, for details take a look at the classes mentioned above in the source code delivered with this paper.

API Controllers and Event Dispatcher

Referring to Section 2.3.1 the *API Controllers* allow direct access to ImporterExporter functionality. The Apollo Blast Plugin makes mainly use of the `DataBaseController` and `LogController`. The `DatabaseController` is used to retrieve database connections, the `LogController` is used to provide user information on the ImporterExporter log panel. Both controller instances are provided by the `ObjectRegistry`. Listing 3.1 shows a minimal example of how the `ObjectRegistry` is accessed and how getting a `Connection` and writing a log message works.

Moreover the usage of the *Global Message Bus* of the ImporterExporter is briefly illustrated in the listing starting from line 10. The `EventDispatcher` singleton is distributed through `ObjectRegistry` just like the controllers. Classes that want to receive events need to implement the `EventHandler` interface and apply the `handleEvent(Event event)` method it dictates (line 16). This function is called every time an event, the class has signed up for (line 11), is triggered some place else in the program (line 20).

```java
// get ObjectRegistry instance
ObjectRegistry or = ObjectRegistry.getInstance();

// Retrieve a databse connection
Connection con = or.getDatabaseController().getConnection();

// Print a example logmessage with LogLevel.INFO
or.getLogController().log( LogLevel.INFO, "Hello world" );

// Sign up for an Event
or.getEventDispatcher().addEventHandler( EventType.STATECHANCE_SIMPAR, this );

// handle Event
@Override
public void handleEvent( Event event ) throws Exception {
  // here goes what happens after the event
}

// Trigger Event
or.getEventDispatcher().triggerEvent( new SimParStateChangeEvent( this ) );
```

Listings 3.1: Code snippets to plugin API controllers and event dispatcher.

3.2.3 Specification of scenario informations

Before a simulation can be performed with the Apollo Blast Plugin a blast scenario needs to be defined. First, all required parameters have to be specified by the user, second the simulation run has to be calibrated to fit both the desired level of detail and the calculation time available. Both steps are carried out on the GUI of the Apollo Blast Plugin. A scenario definition consists of the *explosion location*, an *explosive device*, *calculation options* and *additional informations* on the event it describes. The front-end in charge of theses four tasks is going to be explained here.

Explosive Device

As mentioned in Section 2.5 the explosive devices used in blast simulations performed my the Apollo Blastsimulator are mainly described by *TNT mass* (M_{TNT}). Besides that, a *name* and a *device type* can be specified on the panel as Figure 3.4 shows. Currently three types are available. *WW2 Explosive Device* describes UXOs from world war two, *Improvised Explosive Device* stands for devices manufactured by amateurs like terrorists, *Other Explosive Devices* can be used for everything not fitting the two other categories. Currently the device type only serves informational purpose and helps to organize list of explosive. It does not influence the calculation in any way, though in future releases this field might be used to deliver information on the bombs shape. On right of the panel a 'Save' and a 'Delete' button can be seen. They allow to permanently store or removes devices. The dropdown menu at the bottom lists all currently saved devices and enables their reuse.

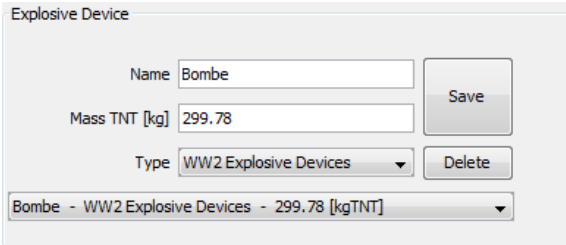


Fig. 3.4: Apollo Blast Plugin Explosive Devices dialog.

Explosive location

The selection of the explosion location is done on the panel depicted in Figure 3.5. The location is described by its *x,y,z coordinates* and the referring *SRS* described by its SRID. All values can be entered manually or determined interactively on a map window by pressing the 'Select Location' button to the right. Though, due to time constraints during the project the interactive selection is not implemented yet. A map window similar to the BBox selection panel of the ImporterExporter is planned here. When coordinates are entered manually, they have to be transformed to the database SRS first, as not automatic transformation is implemented yet. For interactive selection of locations a common online map services like GoogleMaps can be used.

Fig. 3.5: Apollo Blast Plugin Explosion Source dialog.

Calculation options

All calculation options can be set in the panel shown in Figure 3.6. First, the general *calculation approach* should be selected. As described in Section 2.5.1 the options 'free field' and 'with buildings' are available here. If 'free field' is selected, all calculation options despite the radius have no effect, since no voxel model is computed. Second, the *maximum hazard radius* (R_{max}) can be specified. According to Equation (2.1) this values can be computed based on M_{TNT}. Therefore, every time an explosive device is selected from the dropdown menu in the devices panel discussed above, the field is automatically filled with the computed value. Though, the entry can be edited manually as well to allow its specification independently from explosives if required.

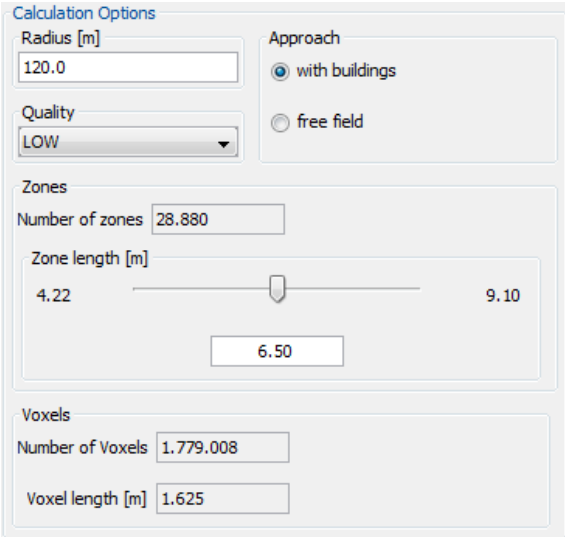


Fig. 3.6: Apollo Blast Plugin Calculation options dialog.

The settings for *quality*, *zone length* and *voxel length* control the generation of the voxel grid. The interval for the *zone length* (L_{zone}) is determined according to Equation (2.4). With the slider in the middle of the panel the user can select a value from that interval. After moving the slider, the 'number of zones' and 'zone length' fields are updated. Smaller values for L_{zone} lead to more zones in the calculation domain and therefore a higher level of detail and longer computation duration. The 'Quality' dropdown menu allows editing the *MaxLevel*. Following Equations (2.5) and (2.6) the *number of voxel per zone edge* (n_{voxel}) and the *voxel length* (L_{voxel}) are computed and displayed at te bottom of the panel. Currently eight *MaxLevels* are available. Their mapping to the 'Quality levels' shown in the dropdown is listed in Table 3.1. This setting greatly affects the quality level of the voxel model and the computation time as the number of cells per zone rises strongly with every additional level as Table 2.4 shows.

Additional scenario information and reuse of settings

Besides the calculation parameters described before, some additional information describing a blast event can be added in the *information panel* displayed on the right in Figure 3.7. The user can specify a *name* and a *description* for a scenario. The *author* and *creation date* fields are gathered from the operation system and cannot be edited on the GUI. Similar to the explosive device panel to the right a 'Save' and a 'Delete' button are placed. They allow to save or delete an entire scenario definition. The dropdown menu at the bottom of the settings panel lists all currently available presets and facilitates their reuse.

Table 3.1: Mapping of Quality levels in the Apollo Blast Plugin to mesh refinement levels ($MaxLevel$).

Quality level	MaxLevel
Zones	0
Very low	1
Low	2
Average	3
Improved	4
High	5
Very high	6
Ultra high	7

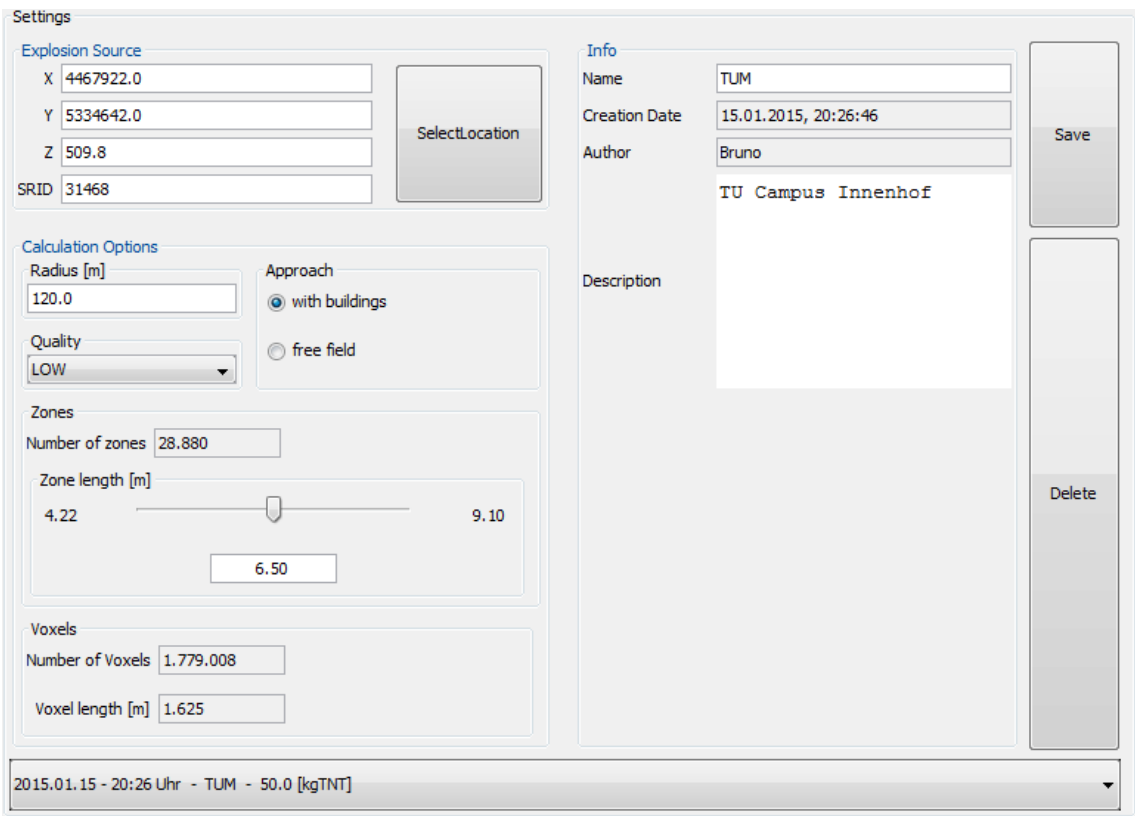


Figure 3.7: Settings panel of the Apollo Blast Plugin.

3.2.4 Data structure, storage and exchange of scenario definitions

The GUI panels described above allow the specification of a scenario, in this section it is discussed how this information is processed in the plugin. Referring to the requirements of the project (Section 1.1.2) the scenario definition needs to be persistently stored on disk and made available for reuse on the front-end. Moreover it should be exchangeable between different systems. All theses requirements can be tackled by saving the scenario information to external XML.

The Apollo Blast Plugin uses separate files for explosive devices and simulation parameters. They are both stored in the plugin directory and are named `exdev.xml` and `simpar.xml`. The first one contains only information on explosive devices, the latter one stores full scenarios including a device and all other simulation parameters. Example instance documents to both files can be found in Appendix E. The XML files can easily be exchanged by simply copying.

In the following the data structures that hold and manage the above information are explained. Since both structures for explosive devices and simulation parameters are identical despite their number, type and naming of attributes, the simpler case has been picked as example for better overview.

Explosive devices data structure

As the UML diagram in Figure 3.9 shows, the class `ExDev` describes according to the fields in the GUI an explosive device by its `name`, TNT mass (`mtnt`) and `type`, where the possible types are listed in the enumeration `ExDevType`.

To allow the storage of multiple devices and XML binding the two classes `ExDevCollection` and `ExDevController` were introduced. The collection class stores an arbitrary number of `ExDev` in an `ArrayList<ExDev>` and provides typical Java list management methods like `add(Exdev)`, `remove(ExDev)` or `contains(ExDev)`. All management functions that edit the list, trigger a `ExDevStateChange` event by calling the method (`triggerEvent()`), to inform other data structures of its change (see Figure 3.9). The Event is implemented with the global message bus of the ImporterExporter as discussed in Section 2.3.1.

`ExDevController` is in charge of saving/loading the devices from the `exdev.xml` (see Listing E.1) file on the hard disk. To do so the Java Architecture for XML Binding 2.0 (JAXB) is used. JAXB is a JAVA library that allow developers to map whole class trees to an XML representation and inverse. Its main operations are *marshal*, which saves objects to XML and *unmarshal* which creates objects from a XML document as illustrated in Figure 3.8. Every time the controller class receives a `ExDevStateChange` from the collection class, `handleEvent(Event)` is called and marshals the changed collection to the XML file. On plugin startup the file is unmarshaled and the resulting collection is distributed around the application (see Figure 3.9).

The library uses Java *annotations* to introduce custom XML namespaces, elements or attributes or define XML root nodes. (Sun Microsystems, 2006) Namespaces are introduced in the `package-info.java` of the `exdev` package, the root element of the XML tree is the collection class. Element names and their attachment to a specific namespace are set in the `EvDev` class. Listing 3.2 gives a brief example on the java annotations used. The resulting instance document can be viewed in Appendix E, Listing E.1.

```
1
2 // Schema and namespce definition - package-info.java
3 @XmlSchema(namespace = "http://bwibo.de", elementFormDefault =
4 XmlNsForm.QUALIFIED, xmlns = {
5 @XmlNs(prefix = "exdev", namespaceURI = "http://bwibo.de/exdev") })
6
7 // XML root element definition - ExDevCollection.java
8 @XmlRootElement(name = "ExplosiveDevices", namespace = "http://bwibo.de/exdev")
9 public class ExplosiveDevices {...}
10
11 // XML element naming - ExDev.java
12 @XmlElement(name = "massTNT_kg", namespace = "http://bwibo.de/exdev")
13 public Double getMasstnt() {
14   return masstnt;
15 }
```

Listings 3.2: JAXB example.

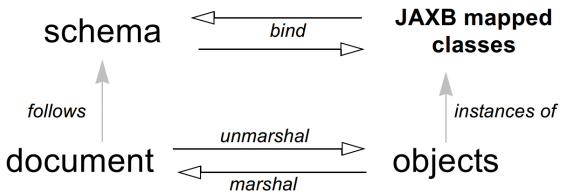


Figure 3.8: JAXB binding of XML to Java objects. Image from: Sun Microsystems, 2006

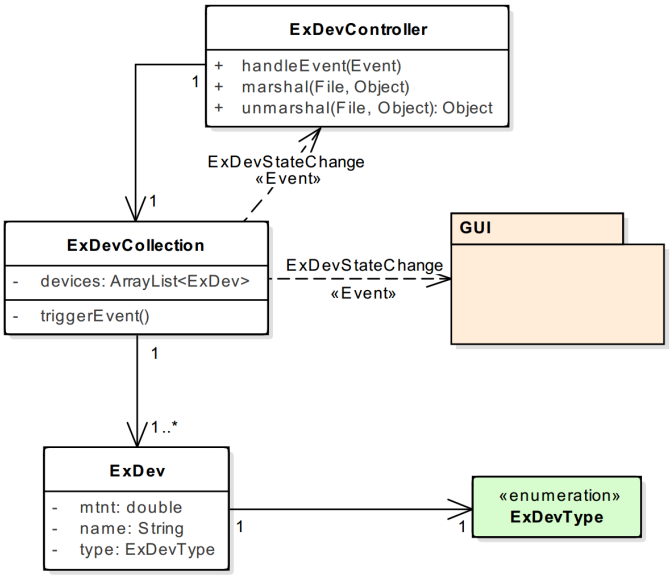


Figure 3.9: Simplified UML model of the Explosive Device data structure.

Simulation parameters data structure

The *simulation parameters* data structure handles full scenario definitions consisting of an explosive device, an explosion location and all other parameters described in Section 3.2.3. Everything works exactly the same as for explosive devices described above. The main classes are named `SimParController`, `SimParCollection` and `SimPar`. For listing the calculation approaches and quality options the enumerations `CalcOption` and `CalcQuality` are used. The explosion location is represented by the class `Point3D`. An example XML instance document can be found in Appendix E, Listing E.2.

3.2.5 Simulation area and voxel grid description

The identification of the simulation area and the definition of the voxel grid are based on the scenario definition. Following the description of the voxel model in Section 2.5.4, the initial parameters are the explosion source (P_{ex}) and the *mass of TNT* (M_{TNT}). They allow according to Equation (2.1) the estimation of the maximum hazard radius ($R_{max,temp}$) for the blast event. With theses parameters and equations a BBox has to be computed, that covers the hazard area and exactly fits the zoning decomposition of the voxel model. To achieve this the following approach was developed. In order to calculate the *ZoneLength* (L_{zone}) the *Volume of the initial BBox* ($V_{bbox,temp}$) is needed. According to Equation (2.2) lowest point of the terrain (z_{min}) is needed for that, it is queried from the database for the initial BBox retrieved from $R_{max,temp}$. Based on $V_{bbox,temp}$ the *ZoneLength* (L_{zone}) is computed. Naturally the BBox around $R_{max,temp}$ will not be an even multiple of L_{zone}. Therefore it is extended as illustrated in Figure 3.11 to fit the zoning by rounding n_{zones} to the next bigger integer:

$$\begin{aligned} n_{zones} &\approx \frac{R_{max,temp}}{L_{zone}} \qquad [-] \\ R_{max} &= n_{zones} \cdot L_{zone} \quad [m] \end{aligned} \tag{3.1}$$

With the new R_{max} the actual BBox is computed and the corresponding z_{min} is queried. Using the given $MaxLevel$ from the scenario definition for calculating the *number of voxel per zone edge* (n_{voxel}) and the *length of a voxel* (L_{voxel}) all parameters for specifying the BBox for selecting the required part of the city model and setting up a voxel grid are collected.

The data structure storing and calculating these informations resides in the `voxelgrid` package, an overview is given in Figure 3.10. It consists of the three classes `VoxelGrid-Bounds`, `BBoxGeo`, `BBoxIJK`. `BBoxGeo` represents a 3D bounding volume, by a PostGIS `box3d`, an `srid` integer and an `srsname` string. Methods for returning the volume and PostGIS 2D BBox are available. `VoxelGridBounds` stores two boxes of this type, one based on the initial BBox, one for the extended one as described above. `BBoxIJK` represents a BBox in IJK coordinates defined by `imin`, `imax`, `jmin`, `jmax`, `kmin`, `kmax` and the length of a cell edge(`celllength`). It offers methods for counting the number of cells in the box and the number of outer cells surfaces, which are needed as an input for the Apollo Blastsimulator. The data type is used twice in `VoxelGridBounds` for the zonal and the voxel grid, as described in Section 2.5.4. `VoxelGridBounds` therefore represents the whole computational

domain, both in world and IJK coordinates around the application. The main purpose of the class is to provide the voxel generation with the required information on the grid. To enable parallel voxel generation the class is able to return a sliced representation of the voxel grid. Moreover all information can be printed for information and debugging purpose. Listing 3.3 gives an example log message from the ImporterExporter. It illustrates a voxel model from the testing area at the TUM campus, which is located at the coordinates given in line 2. Line 4-7 shows the fitting of the calculation domain to the zonal grid. Lines 8-13 show the specification of the zonal grid, line 14-20 the voxel grid based on the MaxLevel from line 3.

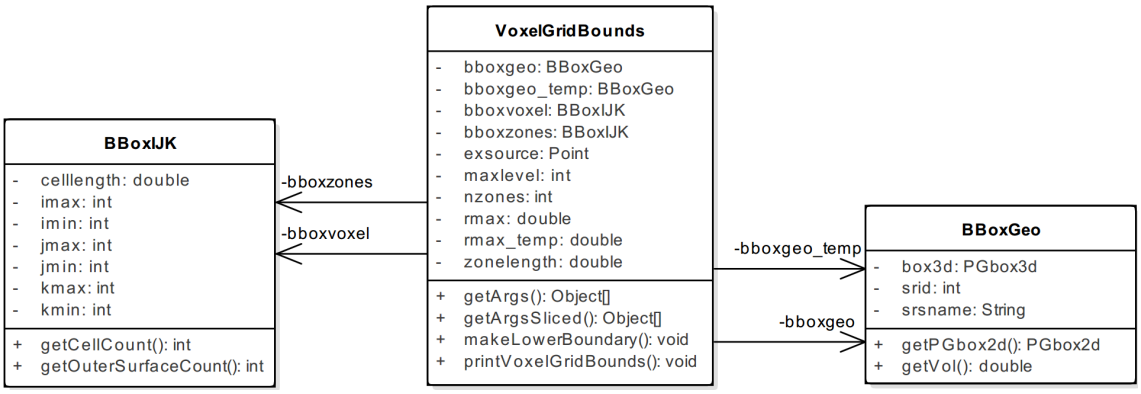


Figure 3.10: Simplified UML-Diagram of the voxel model.

```
[10:41:51 INFO] -- VBG ------------------------------------------------------------
ExSource = SRID=31468;POINT(4467922 5334642 0)
MaxLevel = 3
rMax_temp = 120.0   rMax = 123.5
zMin_temp = 0.0   zMin = 0.0
BBoxGeoTemp = BOX3D(4467802 5334522 0,4468042 5334762 120)
BBoxGeo     = BOX3D(4467798.5 5334518.5 0,4468045.5 5334765.5 123.5)
BBoxZones =
  BBoxIJK[
    celllength=6.5,
    imin=-19, jmin=-19, kmin=0, imax=18, jmax=18, kmax=18,
    ncells_imin=19, ncells_jmin=19, ncells_kmin=0,
    ncells_imax=19, ncells_jmax=19, ncells_kmax=19]
BBVoxels =
  BBoxIJK[
    celllength=1.625,
    imin=-75, jmin=-75, kmin=1, imax=76, jmax=76, kmax=76,
    ncells_imin=76, ncells_jmin=76, ncells_kmin=0,
    ncells_imax=76, ncells_jmax=76, ncells_kmax=76]
-------------------------------------------------------------------------------------
```

Listings 3.3: Example log message of the VoxelGridBounds class.

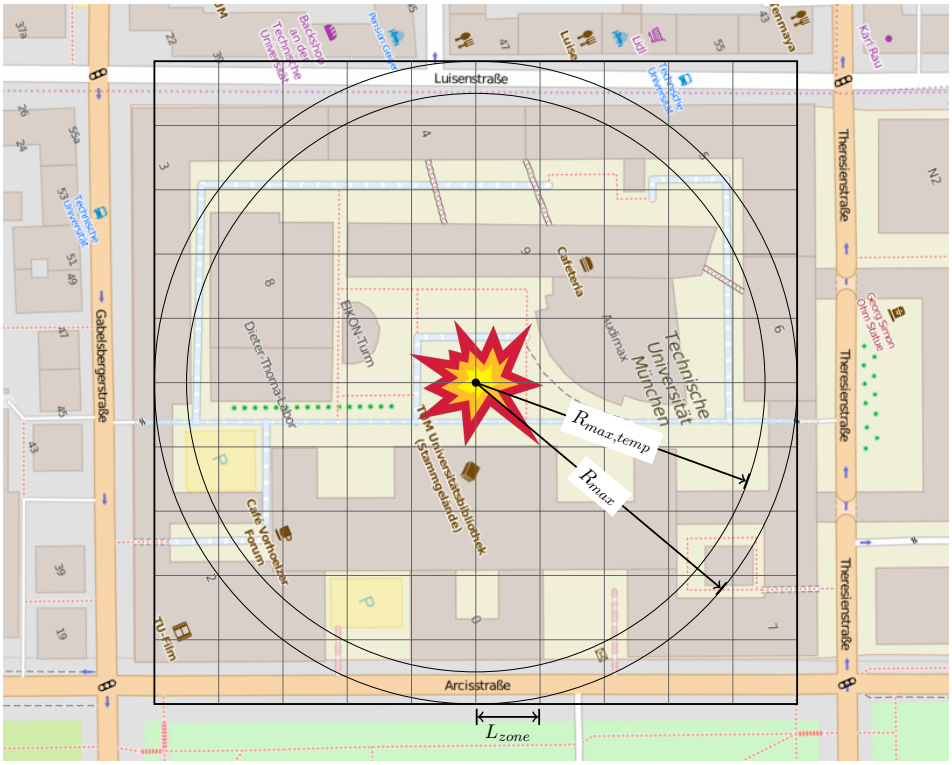


Figure 3.11: Adaption of the max. hazard radius to fit the Apollo Blastsimulator zoning.

3.2.6 Database interaction

The following section is about a central part of the Apollo Blast Plugin, the interaction with the 3DCityDB. Most important in this context is the class `DBController`, as it is involved with literally *any* database interaction of the plugin. At time of writing theses lines the class contained more than 1500 lines of Java code, therefore a detailed explanation would go beyond the demands of this paper. Here only the general concept and the most important functions will be discussed. For detailed information on the implementation feel free to look into the source code delivered with this thesis.

SQL storage in the plugin

Every database interaction of the Apollo Blast Plugin is based on SQL queries sent to the 3DCityDB via the PostgreSQL Java Database Connectivity (JDBC). Therefore lots of SQL strings need to be stored. As they usually span multiple rows they are unhandy to deal with in Java source code. To solve this issue all queries are stored in *Enumerations* allowing to access them by a short, descriptive name. To allow the organization of a big number of single queries the `Queries` interface was introduced, as the UML diagram in Figure 3.12 illustrates. The chart shows an excerpt of query enumerations stored in the `db.sql` package. Enums are organized by type (`PreparedStatement`, `Statement`) and purpose. The dictated method `getSql()` forces enumerations to implement a string field containing the actual query. Listing 3.4 gives an example of how queries are accessed (lines 28, 31). Moreover the handling of `PreparedStatements` is illustrated. Arguments for the statement are placed in an `Object[]` array and passed to the executing method. The DBController internally creates a `PreparedStatement` and sends it to the database (line 26-28).

Requirements to the database interaction concept

The biggest challenge in finding a concept for database interaction was that almost every task the plugin deals with requires it. Despite the scenario definition basically every data structure reads or writes to the 3DCityDB at some point. The main reason for this is that most data processing is performed on the database to avoid excessive network traffic as discussed in Section 2.2.3. To prevent implementing the same database related routines over and over again around the application, a concept was needed dealing with all relevant issues regarding DB access from a central point without limiting functionality. The following list summarizes the demands to the concept:

- **Databse access without code duplication**
  The typical database tasks like querying results or writing data to the DB shall be available everywhere in the application without reimplementation.
- **Management of database ressources**
  Database resources like `Connections`, `Statements`, `ResultSets` need to be properly discarded after use.
- **Transaction handling**
  If a write operation to the database fails, it needs to be rolled back to avoid data inconsistency.

- **Database error handling**
  If errors occur during database operation the plugin needs to deal with them properly.

The solution I developed is based on *public static functions*, `DBController` is a purely static class. It provides methods for all relevant database tasks that can be used everywhere in the application. These methods contain the program logic for their specific assignment, cope with error handling and guarantee safe transactions. Resource management needs to be taken care of by the *calling procedure*. Like this `Connection` objects can be reused in a process several times until they are not needed any more. Listing 3.4 gives an example on the usage of the concept. The calling procedure in `AnyClass` first creates a `Connection`. It is used for executing a prepared update query first, and second for querying a double value from the database. The `getResultSet(Connction, Queries)` method is called for this (line 5-16). Line 12 shows the internal error handling of the `DBController`. Currently errors are logged to the ImporterExporter log panel displaying the error message and the query causing it. Static function from the class `ErrorHandler` from the `util` package take care of this. After that all opened database resources are closed. All steps are conducted using methods provided by the `DBController`.

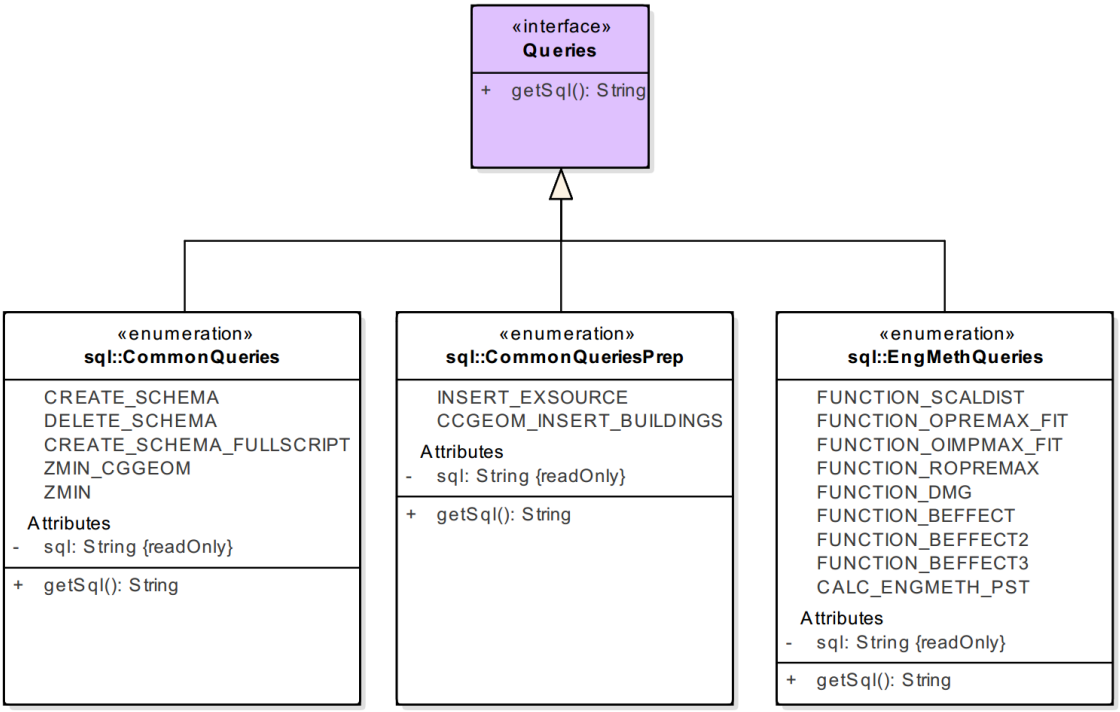


Figure 3.12: Query storage in the Apollo Blast Plugin.

```java
// Class DBController ------------------------------------------------------------
public class DBController {
  // ...
  // Query data from DB
  public static ResultSet getResultSet( Connection con, Queries query ) {
    if ( con != null && query != null ) {
      try {
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery( query.getSql() );
        return rs;
      } catch ( SQLException e ) {
        ErrorHandler.handleSQLExeption( e, query );
      }
    }
    return null;
  }
  // ...
}

// Example program ---------------------------------------------------------------
public class AnyClass {
  public static void main( String[] args ) {
    // get connection
    Connection con = DBController.getConnection();

    // Execute a prepared update statement
    Object[] args = { new Point( 1, 1 ) };
    DBController.executePreparedUpdate( con, CommonQueriesPrep.INSERT_EXSOURCE,
    args);

    // Query zmin
    ResultSet rs = DBController.getResultSet( con, CommonQueries.ZMIN );
    // process results
    double zmin = rs.getDouble( 1 );

    // Close database resources
    DBController.close( rs );
    DBController.close( con );
  }
}
```

Listings 3.4: Minimal Java example of the database concept.

3.3 Derivation of the Apollo Blastsimulator Voxel model from CityGML

According to Section 1.2 one of the main research questions of this thesis is how the voxel model Apollo Blastsimulator works with can be derived from the CityGML data model. This section is going to explain the approach I developed. First the representation of voxels in PostgreSQL is going to be discussed, after that how intersection operators can be used for the model generation. In the end an prototypic implementation for parallelization of the task is introduced. For starters, the general idea of finding a voxel representation of a city model is to intersect CityGML geometries with voxels to get a list of all voxels that represent solids in the model.

3.3.1 Voxel representation and creation with PostGIS

As mentioned in Section 2.5.4 all CityGML geometries have to be represented as an aggregation of finite volumes for simulations with the Apollo Blast Simulator. These volume pixels, short voxels, need to be mapped as an PostGIS spatial object to allow working with them on the 3DCityDB. Basically a voxel is a simple volumetric cube defined by its edge length (L_{voxel}). PostGIS uses a BRep model for geometries. Hence, a cube is described by its outer shell. As explained in Section 2.2.4 PostGIS provides the `PolyhedralSurface` data type which can be used to describe closed surfaces as needed for this application. Following the specifications from EMI for simulation results (see Section 2.5.6) the faces of a voxel are numbered by an ID as Figure 3.13 illustrates. For example the surface on the x_{min} side of the cube receives the $ID = 1$. To keep this information, the faces in the polyhedrons we want to create have to be ordered the same way.

To approximate a part of a city many voxels are required. According to Section 2.5.4 they are organized in an integer based IJK coordinate system. Thus the created voxels must both fit the specifications of the global coordinate system of the city model and the IJK system. The voxel generator developed to meet all these conditions is shown in Listing 3.5. It has been implemented as a PL/pgSQL function. Voxels are therefore created directly on the database and do not have to be transfered to the server. This is essential as big voxel grids can consume a huge amount of data that would be costly to transfer via network connections. The input parameters of the function are the voxel *edge length* (L_{voxel}), the *SRID* of the current world SRS, the *point of origin* of the grid and the *IJK coordinates* of the voxel (IJK_{voxel}) one wants to create. Based on this input a single voxel is made and returned as PostGIS data type `GEOMETRY`. The creation process is straightforward. First a voxel with the given L_{voxel} is written as WKT string. Faces are thereby ordered according to Figure 3.13, a WKT example is given in Listing 2.1. That string is converted to a geometry object with the given SRID with the PostGIS function `ST_GeomFromText`. The resulting object is then translated with `ST_Translate` by IJK_{voxel} times L_{voxel} relative to

the point of origin as follows:

$$\begin{aligned} \Delta x &= x_0 \cdot (i-1) \cdot L_{voxel} \quad [m] \\ \Delta y &= y_0 \cdot (j-1) \cdot L_{voxel} \quad [m] \\ \Delta z &= z_0 \cdot (k-1) \cdot L_{voxel} \quad [m] \end{aligned} \tag{3.2}$$

IJK_{voxel} needs to be reduced by one to fit the EMI specifications for the voxel coordinate system as described in Section 2.5.4.

How whole voxel grids are set up using this function is shown in an example query starting from line 31 in Listing 3.5. The IJK boundaries of a voxel grid are handed to the PostgreSQL function `generate_series()` for each i, j, and k. By cross joining the resulting series all possible IJK_{voxel} coordinate tuples are produced and handed into `makeVoxel()`. In the example four voxels along the x-axis with $j = 1$ and $k = 1$ are made. The results can be observed in line 44 and following. The table shows the coordinates of each voxel, its extent can bee seen in the `BBox3D` column.

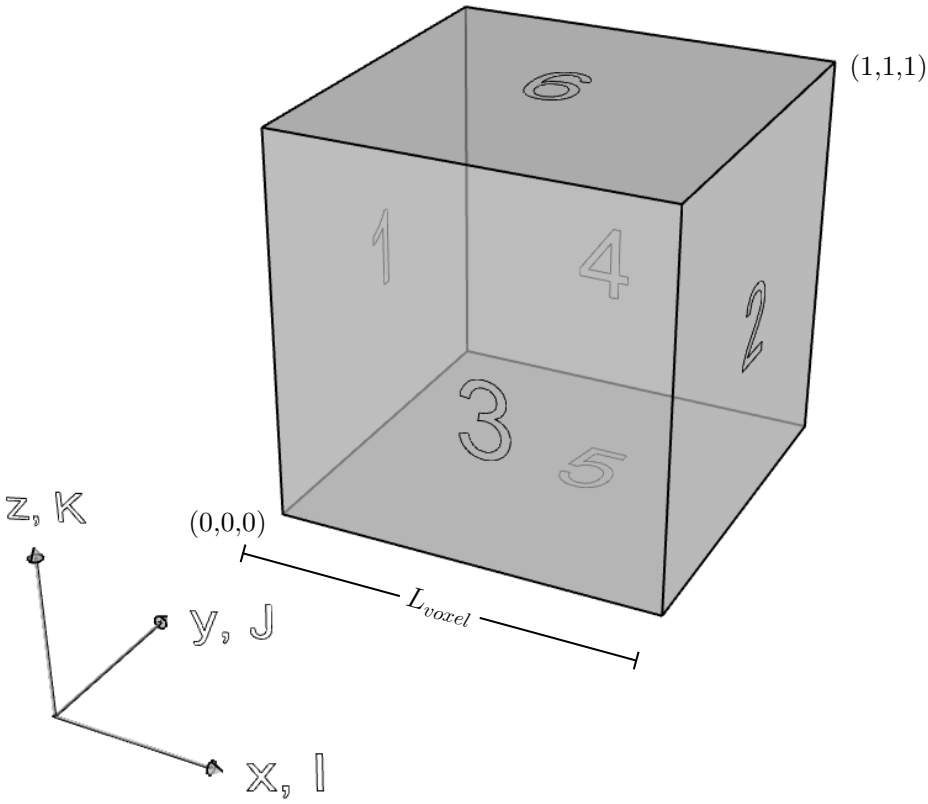


Figure 3.13: Voxel with faces IDs according to EMI specifications. Due to alignment to all three coordinates axis it is represented by its 3D BBox.

```
-- Pl/pgSQL function for voxel generation -------------------------------------
CREATE OR REPLACE FUNCTION apolloblast.makeVoxel(
  edge_length   double precision,
  srid          integer,
  x0            double precision,
  y0            double precision,
  z0            double precision,
  i             bigint,
  j             bigint,
  k             bigint)
RETURNS geometry AS
$$
BEGIN
  RETURN
  ST_Translate(
    ST_GeomFromText(
      'POLYHEDRALSURFACEZ(
        ((0 0 0, 0 0 '||$1||', 0 '||$1||' '||$1||', 0 '||$1||' 0, 0 0 0)),
        ... )', $2
    ),
    $3 + ($6 - 1) * $1,
    $4 + ($7 - 1) * $1,
    $5 + ($8 - 1) * $1
  );
END;
$$ LANGUAGE plpgsql;

-- Query for generating a voxel gird ------------------------------------------
--     BBOX IJK: [I: 1, 4 |J: 1, 1 |K: 1, 1]
--     edge length = 1
--     SRID = 31468, Origin: (0,0,0)
SELECT
  i, j, k,
  apolloblast.makeVoxel(1, 31468, 0, 0, 0, i, j, k) geom,
  Box3D(apolloblast.makeVoxel(1, 31468, 0, 0, 0, i, j, k)) bbox3d
FROM
  generate_series(1, 4) AS i,
  generate_series(1, 1) AS j,
  generate_series(1, 1) AS k;

-- Example output -------------------------------------------------------------
| i | j | k | geom                      | bbox3d                |
| - + - + - + - - - - - - - - - - - - + - - - - - - - - - - +
| 1 | 1 | 1 | POLYHEDRALSURFACE(....) | BBOX3D(0 0 0, 1 1 1) |
| 2 | 1 | 1 | POLYHEDRALSURFACE(....) | BBOX3D(1 0 0, 2 1 1) |
| 3 | 1 | 1 | POLYHEDRALSURFACE(....) | BBOX3D(2 0 0, 3 1 1) |
| 4 | 1 | 1 | POLYHEDRALSURFACE(....) | BBOX3D(3 0 0, 4 1 1) |
```

Listings 3.5: Simplified example of the voxel generation with a Pl/pgSQL function.

3.3.2 3D intersection with PostGIS

As mentioned before, the idea for the generation of the voxel model for the Apollo Blastsimulator is to fill the entire computational domain with voxels and match them with the geometries of the CityGML city model to find intersections between both. After the mesh generation process described before, we are now gonna look at the intersection.

CityGML spatial objects are represented by `polygons` in the 3DCityDB (see Section 2.2.2), voxels are represented by closed *polyhedral surfaces*. To test both for spatial interaction in 3D space PostGIS offers the `ST_3DIntersects` function. It takes two geometries and returns a boolean value. The method evaluates `true` for any intersection type despite disjoint as described in Section 2.2.4.

However, there is one important limitation to this. PostGIS does currently not have full 3D support in terms of dealing with solids. Though a `PolyhedralSurface` can be checked for closeness with `ST_isClosed`, this only refers to its outer shell and does not imply that the enclosed volume with its boundary is recognized as volumetric 3D object. Obviously this leads to a problem when trying to find intersections between finite volumes and polygons. Table 3.2 lists the intersection behavior of both 2D (&&) and 3D (&&&) BBox comparison and `ST_3DIntersects` for the given scenario. As row 3 of the table shows, the intersection evaluates to `false` when the polygon lies fully in the interior of the polyhedron since their surfaces do not intersect. Accordingly in (rare) cases when geometries of the city model are located completely inside voxels intersection will fail. Especially when working with a coarse grid ($L_{voxel} \gg 1\,\mathrm{m}$) this could lead to holes in the voxel representation of the city model and diminish simulation results.

To cope with this issue a work around was developed emulating volumetric properties for voxels during the intersection test. Assuming a voxel is aligned to all three coordinate axes (as depicted in Figure 3.13), it is exactly represented by its 3D BBox. With the &&& operator we can test if another BBox intersects the one representing the voxel. If we used only points for this check, the comparison returns if a point lies inside a voxel or not, as the BBox of a point is the point itself. Hence, if we inspect for all points forming a geometry if at least one of them is inside the 3D BBox representing the voxel, we know for sure that the geometry the point belongs to and the voxel intersect. Row 3 of Table 3.2 shows this special case. While the `ST_3DIntersects` function does not recognize the polygon inside the voxel, the check if any point it is composed of is in the 3D BBox does. If both operations are combined, the desired result is returned.

A consequence of this workaround is, that the voxel grid must be aligned to all three coordinates axis. Therefore, it cannot be rotated to enable a better approximation of buildings by voxels as discussed in Section 2.5.4. Though, the same result is obviously attainable by rotation the CityGML geometries in the table `cggeom`.

Listing 3.6 shows the query performing this intersection approach. The Common Table Expression (CTE) from line 1 to 9 contains the voxel generation as described in Section 3.3.1. They are intersected with the city model geometries from table `cggeom` using a 3D BBox comparison to make use of the 3D GiST index and a 3D intersection test (lines 11-19). Second the workaround is applied. For preparation all points forming the geometries in `cggeom` have been dumped using the PostGIS function `ST_DumpPoints` to the table `point`.

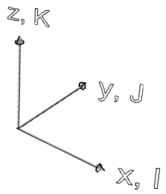
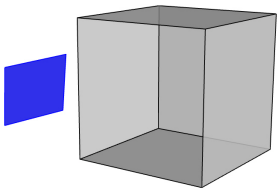
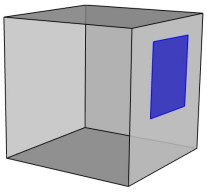
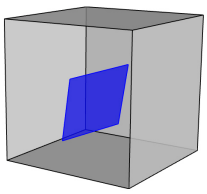
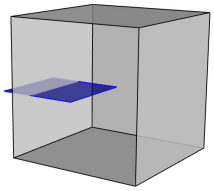
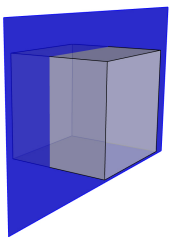
These points are now run against the voxels using the &&& operator (line 21-30). All positive intersections from both steps are stored in the table `intersection_temp` with the `uid` of the `cggeom`, the voxels IJK coordinates and geometry (line 10). After that in a separate step unique voxels are identified by their IJK coordinates and extracted to table `voxel`. The unique relations of `cggeom` and `voxel` are queried from `intersection_temp` as well and stored in the table `intersections`. Now the temporary table can be deleted.

On the one hand this approach guarantees that no voxels are missing in the model, but on the other hand there are some downsides as well. Besides the aforementioned limitations regarding rotation of the voxel grid a lot of duplicates are produced due to the 'double' intersection of points and polygons. However, at some point in the future PostGIS will support 3D volumetric spatial objects and operators to make this workaround obsolete. In many cases where slight errors in the model are negligible or a high resolution is used ($L_{voxel} \ll 1\,\mathrm{m}$) the workaround can be omitted anyway.

```
1  WITH voxel AS (
2    SELECT i,                          -- voxel generation, ? = placeholder
3      j,
4      k,
5      apolloblast.makeVoxel(?, ?, ?, ?, ?, i, j, k) geom
6    FROM generate_series(?, ?) i,
7      generate_series(?, ?) j,
8      generate_series(?, ?) k
9  )
10 INSERT INTO apolloblast.intersection_temp
11 SELECT cg.uid,
12   v.i,
13   v.j,
14   v.k,
15   v.geom
16 FROM apolloblast.cggeom cg,
17   voxel v
18 WHERE cg.geom &&& v.geom AND          -- 3D BBox comparison for 3D GiST usage
19   _ST_3DIntersects(cg.geom, v.geom)   -- 3D intersection
20 UNION ALL
21 SELECT cg.uid,                        -- workaround
22   v.i,
23   v.j,
24   v.k,
25   v.geom
26 FROM apolloblast.cggeom cg,
27   apolloblast.point pt,
28   voxel v
29 WHERE cg.uid = pt.uid_cggeom AND
30   v.geom &&& pt.geom;                 -- 3D BBox comparison voxel <-> point
```

Listings 3.6: Query for creation and intersection of voxels and city model geometries.

Table 3.2: Intersection of a `Polygon` (**P**) and a closed `PolyhedralSurface` (**V**) with different PostGIS operators and functions.

	**P** && **V**	**P** &&& **V**	ST_3D-Intersects(**P**,**V**)	One or more points of **P** in BBox3D of **V**	Combined result column 3 and 4
	✘	✘	✘	✘	✘
	✔	✔	✔	✔	✔
	✔	✔	✘	✔	✔
	✔	✔	✔	✔	✔
	✔	✔	✔	✔	✔

3.3.3 Multi-threaded intersection of voxels and CityGML geometries

As described in Section 2.2.3, PostgreSQL does not have a native multi threading optimization for single queries. Though, like for all modern DBMS, parallel execution of severals queries on different I/O channels is supported. To efficiently exploit this feature for the concurrent processing of a database task some preconditions must be fulfilled. First, the task needs to be divisible into smaller chucks of work that can be executed on their own and deliver in addition the same result as the initial task. Second, the sub tasks must not write to the same data object at the same time. PostgreSQL data consistency mechanisms will block these operations decreasing performance or even lead to failure of the operation. By far the most time consuming process in this project is the intersection of voxels and CityGML geometries. Both aforementioned conditions can be fulfilled for this task.

Writing to the same data objects is not a problem in this case, as no existing data sets need to be updated. The intersection process only reads from the city model and creates new entries for voxels and intersections as described in Section 3.3.2. The division of work is done by splitting up the voxel grid into small subdivisions which are intersected with the static set of city model geometries separate from each other. To do so, the intersection query (see Listing 3.6 on page 63) was designed to allow the creation of variable sized parts of the voxel mesh. As shown in lines 1-9 of the listing, the fraction of the voxel grid created by the query can be controlled with the values given for the placeholders (?) in the `generate_series` functions. For each IJK coordinate the lower and upper boundary can be specified.

The definition of a voxel mesh for the whole computational domain in the Apollo Blast Plugin is defined by the class `VoxelGridBounds` as explained in Section 3.2.5. To facilitate the division of the mesh of the method `getArgsSliced()` was implemented (see Figure 3.10). It returns an array of slices of the full mesh along the I-axis. Figure 3.15 gives an example on this. In a voxel grid bounded by $IJK_{min} = 0$, $IJK_{max} = 2$ the slices for $I = [0, 1, 2]$ would be returned as the colors on the image indicate. Thus, the intersection work is split into three parts of nine voxels each that can be processed on separate I/O channels. Depending on the size of the voxel gird the current implementation of the splitting algorithm delivers few chucks of work containing a big number of voxels. In the future this could easily be changed by for example dividing the mesh along the I and J axis. Finding a good balance between number and size of chucks depends on the hardware the PostgreSQL server operates on and was not further investigated in this study.

The management of the multiple I/O channels is implemented following the *Producer-Consumer design pattern* for multi threaded environments. According to C. Nagel, 2011 the pattern works like a conveyor. *Producers* create *work items* and put them in a *WorkQueue*. A set of *Workers* takes them from there and performs work on them. When all workers are busy, the producers are blocked until a free worker is available again. If no more work items are left in the queue the workers are put to sleep.

As described in Section 2.3.1 the plugin API of the ImporterExporter delivers a generic implementation of the pattern, which can easily be adapted to the project specific problem. The UML charts to the *Plugin API WorkerPool* implementation can be found in Appendix H, the derived classes are shown in Figure 3.14. `VoxelSlice` is the *work item* of the process.

It represents the aforementioned section of a voxel mesh by an array `Object[]` containing the parameters for the intersection query and a `name`. `VoxelSlice` is the replacement for the generic `<T>` of the API's `Worker<T>` and `WorkerPool<T>` generic implementation.

The `VoxelWorker` performing the work on a `VoxelSlice` is inherited from the `DefaultWorkerImpl` class. The default implementation is extended by a `Connection` which is the I/O channel to the database and a `PreparedStatement` containing the intersection query with placeholders. The `doWork` method first updates the `PreparedStatement` with the parameters of the current `VoxelSlices` and executes it after that. When the query has completed, the worker is ready to receive the next `VoxelSlice` and repeat the process. The `Connection` and the `PreparedStatement` remain open for the lifetime of the worker. When the `shutdown` method is called, running queries are canceled and both statement and I/O channel are closed.

The life cycle management of the workers is handled by the `VoxelWorkerPool` which extends the `WorkerPool` of the API implementation. It is initialized with a `corepoolsize`, `maxpoolsize`, `queuesize` and a `VoxelWorkerFactory`. The factory implements the API's `WorkerFactory` interface. It creates new workers when demanded by the pool. The `createWorker()` methods opens a `Connection` to the database and initializes a new `VoxelWorker` with it. The pool size parameters control how many workers should deal with the incoming tasks, `corepoolsize` is the lower, `maxpoolsize` the upper limit. `queuesize` allows us to define the size of the `WorkQueue` internally used by the pool. The pool configuration parameters are currently set to static parameter in the `SimulationController`. Depending on the system the PostgreSQL server operates on, these parameters can be used for optimizing the multi threading. An overview on their influence on the performance is given in Section 4.3.

The whole worker pool implementation is embedded in the `SimulationController`. This class of the Apollo Blast Plugin is the central control point for simulation runs. Both the workflows for the engineering method (`execSim_engmeth()`) and the full simulation run (`execSim_full()`) are implemented here. The required scenario definition (`SimPar`) and voxel grid description (`VoxelGridBounds`) are passed along from the GUI with the method call. For the full simulation run the `SimulationController` serves as the producer of `VoxelSlices` for the worker pool. Moreover, a function for canceling a simulation run (`cancelSim()`) is provided. When it is called, a running `VoxelWorkerPool` is shut down. The GUI is notified on the current state of a simulation run with events.

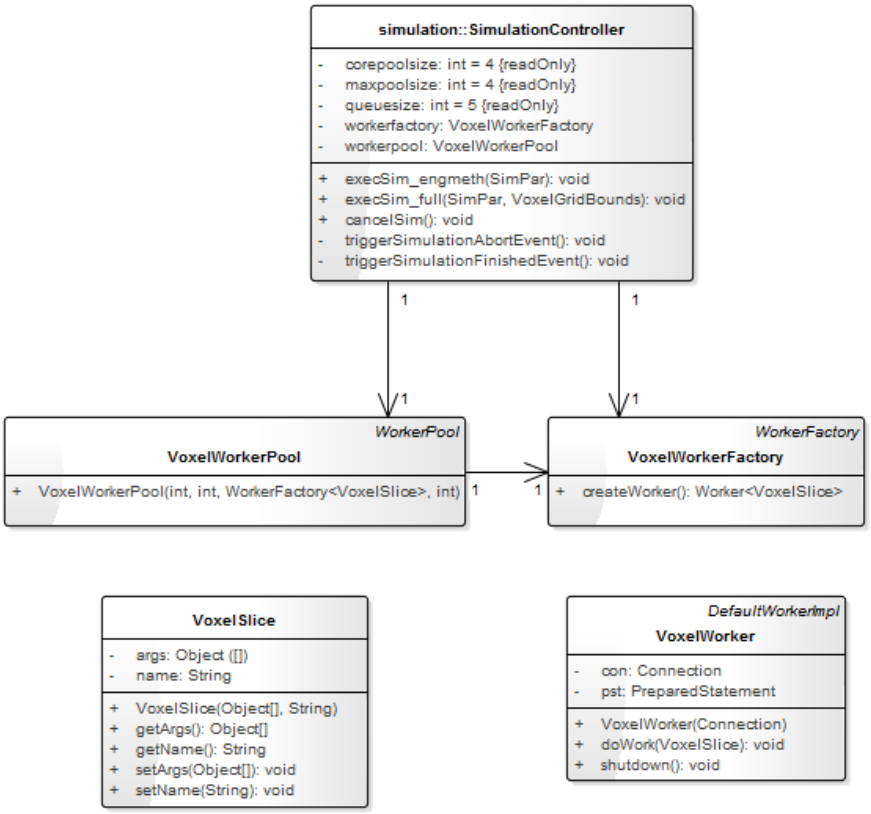


Figure 3.14: Worker Pool implementation UML Chart.

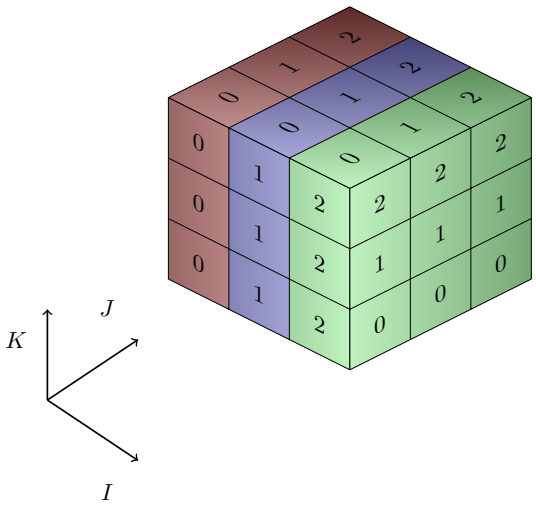


Figure 3.15: Voxel grid with highlighted slices for $I = [0, 1, 2]$.

3.4 Execution and controlling of a simulation run

Simulation runs with the Apollo Blast Plugin are controlled with the 'Simulation' panel shown in Figure 3.16. After all settings have been configured, the user triggers the 'Lock settings' button. The *estimated calculation time* is computed and displayed in the corresponding field and the 'Start' button is enabled. If the calculation time is not acceptable, the user can re-enable the configuration by pressing the 'Unlock settings' button. When everything is fine, a simulation run can be started by triggering the 'Start' button. The plugin starts, depending on the selected calculation approach (see Section 3.2.3), the engineering method or the generation of the voxel model followed by the CFD simulation with the Apollo Blastsimulator. Both approaches can be aborted during runtime with the 'Abort' button. During the simulation run detailed information to the current step of the process is provided. When simulation with buildings (full mode) and all preparation by the plugin is done, the Apollo Blastsimulator is started by a command line call. After that the status of the tool can be monitored in the 'Apollo Blast status' field on the panel.

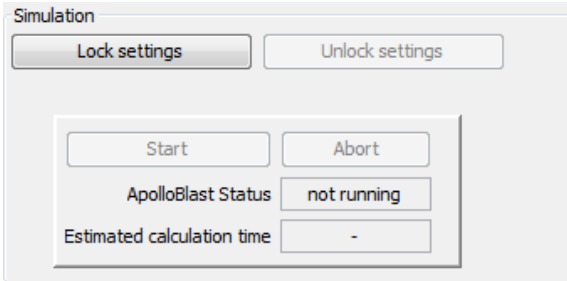


Fig. 3.16: Apollo Blast Plugin Simulation control dialog.

3.4.1 Computation of the engineering method

The engineering method is the simplified calculation the Apollo Blast Plugin supports. It computes very fast as no voxel model has to be generated and no CFD simulation is performed. The built environment is ignored, therefore it is also called 'free field' approach. As described in Section 2.5.1 the calculations are based on *TNT equivalences* and *scaled distances*, hence, the results depend only on the amount of TNT and the distance to the explosion source. The plugin currently evaluates nine damage categories with an *assessment tolerance model* as explained in Section 2.5.6.

The whole computation is performed on the 3DCityDB using PL/pgSQL functions. The empiric physical equations used in the functions are property of EMI and moreover could be easily abused. Therefore they are not published here. In total five functions have been implemented to determine the physical properties and the resulting values for the damage categories. Listing 3.7 gives an example illustrating the general style of the functions. Here the scaled distance is computed based on the TNT mass and the distance to the explosion source.

The distance calculation is performed with the PostGIS methods. `ST_3DDistance` returns the minimum distance in 3D space between two geometries, `ST_3DMaxDistance` accordingly the maximum distance. The average between both is then used for the engineering method evaluation. Distances, physical properties and damage values are all stored in the table `engmeth` (see Section 3.1.3). For each polygon of the city model one set of results is generated.

```
-- Function scaleddist -------------------------------------------------------
--     IN       mass TNT [kg]
--     IN       distance to explosion source [m]
--     RETRUN   scaled distance [m/kg^(1/3)]
-------------------------------------------------------------------------------
CREATE OR REPLACE FUNCTION apolloblast.scaleddist(
  mass double precision,
  dist double precision)
RETURNS double precision AS
$$
  BEGIN
    RETURN $2 / (( 2 * $1 )^0.3333 );
  END;
$$ LANGUAGE plpgsql;
```

Listings 3.7: Pl/pgSQL function for the scaled distance.

3.5 Processing and storage of simulation results in CityGML

The third research question stated in Section 1.2 is how simulation results can be stored in the CityGML city model. Generally two options are available, generics and ADEs. In this project generic attributes were used for mainly practical reasons. By the time of writing theses lines the ADE mechanism was not supported by the 3DCityDB and the SPSHG plugin of the ImporterExporter and therefore not reasonably usable. Though, the release of a new version of the 3DCityDB was imminent so this feature might be included in future releases. Check back to the official homepage[1] of the 3DCityDB for news on the upcoming release.

3.5.1 Simulation results processing

After a simulation has finished the Apollo Blastsimulator returns the results in an ASCII file as described in Section 2.5.6. To transfer theses results back to the city model, they need to be imported to the 3DCityDB first. This is currently done by the CSV import function of the PostgreSQL DBMS. Before that, the file is converted to a readable CSV with a shell script deleting surplus lines, spaces and fixing the number format. The script and an example output file can be found in Appendix F. Though, this is just a quick workaround. In a later version of the plugin the import process will be integrated using Java file reading capabilities. When the results have been imported to the database, they are ready for storage as generic attributes as explained in the following section.

Currently only results for solids are processed. That means that damage values for wall boundary conditions of the computational domain (after line 15 in Listing F.3) are omitted. They are not needed as in this study only buildings were considered for visualization. In a future release, when additional CityGML layers are implemented, these results will be needed to display the damage values on the ground.

1 http://www.3dcitydb.net

3.5.2 Storage as generic attribute

The creation of generic attributes is generally an uncomplicated task. Referring to Section 2.2.1 they are stored in the table `cityobject_genericattrib` on the 3DCityDB. To create an attribute for a city object simply a new row containing the attribute name (`attrname`), type identifier (`datatype`), the attribute value in the column corresponding to the type identifier (e. g. `realval`) and the reference to the city object (`cityobject_id`) needs to be added to the table.

Current limitations and workaround

Generic attributes are an easy-to-use opportunity to enhance the CityGML model, though there are some limitations to keep in mind. One CityGML model based restriction is that generic attributes can only be applied to city objects and not to single geometries. Therefore values that are generated per geometry, like for the engineering method, have to be summarized for geometry aggregations like for example CompositeSurfaces to a single value which is then attached to the corresponding object. Depending on the geometric situation this can strongly diminish the significance of simulation results.

Other limitations are caused by the tools used for data export and visualization of the simulation results. The 3D Web Client and the SPSHG at present support only top level CityGML classes (see Section 2.1) like *Building* or *Transportation* but not lower levels of thematic classes like *Thematic Surfaces*. This causes big problems regarding the presentation of simulation results, as a 'per building' aggregation of blast properties does not make any sense in most cases. Figure 3.17 gives an example on this. The highlighted structure on the left represents one *Building* instance of the city model, on the right a *Thematic Surface* of the building is shown. Below the images the attribute values for a blast event in front of the highlighted *Thematic Surface* can be observed. When comparing for example the averaged peak overpressure for the building and the surface (Figures 3.17(a) and 3.17(b)) it becomes clear that aggregation 'per building' strongly distorts the results. When picking the maximum values the falsification is even worse. This obviously goes back to the spatial positioning of the single walls relative to the explosion source. While the walls in the front are strongly exposed to the detonation, the walls on the back of the building are shadowed.

To cope with this issue a workaround was introduced allowing the selection and attribution of *Thematic Surfaces* for the visualization of the results. Therefore the whole CityGML data set was rewritten redeclaring all *Thematic Surfaces* as *Buildings*. The image on the right in Figure 3.17 displays the resulting model. Since all geometries are *Building* properties now, the highlighting for *Thematic Surfaces* (e. g. *RoofSurface* = red) is lost. To maintain the semantic information the additional generic attributes `surface_type` and `building_id` were introduced as shown in Figure 3.17(b). `building_id` stores the building membership of a surface, `surface_type` displays from which thematic class it originates.

Both the 3D Web Client and the SPSHG are under development at the Chair of Geoinformatics at TUM. Future releases will deal with this issue and make this workaround

obsolete. For news on the development progress please visit the official homepage[1] of the chair.

Generic attributes for the engineering method

The generic attributes for the engineering method are based on the calculations described in Section 3.4.1. Since they are initially computed per *Thematic Surface* no aggregation is required. The values are directly written to the city model after their evaluation. Table 3.3 lists the attributes that are currently available.

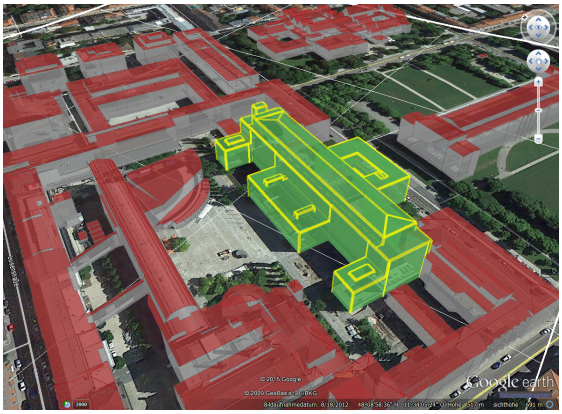
Generic attributes for the Apollo Blastsimulator results

The Apollo Blastsimulator returns a set of result figures for each face of a voxel as described in Section 2.5.6. Though, the reference between voxels and the city models *Thematic Surfaces* is per voxel and surface and not per face and surface. Therefore a two level aggregation is performed to generate the required generic attributes. Currently two strategies are available, either averaging or picking maximum values. First all result sets from the faces are summarized 'per voxel'. Second, all 'per voxel' results are aggregated 'per *Thematic Surfaces*' based on the relations stored in the `intersections` table (see Section 3.1.3). The results for both aggregation strategies are stored as attributes, they can be distinguished by a prefix as shown in Table 3.4, where all attributes currently used for Apollo Blastsimulator results are listed. An example image of the attributes from the 3D Web Client is given in Figure 3.17, a XML instance document of a *WallSurface* can be found in Appendix E, Listing E.3.

Table 3.3: Generic attributes for the engineering method.

Attribute name	Unit	Description
`peak_overpressure`	[Pa]	maximum overpressure of reflected wave
`overpressure_impulse`	[Pa s]	positive overpressure impulse of reflected wave
`window_glass_normal`	[%]	normal window glass destruction probability
`window_glass_hardened`	[%]	hardened window glass destruction probability
`window_glass_laminated`	[%]	laminated window glass destruction probability
`wall_masonry_30cm`	[%]	30 cm masonry wall destruction probability
`wall_concrete_30cm`	[%]	30 cm concrete wall destruction probability
`heavy_building_dmg`	[%]	heavy building damage destruction probability
`leathality_threshold`	[%]	threshold for lethality probability
`leathality_50`	[%]	50 % lethality probability
`eardrum_dmg`	[%]	eardrum damage probability

1 http://www.gis.bgu.tum.de



(a) per *Building* average of **(b)**

(b) AVG per *Thematic Surface*

Fieldname	Value
GMLID	
building_id	
surface_type	
AVG_peak_overpressure	3288.7430923802813
AVG_max_overpressure_impulse	82.20962301069063
AVG_overpressure_impulse	0.5970570649303817
AVG_glass	0.1815877703165184
AVG_safety_glass	0.03613216315260582
AVG_masonry_30cm	0.01500616352597317
AVG_rc_wall_30cm	0.004632151118837694
AVG_person_safe	0.0028061987437574677
AVG_lethality_1_percent	0.00005117254082952251
AVG_lethality_50_percent	0.000001294936947110858 8
AVG_eardrum_damage	0.047435882291884585
MAX_peak_overpressure	13829.474193548389
MAX_max_overpressure_impulse	235.72645161290328
MAX_overpressure_impulse	26.329282795698926
MAX_glass	0.5001763440860215
MAX_safety_glass	0.24366967741935486
MAX_masonry_30cm	0.13613726881720434
MAX_rc_wall_30cm	0.03280322580645161
MAX_person_safe	0.02490967741935484
MAX_lethality_1_percent	0.008268817204301076
MAX_lethality_50_percent	0.0005918279569892472
MAX_eardrum_damage	0.2492952688172044

Fieldname	Value
GMLID	DEBY_LOD2_4959457_f25...
building_id	UUID_d3f0bacf-3211-4df2-...
surface_type	WallSurface
AVG_peak_overpressure	43312.4787280088
AVG_max_overpressure_impulse	516.335596969697
AVG_overpressure_impulse	45.0525317784994
AVG_glass	0.597402597402597
AVG_safety_glass	0.544290883838384
AVG_masonry_30cm	0.532828282828283
AVG_rc_wall_30cm	0.398266226190476
AVG_person_safe	0.246829133192641
AVG_lethality_1_percent	0.00425313852813853
AVG_lethality_50_percent	0.000119134199134199
AVG_eardrum_damage	0.567167712842713
MAX_peak_overpressure	210200
MAX_max_overpressure_impulse	1375
MAX_overpressure_impulse	290.9
MAX_glass	1
MAX_safety_glass	1
MAX_masonry_30cm	1
MAX_rc_wall_30cm	1
MAX_person_safe	1
MAX_lethality_1_percent	0.3845
MAX_lethality_50_percent	0.02752
MAX_eardrum_damage	1

Figure 3.17: Comparison of aggregations of simulation results for *Buildings* and *Thematic Surfaces* for a blast event in front of the highlighted surface in right image.

Table 3.4: Generic attributes for the Apollo Blastsimulator results.

Prefix	Attribute name	Unit	Description
AVG_|MAX_	peak_overpressure	[Pa]	peak overpressure
AVG_|MAX_	max_overpressure_impulse	[Pa s]	maximum overpressure impulse
AVG_|MAX_	overpressure_impulse	[Pa s]	overpressure impulse
AVG_|MAX_	glass	[%]	glass destruction probability
AVG_|MAX_	safety_glass	[%]	safety glass destruction probability
AVG_|MAX_	masonry_30cm	[%]	30 cm masonry wall destruction probability
AVG_|MAX_	rc_wall_30cm	[%]	30 cm concrete wall destruction probability
AVG_|MAX_	person_safe	[%]	person safety probability
AVG_|MAX_	leathality_1_percent	[%]	1 % leathality probability
AVG_|MAX_	leathality_50_percent	[%]	50 % leathality probability
AVG_|MAX_	eardrum_dmg	[%]	eardrum damage probability

3.6 Visualization of Simulation results with the cloud-based 3D Web Client

This section is about the last research question stated in Section 1.2. First, it is explained how the simulation results stored in the CityGML city model as generic attributes can be exported and distributed with the SPSHG. Second, the visualization with the 3D Web Client is going to be discussed. Examples on the analysis performed with the 3D Web Client are given in Chapter 4.

3.6.1 Preparation: Data export and distribution

Before the web-based visualization can be started, the required data need to be exported from the 3DCityDB and made available to end users over the web. As described in section 2.4.1 visual and thematic information is supplied in separate files.

The thematic properties of the city object are provided with a *GoogleSpreadsheet*. They can be exported with the SPSHG as illustrated in Section 2.3.2. First, rules for the columns that should appear in the export need to be defined. In this project only data from generic attributes was exported. An example column definition for the attributes of the engineering method can be observed in Figure 3.18. Each rule contains a column tile, an expression defining its content and an optional comment. The setup only needs to be made once for a project and can be stored as a template. For details on the creation of selection criteria please refer to the official documentation (see Naderi et al., 2012). After the thematic layer and BBox have been specified the spreadsheet can be exported right away to GoogleSpreadsheets with the given login credentials at the bottom of the GUI.

When the file has been successfully uploaded, it should be checked on GoogleDrive if everything was exported as desired. A typical pitfall for example is corrupted number formating. Potentially calculations with simple equations known from spreadsheet tools like Microsoft Excel can be performed to create additional information that should be available in the visualization. Figure 3.19 shows an excerpt of the export specified before. The logical link to the visualization model (`GMLID`) has to be located in the first column.

Column titles should not contain any spaces or special signs. When the data are prepared as desired, they have to be shared with the users group. Google offers various options for this. This video[1] provides a good introduction to the available settings. After the sharing settings are applied, the thematic information is ready to use. Before closing the browser window save the link to the sheet for later use.

The export of the visualization model is conducted with the KML/COLLADA Export plugin (see Section 2.3.3). For this project a geometry level KML file export was produced. COLLADA is not needed as currently no textures for buildings are used. To keep the file size of the single KMZ files low for good performance during viewing automatic tiling was used with a side length of 175.0 m. All other configurations, especially regarding the appearance and highlighting of surfaces were chosen according to the official documentation Kolbe, Yao, et al., 2013, see. The functionality of automatically querying the altitude for buildings from Google's elevation API to fit them on Google Terrain disabled. With the workaround presented in Section 3.5.2 this feature would not work, since buildings are exploded to single surfaces. This would lead to roof surfaces being lowered to the terrain. Instead, the original altitude of the geometries was used. Luckily the data used for this project fit the Google altitude perfectly.

After all settings were applied the KML files were exported. For distribution Dropbox[2] was used. Dropbox offers different models for sharing files. For details on this please refer to the official homepage. For reasons of simplicity in this project the Dropbox 'public' folder was used granting read and write access for everybody. With the KML files uploaded there, everything is set up to initialize the visualization with the 3D Web Client.

1 https://www.youtube.com/watch?v=25CtYkqamIA
2 http://www.dropbox.com/

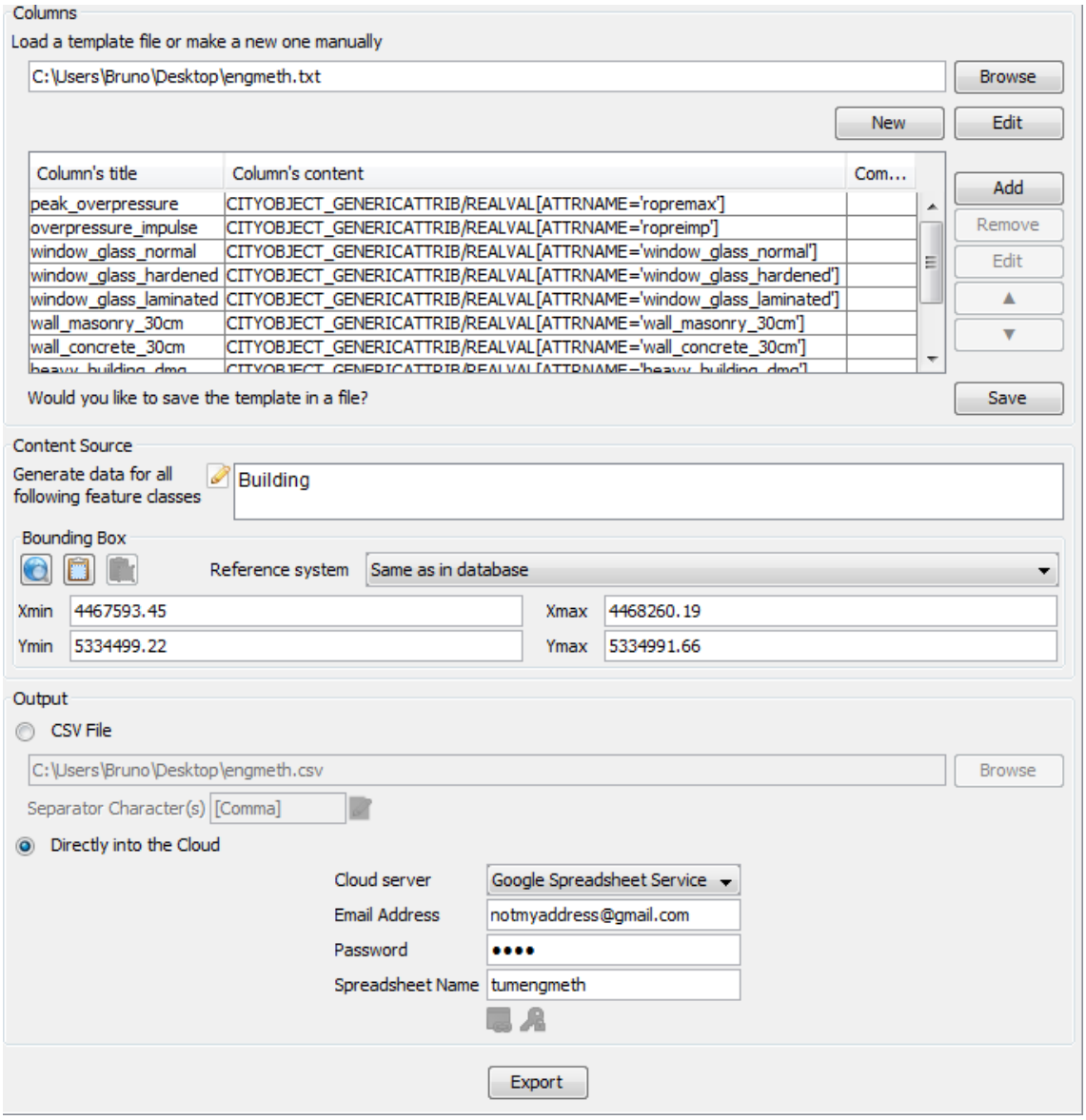


Figure 3.18: Spreadsheet Generator Plugin with example column definition.

	A	B	C	D
1	GMLID	peak_overpressure	max_overpressure_impulse	overpressure_impulse
2	DEBY_LOD2_42364	2665	101,3	-8,693
3	DEBY_LOD2_42364	1932	101,4	0,0403
4	DEBY_LOD2_42364	1745	80,94	0,0403

Figure 3.19: Excerpt of a GoogleSpreadsheet with attributes for the engineering method.

3.6.2 Setting up the 3D Web Client

To work with the 3D Web Client nothing but a standard web browser with an installation of an Earth viewer plugin is needed. In this project the GoogleEarth plugin was used. For starting a session simply the URL to the 3D Web Client has to be opened. Depending on your browser possibly some security options have to be modified to enable all features. For example, pop up windows should be allowed. When the client has loaded its interface (see Figure 3.22) is shown in the browser window and a new layer can be added with the *Add KML/KMZ Layer* dialog. To do so the URLs to the KML visualization of the city model and the attribute spreadsheet that have been distributed over cloud services as described before are needed.

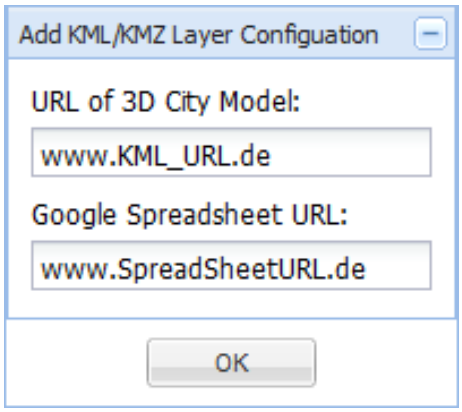


Fig. 3.20: Add *KML/KMZ Layer* dialog.

As shown in Figure 3.20 they have to be copied into the dialog. After pressing the 'OK' button the new layer is loaded. This process is repeated for every desired layer. When successfully loaded, a layer appears in the *Layer Manager*. By right-clicking on an entry of the list the user can open a menu allowing to zoom the 3D view to the layer or remove it. As described in Section 2.4.2 the 3D Web Client offers several other options like the configuration of the of additional Google layers. For example names and borders, terrain, roads or 3D buildings can be displayed. After all settings have been applied and the map is focused to a convenient viewpoint a project should be created. This can be done in the *Project Management* dropdown menu on the *Menu bar*. The 'Save Project as' option allows to store the current configuration and define a project name. Before that, the user needs to log into his Google account with the 'Sign In' button on the right of the *Menu bar*. The configuration is stored in this account in a new spreadsheet. Using the 'Show Project Link' option displays the URL to the project and allows to share it with others. To enable collaborative work on the given task the preference sheets needs proper sharing settings applied, just like the attribute sheet. The 'Show Configuration Spreadsheet' opens it in the Google account where the sharing options are available.

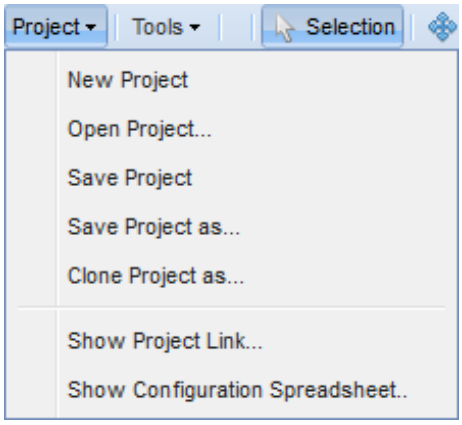


Fig. 3.21: *Project Management* dialog.

The 3D Web Client project is now set up for all permitted users for viewing, analysis or editing. In the next chapter a usage example is presented where theses capabilities are explained in detail.

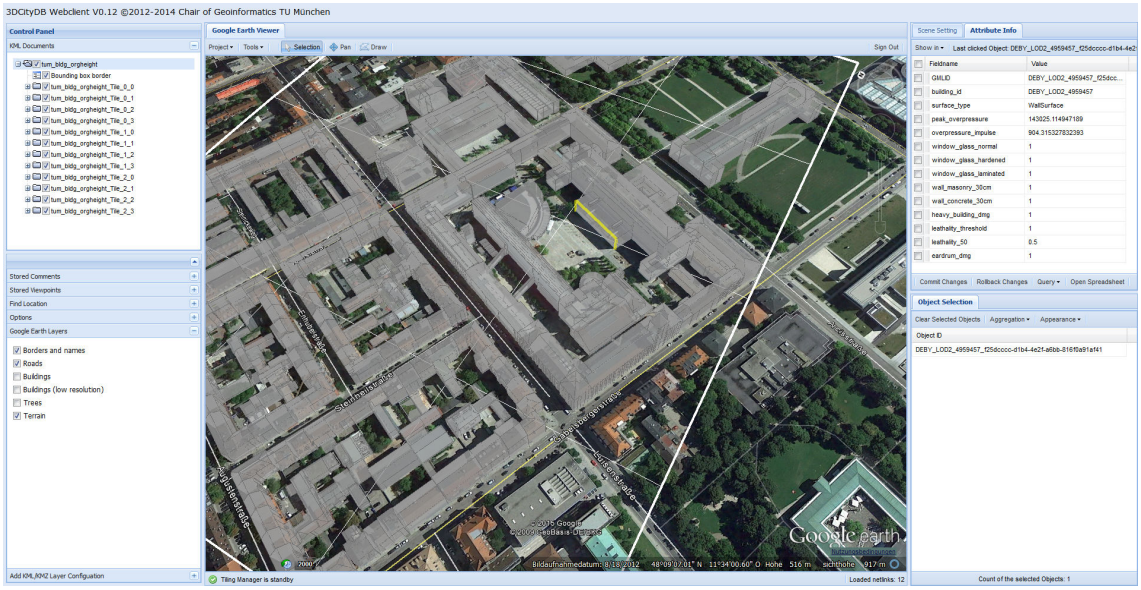


Figure 3.22: GUI of the 3D Web Client.

CHAPTER 4

Evaluation and performance

This chapter will discuss the analytic capabilities of the 3D Web Client with some practical examples. Moreover information regarding the performance and validity of the Apollo Blast Plugin will be presented.

4.1 Evaluation of blast events with the 3D Web Client

In this section a blast simulation is conducted with Apollo Blast Plugin and evaluated with the 3D Web Client.

4.1.1 Description of the test scenario

The example scenario for the following evaluation is located at the campus of the Technical University Munich (TUM). Based on a LoD2 CityGML city model provided by the Bayrisches Landesamt für Digitalisierung, Breitband und Vermessung, 2015 a blast simulation with a hemispheric 250 kg TNT load was performed using the Apollo Blast Plugin and the Apollo Blastsimulator. The initial data set is shown in Figure 4.1. The bomb is located right on campus in front of the main building. For the simulation only the buildings belonging to the main part of the university, not the surrounding blocks, were considered. Regarding the altitude of the city objects simplification was made. Since currently only the CityGML Building layer is supported all structures were placed on plane with zero altitude to guarantee a closed lower boundary of the model. The influence on the results should be negligible as the terrain of the model is almost level anyway. Moreover, the whole area was rotated to align the building's facades with the voxel grid for improved approximation.

For the generation of the voxel model a relatively coarse resolution was picked. The chosen zone length ($L_{zone} = 6.5\,\mathrm{m}$) lead to a number 23 104 zones in the domain. With a selected mesh refinement level of three ($MaxLevel = 3$) a model of 52 849 voxels with an edge length of $L_{voxel} = 1.625\,\mathrm{m}$ was derived. The whole generation process with the Apollo Blast Plugin took 97 s on a standard quad core laptop. Appendix G shows the log file to simulation run. The following CFD simulation with the Apollo Blastsimulator took 804 s.

As Figure 4.2 illustrates there are some errors in the voxel model. They are probably induced by invalid geometries in the city model causing the PostGIS intersection functions to fail. According to the log file (Listing G.1) 8 of the total 1216 city model geometries were

not intersected. The resulting failure rate of 0.66 % is acceptably low, though as observable on the image, a single invalid geometry can have a big influence on the model quality depending on its size and location relative to the explosion source. In the test case these errors could be ignored as they are far away from the blast center. Referring to the work of Alam et al., 2013 and Wagner et al., 2013 geometric and semantic validity is a common problem in 3D city models. The automatic healing of those issues for CityGML data sets is a topic of current research. Due to time constraints this was not further investigated in this thesis.

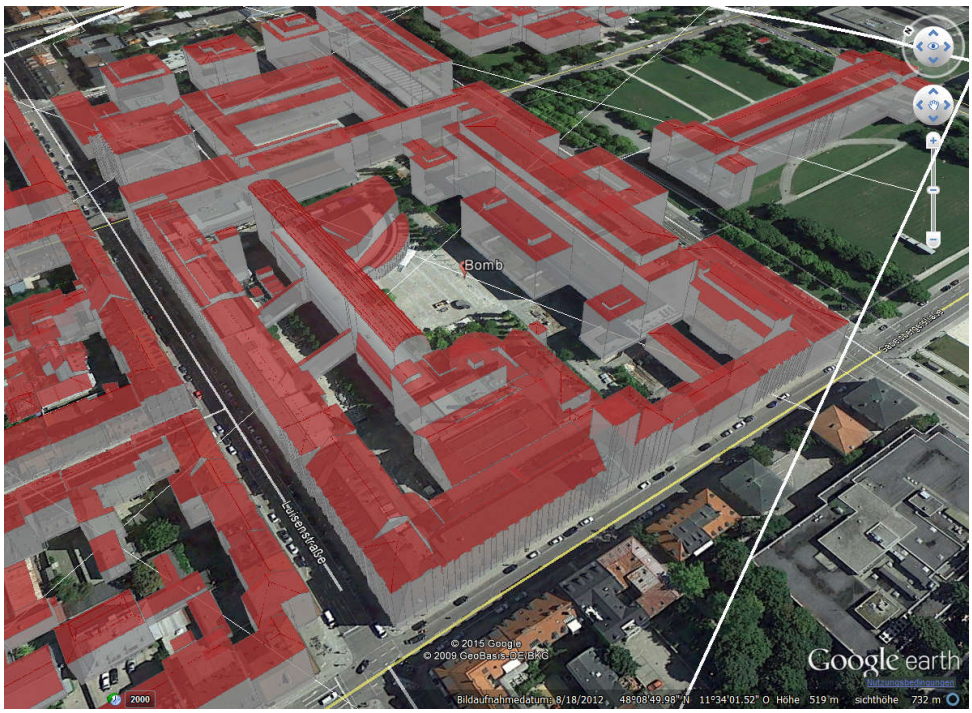


Figure 4.1: LoD2 CityGML city model of the test area around TUM.

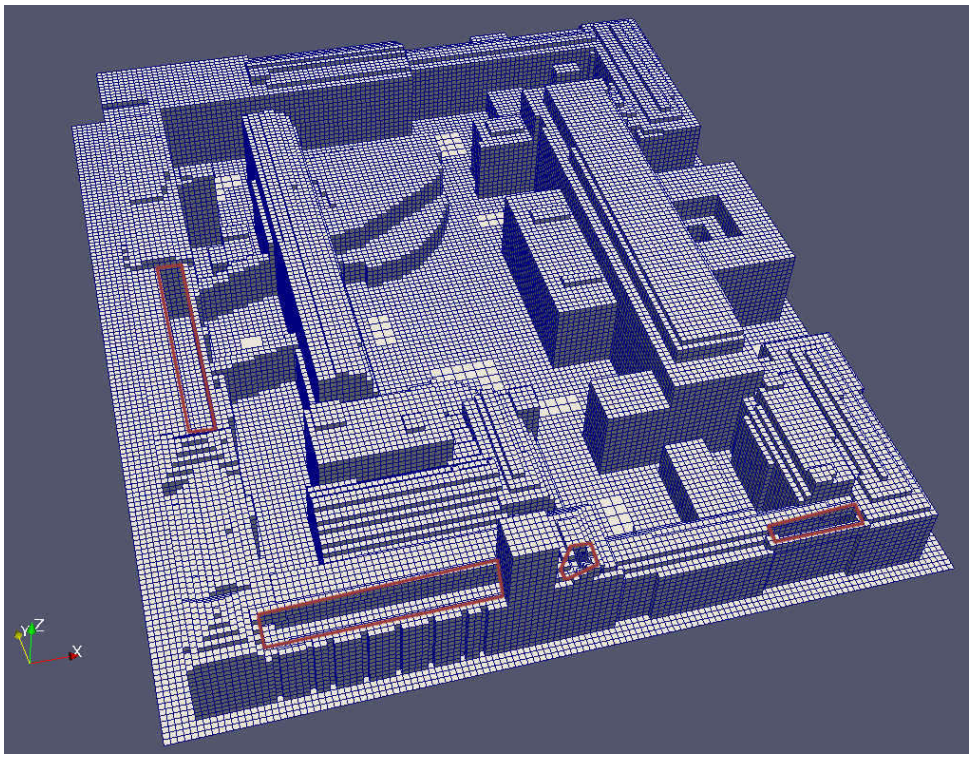


Figure 4.2: Voxel mesh of the test scenario at TUM with highlighted mesh errors, $L_{voxel} = 1.625\,\mathrm{m}$

4.1.2 A practical usage example: Cost estimation for glass breakage with the 3D Web Client

In this showcase analysis task we are going to determine how much cost glass breakage due to the blast event would cause. Thereby we focus only to the main building on site. Furthermore, we assume a window glass price of 50 €/m^2 and that the person conducting the analysis has never been on the campus of TUM before. On the fly the analytic capabilities of the 3D Web Client are explained. Please see Sections 2.4 and 3.6.2 for additional information on the tool.

Analysis with the 3D Web Client

The first step for the thoughtful analyst would be to get an overview on the scenario. Therefore the 3D City Model is disabled in the *Layer Manager* (top left) and all available Google layers are enabled in the *Google Earth Layers Panel* (bottom left). By navigating around the 3D building with the well-known move, pan and zoom functions the GoogleEarth plugin offers, a good impression on the situation can be obtained as Figure 4.6 illustrates.

The first relevant informations gained from this step are that there are no glass roofs on the building and that the portion of window glass of the whole wall surface can roughly be estimated to 40 %.

The second step is to specify criteria for the selection of possibly endangered facades. This is done in the *Attribute Info Panel* of the 3D Web Client (see Figure 4.3). First we adapt the view settings for analysis tasks and enable the 3D city model. To determine it's `building_id`, we select an arbitrary surface of the building. By selecting the field and entering the ID as condition (an operator and a value) we make sure only surfaces of this structure will be returned. Based on the information gained in step one we decide to specify `surface_type = 'WallSurface'` to leave out roof and ground surfaces. Finally we set the value for the probability of glass breakage determined by the blast simulation. We decide for a worst case scenario and set to query for all surface with a peak probability of bigger than 80 %. After all required fields have been selected and the conditions have been entered we use the *Query* menu at the bottom of the *Attribute Info* panel to query all city model geometries fitting the specified criteria for attributes. The returned result can be observed in Figure 4.5. All queried surfaces are highlighted yellow in the 3D view.

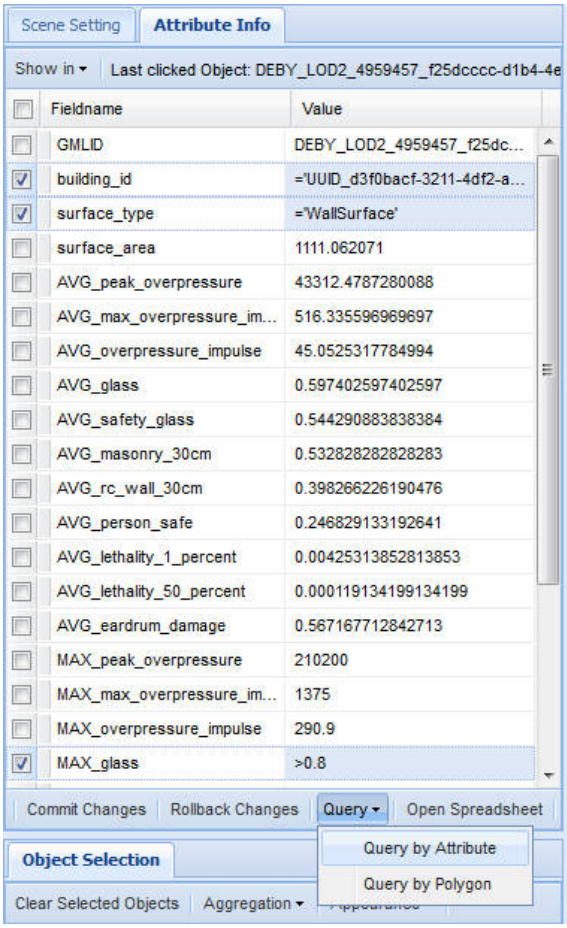


Fig. 4.3: Selection by Attribute.

As the images show, windows facing towards the bomb can be expected to be destroyed. Windows on the building's back side, facing away from the detonation are shadowed from the blast wave and are likely to remain unharmed.

The last step is to calculate the destroyed window area for the cost estimation. The 3D Web Client facilitates this by enabling the user to perform aggregations over the attributes of selected objects. In the bottom right on Figure 4.5 the *Object Selection Panel* is located. After a query, all selected objects are listed here by their GMLID. The right-click menu allows to remove single objects from the selection or hide them from the 3D view. To compute the aggregate sum of the `surface_area` we first select the field in the *Attribute Info Panel*. After that 'Sum up Values' is picked from the *Aggregation* dropdown in the *Object Selection Panel*. The result is shown in the *Attribute Info Panel*, where the headline now displays information on the performed aggregation as shown in the figure. The computed value is the total wall area in which windows would be most likely destroyed. By multiplying the value with the window surface area factor from step one and the glass cost per square meter we get the final result:

$$6585.40\,\mathrm{m}^2 \cdot 50\,€/\mathrm{m}^2 \cdot 0.4 = 131\,708\,€$$

Collaborative work

Finally we want to share the calculated results with our colleges for discussion. To do so we position the 3D view to give a good overview on the surfaces we queried and create a new *Viewpoint* with the *Stored Viewpoints Panel* as displayed in Figure 4.4. With 'Upload to Cloud' we store the viewpoint in the configuration spreadsheet of the 3D Web Client making it available to anybody with sufficient access rights as described in Section 3.6.2. When selecting a viewpoint from the list, the 3D view flies to the saved location. As not only the location, but selected objects are stored in viewpoints, we end up with the exactly the same view we were seeing after our analysis. This makes it easy for others to comprehend what we did and facilitates collaborative work.

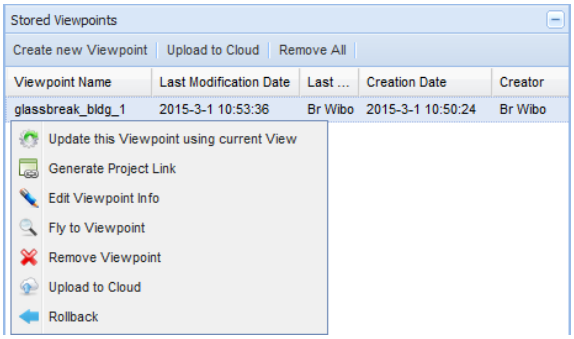


Fig. 4.4: Stored Viewpoint.

Assuming the required data are available in a 3DCityDB and all software has been set up before, the entire workflow from defining a blast scenario, running the simulation within the Apollo Blast Plugin and the Apollo Blastsimulator to exporting and distributing the result files and evaluating them with the 3D Web Client can be completed in short time. Table 4.1 gives an overview on the estimated time consumption for the single steps.

Table 4.1: Estimated time consumption for the example scenario.

Task description	Estimated duration [min]
Blast scenario definition in the Apollo Blast Plugin GUI	5
Model generation according to Section 4.1.1	1.6
Apollo Blastsimulator blast simulation according to Section 4.1.1	13.4
Data export, distribution and access management	10
3D Web Client project setup	10
Result evaluation with the 3D Web Client	15
Total	55

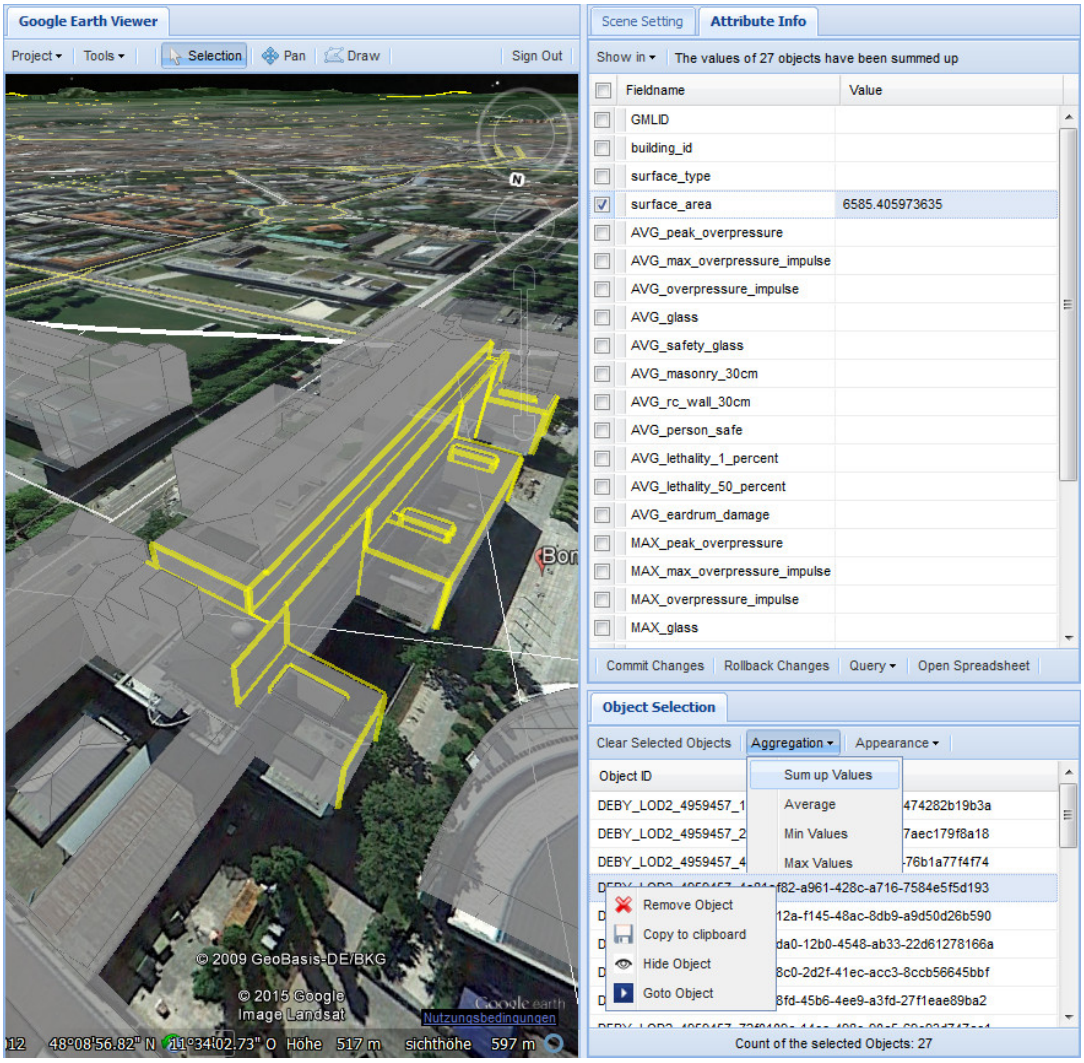


Figure 4.5: Selection results and aggregation of surface area.

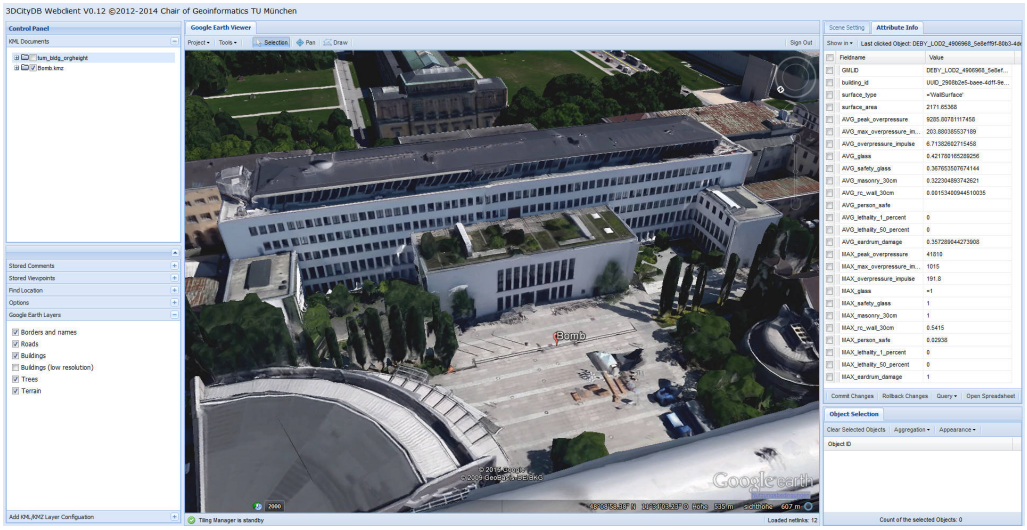


Figure 4.6: The TUM main building in its GoogleEarth 3D building representation.

4.1.3 Comparison of simulation in free field and with buildings

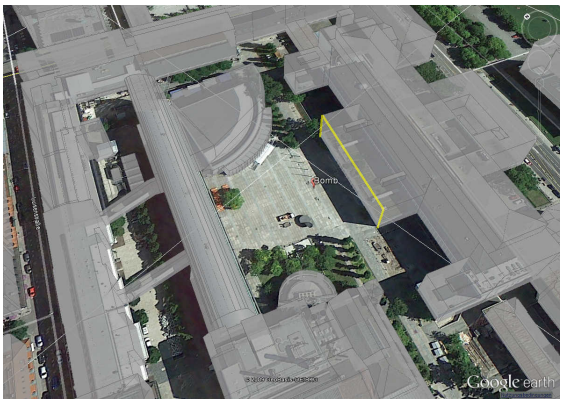
The Apollo Blast Plugin supports two different calculation strategies. The free field approach computes blast effects only depending on the TNT mass and the distance to the explosion source, as discussed in Section 2.5.1. Instead, the full approach performs a CFD simulation including the built environment. For both approaches a 3D Web Client project has been set up to compare their results. The parameter selected for the comparison is the maximum peak overpressure (P_{max}). Querying of the surfaces with the 3D Web Client for the different pressure values was performed like described in Section 4.1.2. Figures 4.7 and 4.8 depict the highlighted surfaces at six pressure levels between 120 000 Pa and 5000 Pa for both calculation approaches. The CFD simulation is shown in the left, the free field approach in the right column of the figures. For each step downwards P_{max} is halved.

Starting at the explosion site, where the highest overpressure occurs, both methods deliver about the same results. Only the surface right next to the detonation exceeds 120 000 Pa. Though, when looking at the exact values for the highlighted surface in Figures 4.7(a) and 4.7(b) we learn that the full simulation (210 200 Pa) returns a significantly higher P_{max} than the engineering method (143 025 Pa).

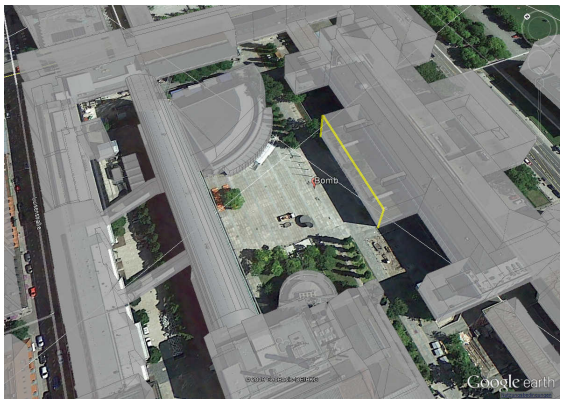
Overpressure of more than 60 000 Pa appear, as Figures 4.7(c) and 4.7(d) show, only on facades in the close perimeter of the explosion. When comparing both images we can see the shadowing effect of solid structures in the full simulation. While for the engineering method the two blocks on the roof right to the bomb exceeded $P_{max} = 60\,000$ Pa, they don't for the CFD simulation. The amount of the shadowing effect is huge. Following the images with decreasing P_{max} this effect is observable down to $P_{max} > 20\,000$ Pa. The exact figures for the top surfaces of the blocks are about 64 000 Pa for the engineering method and about 19 000 Pa for the CFD simulation.

When looking at the images on Figure 4.8 the shadowing effect becomes even more clear. The extent of the blast wave never reaches out of the campus area, even for low overpressure of lower than 10 000 Pa for the CFD simulation. Instead, the effect radius of the engineering method grows circular around the explosion source with lowering P_{max} to over 200 meters at $P_{max} < 10\,000$ Pa.

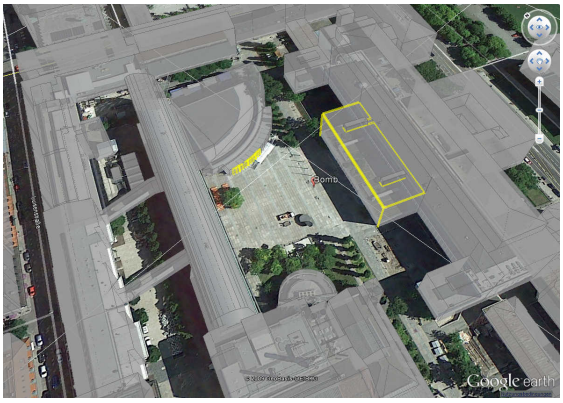
Concluding we can say that the engineering method delivers acceptable results, as long as the path of the blast wave is not obstructed by solid structures. As the images on Figure 4.7 indicate, for such a case the results are similar to the CFD approach. If the blast wave is deflected, the engineering method becomes unusable as Figure 4.8 illustrates. Following Klomfass and Zweigle, 2013, it can only serve to identify possibly critical scenarios, which have to be further investigated using a CFD simulation.



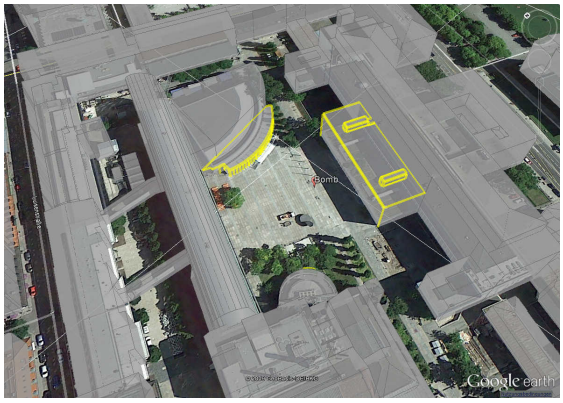
(a) with buildings, $P_{max} > 120\,000\,\mathrm{Pa}$



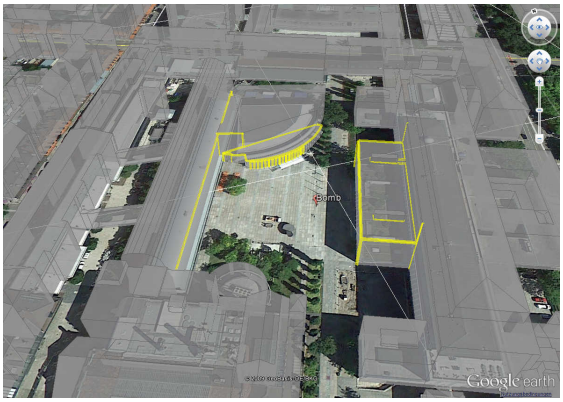
(b) free field, $P_{max} > 120\,000\,\mathrm{Pa}$



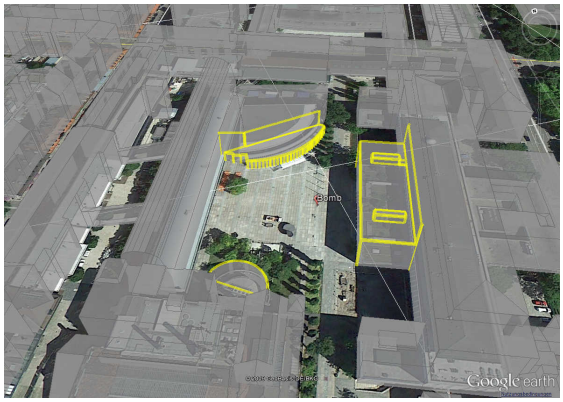
(c) with buildings, $P_{max} > 60\,000\,\mathrm{Pa}$



(d) free field, $P_{max} > 60\,000\,\mathrm{Pa}$

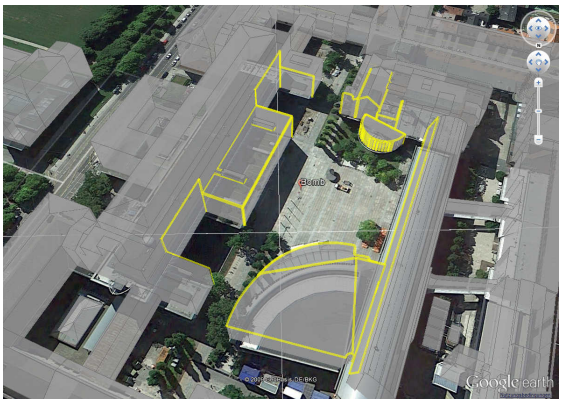


(e) with buildings, $P_{max} > 40\,000\,\mathrm{Pa}$

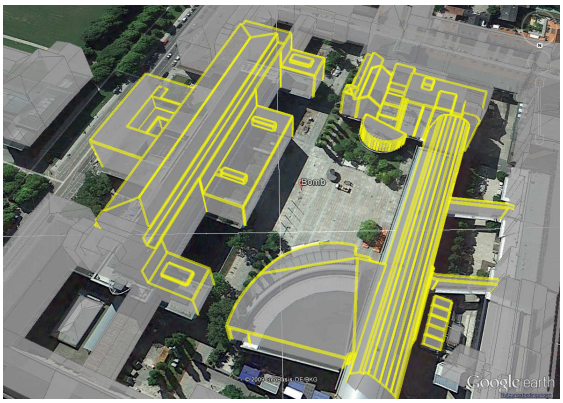


(f) free field, $P_{max} > 40\,000\,\mathrm{Pa}$

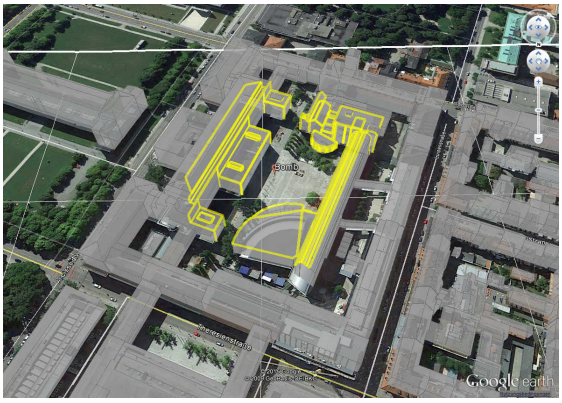
Figure 4.7: Distribution of the peak overpressure (P_{max}) compared for simulation in free field and with buildings. (Part 1)



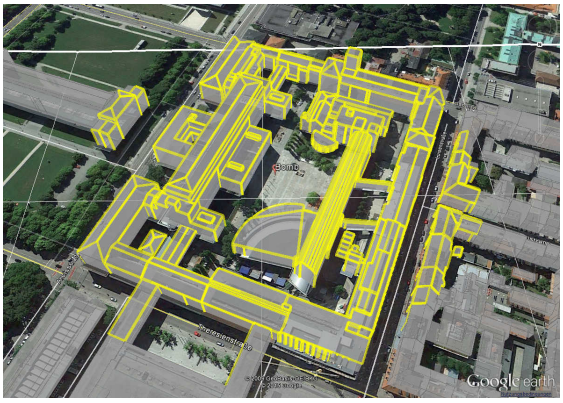
(a) with buildings, $P_{max} > 20\,000\,\mathrm{Pa}$



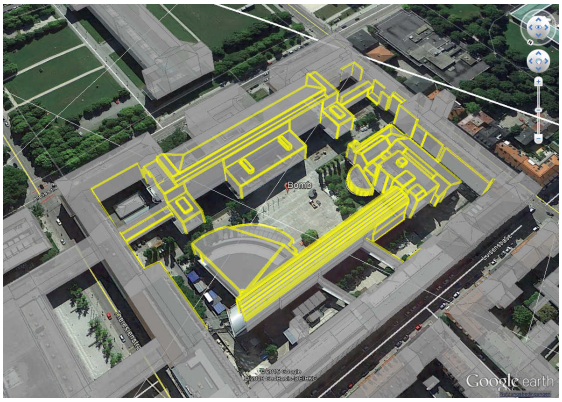
(b) free field, $P_{max} > 20\,000\,\mathrm{Pa}$



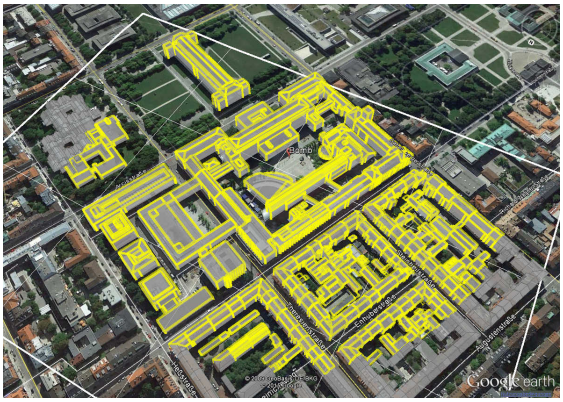
(c) with buildings, $P_{max} > 10\,000\,\mathrm{Pa}$



(d) free field, $P_{max} > 10\,000\,\mathrm{Pa}$



(e) with buildings, $P_{max} > 5000\,\mathrm{Pa}$



(f) free field, $P_{max} > 5000\,\mathrm{Pa}$

Figure 4.8: Distribution of the peak overpressure (P_{max}) compared for simulation in free field and with buildings. (Part 2)

4.2 Influence of the mesh resolution on simulation results

The resolution of the voxel mesh created by the Apollo Blast Plugin can be influenced by three parameters as discussed in Section 2.5.4. The *Zone length* (L_{zone}) defines the edge length of the initial zoning of the calculation domain. Its value is prescribed by the size of the calculation domain according to Equation (2.4). The $MaxLevel$ describes the highest possible refinement of zones by the Apollo Blastsimulator during a simulation run (see Section 2.5.3) and allows us to compute the *Voxel length* (L_{voxel}) with Equation (2.6). The Apollo Blast Plugin always produces a mesh based on L_{voxel}, because solids are embedded on the finest mesh level by the Apollo Blastsimulator.

To evaluate the influence of the mesh resolution on the simulation results, three voxel grids for $MaxLevel = 3$, 4, 5 and $L_{zone} = 6.5\,\mathrm{m}$ were created in this study. An overview on the different levels is given in Table 4.2. The quality levels were compared by the *Glass Breakage Probability*. For the visualization of this parameter the VTK result files of the Apollo Blastsimulator were displayed with ParaView[1]. Additionally a 3D Web Client project was set up for each quality level to control the mapping of the simulation results to CityGML.

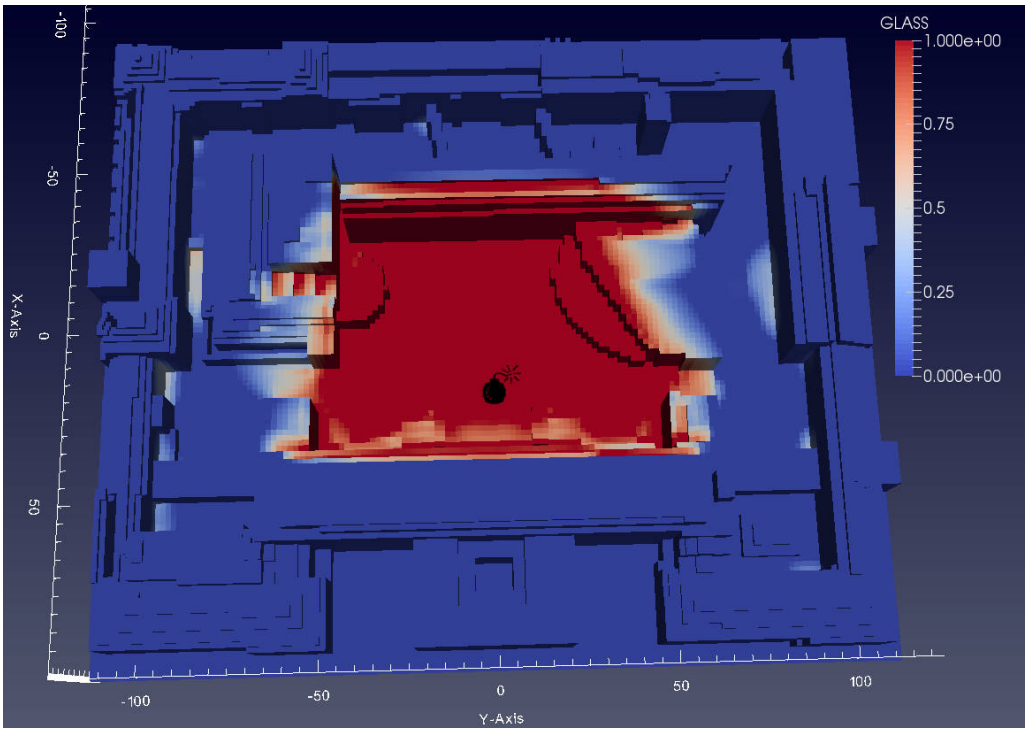
Images from the ParaView representation are given in Figures 4.9(a), 4.10(a) and 4.11(a) starting with the lowest resolution. Blue stands for 0 % glass breakage probability, red for 100 %. The increasing mesh resolution can been observed best at the circular shaped building top right of the bomb. When looking at the glass breakage probability on the three images we can see that for the campus area, where the blast wave is not obstructed by solids, all mesh resolutions deliver similar results. Where buildings are hit by the wave slight differences between the mesh levels can be noted. Generally it seems that higher grid resolutions lead to a higher probability. This is well observable when comparing the building located top left of the bomb ($X \approx 54, Y \approx -52$) for the three stages. As theses differences occur right at the border of the area of maximum blast effects and are of small extent, they don't influence the results a lot. Therefore we can state that for the close perimeter of the detonation, where the blast effects reach their peak, higher mesh resolutions don't improve the results significantly.

When moving away from the core of the explosion to the border of the building complex we can observe several spots with locally high probabilities. They are caused by focusing effects of the blast wave due to the spatial arrangement of the buildings. The number and probability values for theses spots significantly increasing with raising mesh quality. On $MaxLevel = 3$ only a few spots, mainly to the far left and right on the image, with low to medium probabilities can be monitored. For level 4 and 5 theses spots show strongly increased probabilities. Moreover in the corners of the campus and at the top middle new spots with medium to high probabilities appear. This is where the high resolution CFD simulation has its biggest advantages. The predictions obtained for these zones are not available when working with low resolutions or a simplified calculation approach like the engineering method.

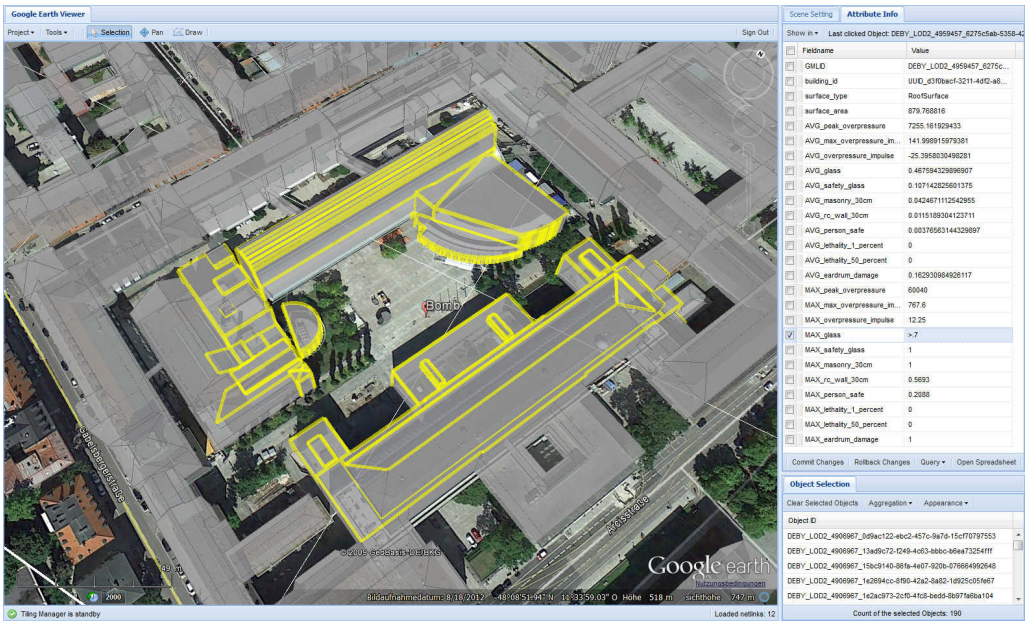
Figures 4.9(b), 4.10(b) and 4.11(b) show the 3D Web Client projects for each of the

1 http://www.paraview.org/

quality levels. For all three cases surfaces with a peak glass breakage probability of greater than 70 % have been queried. Compared with the corresponding VTK visualizations we can see that both results match quite well. Hence, the mapping of the Apollo Blastsimulator simulation results to CityGML seems to be working. Regarding the influence of the mesh resolution the same effects as described above can be monitored. While for $MaxLevel = 3$ only surface close to the explosion source are highlighted, for level 4 and 5 a growing number of areas at the edge is marked. When comparing the 3D Web Client visualizations for mesh level 3 and 5 on Figures 4.9(b) and 4.11(b) the increased prediction quality of the high resolution simulation run becomes most evident. By manual selection and aggregation in the 3D Web Client an additional wall surface area endangered from glass breakage of over 7000 m^2 was computed, that would not have been predicted at level 3. An illustration to this is given in Figure 4.12.

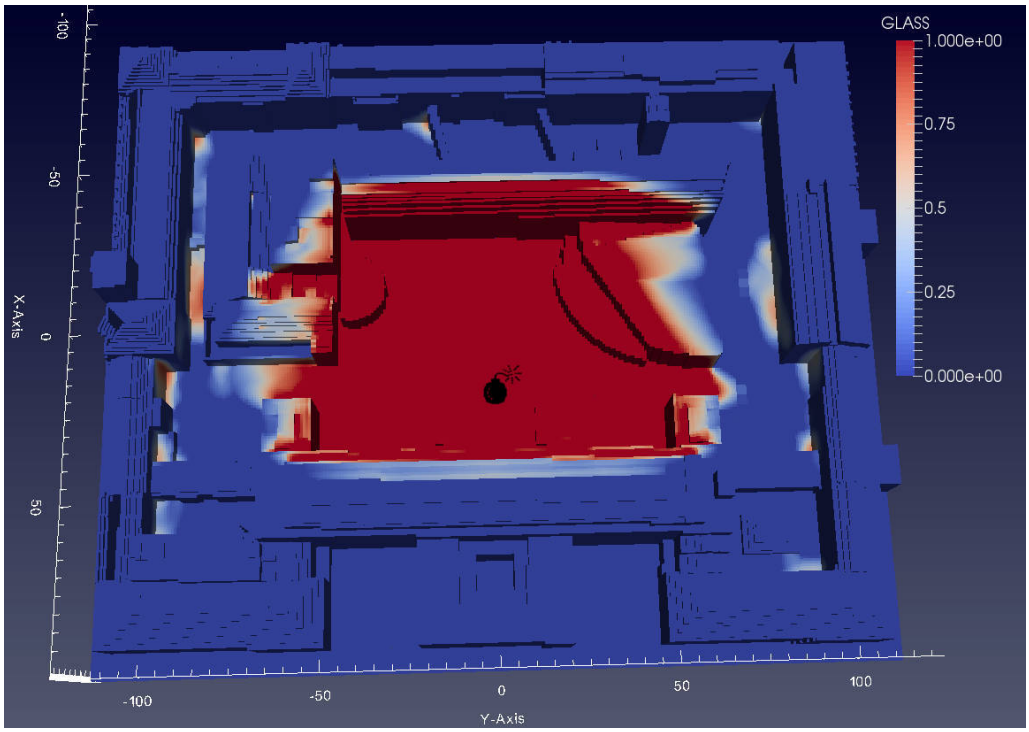


(a) ParaView VTK file: Glass breakage probability

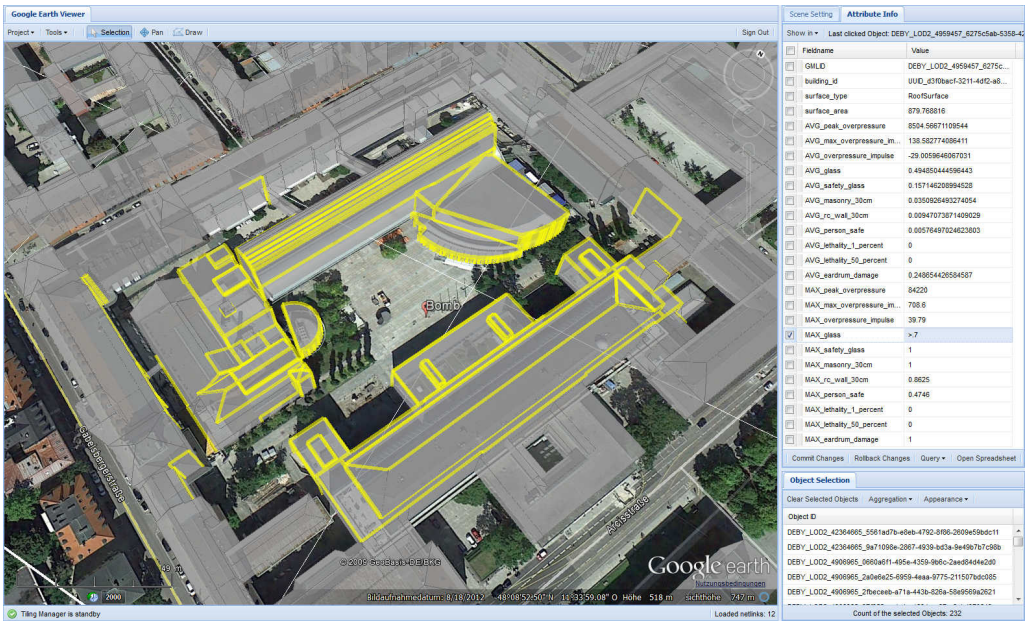


(b) 3D Web Client: Peak glass breakage probability $> 70\,\%$

Figure 4.9: Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 3$, $L_{voxel} = 1.625\,\mathrm{m}$.

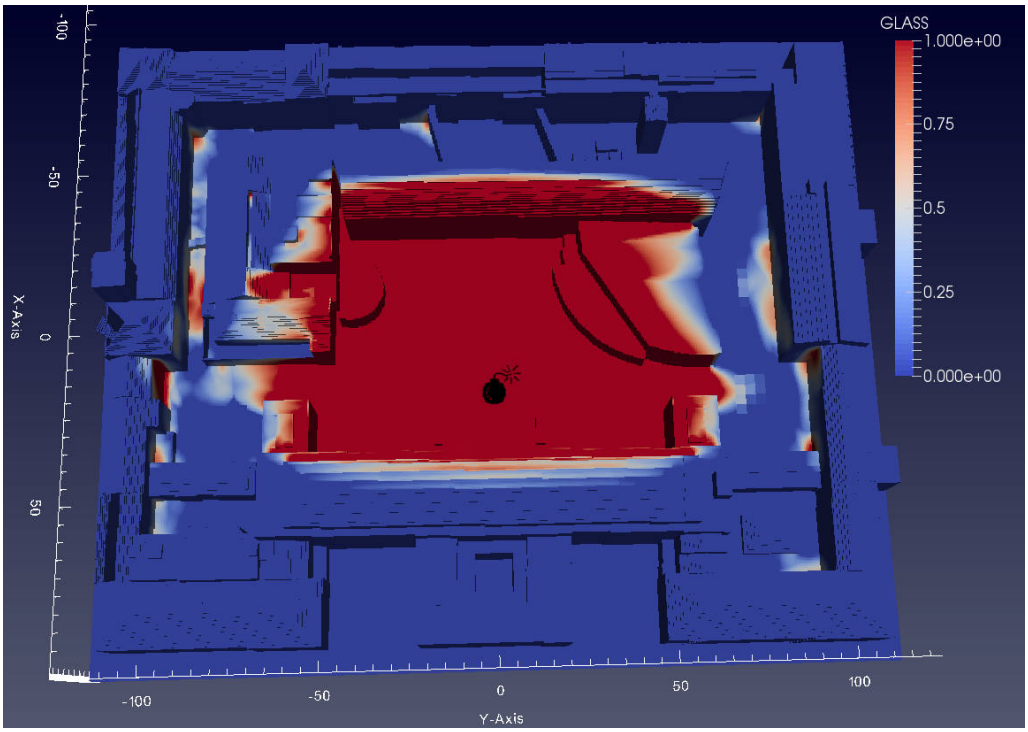


(a) ParaView VTK file: Glass breakage probability

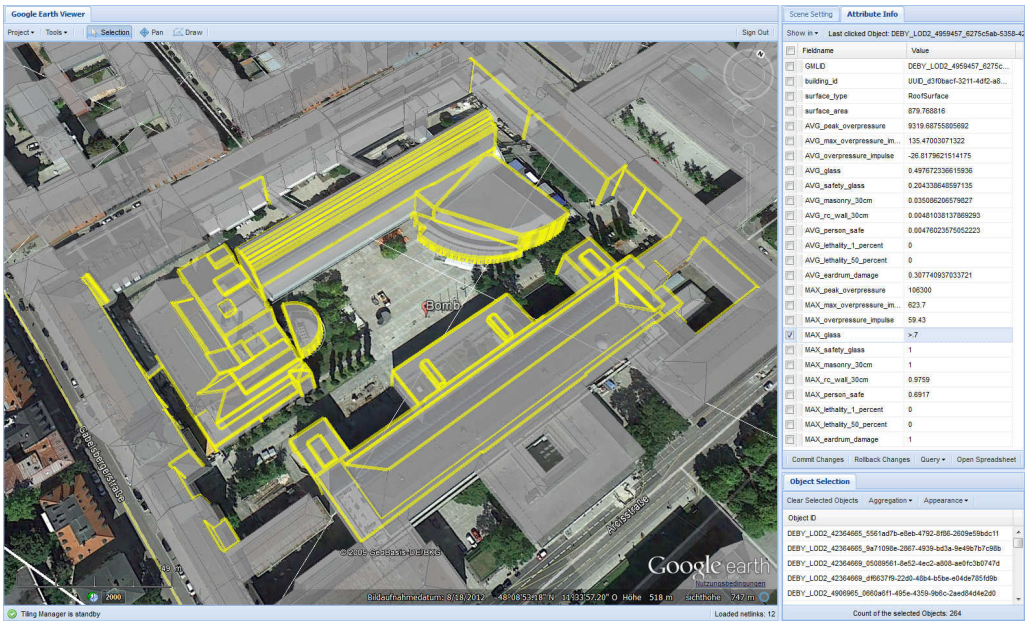


(b) 3D Web Client: Peak glass breakage probability > 70 %

Figure 4.10: Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 4$, $L_{voxel} = 0.8125\,\mathrm{m}$.



(a) ParaView VTK file: Glass breakage probability



(b) 3D Web Client: Peak glass breakage probability $> 70\,\%$

Figure 4.11: Comparison of peak glass breakage probability for a voxel mesh with $L_{zone} = 6.5\,\mathrm{m}$, $MaxLevel = 5$, $L_{voxel} = 0.406\,25\,\mathrm{m}$.

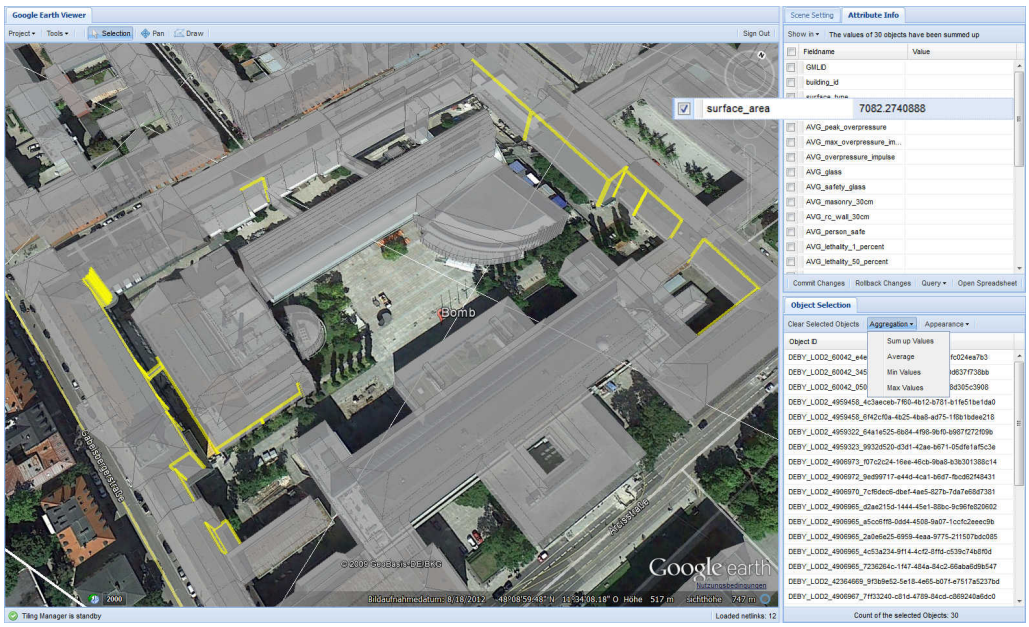


Figure 4.12: Additional surfaces endangered of glass breakage from $MaxLevel = 3$ to $MaxLevel = 5$ voxel mesh.

4.3 Performance

The two relevant operations regarding performance are the mesh generation and the simulation run with the Apollo Blastsimulator. The computational effort for both processes is defined by the mesh resolution and the size of the computational domain. As described in Section 2.5.3 the parameter determining the mesh resolution is the maximum mesh refinement level ($MaxLevel$). By changing it, the user can strongly influence both the system performance and the mesh quality as discussed before.

To evaluate the performance of the system, four simulation runs at different quality levels were performed with the test area at TUM. The mesh generation was conducted on the system described below, the CFD simulation with the Apollo Blastsimulator was performed at EMI on an unknown system.

- Intel Core i5-4200M 2.5GHz, 2 Cores, 4 Threads
- 4GB RAM
- SSD-HDD
- Windows 7 x64, PostgreSQL 9.3 x64, PostGIS 2.1.5

The size of the test area is $\approx 120\,\mathrm{m} \times 120\,\mathrm{m}$ with a height of 120 m, the zone length (L_{zone}) was 6.5 m for all tests. The resulting voxel length, number of voxels in the model, number of voxels theoretically possible for the whole computational domain and the corresponding duration are listed in Table 4.2.

As expected, both the time consumption for mesh generation and simulation with the Apollo Blastsimulator highly increases with raising mesh quality. Compared to the duration of the blast simulation the time consumption of the mesh generation is moderate. When looking at the influence of the mesh quality on the simulation results as described in the preceding section, we can state that a scenario of the given size can be processed with acceptable accuracy ($MaxLevel = 4$) for a broad range of applications in about three hours. If only the close perimeter of a blast is relevant or inaccuracy regarding shadowing and focusing effects of the blast wave is acceptable for the scenario, results can be delivered within minutes for the given setup.

Table 4.2: Voxel meshes with $L_{zone} = 6.5\,\mathrm{m}$ for the TUM test area.

$MaxLevel$ [–]	L_{voxel} [m]	Number of voxels in model [–]	Number of voxel in domain [–]	Duration mesh generation	Duration Apollo Blast-simulator
2	3.25	12 606	219 488	0.2 min	1.0 min
3	1.625	52 849	1 755 904	1.5 min	13.0 min
4	0.8125	215 425	14 047 232	11.2 min	146.2 min
5	0.40625	867 449	112 377 856	87.8 min	29.8 h

4.3.1 Multi-threaded 3D intersection performance

The influence of the multi-threading approach (see Section 3.3.3) for the mesh generation performance was measured along with the tests described before. Each quality level ($MaxLevel$) was computed four times with a number of `Workers` from one to four, where each `Wroker` represents an I/O channel to the database. The `queue size` of the `WorkerPool` was set to 8 for all tests.

Figures 4.13 and 4.14 illustrate the duration for the different setups. For all quality levels a performance gain of a factor of ≈ 1.7 between one and two `Workers` can be observed, which matches the number of available cores on the test system. More than two `Workers` do not bring a relevant improvement, probably because no additional CPU power is available.

Due to time constraints at the end of the project no further testing was done here. On the first look it seems like the multi-threading is working, though this needs more testing and evaluation.

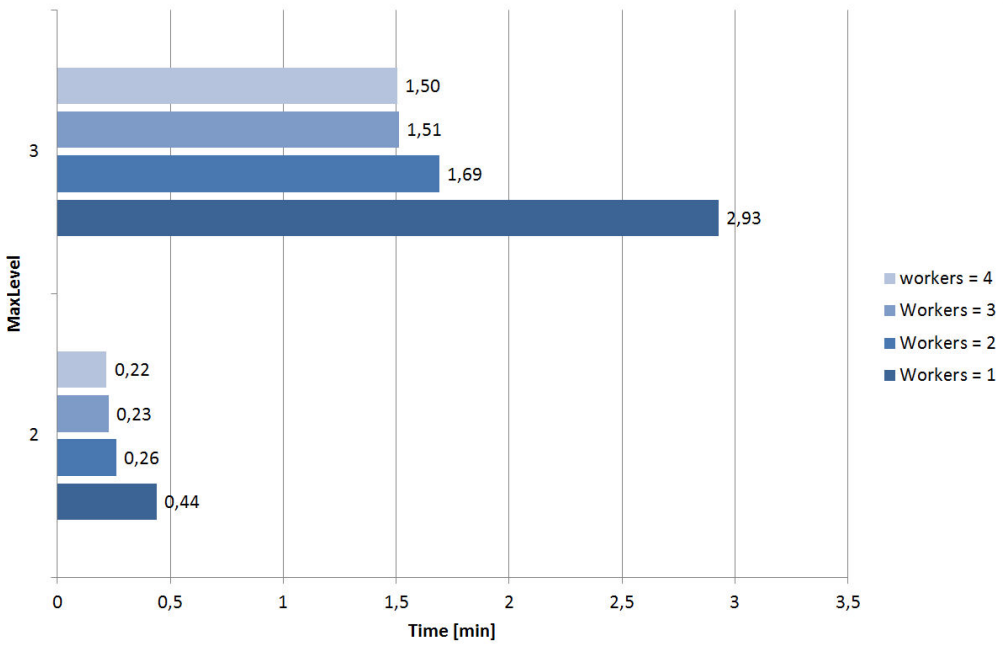


Figure 4.13: Time consumption of mesh generation for one to four workers, $MaxLevel = 2, 3$.

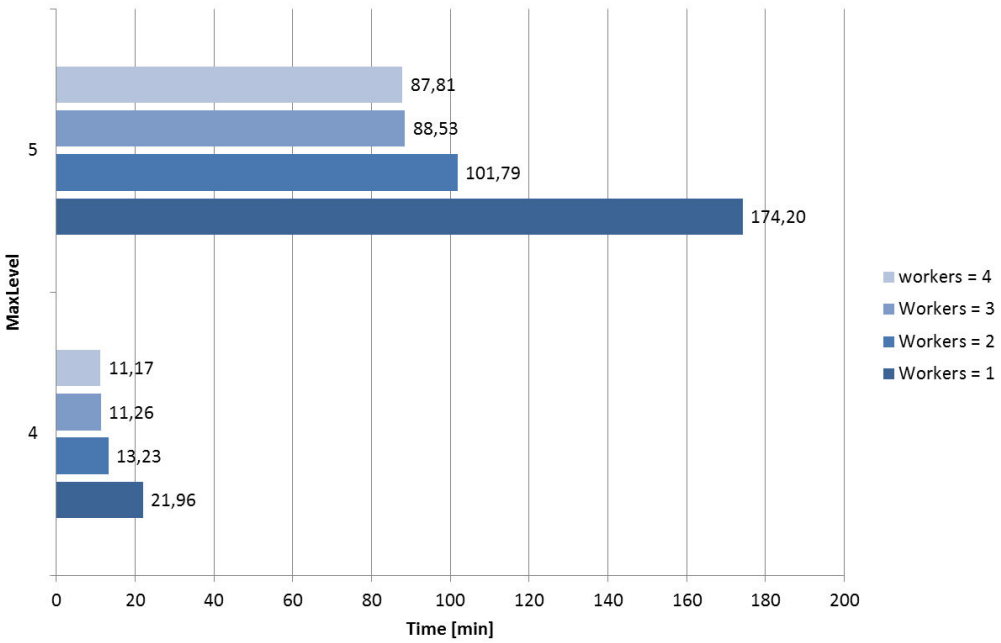


Figure 4.14: Time consumption of mesh generation for one to four workers, $MaxLevel = 4, 5$.

CHAPTER 5

Final considerations and outlook

5.1 Conclusion

According to Li et al., 2007, in the past 30 year both Geographical Information Systems (GIS) and Computational Fluid Dynamics (CFD) simulations have evolved to essential tools in planing, decision making and risk assessment. Yet, both have weaknesses caused by strong specialization to their specific domain. Today's GIS are highly developed in handling spatial information, but they mostly lack the representation of time in their models. Therefore they cannot map dynamic processes. CFD simulations on the other hand are specialized to the modeling and simulation of dynamic events. However, the mechanical models they work with are usually not available and have to be created for each scenario one wants to evaluate. Virtual 3D city models pose an attractive data source for that as they are increasingly available. The main purpose of this study was to combine both branches and make them benefit from each other on the example of CityGML city models and blast simulations performed by the Apollo Blastsimulator.

The preconditions to this project were full conformity to CityGML and to keep the project open source. The main objective was to develop a tool enabling users with expert knowledge in neither GIS or CFD to perform and evaluate blast simulations based on CityGML city models and the Apollo Blastsimulator.

Four main challenges had to be resolved to achieve this. First, a workflow needed to be created allowing to transform data from CityGML to the mechanical model of the simulation tool, perform the simulation, re-import the results back to the city model and provide a platform for their visualization and evaluation. The answer was the 3DCityDB as data backbone for CityGML and the tool chain that comes with it. The key element of this workflow, the main outcome of this study, is the *Apollo Blast Plugin*, an extension to the 3D City Database Importer/Exporter (ImporterExporter) tool. Its main purpose is to control the whole process based input the user provides over an uncomplicated GUI.

The second challenge was the data conversion from CityGML to the mechanical model of the Apollo Blastsimulator. Controlled by the plugin a solution was found using PostGIS, the spatial extension for the PostgreSQL DBMS the 3DCityDB operates on. With the spatial data types and operators of PostGIS the so called voxel model for the Apollo Blastsimulator could be described and derived from CityGML geometries. The developed

approach features variable settings for controlling mesh quality and computational effort. To guarantee decent performance and scalability multi-threading was applied.

Storing the results of a simulation run conform to CityGML allowing their persistent storage in the city model and enabling their further use with tools tailored for the standard was the third challenge of this study. This was realized by extending CityGML with generic attributes containing the blast properties for every building surface of the city model.

Finally the simulation results needed to be distributed, visualized and analyzed. Herby three tools from the aforementioned CityGML tool chain came to use, facilitating a cloud-based approach for visualization and collaborative assessment of the provided data on a blast event. Data export and sharing was performed with two plugins of the ImporterExporter and common cloud services. The 3D visualization model was exported with the KML/COLLADA plugin and stored in Dropbox, thematic attributes to the blast event were exported with the Spreadsheet Generator plugin and stored in a GoogleSpreadsheet. For visualization the cloud-based 3D Web Client was used. Based on the two exported files the city model is visualized in 3D with the GoogleEarth plugin in a standard web browser. With the thematic information attached in the spreadsheet the blast properties can be queried and evaluated based on attribute values in an easy-to-use environment, without extensive knowledge on the preceding GIS and CFD processes. Project management capabilities of the client and access control by the sharing settings of the cloud services facilitate collaborative work.

In summary I can say that all research questions of this study could be answered, as described above. With the *Apollo Blast Plugin* for the ImporterExporter an easy-to-use tool for blast analysis with CityGML city models and the Apollo Blastsimulator was designed and implemented. However, the current state of the implementation is prototypic. There are many aspects that can be optimized or added. Some of them are described in the following section.

5.2 Outlook

When it comes to software projects there is one simple truth. They are *never* finished! Obviously this project can not be excluded from this. There are many things regarding both concepts and implementation that can be improved or added.

Big potential for upgrading lies in the usability of the plugin. The selection of the simulation area and explosion location is at this point rather unhandy. Currently the coordinates for the explosion source have to be picked and transformed to the SRS of the database manually. The BBox is computed based on the explosion location and the blast load and can therefore not be fitted to the characteristics of built environment. Moreover, currently it's not possible to rotate the selected area in the plugin. As this can strongly improve the approximation of structures by the voxel mesh it should be added as well. To solve all theses issues an interactive scenario selection panel is planned. It's based on the current implementation of the BBox selection panel of the 3DCityDB ImporterExporter, which is shown in Figure 5.1. The selection of the simulation domain and explosion location will be allowed by drawing on an interactive map as the purple box on the image indicates. Rotation of the area will be implemented with this as well.

An important feature that is currently missing is the storage and visualization of

simulation results on the ground. A quick solution could be the generation of raster images out of the result data sets. After georeferencing the images they could be loaded as additional KML layers in the 3D Web Client. Though, this is only a temporary solution. An approach for storing the simulation results on the ground based on the CityGML thematic class `Relief` needs to be developed.

Another topic of the future development is the extension of the model for higher LoD levels than the currently supported LoD2. With *Openings* in buildings introduced in LoD3 and *Rooms* supported in LoD4, the effects of blast waves on the inside of buildings could be modeled.

Big improvement can also be made regarding performance. The algorithm for the creation and intersections of voxels with the city model gives lots of room for optimization. At this time the whole computational domain is tested for intersection with the city model causing a large number of tests, that are not necessary. Currently an approach is being developed that reduces the intersection tests to the 3D BBoxes of buildings. Further performance improvements could be realized by implementing an *Octree*[1] for the remaining comparisons.

Moreover, in the future improved intersection tests will become available providing full support for 3D solids. PostGIS already provides an unreleased version[2] based on the SFCGAL spatial library. According to the official homepage[3] the library adds full support for ISO 19107:2013 and OGC Simple Features Access 1.2 3D operations. Porting the current implementation to this version of PostGIS should be possible without major modifications and is aready in planing.

1 http://de.wikipedia.org/wiki/Octree
2 http://postgis.net/windows_downloads
3 http://oslandia.github.io/SFCGAL/

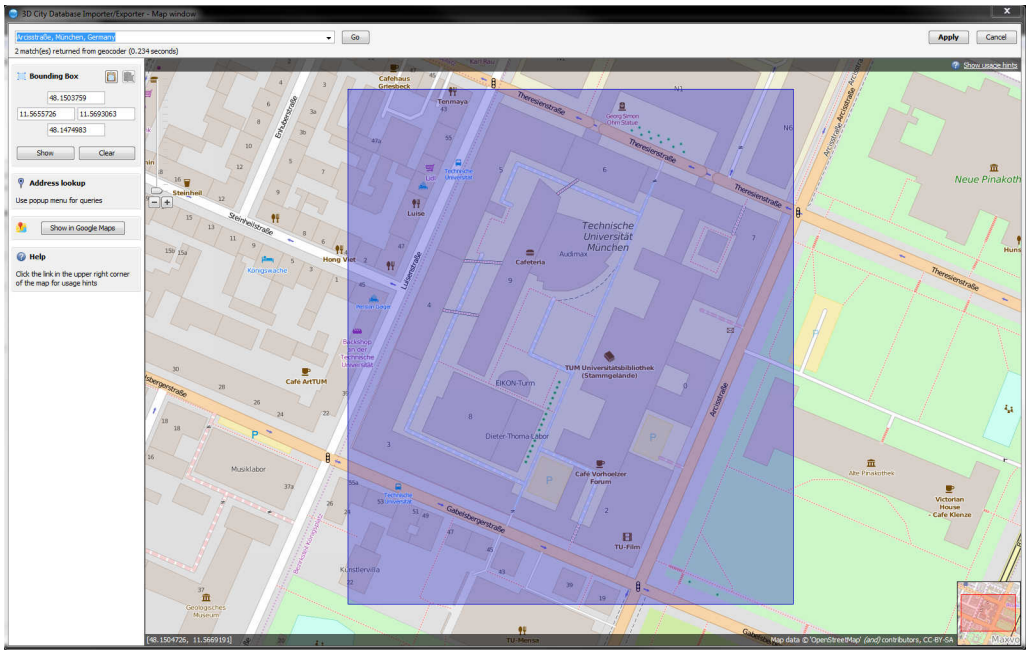


Figure 5.1: BBox selection dialog of the 3DCityDB ImporterExporter.

Bibliography

Alam, N., D. Wagner, M. Wewetzer, J. von Falkenhausen, V. Coors, and M. Pries (2013): 'Towards automatic validation and healing of CityGML models for geometric and semantic consistency'. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-2/W1: pp. 1–6. ISSN: 2194-9050 (cit. on p. 80).

Bayrisches Landesamt für Digitalisierung, Breitband und Vermessung (2015): *LOD2 Arbeitsstand*. URL: http://www.geodaten.bayern.de/download/uebersicht_LOD/LoD2_InternetUebersicht.pdf (visited on 01/23/2015) (cit. on pp. 2, 79).

Burgherr, P. (2015): *Unfallrisiken von Erdgas: eine umfassende Analyse*. Ed. by Paul Scherrer Institut. CH-5232 Villigen. URL: http://www.gat-dvgw.de/fileadmin/gat/PDF_Vortragsarchiv_05/burgherr_m05_01.pdf (visited on 01/23/2015) (cit. on p. 1).

C. Nagel Kaden, R. (2011): 'Importer/Exporter - Plugin API'. Unpublished slides for internal discussion (cit. on pp. 21, 22, 65).

Chen, A.S., B. Evans, S. Djordjevic, and D. A. Savic (2012): 'Multi-layered coarse grid modelling in 2D urban flood simulations'. *Journal of Hydology*, vol. 470: pp. 1–11 (cit. on p. 1).

Corti, P. (2014): *PostGIS 2.0 Cookbook*. Packt Publishing. ISBN: 9781849518666 (cit. on p. 17).

Görl, Wolfgang (2013): *Was auf den Knall folgte*. Ed. by Sueddeutsche.de. URL: http://www.sueddeutsche.de/muenchen/fliegerbombe-in-schwabing-was-auf-den-knall-folgte-1.1754600 (visited on 06/01/2014) (cit. on p. 1).

Herring, J. (2001): *OGC Abstract Specifications Topic 1—Feature Geometry (ISO 19107 Spatial Schema)*. OGC Document Number 01-101 (cit. on p. 9).

Klomfass, A. and O. Herzog (2010): 'A Reliable, Fast and Easy-to-use Simulation Tool for Blast Waves in Urban Scenarios'. Ed. by Future Security. Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute. Freiburg, Germany (cit. on pp. 30–33).

Klomfass, A. and K. Thoma (2012): 'CFD and Holistic Methods for Explosive Safety and Risk Analysis'. Ed. by Verband der deutschen Chemischen Industrie (cit. on p. 30).

Klomfass, A. and T. Zweigle (2013): 'A New AMR/ALE-Solver for the efficient Simulation of Detonations and Blast Waves'. Ed. by Intern. Symposium of Interaction of Effects of Munitions with Structures. Fraunhofer Institute for High-Speed Dynamics, Ernst-Mach-Institute. Freiburg, Germany (cit. on pp. 1, 30–32, 86).

Kolbe, T. H. (2009): 'Representing and Exchanging 3D City Models with CityGML'. *3D Geo-Information Sciences*. Ed. by Lee, Jiyeong and Sisi Zlatanova. Lecture notes in geoinformation and cartography. Berlin, Heidelberg: Springer: pp. 15–31. isbn: 978-3-540-87394-5 (cit. on pp. 5, 6, 8, 9).

Kolbe, T. H., G. Koenig, and A. Nagel C.and Stadler (2009): *3D-Geo-Database for CityGML*. 2.0.1. Institute for Geodesy and Geoinformation Science, Technical University Berlin. url: http://www.3dcitydb.net/3dcitydb/fileadmin/downloaddata/3DCityDB-Documentation-v2_0.pdf (visited on 01/27/2015) (cit. on pp. 11–13, 21, 130, 131).

Kolbe, T. H., Z. Yao, J. Herreruela, C. Nagel, F. Kunde, P. Willkomm, and G. Hudra (2013): *Addendum to the 3D City Database Documentation Version 2.0.1*. Institute for Geodesy and Geoinformation Science, Technical University Berlin. url: http://www.3dcitydb.net/3dcitydb/fileadmin/downloaddata/3DCityDB-Documentation-Addendum-v2_1.pdf (visited on 01/27/2015) (cit. on pp. 21, 22, 24, 25, 74).

Li, J., Andrea G. Fabbri, and S. Zlatanova (2007): *Geomatics solutions for disaster management*. Lecture notes in geoinformation and cartography. New York: Springer. isbn: 978-3-540-72106-2 (cit. on pp. 2, 4, 99).

Naderi, B., J. Herreruela, and T. H. Kolbe (2012): *Spreadsheet Generator Version 1 .0.0 Plugin for Importer/Exporter Version 1.4.0*. Institute for Geodesy and Geoinformation Science, Technical University Berlin (cit. on pp. 24, 27, 29, 73).

Obe, R. O. and L. S. Hsu (2011): *PostGIS in action*. Greenwich, Conn. and London: Manning and Pearson Education. isbn: 9781935182269 (cit. on p. 17).

Open Geospatial Consortium (2012): *OGC City Geography Markup Language (CityGML) Encoding Standard*. Ed. by Groeger, G., T. H. Kolbe, C. Nagel, and K. H. Haefele. 2.0.0. OGC 12-019. url: http://www.opengis.net/spec/citygml/2.0 (visited on 01/27/2015) (cit. on pp. 5, 6, 8–10, 126, 127).

– (2011): *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. Ed. by Herring, J. R. 1.2.1. OGC 06-103r4. url: http://www.opengeospatial.org/standards/sfa (visited on 01/27/2015) (cit. on pp. 15–18).

– (2010): *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*. Ed. by Herring, J. R. 1.2.1. OGC 06-104r4. url: http://www.opengeospatial.org/standards/sfs (visited on 01/27/2015) (cit. on pp. 15–17).

PostGIS Project Steering Committee (2015): *PostGIS 2.2.0dev Manual*. 2.2.0dev. url: http://postgis.net/documentation (visited on 01/27/2015) (cit. on pp. 15, 17–20).

Stadler, A., C. Nagel, G. König, and T. H. Kolbe (2009): 'Making Interoperability Persistent: A 3D Geo Database Based on CityGML'. *3D Geo-Information Sciences*. Ed. by Lee, Jiyeong and Sisi Zlatanova. Lecture notes in geoinformation and cartography. Berlin, Heidelberg: Springer: pp. 175–192. isbn: 978-3-540-87394-5 (cit. on pp. 11, 13).

Sun Microsystems, Inc. (2006): *The Java™ Architecture for XML Binding (JAXB) 2.0*. Ed. by Vajjhala, Sekhar and Jeo Fialli. Santa Clara, CA 95054 USA. url: https://jaxb.java.net/ (visited on 01/27/2015) (cit. on pp. 51, 52).

The PostgreSQL Global Development Group (2011): *Parallel Query Execution*. URL: https://wiki.postgresql.org/wiki/Parallel_Query_Execution (visited on 01/27/2015) (cit. on p. 15).

– (2015a): *PostgreSQL 9.3.6 Documentation*. URL: http://www.postgresql.org/docs/9.3/static/index.html (visited on 01/27/2015) (cit. on pp. 14, 15).

– (2015b): *PostgreSQL:About*. URL: http://www.postgresql.org/about/ (visited on 01/27/2015) (cit. on p. 14).

Trometer, S. and M. Mensinger (2014): 'Simulation von Detonationsszenarien im urbanen Umfeld'. *Geoinformationssysteme 2014, : Beiträge zur 1. Münchner GI-Runde*. Ed. by Kolbe, T. H., R. Bill, and A. Donaubauer. Berlin: Wichmann: pp. 150–164. ISBN: 9783879075379 (cit. on p. 4).

Wagner, D., N. Alam, and V. Coors (2013): 'Geometric validation of 3D city models based on standardized quality criteria'. *Urban and Regional Data Management*. Ed. by Laurini, Robert. CRC Press: pp. 197–210. ISBN: 978-1-138-00063-6 (cit. on p. 80).

Weiss, H. (2012): *"Den entschärfe ich nicht, ich bin doch nicht lebensmüde"*. Ed. by Welt.de. URL: http://goo.gl/hkkmFG (visited on 06/01/2014) (cit. on p. 1).

Wikipedia (2015a): *Blindgaenger — Wikipedia, The Free Encyclopedia*. URL: http://de.wikipedia.org/wiki/Blindg%C3%83%C2%A4nger%5C#Deutschland (visited on 01/23/2015) (cit. on p. 1).

– (2015b): *Unexploded Ordnance — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/wiki/Unexploded_ordnance (visited on 01/23/2015) (cit. on p. 1).

Yao, Z. and T. H. Kolbe (2014): 'Cloud-basierter Webclient für kollaboratives Arbeiten auf semantischen 3D-Stadtmodellen'. *Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V.* Vol. 23 / 2014. Potsdam (cit. on pp. 27, 29).

Yao, Z., M. Sindram, R. Kaden, and T. H. Kolbe (2014): 'Cloudbasierter 3D-Webclient zur kollaborativen Planung energetischer Maßnahmen am Beispiel von Berlin und London'. *Geoinformationssysteme 2014, : Beiträge zur 1. Münchner GI-Runde*. Ed. by Kolbe, T. H., R. Bill, and A. Donaubauer. Berlin: Wichmann: pp. 40–52. ISBN: 9783879075379 (cit. on p. 27).

APPENDIX A

EMI Concept paper

Erläuterungen zum Konzeptentwurf

Ziele und Vorgehensweise

- Ermöglichung der Berechnung und visuellen Darstellung von Gefährdungsbereichen innerhalb eines virtuellen Stadtmodells im CityGML Format
- Erweiterung eines bestehenden CityGML Viewers um eine Eingabemaske, Berechnungsablaufsteuerung und Visualisierungsoptionen

Erläuterungen zu den Berechnungsmodellen

Gefährdungsbereiche hängen ab von:

- Ort der Explosionsquelle, Explosionsstärke (z.B. Angabe kg TNT Äquivalent)
- Geometrie der Bebauung (äußere Gebäudeoberflächen, Bauweise spielt dabei keine Rolle; Gelände- und Straßenoberflächen)
- Ermittlung der Gefährdungsbereiche auf Basis von »What-If« Analysen
  - Welche Gefährdung besteht für Gebäudewände falls diese a) aus Mauerwerk, oder b) aus Beton bestehen, c) Glasfassaden oder Fenster besitzen (Unterscheidung jeweils in keine Gefährdung, leichte Beschädigung, Zerstörung/Einsturzgefahr)
  - Welche Gefährdung existiert für Personen im Freien innerhalb von bestimmten Bereichen (Unterscheidung nach keine, leichte, letale Gefährdung)

Für die Ermittlung der Gefährdungsbereiche stehen zwei verschiedene Berechnungsmodelle zur Verfügung:

- Berechnung ohne Berücksichtigung der tatsächlichen Bebauung (Ausführungsdauer der Berechnung: Wenige Sekunden)
- Berechnung unter Berücksichtigung des Geländes und der tatsächlichen Bebauung durch Einbeziehung der äußeren Gebäudeoberflächen und der Boden- bzw. Straßenoberflächen → Erfassung von Abschattungs- und Fokussierungseffekten der Druckwelle (Dauer der Berechnung: Minuten bis Stunden)

Beide Berechnungsmodelle sind in einem Berechnungsprogramm integriert, welches als Executable (unabhängig vom Viewer) vorliegt, über Inputdateien gesteuert wird und Outputdateien erzeugt.

Ziele des Konzeptentwurfs

- Darstellung der erforderlichen Erweiterungen des CityGML Viewers
- Beschreibung des Berechnungsablaufs, der erforderlichen Berechnungsablaufsteuerung und der Datenübertragung zwischen den Berechnungs- und Darstellungsformaten

Konzeptbeschreibung

1. Userinterface für Szenariodefinition und Berechnungssteuerung

Spezifikation der Explosionsquelle

1) Position der Explosionsquelle (Angabe via Straßenname und Hausnummer oder Koordinaten oder interaktiv per Maus)

2) Art der Explosionsquelle (Auswahlliste: z.B. Improvised Explosive Device, WWII Bombenfund)

3) Explosivstoffmenge (abhängig von der Auswahl unter 2): Angabe durch kg TNT oder Bombentyp)

4) In der Endversion würden noch weitere Details abgefragt

Auswahl der alternativen Berechnungsmethoden

1) Berechnung ohne Berücksichtigung der Bebauung (ohne weitere Optionen)

2) Berechnung für das bebaute Umfeld; mit Optionen für die Wahl der Berechnungsqualität:
- Schnell (a), typisch (b), genau (c)
- in der Endversion würden noch weitere Optionen abgefragt.

Die User-Eingabe zur Auswahl der Berechnungsmethode löst aus:
- Anzeige des geschätzten maximalen Gefährdungsradius
- Anzeige der geschätzten Dauer (min, h) der Berechnung

Die mathematische Methode zur Ermittlung des max. Gefährdungsradius und der Berechnungsdauer soll in den Viewer Code integriert werden (kurze Berechnungsformeln).

Start/Stop der Berechnung und Statusanzeige

- Start über einen Button. Dieser löst aus:
    - Erzeugen notwendiger Input Dateien für das Berechnungsprogramm
    - Ausführung des Executables des Berechnungsprogramms (via OS Kommando)
    - Nach Berechnungsende: Konvertierung des Outputs in das CityGML Format
- Button zum Abbruch der Berechnung
- Darstellung einer Fortschrittsanzeige (z. B. Anzeige Restlaufzeit oder Fortschrittsbalken),
- Anzeige eines Berechnungsstatus (Rot / Grün == Berechnung läuft / ist abgeschlossen)
- Status bzw. Restlaufzeit kann aus einer STATUS Datei ausgelesen werden, die vom Berechnungsprogramm angelegt und regelmäßig (alle paar Sekunden) aktualisiert wird.
- Nach Berechnungsende: Konvertierung des Ergebnisses in das CityGML Format durch Auslösen einer im Viewer integrierten Methode.

Speichern der Eingaben und Ergebnisse

- Abspeichern der User-Eingaben und der Berechnungsergebnisse zur späteren Wiedervorlage und Visualisierung ohne erforderliche erneute Berechnung

2. User-Interface für die Visualisierung der Ergebnisse

- Auswahl eines Berechnungsergebnisses im CityGML Format (=Ausschnitts eines Stadtmodells mit zusätzlichen generischen Attributen; aus aktueller Berechnung oder früher durchgeführten, abgespeicherten Berechnungen)
- Auswahl eines darzustellenden Attributes aus einer Liste, zB:
  1) Überdruckamplitude
  2) Überdruckimpuls
  3) Personengefährdung
  4) Gebäudeschäden
- Visuelle Darstellung der Gefährdungsbereiche durch Farbkodierung von äußeren Gebäudeoberflächen und Geländeoberflächen (Boden/Straßen). Anzeigen einer zugeordneten Legende.

  Farbkodierungen und Legenden
  - quasi-kontinuierliche Wertebereiche für 1) und 2)
  - Diskrete Wertebereiche für 3) 0=keine Gefährdung, X1=leichte Gefährdung, X2=signifikante Gefährdung, 1=Letale Gefährdung.
  - Diskrete Wertebereiche für 4) 0=keine Gefährdung, X1=Glasbruch, X2=Mauerwerk beschädigt, X3=Betonwände beschädigt, 1=Schwere Gebäudeschäden, Einsturzgefährdung.
- Anwendung von Visualisierungsfiltern (z.B. Aus/Einblendung von Objekten mit bestimmten Gefährdungsgraden bzw. Wertebereichen)

Dateiinhalte und Formate

Inputdatei für das Berechnungsprogramm (ASCII Datei)

- Explosionsort (XYZ)
- Explosionsstärke (kg TNT)
- Berechnungsmethode (1 oder 2a, 2b, 2c)
- Maximaler Gefährdungsradius (Rmax)
- Voxelmodell (erforderlich für Berechnungsmethode 2):
  - Referenzposition für das Voxelgitter (IJK=000 -> XYZ=XYZ000)
  - Räumliche Auflösung des Gitters DX=DY=DZ (ergibt sich aus Rmax in Abhängigkeit von der gewählten Genauigkeitsstufe 2a, 2b oder 2c)
  - Werteintervalle des Voxelmodells (Imin, Imax, Jmin, Jmax, Kmin, Kmax)
  - Liste der Voxel, welche von relevabten Oberflächen (Gebäudeaußenwände, Straßen, Gelände) geschnitten werden:

    I, J, K,Ident# (Identifikation der zugeordneten Oberfläche im CityGML Datensatz)

Erstellung des Voxelmodells

Vorgabe: Explosionsort XYZ, Maximaler Gefährdungsradius Rmax, Räumliche Auflösung DX

- Festlegung einer kartesischen Bounding Box in der horizontalen Ebene (XY Min: XY Max); die Bounding Box ergibt sich aus dem kreisförmigen Gebiet mit Radius Rmax um den Explosionsort; der Höhenbereich ist zunächst offen (Z = -infty : +infty)
- Loop über alle relevanten Oberflächenelemente des Stadtmodells, welche innerhalb der kartesischen Bounding Box liegen:
  - Falls erforderlich: Zerlegung des Oberflächenelements in Teilflächen mit charakteristischer Abmessung L < DX (unterhalb der Auflösung des Voxelmodells, stellt sicher, dass im Wesentlichen alle geschnittenen Voxel erkannt werden)
  - Mittelpunkt einer Teilfläche → liefert approximierte Voxelkoordinate (IJK) der Teilfläche
  - Abspeichern der Voxelkoordinaten IJK und der zugeordneten Oberflächen Ident#
  - Wenn ein Voxel von mehreren Oberflächen geschnitten wird, so genügt es meistens eine Oberflächen Ident# zu speichern
- Speichern aller Voxelkoordinaten und Ident#n in der Datei

Output (Berechnungsmethode 1)

Liste der verzeichneten Attribute und Bezeichnungen für die Legende

- Anzahl N der Attribute;
- Legendentitel für jedes Attribut

Tabellarische Auswertung der Attributwerte A_1 bis A_N für diskrete radiale Abstände vom Explosionsort

$R_{i,}$ A_1 (R_i) …. A_N (R_i) i=1, Imax; R ε [0:Rmax] /Typischerweise Imax = ca. 100 - 1000

Output (Berechnungsmethode 2)

Liste der verzeichneten Attribute und Bezeichnungen für die Legende

- Anzahl N der Attribute;
- Legendentitel für jedes Attribut

Zuordnung der Attributwerte zu den Voxeln des Voxelmodells

I, J, K, Ident#, A_1 … A_N

Konvertierung in das CityGML Format (Generierung einer Ergebnisdatei unter Verwendung generischer Attribute)

Es wird eine Kopie der Datei des Stadtmodells angelegt, welche

- einen Ausschnitt des gesamten Stadtmodells beinhaltet (zB im Bereich 2*Rmax)
- um generische Attribute zur Visualisierung der Gefährdungsbereiche erweitert wird, entsprechend den Attributen im Output File

Für Berechnungsmodell 1:
- Interpolation der Attributwerte aus den Werten der Output Datei
- Effektiver Abstand der Oberfläche vom Explosionsort
    - Mindestabstand der Oberfläche (via Eckpunkt mit geringstem Abstand)
    - Mittlerer Abstand der Oberfläche (via Mittelwert aller Eckpunkte)
    - Unterschiedliche Präferenz für verschiedene Attribute möglich

Für Berechnungsmodell 2:
- Attributwerte aus mehreren Voxeln mit gleicher Flächenzuordnung
    - Verwendung des auftretenden Maximalwertes
    - Verwendung eines Mittelwertes
    - Unterschiedliche Präferenz für verschiedene Attribute möglich

Aufgaben

Lehrstuhl für Geoinformatik

- Design und Integration der User Interfaces in die Viewer Software
- Programmierung und Integration der Methode zur Generierung der Inputfiles, insbesondere des Voxelmodells
- Integration der Ablaufsteuerung in die Viewer Software (Starten/Abbrechen des Berechnungsprogramms, Auslesen der Status Datei und Status Anzeige im Vierwer)
- Programmierung und Integration der Methode zur Übertragung der Ergebnisdateien der Berechnungsprogramme auf das CityGML Format via generischer Attribute

Fraunhofer EMI

- Ableitung und Bereitstellung der notwendigen Berechnungsformeln für max. Gefährdungsradius Rmax, Voxel Auflösung, Zeitschätzung der Berechnungsdauer
- Integration aller Berechnungsprogramme in ein einzelnes Executable;
- Modifikation der Berechnungsprogramme zur Verarbeitung der Inputformate und zur regelbasierten Spezifikation weiterer erforderlicher Inputparameter, die zur Steuerung der Berechnungen erforderlich sind
- Implementierung der Erzeugung des Statusfiles

APPENDIX B

EMI specifications for the voxel model

Voxelmodell: Dateiformat

```
1          VersionIdent                                                    (Int)
2          MaxLevel                                                        (Int)
3          RefLength                                                       (Float)
4          X0 Y0 Z0                                                        (3x Float)
5          IMin JMin KMin IMax JMax KMax                                   (6x Int)
6          NumZones                                                        (Int)
7          NumSurfs                                                        (Int)
8          NumParts                                                        (Int)
9          "Partname(1)" "Partname(2)" ..... "Partname(NumParts)"          (N x Character)
10         Format                                                          (Int)
11         RBImin RBImax RBJMin RBJMax RBKmin RBKmax                       (6x Int)
12         NumObjects                                                      (Int)
13         "Objname(1)" "Objname(2)" ..... "Objname(NumObjects)"           (N x Character)
14         ivox(1) jvox(1) kvox(1)  obj#(1)                                (4x Int)
.....
14         ivox(max) jvox(max) kvox(max)  obj#(max)                        (4x Int)
```

Erläuterungen auf den nachfolgenden Seiten

Erläuterungen zum Dateiformat

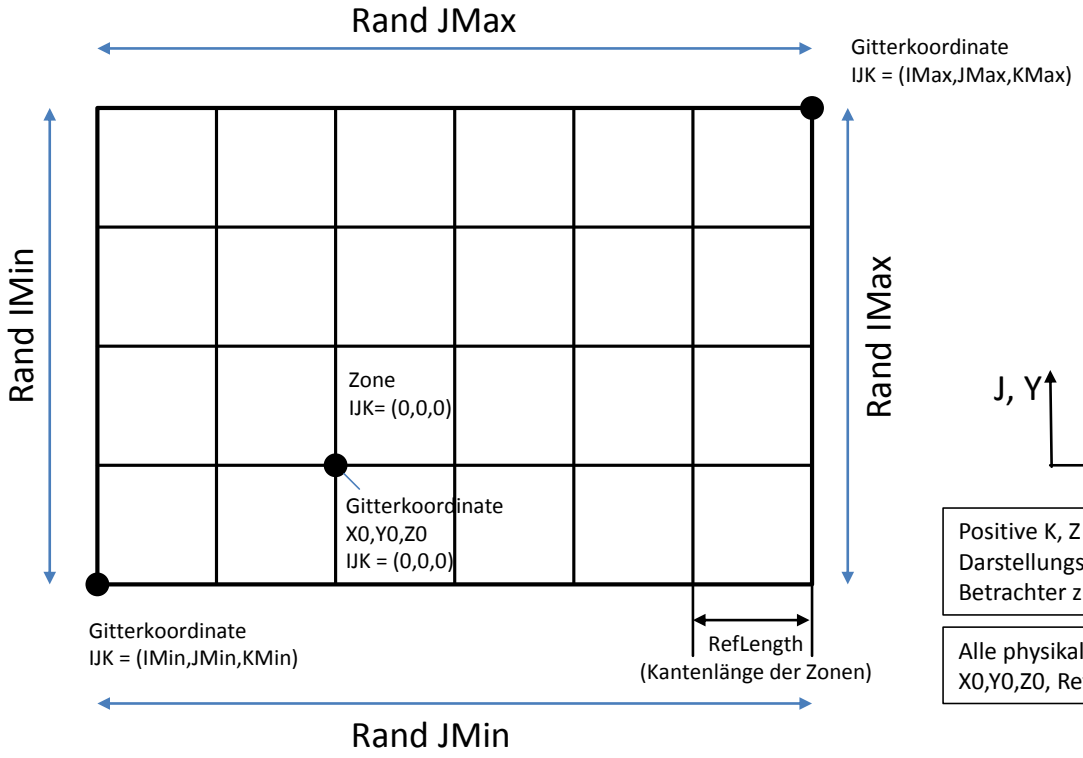
Allgemein

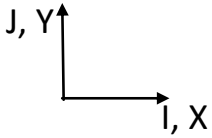
- Zeilennummern dienen nur zur Erläuterung und tauchen im File nicht auf
- Fileformat: ASCII Format, innerhalb einer Zeile frei formatiert (Trennzeichen = Leerzeichen)
- Alle CharackterStrings in Hochkomma (ermöglicht unformatiertes Einlesen in Fortran)
- Maximale Stringlänge für Namen = 16 (Zeile 9, Zeile 13)

Angaben zu den Einträgen

- Zeile 1: 201501 (dies ist die Versionskennziffer)
- Zeilen 2-4: Max Level für die Simulation, Zonenlänge, Referenzkoordinate (siehe Skizze S.3)
- Zeile 5: Bounding Box in Gitterkoordinaten (siehe Skizze S.3)
- Zeilen 6-9: Anzahl der Zonen im Berechnungsgebiet, Anzahl der äußeren Zonenoberflächen im Berechnungsgebiet, Anzahl der Parts (Zonen können zu Parts gruppiert werden, deren Eigenschaften in APOLLO individuell ausgewertet werden können); Namen der Parts
- Zeile 10: 1 (dies ist die Kennziffer für das einfache Beschreibungsformat – anwendbar für ein quaderförmiges Berechnungsgebiet); Erfordert:
  - NumZones =!= (Imax-Imin) x (Jmax-Jmin) x (Kmax-Kmin) mit Imin, Imax, … aus Zeile 5
  - NumSurf =!= 2*(Imax-Imin) x (Jmax-Jmin) + 2*(Imax-Imin) x (Kmax-Kmin)+ 2*(Kmax-Kmin) x (Jmax-Jmin) mit Imin, Imax, … aus Zeile 5
  - NumParts =!= 1; Partname(1) = z.B. = "Domain"
- Zeile 11: Randbedingungen an Imin, Imax, Jmin, Jmax, Kmin, Kmax des Berechnungsgebietes:
  - Offener Rand => 3, Wand => 1. Ebenes Terrain mit Boden an Z=Zmin: => Eintrag Zeile 11: 3 3 3 3 1 3, siehe auch Skizze S.5
- Zeilen 12-13: Eingebettete Voxel können zu Objekten gruppiert werden, deren Eigenschaften in APOLLO individuell ausgewertet werden können.
  - Sinnvoll für eine übersichtliche Anzahl von Objekten; Nutzung im vorliegenden Kontext nur bedingt sinnvoll (große Anzahl von Gebäuden)
  - Vorschlag: Feature wird hier zunächst nicht verwendet -> NumObjects = 1; ObjectName(1) = „AllObjects"
- Zeilen 14 : Liste der eingebetteten Voxel (beliebige Anzahl)
  - Alle Objekte werden auf dem feinsten Gitter eingebettet (Gitter mit Auflösung MaxLevel)
  - Konvention für IJK Positionen: siehe Skizze S.4
  - Reihenfolge der Listung der Voxel egal; aus Rechenzeitgründen kann aber eine Clusterung nach Zonen sinnvoll sein. (In APOLLO wird zonenweise abgearbeitet)
  - Obj#: Referenz zu einer m Objekt; Kann auch Referenz zu einem Element des Stadtmodells sein für spätere Übertragung des Ergebnisses auf das referenzierte Element des Stadtmodells; Wertebereich: beliebige Integer; Die Referenz zum Element des Stadtmodells kann aber auch ausserhalb von APOLLO gespeichert werden I,J,K -> StadtElemenet#

Erläuterungen zum Voxelmodell: Berechnungsgebiet

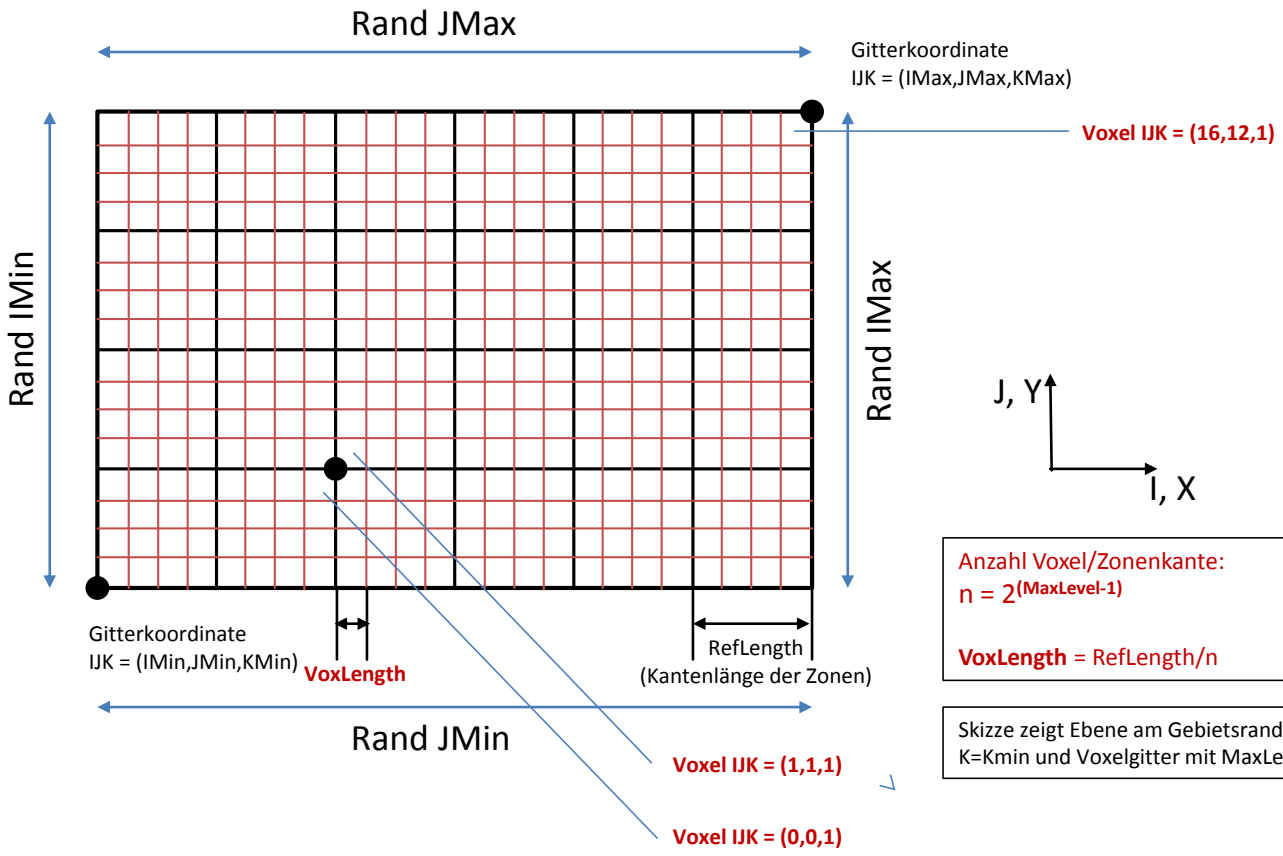




Positive K, Z Richtung zeigt aus der Darstellungsebene heraus (auf den Betrachter zu)

Alle physikalischen Längen in Meter [m]: X0,Y0,Z0, RefLength

Erläuterungen zum Voxelmodell: Voxelkoordinaten

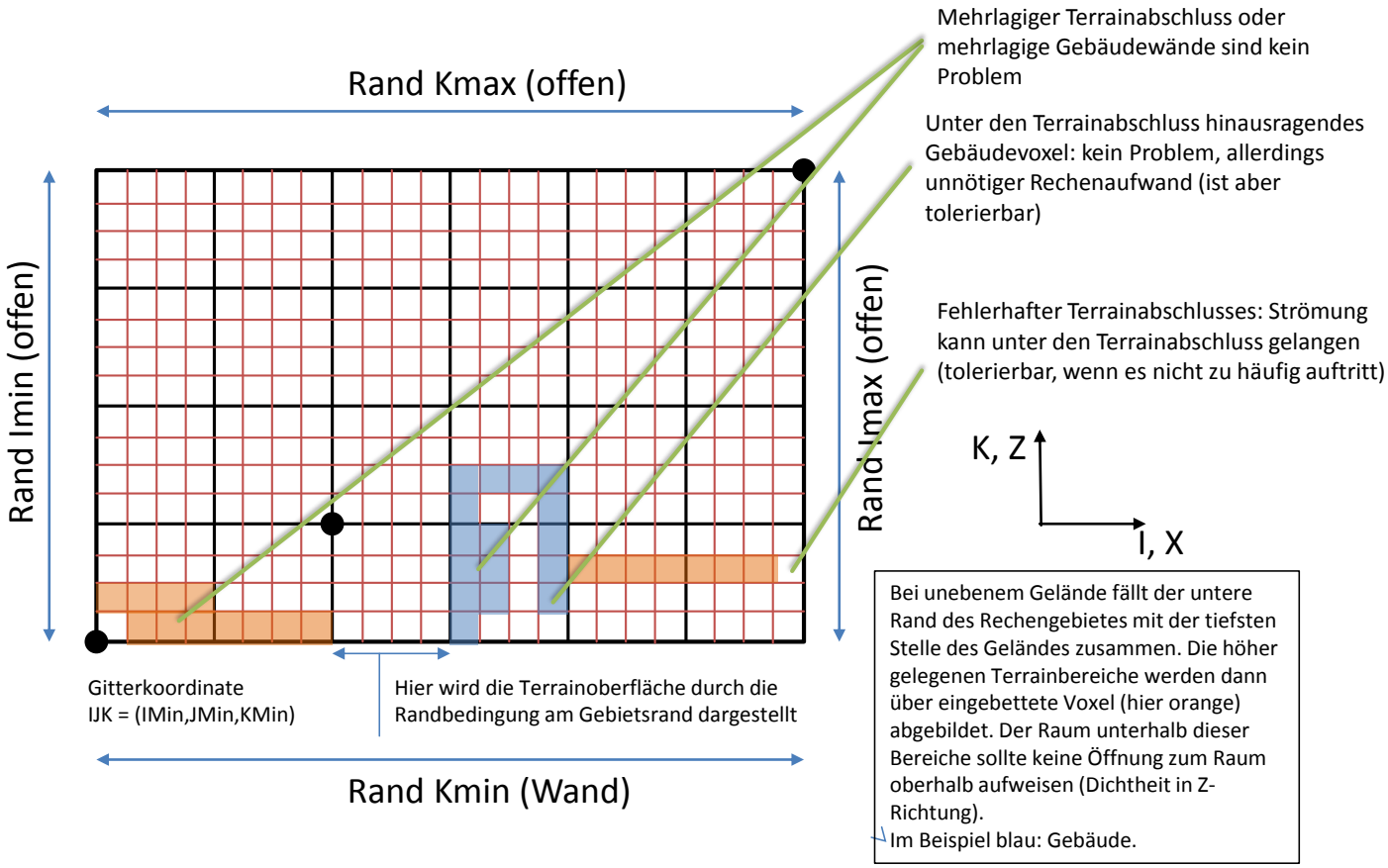


Anzahl Voxel/Zonenkante:
$n = 2^{(MaxLevel-1)}$

VoxLength = RefLength/n

Skizze zeigt Ebene am Gebietsrand
K=Kmin und Voxelgitter mit MaxLevel=3

Erläuterungen zum Voxelmodell: Terrainabschluss (Dichtheit in Z)



Bei unebenem Gelände fällt der untere Rand des Rechengebietes mit der tiefsten Stelle des Geländes zusammen. Die höher gelegenen Terrainbereiche werden dann über eingebettete Voxel (hier orange) abgebildet. Der Raum unterhalb dieser Bereiche sollte keine Öffnung zum Raum oberhalb aufweisen (Dichtheit in Z-Richtung).
Im Beispiel blau: Gebäude.

Weitere Hinweise zum Voxelmodell

Alternatives Format für Zonen bei unregelmäßigen oder zusammengesetzten Berechnungsgebieten (Zeile 10 – 11)

- Falls eine große Zahl von Zonen vollständig unbenutzt ist, z.B. unterhalb eines sehr schrägen Terrains liegt oder sehr große Innenräume von großen Gebäuden existieren, kann ein alternatives Format zur Beschreibung des Berechnungsgebietes verwendet werden:

```
10          Format                                                    (Int)
11a         Izone(1), Jzone(1), Kzone(1), Part#(1)                    (4x Int)
...
11a         Izone(NumZone), Jzone(NumZone), Kzone(NumZone), Part#(NumZone)    (4x Int)
11b         Izone(1), Jzone(1), Kzone(1), SurfZone(1), Part#(1), RB(1)        (4x Int)
. . .
11b         Izone(NumSurf), Jzone(NumSurf), Kzone(NumSurf), SurfZone(NumSurf), Part#(NumSurf),
            RB(NumSurf)           (6x Int – in einer Zeile)
```

- Format = 2
- In Zeilen 11a werden die Zonen einzeln aufgelistet die das Berechnungsgebiet bilden (Reihenfolge egal), die zugeordnete Part# wird angegeben
- In Zeilen 11b werden die Zonen aufgelistet, die freie Oberflächen haben, an denen Randbedingungen definiert werden müssen (auch innere Ränder, falls benachbarte Zonen nicht gejoined werden)
  - Die Surface Number (SurfZone) spezifiziert die die Zonenfläche: 1,2,3,4,5,6 => Imin, IMAx, Jmin, Jmax, Kmin, Kmax,
  - RB bezeichnet den Typ der Randbedingung: RB = 1 oder 3 (für Wand oder offener Rand)
- Dieses Format reduziert den Rechenaufwand, da es keine unbenutzten Zonen im Berechnungsgebiet gibt. Unbenutzte Zonen haben aber einen sehr kleinen Aufwand, so dass sich dieses Format vor allem lohnt, wenn die Zahl der unbenutzten Zonen groß ist, zB > 50%)

Weitere Hinweise zum Voxelmodell

Festlegung der Referenzkoordinate (Ursprung des Koordinatensystems des Zonengitters und des Voxelgitters: X0, Y0, Z0 in Zeile 4)

Dem Rechenverfahren sind die absoluten Koordinaten prinzipiell egal.

Die Güte der Approximation des Stadtmodells auf das Voxelgitter kann jedoch von der Lage der Koordinatenursprungs abhängen: Eine Verschiebung um einen Bruchteil einer Voxellänge kann eine Verbesserung oder Verschlechterung der Abbildung bewirken.

Beispiel: Ein Gebäude besitzt Abmessungen, die exakt ganzzahligen Vielfachen eines Voxels entsprechen. Dann kann das Gebäude exakt abgebildet werden, wenn der Ursprung des Voxelgitters auf einer Gebäudeecke liegt.

Weitere Hinweise zum Voxelmodell

Festlegung der Referenzlänge (Zonenlänge) und des Max. Levels

Damit das Rechenverfahren eine hohe Effizienz entwickeln kann, muss die Anzahl der Zonen im Bereich zwischen etwa 10.000 und 100.000 liegen.

Die Festlegung des maximalen Levels wird aus der User-Eingabe für die Rechengenauigkeit bzw. den Rechenzeitbedarf (1,2,3 -> schnell, typisch, genau) abgeleitet.

Somit:

1: Festlegung eines relevanten Berechnungsgebietes
- Ladungsmenge (kg TNT, user eingabe)
- Rbb = maximaler interessierender Radius, aus:
    - zB Spitzenüberdruck = 2 kPa -> Rbb = (M_tnt)^(1/3) * 40 [m]
- Bounding Box enthält Kreis mit Rbb um Explosionsort
    - Höhe des Berechnungsgebietes: 1 Rbb über dem Explosionsort; Abschluss nach unten: tiefster Punkt des Terrains in der Bounding Box

2: Festlegung der Zonenlänge
- Lref liegt im Wertebereich zwischen (Volumen der Bounding Box / 10000)^(1/3) bis (Volumen der Bounding Box / 100000)^(1/3)
- In diesem Bereich kann gemittelt und/oder gerundet werden (ganze Zahlen, zB 2m, 3m bringen jedoch keine Vorteile gegenüber nicht gerundeten Werten)

3: Die Formel für die Festlegung von MaxLevel muss ich noch liefern. Typischerweise werden Werte zwischen 3 bis 5 auftreten (4 bis 16 Voxel pro Kantenlänge der Zone)

Ergebnisdateien (stehen nach Ende des Simulationslaufs zur Verfügung)

Binär, VTK Format: PROJECT_modeff.vtk
ASCII Datei: PROJECT_modeff.dat

```
1  Comment Line (= # Blast effects record)                              -
2  NumField (Anzahl der Ergebnisgrößen)                                 (Int)
3  FieldName(1), …, Fieldname(NumField)                                 (N x Character)
4  X0 Y0 Z0 (identisch X0,Y0,Z0 aus Modell File)                        (3x Float)
5  VoxLength (identisch VoxLength in Modell File via RefLength,HiLev)   (1x Float)
6  Comment Line (= #Record for embedded voxel surfaces)                 -
7  NumVoxSurf (Anzahl Voxel Surfaces mit Fluid Kontakt)                 (Int)
8  Comment Line
9  ivox(1) jvox(1) kvox(1) surf(1) obj#(1) Results(1:NField,1)          (5xInt, NumField x Float)
.....
9  ivox(NumVoxSurf) jvox() kvox() surf() obj#() Results(1:NField, )     (5xInt, NumField x Float)

10 Comment Line (= # Results for Solid Domain Surfaces)                 -
11 NumDomSurf (Anzahl Domain Surfaces with Wall B/CV)                   (Int)
12 Comment Line                                                         -
13 ivox(1) jvox(1) kvox(1) surf(1) NVox Results(1:NField,1)             (5xInt, NumField x Float)
.....
13 ivox(NumDomSurf) jvox() kvox() surf() NVox Results(1:NField, )       (5xInt, NumField x Float)
```

Die Anzahl der in das Stadtmodell zum Zweck der Visualisierung zurück zu übertragenden Ergebnisse ist NumField. Die als Bezeichnungen (Legendentitel) zu verwendenden Namen sind in Zeile 3 aufgelistet.

Zeilen 4, 5 definieren das Voxelgitter (identisch zur Spezifikation im Modellfile)

In Zeile 6 – 9(ff) sind alle von Fluid benetzten Voxeloberflächen aufgelistet. Das Voxel wird durch ivox, jvox, kvox identifiziert und die jeweilige Fläche durch die Surf Variable (Surf=1,2,3,4,5,6): 1,2,3,4,5,6 => Imin, Imax, Jmin, Jmax, Kmin, Kmax

In Zeile 10-13(ff) sind alle Domain Oberflächen mit Randbedingung Wand aufgelistet. Diese werden zB für den Terrainabschluss bei ebenem Boden relevant, wenn nicht eine durchgehende Voxel-Schicht als Boden verwendet wird. Da hier die automatische Gitteranpassung wirksam ist, können anstelle einzelner Voxeloberflächen (mit Kantenlänge VoxLength) auch Patchcs aus NVox x NVox Voxelflächen zusammengefasst aufgelistet sein.

APPENDIX C

CityGML UML diagrams

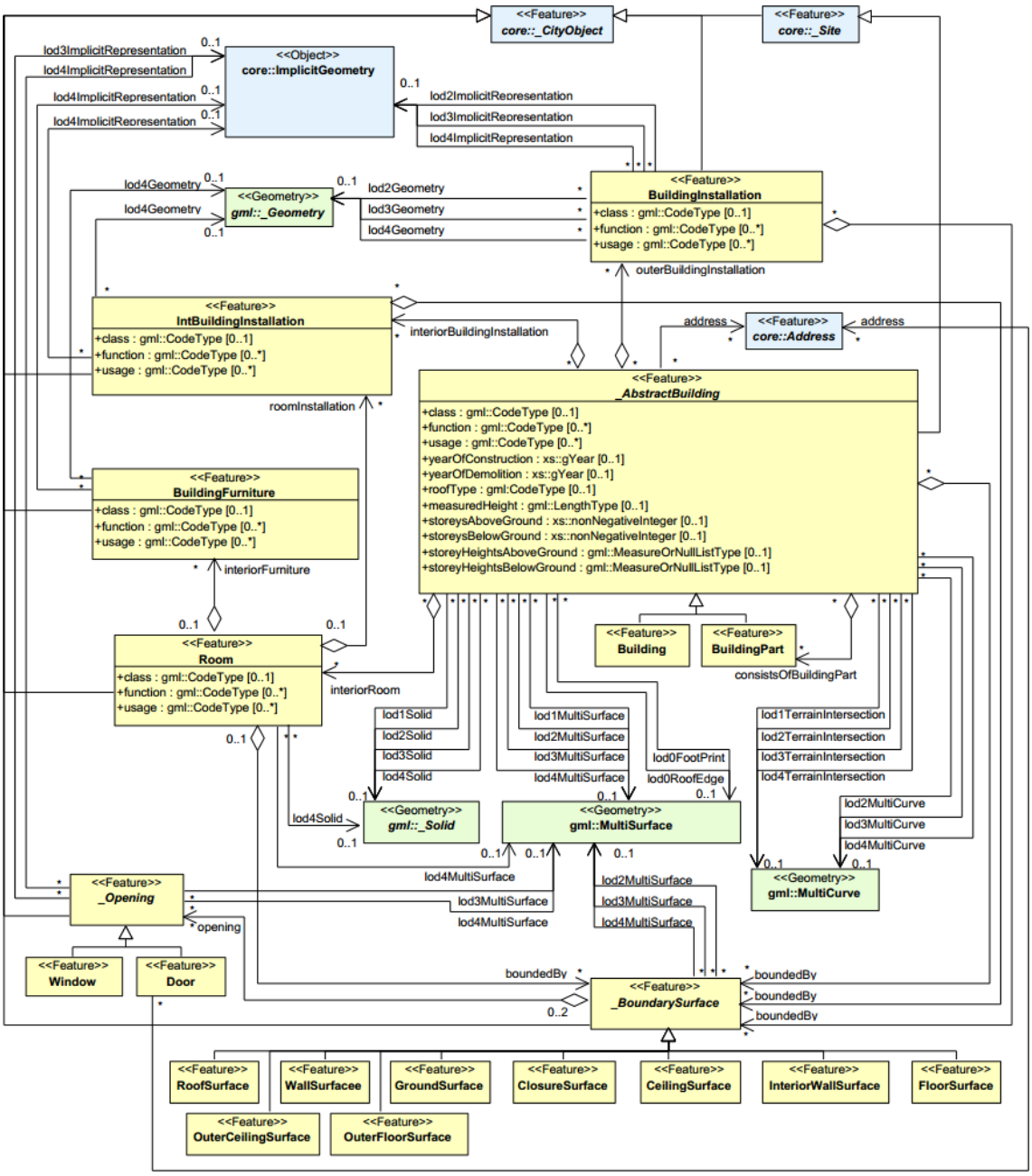


Figure C.1: UML diagram of CityGML's bulding model. Image from: (Open Geospatial Consortium, 2012)

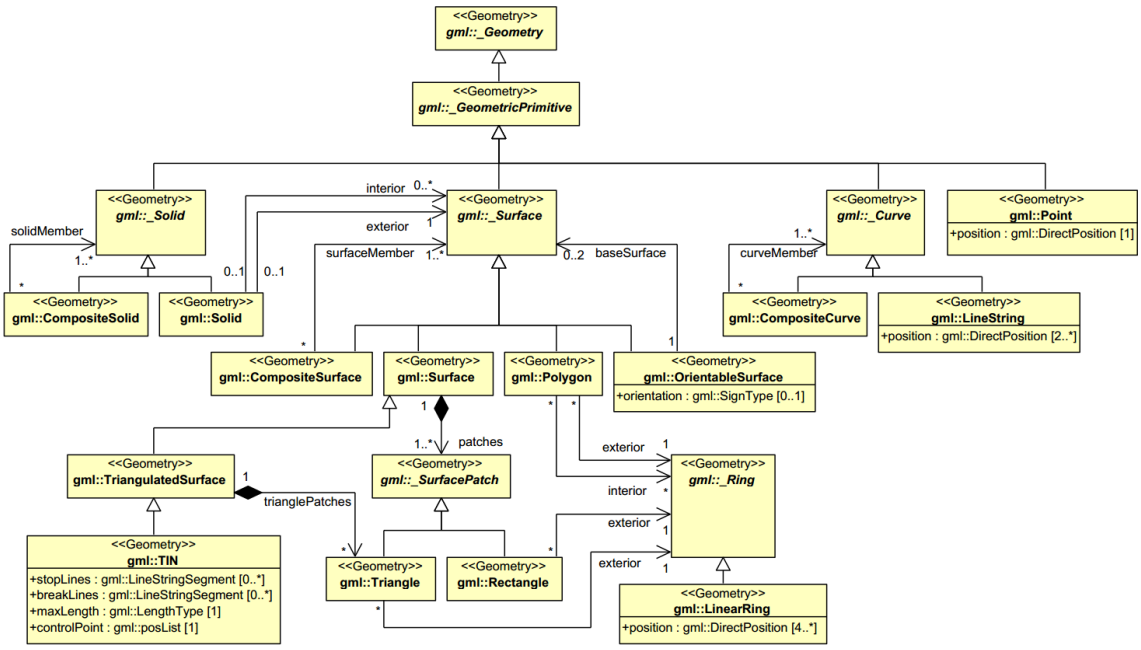


Figure C.2: UML diagram of CityGML's geometry model. Image from: (Open Geospatial Consortium, 2012)

APPENDIX D

3DCityDB UML diagrams

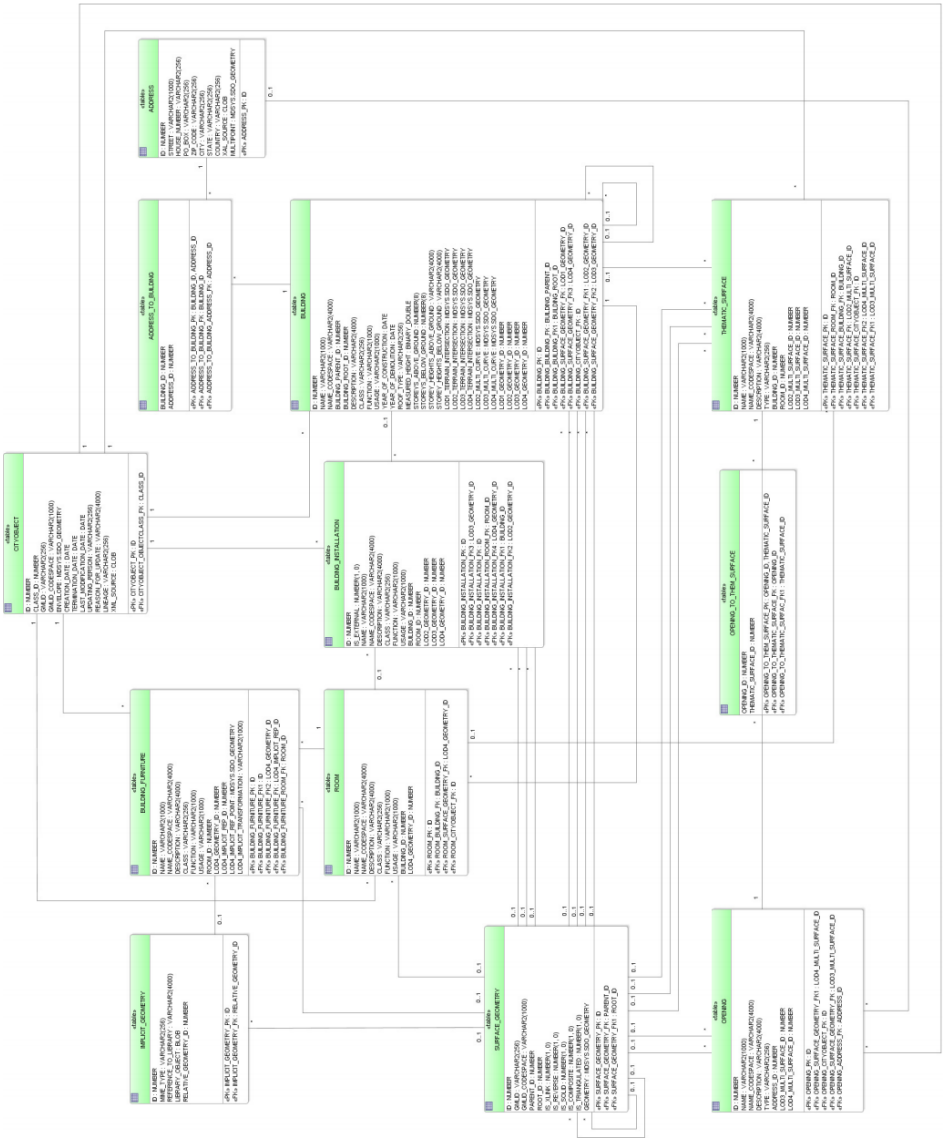


Figure D.1: Database schema of the building model. Image from: Kolbe, Koenig, et al., 2009

Figure D.2: Database schema of the generic objects model. Image from: Kolbe, Koenig, et al., 2009

APPENDIX E

XML examples

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<exdev:ExplosiveDevices xmlns:exdev="http://bwibo.de/exdev">
    <exdev:ExplosiveDevice>
        <exdev:massTNT_kg>299.78</exdev:massTNT_kg>
        <exdev:name>Bombe</exdev:name>
        <exdev:type>WW2</exdev:type>
    </exdev:ExplosiveDevice>
    <exdev:ExplosiveDevice>
        <exdev:massTNT_kg>10.0</exdev:massTNT_kg>
        <exdev:name>Rohrbombe</exdev:name>
        <exdev:type>IMPROVISED</exdev:type>
    </exdev:ExplosiveDevice>
    <exdev:ExplosiveDevice>
        <exdev:massTNT_kg>999.0</exdev:massTNT_kg>
        <exdev:name>BigBomb</exdev:name>
        <exdev:type>WW2</exdev:type>
    </exdev:ExplosiveDevice>
    <exdev:ExplosiveDevice>
        <exdev:massTNT_kg>50.0</exdev:massTNT_kg>
        <exdev:name>OtherBomb</exdev:name>
        <exdev:type>OTHER</exdev:type>
    </exdev:ExplosiveDevice>
</exdev:ExplosiveDevices>
```

Listings E.1: XML instance document of the explosive device data structure.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<simpar:SimulationParameters xmlns:exdev="http://bwibo.de/exdev"
    xmlns:exsrc="http://bwibo.de/exsrc" xmlns:simpar="http://bwibo.de/simpar">
    <simpar:SimulationParameter>
        <simpar:Name>TUM</simpar:Name>
        <simpar:Author>Bruno</simpar:Author>
        <simpar:CreationDate>2015-01-15T20:26:46.042+01:00</simpar:CreationDate>
        <simpar:Description>TU Campus Innenhof</simpar:Description>
        <simpar:MaxLevel>2</simpar:MaxLevel>
        <simpar:Rmax>100.0</simpar:Rmax>
        <simpar:CalcualtionOption>FREE_FIELD</simpar:CalcualtionOption>
        <simpar:CalculationQuality>VERYLOW</simpar:CalculationQuality>
    <exdev:ExplosiveDevice>
        <exdev:massTNT_kg>50.0</exdev:massTNT_kg>
        <exdev:name>OtherBomb</exdev:name>
        <exdev:type>OTHER</exdev:type>
    </exdev:ExplosiveDevice>
        <exsrc:ExplosionSource>
            <exsrc:x>4467922.0</exsrc:x>
            <exsrc:y>5334642.0</exsrc:y>
            <exsrc:z>509.8</exsrc:z>
            <exsrc:SRID>31468</exsrc:SRID>
        </exsrc:ExplosionSource>
    </simpar:SimulationParameter>
</simpar:SimulationParameters>
```

Listings E.2: XML instance document of the simulation parameters data structure.

```
<!--...-->
<bldg:WallSurface gml:id="DEBY_Lxxxx">
  <gml:boundedBy>
    <gml:Envelope srsName="EPSG:31468" srsDimension="3">
      <gml:lowerCorner>4467838.73 5334699.44 0.0</gml:lowerCorner>
      <gml:upperCorner>4467838.91 5334699.52 22.93</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <creationDate>2015-01-16</creationDate>
  <gen:doubleAttribute name="AVG_peak_overpressure">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="AVG_max_overpressure_impulse">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_overpressure_impulse">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_glass">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_safety_glass">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_masonry_30cm">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_rc_wall_30cm">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_person_safe">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_leathality_1_percent">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_leathality_50_percent">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="AVG_eardrum_dmg">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="MAX_peak_overpressure">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
  <gen:doubleAttribute name="MAX_max_overpressure_impulse">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="MAX_overpressure_impulse">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
    <gen:doubleAttribute name="MAX_glass">
    <gen:value>0.0</gen:value>
  </gen:doubleAttribute>
```

```
55      <gen:doubleAttribute name="MAX_safety_glass">
56      <gen:value>0.0</gen:value>
57    </gen:doubleAttribute>
58      <gen:doubleAttribute name="MAX_masonry_30cm">
59      <gen:value>0.0</gen:value>
60    </gen:doubleAttribute>
61      <gen:doubleAttribute name="MAX_rc_wall_30cm">
62      <gen:value>0.0</gen:value>
63    </gen:doubleAttribute>
64      <gen:doubleAttribute name="MAX_person_safe">
65      <gen:value>0.0</gen:value>
66    </gen:doubleAttribute>
67      <gen:doubleAttribute name="MAX_leathality_1_percent">
68      <gen:value>0.0</gen:value>
69    </gen:doubleAttribute>
70      <gen:doubleAttribute name="MAX_leathality_50_percent">
71      <gen:value>0.0</gen:value>
72    </gen:doubleAttribute>
73      <gen:doubleAttribute name="MAX_eardrum_dmg">
74      <gen:value>0.0</gen:value>
75    </gen:doubleAttribute>
76    <bldg:lod2MultiSurface>
77      <gml:MultiSurface gml:id="UUID_f1c12f4e-5a7f-4e9a-b469-fb59009e6bf8">
78        <gml:surfaceMember>
79          <gml:Polygon gml:id="UUID_f965419e-eba5-4b20-a445-8e9c7405e548">
80            <gml:exterior>
81              <gml:LinearRing gml:id="UUID_f965419e-eba5-4b20-a445-8e9c7405e548_0_">
82                <gml:posList srsDimension="3">... ... ...</gml:posList>
83              </gml:LinearRing>
84            </gml:exterior>
85          </gml:Polygon>
86        </gml:surfaceMember>
87      </gml:MultiSurface>
88    </bldg:lod2MultiSurface>
89  </bldg:WallSurface>
90  <!--....-->
```

Listings E.3: XML instance example of a CityGML *WallSurface* with generic attributes of the Apollo Blast Plugin.

APPENDIX F

Apollo Blastsimulator results import scripts and example

```
#!/bin/sh
# Prepare ApolloBlast results file for DB import ------------------------------
#   - CSV import, space = separator
# ------------------------------------------------------------------------------
sed -i.bak '1,8d' $1            # delete file header (row 1-8)
sed -i 's/ \+/ /g' $1           # multiple space -> single space
sed -i 's/^ //g' $1             # remove spaces at beginning of a line
sed -i 's/ $//g' $1             # remove spaces at end of a line

# delete unneeded results for boundary of domain
# delete all file after match
sed -i -n -e '1,/# RESULTS FOR SOLID DOMAIN SURFACES/p' $1
# delete matched row
sed -i '/^# RESULTS FOR SOLID DOMAIN SURFACES$/d' $1
```

Listings F.1: Shell script for Apollo Blastsimulator results file CSV import preparation.

```
psql \copy apolloblast.result FROM 'C:/csv/file.csv' DELIMITER ' ' CSV
```

Listings F.2: Batch cmd line for CSV import ot PostgreSQL.

```
# BLAST EFFECTS RECORD
11  ...number of result variables, see names below
"peak overpressure", "max.overpressure impulse", "overpressure impulse", "glass", "safety glass", "masonry 30cm",
 "rc-wall 30 cm","person safe", "lethality 1%", "lethality 50%", "eardrum damage"
0.0000E+00  0.0000E+00  0.0000E+00  ...x0, y0, z0 (= ijknode = 0 0 0, voxel next to this node: ijk = 1 1 1)
0.16250E+01 ... voxel length
# RECORD FOR EMBEDDED VOXEL SURFACES
84694  ...number of voxel surfaces
# Data below: voxel i, j, k, surf#, obj#, results values(1:11)
-62 -72 1 3 1 0.1679E+03 0.1425E+02 0.9934E+01 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-61 -72 1 3 1 0.1804E+03 0.1492E+02 0.1027E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-60 -72 1 3 1 0.1805E+03 0.1504E+02 0.1026E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-62 -72 1 1 1 0.2302E+03 0.1613E+02 0.1031E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
...
...
# RESULTS FOR SOLID DOMAIN SURFACES
18886  ...number of domain surfaces
# Data below: i, j, k, surf#, Voxels/Surface, results values(1:11)
-67 -75 1 5 1 0.1721E+03 0.1683E+02 0.1238E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-66 -75 1 5 1 0.1634E+03 0.1574E+02 0.1152E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-65 -75 1 5 1 0.1695E+03 0.1577E+02 0.1139E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
-64 -75 1 5 1 0.1758E+03 0.1579E+02 0.1131E+02 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
0.0000E+00 0.0000E+00
...
...
```

Listings F.3: Sample results file of the ApolloBlast Simulator.

APPENDIX G

Apollo Blast Plugin sample log file

```
[08:34:03 INFO] Connecting to database profile 'tum3'.
[08:34:03 INFO] Database connection established.
[08:34:03 INFO] PostgreSQL, 9.3.2
[08:34:03 INFO] SRID: 31468 (Projected)
[08:34:03 INFO] SRS: DHDN / 3-degree Gauss-Kruger zone 4
[08:34:03 INFO] gml:srsName: EPSG:31468
[08:34:03 DEBUG] Reference system '[Default] WGS 84' (SRID: 4326) supported.

[08:34:16 INFO] -- VBG -------------------------------------------
ExSource = SRID=31468;POINT(4467922 5334642 0)
MaxLevel = 3
rMax_temp = 120.0   rMax = 123.5
zMin_temp = 0.0   zMin = 0.0

BBoxGeoTemp = BOX3D(4467802 5334522 0,4468042 5334762 120)
BBoxGeo     = BOX3D(4467798.5 5334518.5 0,4468045.5 5334765.5 123.5)

BBoxZones =
  BBoxIJK[
    celllength=6.5,
    imin=-19, jmin=-19, kmin=0, imax=18, jmax=18, kmax=18,
    ncells_imin=19, ncells_jmin=19, ncells_kmin=0,
    ncells_imax=19, ncells_jmax=19, ncells_kmax=19
  ]

BBVoxels =
  BBoxIJK[
    celllength=1.625,
    imin=-75, jmin=-75, kmin=1, imax=76, jmax=76, kmax=76,
    ncells_imin=76, ncells_jmin=76, ncells_kmin=0,
    ncells_imax=76, ncells_jmax=76, ncells_kmax=76
  ]
--------------------------------------------------
[08:34:16 INFO] Running Simulation...
```

```
[08:34:16 INFO] Preparing...
[08:34:16 INFO] Preparing...done!
[08:34:16 INFO] Selecting CG geometries...
[08:34:17 INFO] Selecting CG geometries...done!
[08:34:17 INFO] Intersect voxel grid <-> CG geometries...
[08:35:52 INFO] Intersect voxel grid <-> CG geometries...done!
[08:35:52 INFO] Write ApolloBlast input file...
[08:35:53 INFO] Write ApolloBlast input file...done!
[08:35:53 INFO]
- Intersection Report CCGEOM ----------
  total             1216
  intersected       1208
  not intersected   8
--------------------------------------
[08:35:53 INFO]
- Intersection Report VOXEL -----------
  total             52849
  intersected       52849
  not intersected   0
--------------------------------------
[08:35:53 INFO] Running Simulation...done!
```

Listings G.1: Sample ImporterExport log of a simulation run.

APPENDIX H

3DCityDB ImporterExporter WorkerPool UML Charts

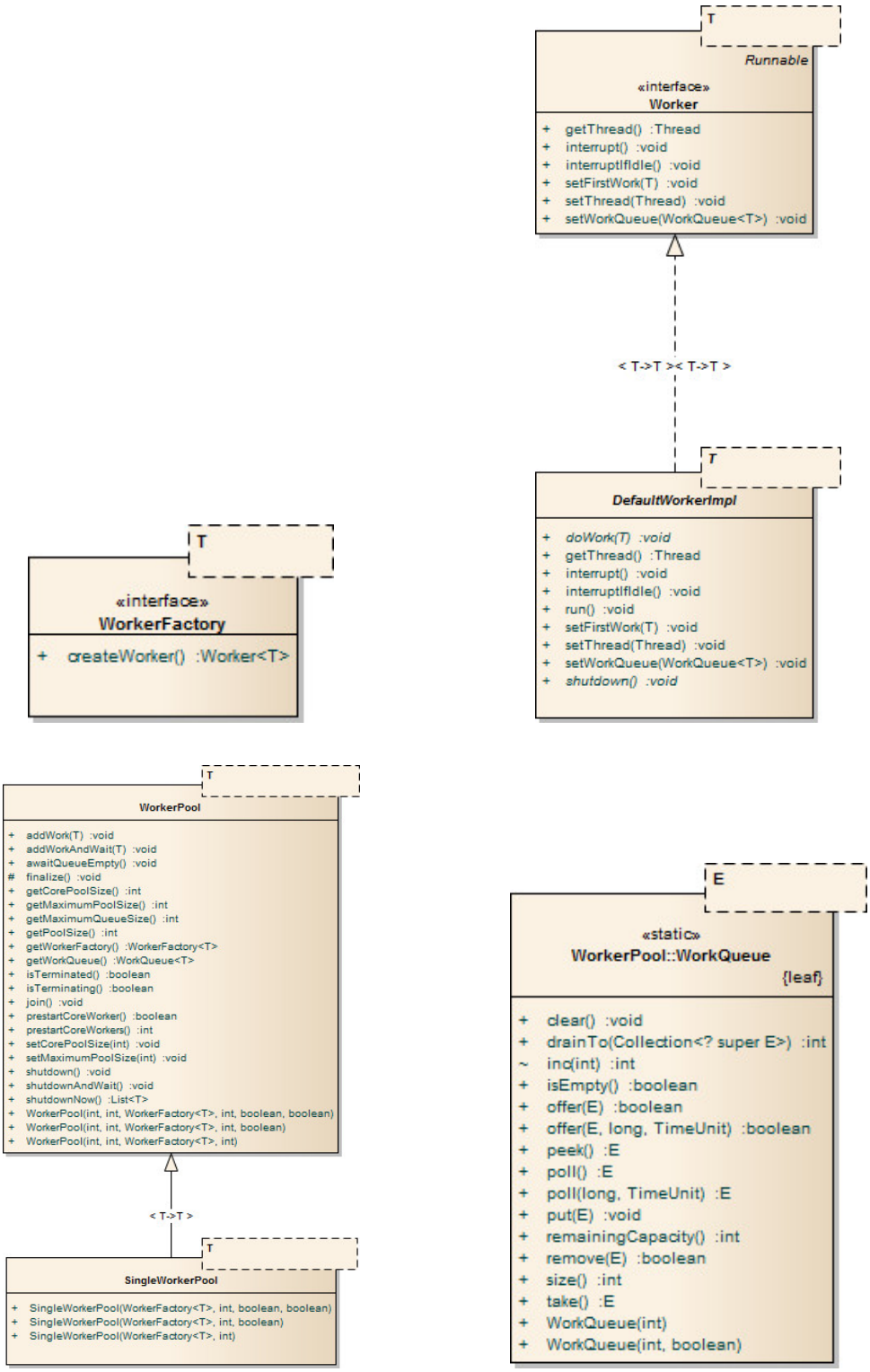


Figure H.1: 3D CityDB ImporterExporter Plugin API Worker Pool implementation.