Technische Universität München

Lehrstuhl für Computation in Engineering

A Parallel, Multi-Resolution Framework for Handling Large Sets of Complex Data, from Exploration and Visualisation to Simulation

Vasco Varduhn

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. André Borrmann

Prüfer der Dissertation:

1. Univ.-Prof. Dr. rer. nat. Ernst Rank
2. Univ.-Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz

Die Dissertation wurde am 26.03.2014 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 05.05.2014 angenommen.

Abstract

In this work, techniques from computational science and engineering are applied to demanding questions and challenges from civil engineering concerning our cities. Many complex features of urban regions can be described by fusing and integrating highly detailed multi-scale data. A parallel data access framework with clearly defined and proven interfaces to all parts of the simulation pipeline such as preprocessing, numerical simulation and postprocessing is developed. This framework is capable of storing, handling and providing access to large amounts of highly detailed data from constructions, built infrastructure, geographical data and infrastructure networks.
The strict usage of parallelisation techniques and efficient algorithms is necessary and introduced in detail. These techniques make it possible to create a framework which can provide access to multi-resolution representations of large data sets fast enough to put into practice a complete exploration and simulation pipeline, which ranges from a multi-monitor or CAVE-based real-time visualisation to multi-scale simulation scenarios such as flooding of urban regions.
Finally, the application of an urban flooding simulation with the incorporation of pipe network interaction is given. The capabilities of the framework allow the coupling of the different simulations to be handled efficiently and insight is gained over all scales from the global flow behaviour to the impact on single construction entities.

Zusammenfassung

In dieser Arbeit werden Ansätze der Ingenieursinformatik sowie des wissenschaftlichen Rechnens auf Fragestellungen des Bauingenieurwesens zur Untersuchung hoch detaillierter Stadtmodelle angewendet, um Beiträge zur Beantwortung dringender Fragen und Herausforderungen unserer stetig wachsenden Städte zu leisten.
Ein parallelisiertes Datenhaltungskonzept wird vorgestellt, das klar definierte Schnittstellen zu einzelnen Schritten der numerischen Simulation, wie zum Beispiel effiziente Datenspeicherung oder Verarbeitung, zur Verfügung stellt. Dieses Datenhaltungskonzept bietet die Möglichkeit, große Mengen von Produktmodelldaten vorzuhalten und in Echtzeit zu verarbeiten. Diese Daten umfassen die vollständige geometrische und semantische Beschreibung von Städten, einschließlich Gebäuden und Konstruktionen, kartografischen Oberflächen und Spezifikationen von Rohrleitungs- und Versorgungsnetzen.
Diese Datenbasis wird in einem verteilten Konzept zusammengeführt und effiziente Schnittstellen zu Simulation und Visualisierung werden implementiert und detailliert beschrieben. Diese Schnittstellen reichen von der Anbindung einer komplexen Visualisierungs- und Auswertungsumgebung, einschließlich der Möglichkeit der interaktiven Erkundung der Daten, bis hin zur numerischen Simulation auf großen Parallelrechnern.
Das entwickelte Gesamtkonzept wird anhand einer großflächigen Überflutung von Städten

demonstriert, wobei neben der Oberflächenströmung auch das unterirdische Kanalnetzwerk und deren Interaktion berücksichtigt wird. Durch die detaillierte Auswertung der numerisch gewonnenen Daten auf verschiedenen Skalen kann eine Untersuchung der Ausbreitung des Wassers Auswirkungen sowohl auf der Ebene der ganzen Stadt, als auch auf der einzelner Gebäude oder gar einzelner Bauwerksdetails genau aufzeigen und dementsprechend bewertet werden.

Vorwort

Die vorliegende Arbeit entstand während meiner fünfjährigen Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Computation in Engineering an der Technischen Universität München (TUM) und in dem interdisziplinären Forschungsprojekt "Virtual Arabia" zwischen der King Abdullah University of Science and Technology (KAUST) und der TUM.

Mein besonderer Dank gilt meinem Doktorvater Herrn Prof. Dr. Ernst Rank für seine fachliche Expertise und seine tatkräftige Unterstützung zum Gelingen meiner Arbeit. Durch sein Vertrauen in mich und meine Forschung sowie durch die mir übertragene und anvertraute Lehrtätigkeit konnte ich mich weiter entwickeln.

Ich danke Herrn Prof. Dr. Hans-Joachim Bungartz sehr für die Übernahme des Koreferats, seine kritischen Kommentare zu meiner Arbeit und der vorliegenden Monographie, sowie seine fachliche Unterstützung als Principal Invesitgator des Virtual Arabia Projektes.

Ich danke allen Kollegen am Lehrstuhl, insbesondere Hanne Cornils, Dr. Ralf-Peter Mundani und Jérôme Frisch, Dr. Angelika Kneidl und Nils Zander, sowie den Projektkollegen Dr. Jens Schneider, Amal Benzina, Gerrit Buse, Daniel Butnaru, Alin Murarasu und Marc Treib für die intensive Zusammenarbeit und ihre persönliche Unterstützung.

Mein größter Dank gilt meiner Familie und meinen Lieben für ihre Unterstützung während der gesamten Zeit und insbesondere in der Endphase.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goal . . . 2
1.3 Structure . . . 3

2 Input Data and Preprocessing **5**
2.1 Building Information Modelling . . . 6
2.1.1 Industry Foundation Classes . . . 7
2.1.2 CAD Data and 3rd Party Building Information Models . . . 8
2.2 3D City Model . . . 8
2.3 Geographic Information System . . . 9
2.3.1 Terrain Specification . . . 10
2.3.2 Pipe and Supply Network Data . . . 10
2.4 Levels of Detail . . . 11
2.4.1 Multi-Resolution Raster Data . . . 12
2.4.2 Stepwise Coarsened Product Model Data . . . 13
2.4.3 GPU-based Generation of Levels of Detail . . . 15
2.5 Multi Resolution Meta Format . . . 22
2.6 Data Fusion and Set Augmentation . . . 25

3 Framework **27**
3.1 Hierarchical Data Representation . . . 29
3.1.1 Local Scale - Product Model Details . . . 32
3.1.2 Global Scale - Location Awareness . . . 35
3.2 A Scalable Hybrid Parallel Approach . . . 36
3.2.1 Multi-Level Parallelisation . . . 38
3.2.2 Building Up the Framework . . . 40
3.2.3 Interfacing Multi-Resolution Geometry . . . 43
3.2.4 Interfacing Auxiliary Information . . . 46
3.2.5 Interfacing Linearised Octree Representations . . . 48
3.2.6 Interfacing Voxel Representation . . . 49

3.2.7 Interfacing Voxel Query . . . 52
3.2.8 Communication Protocol of the Framework Access . . . 53
3.3 Load Distribution . . . 54
3.3.1 Probabilistic Mapping . . . 56
3.3.2 Modified Space Filling Curves . . . 56
3.3.3 Distribution Quality . . . 59

4 Visualisation and Exploration 63
4.1 Visualisation . . . 64
4.1.1 Visualisation of Hierarchical Ordered Product Model Data . . . 64
4.1.2 Integration of Simulation Data . . . 70
4.1.3 Parallel Visualisation in CAVE-like Environments . . . 71
4.2 Navigation and Data Exploration . . . 74
4.2.1 Interacting Handheld Devices . . . 76
4.2.2 User Tracking and Tracked Interaction . . . 77
4.2.3 Product Model Investigation and Engineering Applications. . . . 80

5 Coupling Grid-Generation and Simulation Frameworks 85
5.1 Coupling the Sierpinski Framework . . . 86
5.1.1 Derivation of the Bathymetry Discretisation . . . 87
5.2 Coupling the Peano Framework . . . 91
5.2.1 Interfacing the Geometry Description . . . 92
5.3 Coupling OpenFOAM . . . 95
5.3.1 Preprocessing the Solid/Fluid Discretisation . . . 97
5.3.2 Mesh Generation . . . 98
5.3.3 Parallel Mesh Generation and Domain Decomposition . . . 102

6 Multi-Resolution Parallel Numerical Simulation 105
6.1 Fluid Flow Simulation . . . 106
6.1.1 Three-Dimensional Potential Flow . . . 106
6.1.2 Two-Dimensional Shallow Water Equation . . . 106
6.1.3 Three-dimensional Incompressible Navier-Stokes Equation . . . 107
6.1.4 Three-dimensional Free Surface Simulation . . . 108
6.2 Resolution Refinement . . . 110
6.2.1 Resolution Adaption . . . 111
6.3 Postprocessing - Propagation to Product Model Data . . . 117
6.4 Performance Results . . . 119

7 Urban Flooding Simulation with Pipe Network Interaction 121
7.1 Existing Work . . . 122
7.2 A Holistic Pipe Network and Surface Flow Approach . . . 124

7.2.1 Surface Flow . . . 125
7.2.2 Pipe Network Flow . . . 127
7.2.3 Interplay of Surface and Pipe Network Flow . . . 130
7.3 The Scenario . . . 131
7.3.1 Data Basis and Discretisation . . . 131
7.3.2 Boundary Conditions . . . 131
7.3.3 Results . . . 131

8 Conclusion and Outlook 139

A Appendix 143
A.1 Data Fusion and Set Augmentation . . . 143
A.2 CiE Sandstorm Cluster . . . 146
A.3 Shaheen . . . 147
A.3.1 KAUST Blue Gene/P 'Shaheen' . . . 147
A.4 OpenFOAM Mesh Specification . . . 148
A.4.1 points . . . 148
A.4.2 faces . . . 148
A.4.3 boundary . . . 149
A.4.4 owner . . . 150
A.4.5 neighbour . . . 151
A.5 OpenFOAM Case Definition . . . 152
A.5.1 system/controlDict . . . 152
A.5.2 system/decomposeParDict . . . 153
A.5.3 system/fvSchemes . . . 154
A.5.4 system/fvSolution . . . 155
A.5.5 setFieldsDict . . . 156

Chapter 1

Introduction

1.1 Motivation

Today, more people live in cities and metropolitan areas than in the countryside. In emerging countries such as China, in particular, there is a strong trend towards developing mega-cities. Simultaneously, larger cities present tremendous challenges for the future. The ongoing shortage of fossil resources requires the implementation of energy-efficient building and city concepts. The high population density in cities demands proper infrastructure for public transportation and supply networks. Natural disasters such as the European flood in 2013 affect the people of a whole city and the constructions and built infrastructure there. These examples show that planning on the city scale is of the utmost importance. Whereas this is a very large scale, the granularity of the entities such as constructions and built infrastructure is very fine. Individual buildings are complex structures, cities cover thousands of buildings. Together with the infrastructure, a city means a very extensive scope for the investigation of physical phenomena, performance evaluation, identification of optimisation possibilities, or mitigation of risk potential. In these application scenarios the challenges of the future have to be tackled using efficient approaches.

Computational science and engineering has become the third pillar of research in addition to theory and experiment. There are even fields where it is the only applicable approach. The investigation of flooding in a city can obviously not be performed in an experiment. Neither can the damaging effect of substances on the ozone layer be investigated by performing a real experiment. This shows that there is an emerging demand to apply numerical simulation to questions concerning cities and their buildings as a whole.

In the field of computational science and engineering, there is a great variety of efficient, massively parallel and highly detailed applications for performing engineering relevant tasks from the fields of data exploration, numerical simulation and postprocessing. As has been

known for many years from disciplines such as automotive and aeronautical research, a strict exploitation of highly detailed product model descriptions, accurate numerical simulation and the examination of efficiently performed analyses in the early stages of the planning phases give engineers and scientists a tool in hand with which to gain an insight into the behaviour, potential and weaknesses of new constructions even before the first prototype has been built. A strict adherence to these techniques has the potential to achieve an optimisation of new constructions, taking place even before they have been built or helping to evaluate existing constructions.

Although all the potentials mentioned are there even or especially in the field of civil engineering, a strict exploitation from computational science and engineering has not yet been put into practice. But the very high complexity in civil engineering, especially in the field of city models and their buildings and built infrastructure exerts much more pressure to do so. In order to investigate the effects of natural disasters, to perform energy-efficient planning of buildings, to plan public transportation objects in a city, it is not only the assembly of the constructions with respect to each other which is important, but also the fully detailed description of the individual buildings. For an urban flooding investigation, for example, the global flow behaviour has to be investigated on scales in the range of tens of kilometres, but the impact on constructions and built infrastructure has to be examined in the range of metres and centimetres. Civil engineering applications are extremely multi-scale in nature.

Over the last two decades, there have been significant developments in the field of building and construction modelling, digital city models and large-scale geospatial services. Today, the fully detailed product models of constructions and built infrastructure can be described, digital city models examining the assembly of all buildings in a city are available, as are geospatial descriptions of the terrain, the surface and the land usage.

To put it in a nutshell, in civil engineering the multi-scale fully detailed description of cities, landscapes, constructions and built infrastructure is possible, together with a strong demand for the efficient investigation of real-life problems on the basis of these highly complex data by using the approaches of computational science.

1.2 Goal

The aim of this work is to apply techniques from computational science and engineering to the approaches of civil engineering and make a contribution to taking a further step towards answering the demanding questions and challenges of our cities.

In order to achieve this, a parallel data access framework with clearly defined and proven interfaces to all parts of the simulation pipeline such as preprocessing, numerical simulation and postprocessing is developed. This framework is capable of storing, handling and provid-

ing access to large amounts of highly detailed data from constructions, built infrastructure, geographical data and infrastructure networks.

In order to put such a framework into practice, the strict usage of parallelisation techniques and efficient algorithms is necessary and is introduced in detail. These techniques make it possible to create a framework which can provide access to multi-resolution representations of large data sets fast enough to put into practice a complete exploration and simulation pipeline, which ranges from a multi-monitor or CAVE-based real-time visualisation to multi-scale simulation scenarios such as flooding of urban regions.

Since it is clear that the expert knowledge of engineers is indispensable in order to set up simulation scenarios which deliver accurate and reliable results and insight, the general interoperability with existing visualisation and simulation environments is shown for various scenarios. Finally, the application of an urban flooding simulation with the incorporation of pipe network interaction is given. The capabilities of the framework allow the coupling of the different simulations to be handled efficiently, and insight is gathered over all scales from the global flow behaviour to the impact on single construction entities.

1.3 Structure

This work is organised as follows. In Sec. 2 the various data sources ranging from fully detailed product model descriptions of constructions and built infrastructure, and digital city models through to geospatial large-scale descriptions are given. These different sources are used to form a complex multi-scale data basis by fusing all input data to a common data set and enriching the coarse scale data with the fully detailed product model descriptions. Furthermore, a multi-resolution description of the complete data set is derived by introducing a level of detail approach to the whole data assembly.

In Sec. 3 the centrepiece of this work is presented: a parallel data access and processing framework which is capable of handling the complete model of a city with all constructions and built infrastructure. The hybrid parallelisation concept for this framework follows; this is essential to handle such large amounts of complex multi-resolution data. In order to show the applicability of the framework and prepare real-life applications, coupling interfaces to data exploration, visualisation and numerical simulation approaches are developed within this dissertation project. This chapter closes with a load distribution strategy based on space-filling curves and an investigation of its quality.

In Sec. 4 the visualisation and exploration of the framework presented is developed. Starting from the visualisation of the hierarchically ordered multi-resolution product model data, the integration of simulation data is prepared. All approaches are transferred to multi-monitor CAVE-like installations. In order to investigate the full complexity of the underlying data,

adequate hand-held navigation for such visualisation environments is introduced.

In order to open the door for various numerical simulations, the multi-scale representation of the underlying data is interfaced in Sec. 5 in order to drive general purpose grid-generation and parallel numerical simulation frameworks. Based on the coupled frameworks, various numerical simulations are introduced and performed in Sec. 6. These simulations use the inherent multi-representation approach of the framework presented and therefore allow simulations based on almost any computational domains and resolutions.

In Sec. 7 the applicability of the complete approach presented in this work is shown by simulating urban flooding with the pipe network interaction of a city and its buildings. In order to investigate the effects of a drainage system collapsing due to a heavy rain scenario in a city, a three-dimensional parallel free surface flow simulation is incorporated with the interaction of the one-dimensional pipe-network flow. The holistic approach enables the behaviour to be investigated on the city-wide scale and allows the adaptive investigation down to the scale of buildings and construction details.

Chapter 2

Input Data and Preprocessing

In this chapter, the data basis for the framework with its supported input formats is introduced and the compatibility with almost any data format is shown. Afterwards, the preprocessing steps performed are presented, which cover the generation of stepwise coarsened representations of the fully detailed data. The definition of a metaformat is then introduced, in which all data formats supported are stored in an efficient and fast manner, and the conversion algorithms to this format are given. Finally, a data fusion algorithm is presented, which organises data originating from different scales and fuses them to one basis as a large set of complex data.

The data basis covers, on the one hand, data for constructions and built infrastructure, which describe buildings, bridges, etc. Furthermore, geographic data are integrated, from which large-scale information such as the elevation and the texture of the surface and land usage information, and also pipe and power supply networks are derived. The introduction of data from numerical simulation follows with their creation and is therefore postponed to the following sections.

After the description of the data types supported, level of detail (LoD) algorithms are introduced. These algorithms make it possible to stepwise coarsen a complex structure by means of a simpler representation such that the coarsening error made is acceptable in the situation to which it is applied. A good example is the stepwise coarsening of the terrain during visualisation. When exploring individual details, a fully resolved mesh is used for visualisation. The larger the scale of investigation, the coarser the mesh can be made, for example by collapsing individual elements and approximating them with the average of the collapsed elements without obtaining a not negligible error - at least for visualisation. Three level-of-detail algorithms will be introduced, which cover different aspects and application scenarios. The first algorithm implements LoDs for product model data and coarsens the fully detailed description of a construction by approximating individual complex parts with their bounding box. The second algorithm computes the outer shell of the product model

and exploits the Graphics Processing Unit (GPU) to speed up the computation. A third algorithm is introduced, which relies on approximating complicated structures with textured polyhedra. This computation is also performed on the GPU and provides good speed-up possibilities for visualisation purposes.

After introducing the data types supported and levels of detail, a meta-format is presented, which is capable of storing and providing read access to a single model with all its LoDs and auxiliary information in an efficient and fast manner. This binary format makes it possible to read individual parts such as an individual LoD or specific auxiliary information without reading the whole file into main memory and therefore it is well suited for interactive and parallel access.

Finally, the topic of enriched data sets is covered. This comes from the fact that one of the application scenarios of the framework presented is to develop a system which makes it possible to investigate urban flooding on multiple scales, ranging from the overall flooding behaviour in the city to the effects on specific construction details. Geographic information systems (GIS) provide information on the elevation and usage of the surface, whereas city models give the assembly of buildings within their surroundings, and Building Information Models (BIM) define a single building in full detail. Building Information Models distinguish from geometric models by the fact, that besides the geometric representation particularly the rich semantic description is stored.

What is not available is a complete, fully detailed specification of a city or even a district with all its buildings, but this is imperative for the framework presented in order to show the applicability and feasibility of the approach. In order to overcome this obstacle, an algorithm has been developed and implemented which enriches a given city model with building product model data based on a similarity criterion. This algorithm is based on a metric and replaces the coarse representation of single buildings with fully detailed synthetic product models and results in a scenario which has the complexity of a necessary, but not available real world data set. Based on this enriched data set, which has been generated for the city centre of Munich, the algorithms and approaches presented in the following are tested and applied.

2.1 Building Information Modelling

Building Information Modelling (BIM) is the ongoing process of describing the data of a construction in a digital product model over the whole life cycle [1] of the construction. This ranges from the planning process and the construction work through to the maintenance during the life time of the building and also to the deconstruction of the building. It must be pointed out that BIM is not restricted to a single purpose or a single discipline such as architecture, construction or maintenance [2] but is a holistic approach for handling building

data in the architecture, engineering and construction (AEC) industry [3, 4, 5, 6, 7]. A strict adherence to BIM can overcome the problem that information which is not managed efficiently loses its value, leading to a lack of insight due to disjointed or incompatible information handling [8, 9]. In the latest research, BIM has been extended from a static description of constructions to a dynamically evolving information database, supporting 4D modelling and parametric descriptions of relations in order to cope with the challenges of the development processes [10, 11, 12, 13].

2.1.1 Industry Foundation Classes

Industry Foundation Classes (IFC) are an open standard and provide a specification for performing Building Information Modelling. It is an object-oriented data model for interoperability in the AEC industry and is maintained by buildingSMART [14].

IFC use the Standard for the Exchange of Product model data (STEP) [15] ISO 10303 for storing the relational description of the product model. A product model for the main building of Technische Universität München, Munich, Germany is given in Fig. 2.1. The model has been developed in a student project conducted at the Chair of Computational Modelling and Simulation at Technische Universität München.

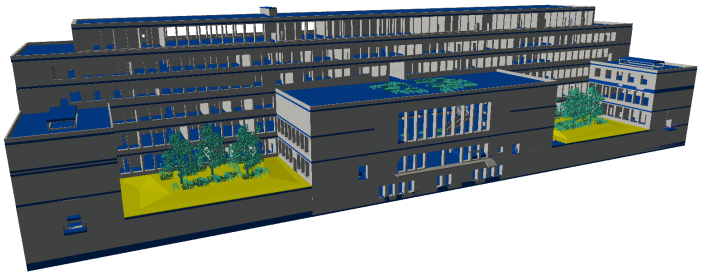

Figure 2.1: IFC product model of the main building of TUM [16].

IFC are currently the best choice and the industry standard for performing BIM and exchanging product model data for constructions. Over the last decades, IFC have undergone a development process and this is still ongoing. In March 2013 the latest update of the standard, IFC4, was launched and relies on XML as the data format [14].

IFC can deliver the complete specification of a construction or built infrastructure, ranging from the fully detailed geometric representation to the auxiliary information such as relations, a window as an opening element of a wall, for example, or measured data such as the floorspace of a room, insulation class of the glass in a window or the address of an installed network port. IFC is therefore well suited as one data source for the purpose of the framework presented as it provides feature-rich data and allows general interoperability by being the standard for storing building product model data.

On the other hand, it has to be mentioned that IFC is also still in a development process. There are still unmet challenges in describing construction details properly. For example, the intersection of walls is not defined exactly. For a detailed review of the unanswered modelling questions in IFC, the reader is referred to [17, 18].

Besides the specification of buildings IFC are extended to the description of constructions and built infrastructure facilities such as bridges, tunnel segments and roadways [19, 20]. The specific modelling demands of different types of built infrastructure are an active field of research.

2.1.2 CAD Data and 3rd Party Building Information Models

As described in Sec. 2.1.1, IFC are a standard data format for describing the product model definition of constructions and built infrastructure. In order to bring BIM to life, it has to be supported by industry, and software vendors have gone to a lot of effort to incorporate and support IFC. All large developers of computer-aided design (CAD) software provide and are actively developing import and export interfaces to and from IFC.

At the moment there is already support for the major and widely distributed software tools such as AutoDesk Revit or Nemetschek Allplan, to name but a few, but the consistency in respect of the import and export features is still limited. It is therefore still only a semi-automated process to bring product model data from a construction tool to IFC, and not the whole feature set of all construction tools is compatible and transferable to and from IFC. For a detailed review of the interoperability of IFC with specific CAD tools, the reader is referred to [21].

Nevertheless, there is the process of transferring basically any 3D models, drawings and construction plans with their auxiliary information from all major software tools to IFC. For the purpose of the work presented it is therefore sufficient to stick to incorporating IFC as a data source for product model data from constructions and built infrastructure.

2.2 3D City Model

3D city models are schemes for representing a district, a city or even a country with its buildings, land usage and infrastructure by assembling the entities covered in the area of interest. Today, many institutions provide detailed 3D models of cities and the number is still increasing; 3D models of Berlin [22, 23] and London [24] are available, to name but two out of many.

The common standard for 3D city models is City Geography Markup Language (CityGML) [25, 26, 27] which is based on and extends the Keyhole Markup Language (KML) [28]. CityGML

fuses data from different sources in order to provide a product model for cities with their 3D objects. It must be pointed out that CityGML does not only focus on the geometric representation of buildings and their visualisation, but also covers semantic and relational information and therefore functions as a common interface model for virtual 3D city and landscape models [29]. Fig. 2.2 shows the city model of Berlin, Germany [22].

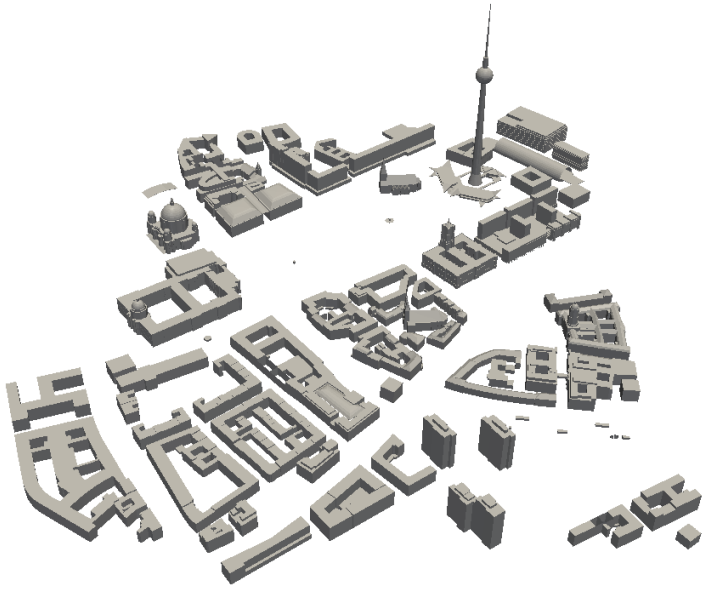

Figure 2.2: CityGML model of Berlin, Germany [22].

Its definition, growing coverage, and availability for domains of interest make CityGML well suited for many applications beside visualisation as shown in [30] and these range from urban planning [31, 32] and disaster management [33] through to public affairs.

Nevertheless, it must be pointed out that there are unresolved challenges in implementing CityGML which are addressed differently [34]. Files tend to become very large and can easily reach many gigabytes in size for complex scenarios; furthermore, city models can also be distributed over multiple files and then the linking between elements over different files becomes an issue.

2.3 Geographic Information System

A Geographic Information System (GIS) is a data processing system which focuses on handling spatial data. In [35] a GIS is defined as follows: *"A GIS is a computer-based system to aid in the collection, maintenance, storage, analysis, output and distribution of spatial data".*

A GIS basically handles two different types of data, raster data and vector data. Raster data are sets of discretised values on a usually uniform, two-dimensional grid. Typical examples of

raster data are elevation maps and orthophotos for specifying the shape and representation of the surface of the earth. Vector data are in general a set of geometric primitives such as lines, polygons, or points and so on.

Another important aspect of GIS modelling is georeferenciation and the processes of mapping input data coordinates to spatial real world coordinates. As the surface of the earth is similar to the surface of a sphere, the question arises of how to map 2D coordinates of a map, for example, to the 3D coordinates of the earth. In general, two different coordinate systems are used. The first one is the geographic coordinate system and in this method, the coordinate is defined by the two angles of the polar coordinates of the spherical surface. The second one is a projected coordinate system as is used for the Gauss Krüger coordinates; for projected coordinates the 2D plane on which the points are defined is projected to the surface of the earth [36, 37, 35].

2.3.1 Terrain Specification

The terrain specification is a typical type of raster data in GIS, consisting of the elevation map, giving the height values of the surface on a (regular) grid, and orthophotos generated by aerial picture taking. From the elevation map a 3D mesh is generated on which the texture derived by the aerial photos is projected.

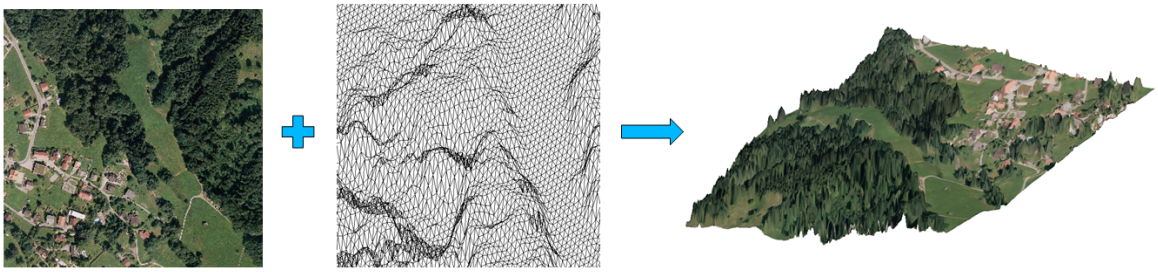

Figure 2.3: Realistic terrain representation is achieved by mapping the conforming texture to a surface definition given on a (uniform) grid.

Fig. 2.3 shows a dataset provided by the Chair for Geoinformatics at Technische Universität München, which very accurately describes the area of Voralberg in Austria. The dataset has been generated and provided by the federal state [38] and is provided for the scientific community; a further source for GIS data is [39] and the integration of CityGML models [40].

2.3.2 Pipe and Supply Network Data

Pipe and supply network data are typical types of vector data. Usually networks are defined as a graph, where the nodes are the intersections of the pipes and the edges are the pipes. The

graph can be represented by a node list, which provides the exact position of the intersection of the network, and an adjacency matrix. The adjacency matrix has the dimension of the length of the node list and stores k_{ij} at an entry if there is a connection between node i and node j. Usually 0 indicates no connection and a value different from 0 indicates a connection. These values can be encoded only with yes or no if a connection exists, store the weight of the edge, or be a lookup key table where, in the case of a pipe network, the diameter of the pipes, roughness values or the material the pipes are made of is stored.

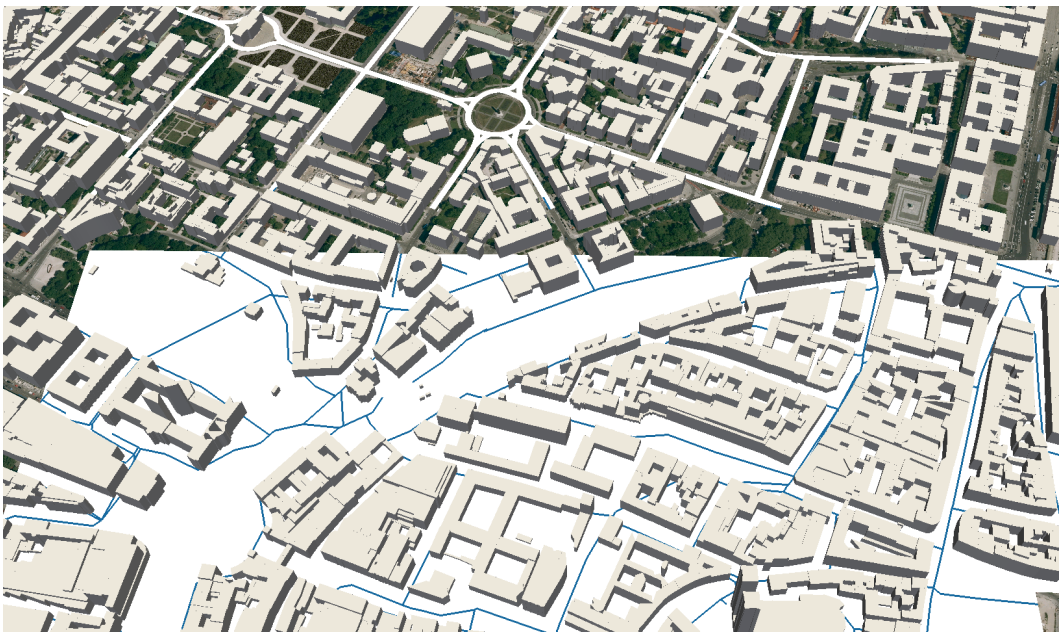

Figure 2.4: The pipe network of the sewer system in the city centre of Munich, Germany for a domain of approximately 2 square kilometres.

Fig. 2.4 shows the pipe network [41] of the sewer system in the city centre of Munich measuring approximately 2.6 km by 0.8 km. The dataset is provided by the Chair for Geoinformatics at Technische Universität München. It consists of over $1,800$ pipe segments having a total length of over 38km.

2.4 Levels of Detail

In order to process large data sets and complex geometries, one of the essential issues is how to select the part which is of interest and necessary for the application from a vast amount of information, and how to work out which part can be ignored. Furthermore, a decision has to be made as to the resolution at which the individual data of the set investigated have to be processed.

Of course this is a highly situation-specific and application-specific question, one of the driving parameters is the scale of the domain of interest. On a large scale, the individual elements generally have a coarse representation or resolution, as the number of individual elements is high. On a fine scale, only a couple of elements will be investigated, but these elements are highly resolved.

A good example for a description using the concept of levels of detail is the visualisation of a city model. In order to visualise the whole model of a city, individual houses can be approximated by their coarse outer shape and details such as the exact outline of the windows can be neglected. Even finer details such as the handrail of the stairs inside the building are not even visible. This is just one example, but it already shows that a model has to have multiple representations - called levels of detail (LoD) - in order to decide which level to apply at the moment the model is processed.

In the literature, it is common to use five different levels of detail [27, 42] for buildings and the assembly of buildings in city models, starting from the coarsest and progressing to the finest. This is also known as aggregation and generalisation approach.

1. LoD0 holds just the elevation model as an extruded (textured) surface mesh.
2. LoD1 holds approximations of individual buildings with their bounding box.
3. LoD2 extends LoD1 with the shape of the roof and textures for the facade.
4. LoD3 contains the full outer geometry of every individual building.
5. LoD4 holds the fully detailed description of the buildings including interior and installations.

In the following these LoD definitions will be introduced and additional formulations given in order to cover all the different demands made by the current setting for which this framework is used.

2.4.1 Multi-Resolution Raster Data

As introduced, LoD0 gives an approximation of the domain of interest with a textured surface mesh. From the GIS sources introduced in Sec. 2.3.1, a mesh for the elevation and a texture can be derived which leads directly to LoD0.

Even if this is the coarsest LoD defined, it does not indicate a low density or easy to process data. Terrain data are available at very fine resolutions as shown in Sec. 2.3.1. In order to provide the right resolution of raster data for a specific scenario, down sampling methods have been implemented within this dissertation project.

For input raster data $M^0 = m_{i,j}$ a down sampling of level k is defined by $M^k = m^k_{i,j} = m_{k \cdot i, k \cdot j}$, where the number of raster points of M^k decreases as the square over k. Values are then interpolated linearly over the domain of M^1. Fig. 2.5 gives two raster data sets, a surface mesh and a terrain texture, with three coarsening steps.

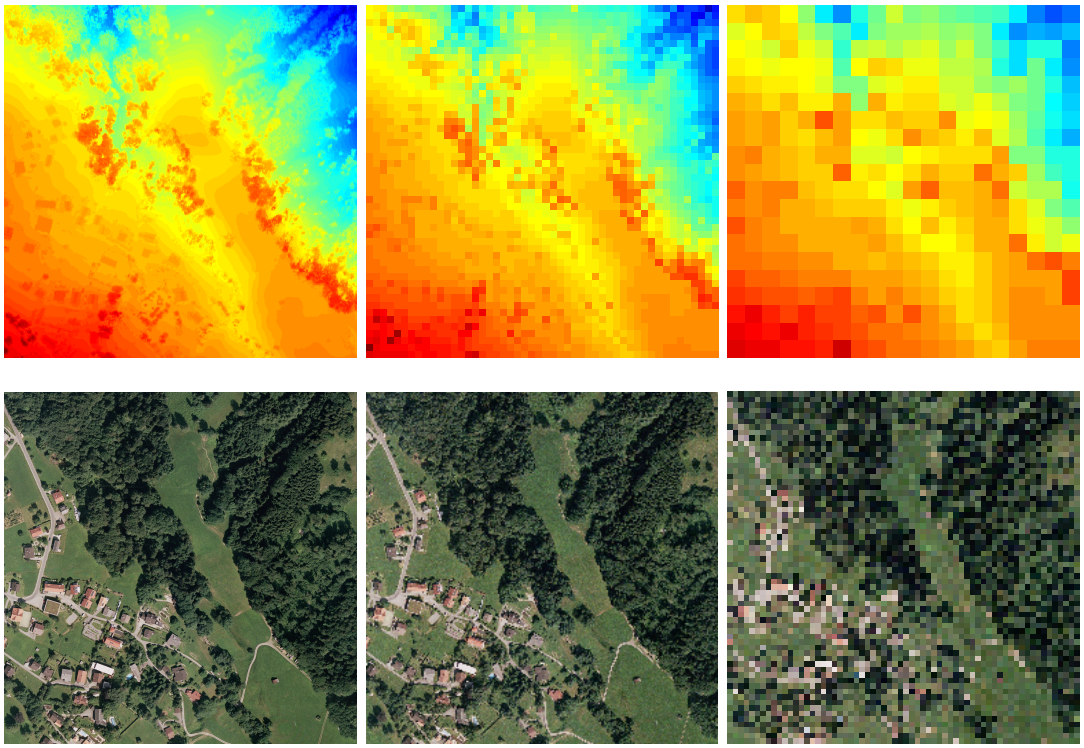

Figure 2.5: Two raster data sets describing a surface (top) and the corresponding texture (bottom), both with a stepwise down sampling for $k = [1, 200, 800]$ (f.l.t.r.). The dataset is provided by the Chair for Geoinformatics at Technische Universität München.

There are approaches in which highly resolved terrain elevation and texture data are pre-processed by using efficient encoding algorithms and therefore terra scale datasets can be processed in real-time [43, 44].

2.4.2 Stepwise Coarsened Product Model Data

As introduced, LoD4 contains the fully detailed description of a building including its interior and installations. As this LoD consists of the non-coarsened data, it can be derived from BIM data as introduced in Sec. 2.1 directly, but BIM is capable of storing much more detailed information. Besides the detailed geometric representation, auxiliary information such as measured data, material parameters or performance values are also available.

As IFC are used in this work for handling BIM data, the derivation of the explicit geometry and auxiliary information has to be performed. This is done by using the IFCEngine [45]

library. IFCEngine interprets a given IFC file and delivers the geometric representation as an indexed 3D triangular mesh together with the mapping of auxiliary information to the identifiers of the mesh, which are the individual building elements. The triangular mesh can be stored directly and the mapping of the auxiliary information is stored as a set of arrays where each holds the information for a single element of the product model. The UML specification of the resulting discretised product model representation is given in Fig. 2.6.

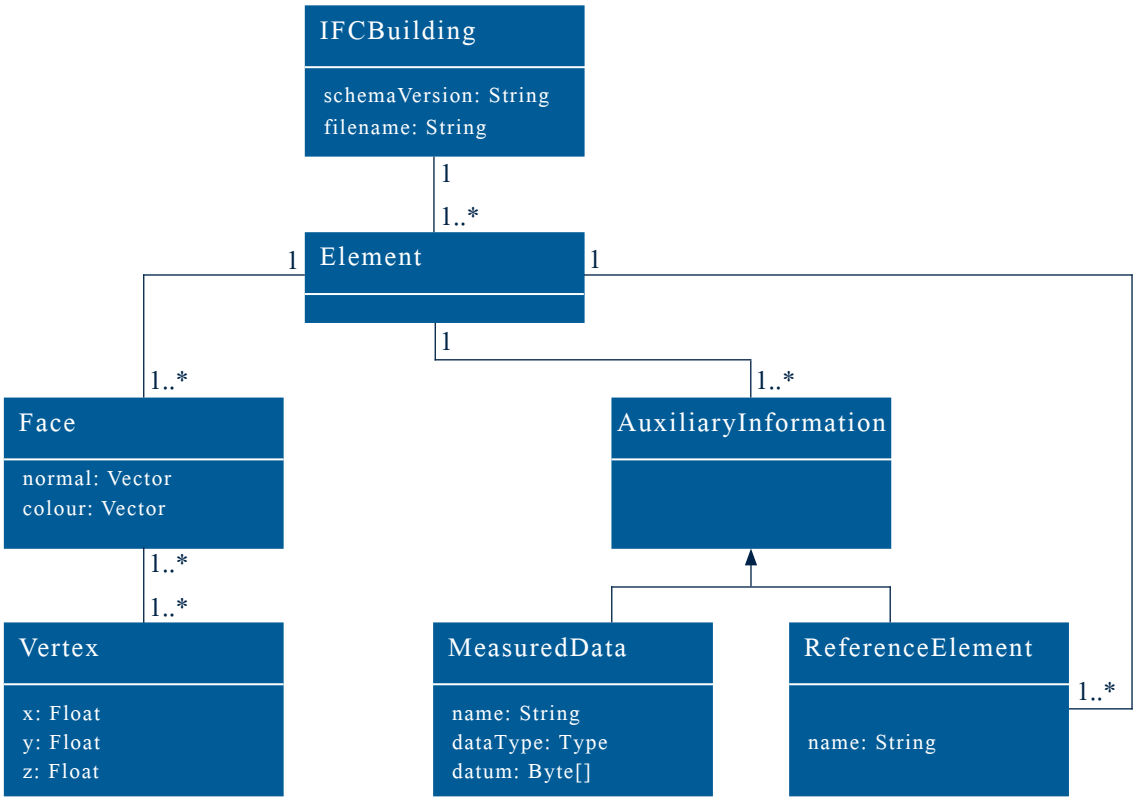

Figure 2.6: A discretised product model representation of an IFC file stores the list of indexed faces, which give the fully detailed geometry of the model. For every face, the identifier gives the mapping to the corresponding building element. Every building element is stored with the full set of its auxiliary information.

This representation now provides LoD4 directly for a given IFC model, LoD1 and LoD2 are derived by performing stepwise coarsening of the LoD4 data.

In a first step, all individual elements such as doors or windows are approximated individually with their bounding boxes. In a second step, all elements of the same type are approximated with their bounding boxes. In a third step, the group of elements is approximated with its bounding box which provides the bounding box of the model. The pseudocode of the algorithm is given in Alg. 1.

Fig. 2.7 shows the geometry derived and all three coarsening steps for an IFC model with a representation of 1.1million triangles and 3.4million vertices. Furthermore, the model con-

Algorithm 1 *calculateBBoxApproximation*

```
void calculateBBoxApproximation(Object building){

 BBox buildingElements[];
 BBox buildingElementGroups[];
 BBox buildingOutline;

 for (each e in building.elements())
  buildingElements[e.elementID].adaptBBoxWith(e);
 for (each be in buildingElements)
  buildingElementGroups[be.elementTypeID].adaptBBoxWith(be);
 for (each beg in buildingElementGroups)
  buildingOutline.adaptBBoxWith(beg);
}
```

tains a set of over 100k pieces of auxiliary information. It should be pointed out that this information covers not only measured data but also the mapping between elements. By having this for a window, for example, information is stored as to which wall contains this opening element, in which building the floor is located, and what the coordinates of the building are.

The algorithm presented for generating levels of detail for product model data is performed per IFC file and at the preprocessing stage of the framework. In Tab. 2.1 a listing of some of the processed IFC files is given together with the coarsening achieved concerning the quantity of triangles needed for the respective LoD representation.

2.4.3 GPU-based Generation of Levels of Detail

In the following two further level of detail formulations are presented. These algorithms are well suited for graphics accelerators and therefore their GPU-based implementation is given, even if the formulations are independent of the specific hardware they are performed on.

Generation of the Outer Shell

As introduced, LoD3 contains the fully detailed description of the outer shell of a building. In order to generate this LoD of a product model, an algorithm has been developed which uses the capabilities of the GPU to retrieve the subset of the fully detailed geometric description which forms the outer shell. Usually a reduction of up to 90% is achieved. This reduction leads to a representation which is loss free as long as the domain of interest is on the outside of the buildings. Furthermore, the complement of the outer shell provides the representation of the inner (structure) of the product model, see Fig. 2.8.

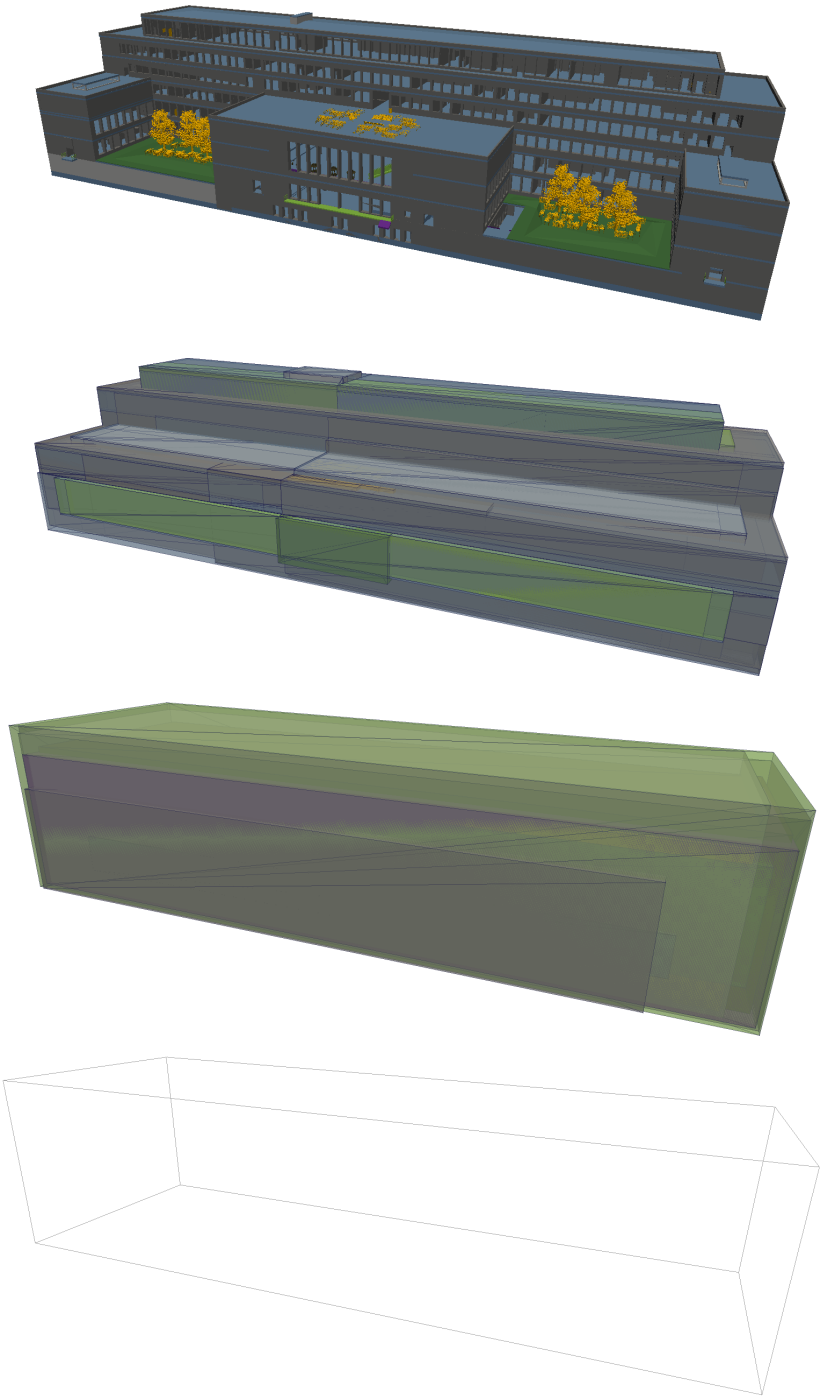

Figure 2.7: The fully detailed geometric representation of a building product model is coarsened by the set of bounding boxes for each individual building element. All bounding boxes of the same type of element are grouped and approximated by their bounding boxes. The bounding box of all groups of element types gives the bounding box of the product model (f.t.t.b).

The algorithm developed works as follows. As introduced in Sec. 2.4, the fully detailed representation of the product model is derived and stored as an index triangular mesh. The mesh is scaled and translated such that its bounding box fits the 3D unit interval. From

Description	#Triangles init	step 1	step 2	step 3
Studienarbeit BIC Gefägnis	307532	10436	3402	48
BIC Studienarbeit Haus RG	165614	6450	1026	48
Studienarbeit CAD-Haus RB	73657	2690	978	48
BIC Bürogebäude-JG	55922	1442	834	36
ProjektBIC Villa-MP	53659	1919	762	42
Testat-Bic Haus-AS	50193	1693	564	48
Einfamilienhaus	46929	1355	750	36
Studienarbeit Haus-LT	45223	1121	714	54
3D BIC Haus-MH	44874	1498	714	36
3D Haus-RP	44307	1510	684	36
Bic-Studienarbeit-Haus LG	42648	1305	564	48
Studienarbeit Bic (Haus)	41170	1469	624	42
BIC Einfamilienhaus-PW	36358	1286	540	36
3D-BIC Haus-MW	35094	1172	594	42
Einfamilienhaus-YP	29841	1059	600	48
BIC Studienarbeit Wohnhaus-LK	29788	927	498	48
Studienarbeit Haus-NY	28235	799	390	24

Table 2.1: By performing the stepwise level of detail coarsification presented, the number of triangles to be processed is reduced iteratively.

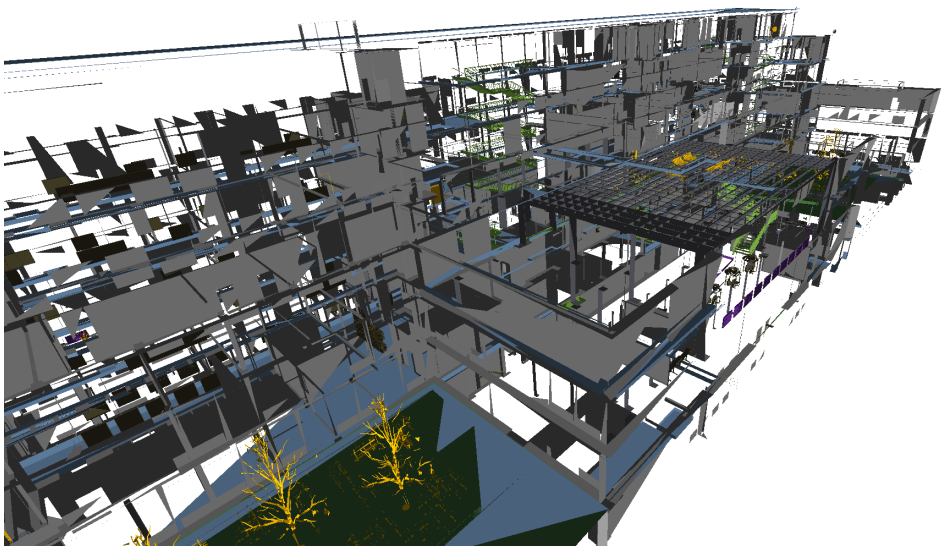

Figure 2.8: After identifying the outer shell of a fully detailed product model, the complement of the shell with respect to the product model gives the inner structure of the construction.

this triangular mesh all windows are identified; this is directly provided by the auxiliary information. Before processing the data further, all windows are removed from the mesh and the mesh is re-indexed with one identifier per triangle, not per building element, in ascending order starting with 0. The algorithm takes advantage of the GPU's very fast ability to detect

hidden pixels. Every triangle of a given mesh is assigned a unique colour value. All triangles are processed as for visualising, and from the colour of the visible triangles the identifier (coded in its colour) is retrieved as given in Fig. 2.9.

Now, the visible triangles at that special orientation of the product model are derived. Then, the product model is rotated along all axes and the indices are collected. By doing so, the indices of all triangles are identified which are visible from the outside.

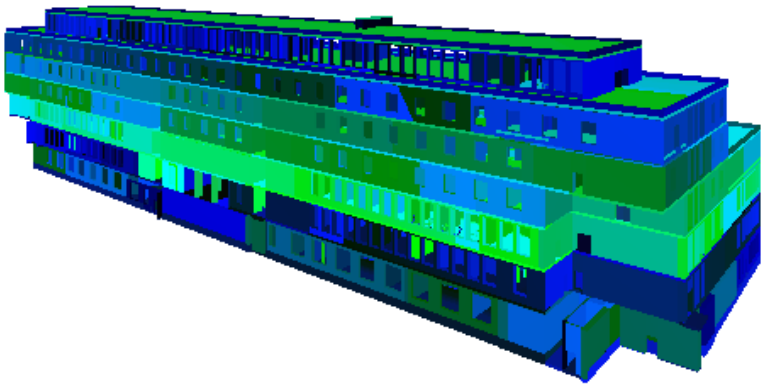

Figure 2.9: The indexed mesh of a product model is rendered with colour-encoded identifiers of the triangles. The frame buffer of the scene rendered contains the colour value of the visible triangle for every pixel and therefore the information of all visible triangles for that view.

Alg. 2 gives the pseudocode for the algorithm. Fig. 2.10 gives the increasing number of identified triangles over the rotation steps. Both rotation angles are discretised with 72 steps each. It can be seen that by rotating around the horizontal axis in the inner loop more triangles are already identified with the first full inner rotation. By performing the horizontal rotation on the outer loop, the identification of the triangles increases on reaching each main axis of the building.

This algorithm is performed in a preprocessing step and the accuracy can be adapted in the following way. As the GPU is not limited to the resolution of a monitor and can also be used without, the resolution of the frame buffer can be set to a chosen value in order to achieve the desired accuracy. In Fig. 2.11 the derived outer shell with a reduction of over 90% for a building product model is depicted.

Tab. 2.2 gives a listing of some of the processed IFC files together with the reduction achieved in relation to the number of triangles needed for the corresponding LoD representation.

Textured Polyhedra

One of the key applications for level of detail formulations is to speed up visualisation. Therefore an LoD has been implemented within the dissertation project which brings about a great simplification of the product model by representing its outer view.

Algorithm 2 calculateOuterShell

```
void calculateOuterShell(Object building){

 vector[bool] visibleTriangles[building.triangleCount()] = false;

 for (each e in building.elements())
  buildingElements[e.elementID].color = toBasis256(e.elementID);

 for (float angleX = 0; angleX<360; angleX++){
  for (float angleZ = 0; angleZ<360; angleZ++){

   e.rotateMesh(angleX, angleZ);
   frameBuffer = render(e.mesh());
   for (each p in frameBuffer.pixels())
    visibleTriangles[toBasis10(p.color)] = true;
   }
  }
 }
}
```

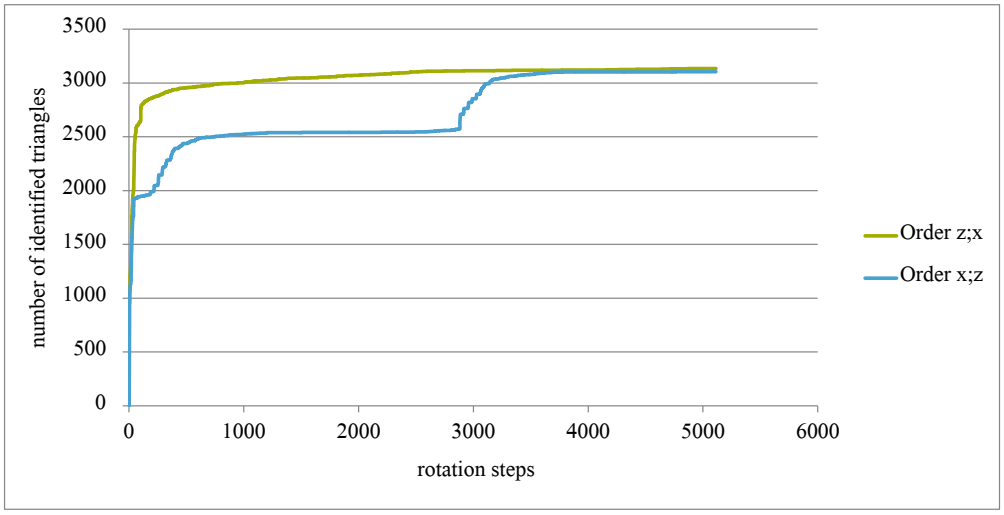

Figure 2.10: The rotation of the product model is discretised over the two rotation angles. As for every single rotation step the visible triangles are identified and added to the triangles already found, the number of triangles increases.

One of the most prominent representations of complex geometries is the derivation of textured polyhedra [46]. Especially for building models which consist primarily of glass, the highly detailed small-scale representation of the outside still ends up in a high number of primitives, as most of the complex inner parts are also visible from the outside. In order to achieve a reduced representation, the complete product model is approximated by a simple geometry such as its bounding box or its convex hull, and every face of this simplified representation is textured with the corresponding view of the original geometry.

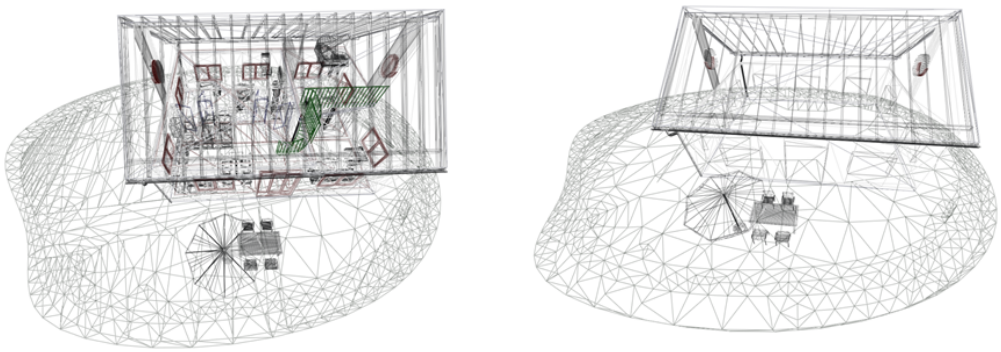

Figure 2.11: On a fully detailed BIM model (left) the outer shell is generated using the GPU-based algorithm for identifying the visible triangles forming the outside. In order to gain an insight into the reduction of about 90% from 84578 to 6686 triangles, the wire frame representation is given (right).

	#Triangles	
Description	full	outer shell
TUM Building1	1097258	213715
BIC Haus-MD	109626	24418
CAD-Studienarbeit-Haus RS	7556	1731
Studienarbeit Bic Haus-NT	53834	12701
3D-Ansicht Haus-DH	6668	1579
BIC Studienarbeit Haus RS	138541	33024
Revit Haus-MG	88108	21006
Studienarbeit CAD Haus-KF	82985	20293
BIC Studienarbeit Haus-SK	80085	20847
BIC EF-Haus	57941	15362
Bic-Studienarbeit Haus-GW	1356	365
Reihengarage	1418	391

Table 2.2: By performing outer shell calculation of fully detailed product models, a reduction of up to 90%can be achieved.

These views can originate from real data such as photographs [47] or be derived from the discretised product model representation. When using photographs, texture mapping is a semi-manual process where the photographs are usually provided or taken by the user and then assigned to the simplified geometry.

In order to automatically derive a texture for a product model, the simplified geometry has first to be identified. Fig. 2.12 shows a product model with its bounding box and convex hull. For the simplified geometry, the projection of the view along the normals of every face is computed and stored as a texture. For reasons of convenience, all textures of the faces are usually stored in a single picture and can be referenced by their relative position in the texture. Fig. 2.13 gives the resulting combined texture of a product model's bounding box.

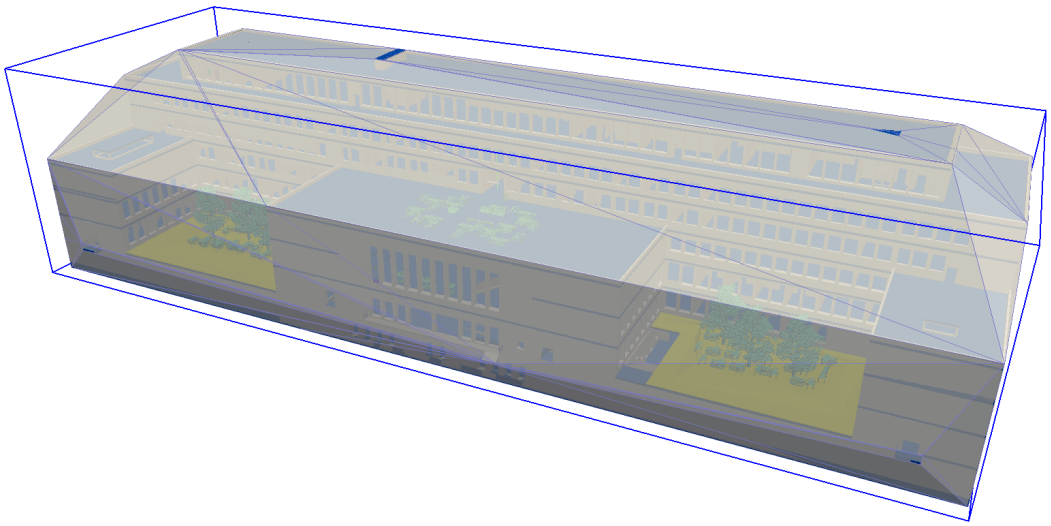

Figure 2.12: In order to derive a textured representation of a product model, the complete model is approximated by a simplified geometric representation such as its bounding box or the convex hull.

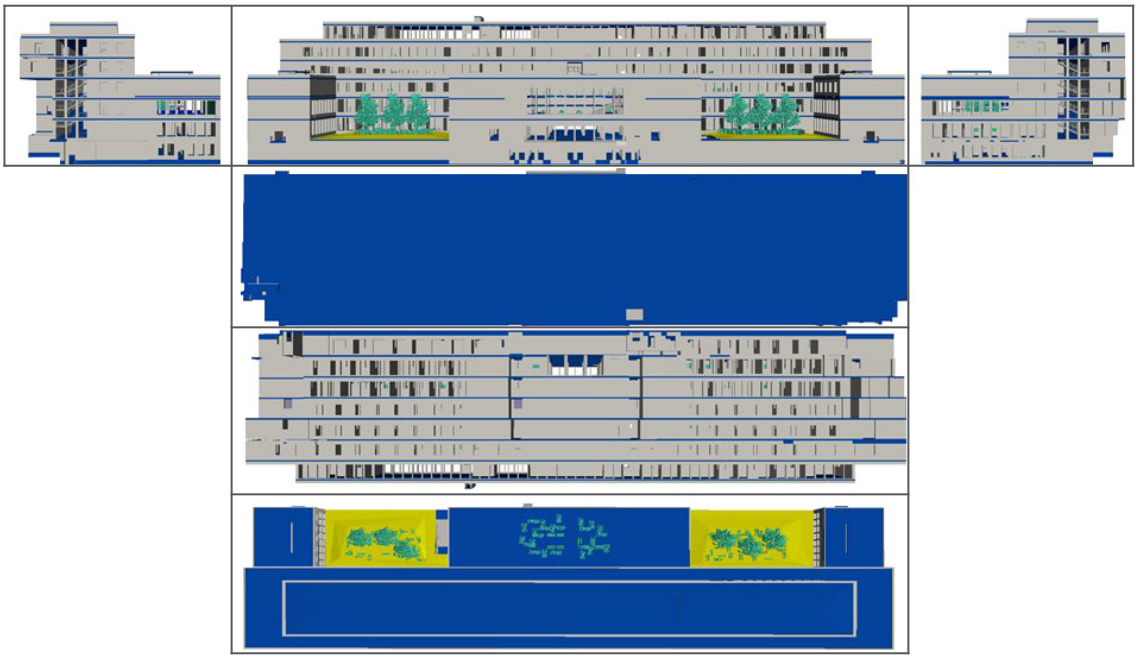

Figure 2.13: The bounding box of a product model identifies the - in this case six - projection planes for the generation of the textures. The projection of the view along the normal of the face is computed for every face and all textures are combined to the representation of the model.

It should be pointed out that besides the bounding box and the convex hull as a simplex for texturing, almost any geometric representation can be chosen including hierarchical representations. The process of computing the texture along the normal of every face applies to general solids.

2.5 Multi Resolution Meta Format

In order to form an efficient data basis, a meta-format for storing and accessing multi-resolution geometries has been developed and implemented. The key demands for this data basis and therefore properties of the format are as follows:

- Conversion and storage routines for all data types introduced
- Storage of all LoDs introduced
- Support of auxiliary information
- Binary storage for fast access to the data especially for read access
- Direct access to individual information without the necessity to read the whole file and search through the data

Particular attention should be drawn to the direct access to individual parts of the data. When processing large data sets, many of the individual elements are needed and processed only on a very coarse LoD. Therefore it has to be possible to access a single level of detail of an individual product model without reading and storing the usually large, complete specification of the model in the main memory. In the literature, it is common to apply level of detail approaches and use hierarchical storage schemes for geometries. [48, 44]. The need for covering multiple geometric representations where each of them has a sub-structure which groups single elements, augments them with additional information and links them to each other, led to the implementation of the meta-format presented.

Looking at the coarse structure of the format it consists of four parts: header, polygonal representation of the geometry, texture data, and auxiliary information. The header contains an identifier of the file which is unique among all files processed by the framework, a short description, and the information as to whether one geometry is split up among multiple files. Splitting up data and not combining them to a single file can be of interest especially for large-scale data such as terrain data or the definition of pipe networks.

	Field Name	Size[byte]
Header	fileId	Integer
	description	Char[256]
	numberOfFile	Integer
	countOfFiles	Integer
	sizeOfPolygonalData	Integer
	sizeOfTextureData	Integer
	sizeOfAuxiliaryData	Integer

The header has a size of $256 + 6 \cdot sizeof(Integer) = 280$ bytes. When opening a file, initially the first 280 bytes are read. From the information in the header the total file size is available. Therefore direct access to specific data is ensured on this first coarse level. In order to load just the polygonal description of the geometry with all levels of detail, only the segment

$$[280 + 1; 280 + sizeOfPolygonalData]$$

bytes has to be loaded. In order to load just the texture data with all levels of detail, only the segment

$$[280 + sizeOfPolygonalData + 1; 280 + sizeOfPolygonalData + sizeOfTextureData]$$

bytes has to be loaded. In order to load just the auxiliary information, only the segment starting at $280 + sizeOfPolygonalData + sizeOfTextureData + 1$ bytes has to be loaded.

The polygonal data contain the geometric representation of all five LoDs generated for a model. At the beginning of the block, two arrays with five integer values each are stored; they contain the number of vertices and the number of triangles the LoDs consists of.

This header is followed by the list of all vertices, the order being such that initially all vertices of LoD0 and then LoD1 until LoD4 are stored. Every individual vertex is stored with its coordinates, the vertex normal, the colour value, and the texture coordinates.

The list of the vertices is followed by the list of the faces. The specification of a face contains an identifier and the list of the vertices forming this triangle. In order to also store dimensionally reduced information such as the definition of a pipe network as a graph, how many vertices form a face is also stored. For triangular elements this is three, for the definition of a pipe network as line elements this is two.

With these data the geometric description of an LoD of a model is complete, further information is stored as auxiliary information and will be dealt with later in this Section.

	Field Name	Size[byte]
Polygonal Data	countOfVertices	Integer[5]
	countOfFaces	Integer[5]
	verticesPerFace	Integer[5]
	vertexCoordinates	Float[3]
	vertexNormal	Float[3]
	vertexColor	Float[4]
	textureCoordinates	Float[6]
	faceId	Integer
	faceVertices	Integer[3]
	textureId	Integer[3]

Accessing a single LoD now follows the exact same scheme of identifying the different segments as for the polygonal, texture and auxiliary data.

It should be mentioned that this storage scheme of course contains redundant information. Usually the vertex information of a coarse level of detail is a subset of the fine representation, and the list of vertices could be saved as one single list of all vertices needed for the different LoDs. Nevertheless, the data are saved for each LoD in their entirety since this makes it possible to read a single level of detail from a file without having to read the full list of all vertices. In this work the possibility of accessing data directly is preferred over saving disk space with a non-redundant storage of the vertices.

The definition of the polygonal representation is followed by the texture data. The texture data consist of a set of images where the dimension is stored for each single image. Every texture is referenced by the texture ID which is saved with the face information.

	Field Name	Size[byte]
Polygonal Data	countOfTextures	Integer
	widthOfTexture	Integer[countOfTextures]
	heightOfTexture	Integer[countOfTextures]
	data	Float[4]

Finally, the set of auxiliary information is stored. This information is basically an array of structs, consisting of the face ID, name of the property, the data type with the length of the value and the data. The data type is encoded using a single byte indicating the enumerator of (Float, Double, Integer, Char, void*).

	Field Name	Size[byte]
Auxiliary Data	countOfRecords	Integer
	propertyName	Char[256]
	propertyDataType	Char
	propertyDataLength	Integer
	data	sizeof(propertyDataType)*propertyDataLength

The introduction of the different data sources and file types at the beginning of the Chapter has been kept short by intention. This is inspired by the fact that it will be shown that the approaches and algorithms developed and presented apply to complex data in general and do not exploit special features of specific data formats. Furthermore, it is not feasible in such a work to focus on all possible data formats and types in detail.

At this point, a data storage format as the basis of the framework introduced is available, and all further research and investigation is restricted to data of this type.

It should not be denied that this format limits the possible data sources of the framework presented and therefore the applicability in multiple ways. The geometric representation covers tessellated surfaces only. The derivation of volumetric elements has to be performed

by a subsequent application; free-form or high-order descriptions are not supported. Also the type of auxiliary information is limited to (sets of) data arrays which are assigned to construction entities. More complex specifications of buildings and their construction parts such as life-cycle information, histograms, simulation profiles, or parametric data require for more advanced storage routines. But the meta-format presented is a good starting point for further approaches and enables the demonstration of the applicability of the framework presented.

2.6 Data Fusion and Set Augmentation

At this point almost all data from three different sources have been introduced and are available in a common storage format. Fully detailed models of buildings and constructions are accessible not only with their geometric information but also with the auxiliary information of a full attributed product model. 3D city models are accessible on larger scales which give the assembly of the buildings in the urban context. Geospatial sources deliver large-scale information such as the terrain with its elevation and the texture. Furthermore, pipe network information for the city rain water drainage system is also available.

What is not yet explained but crucial for the work presented is the assembly of a sufficiently large data set in order to show the applicability of this approach. It should be mentioned that this is not self evident. The Chair for Computation in Engineering and the Chair of Computational Modelling and Simulation at Technische Universität München conduct research projects in order to derive high quality product model definitions. In Finland and Singapore, for example, it is required to provide the IFC description of public buildings.

The algorithm developed within this dissertation project is given in A.1.

Chapter 3

Framework

This chapter presents one of the key parts of this work: a data access framework which is capable of handling large sets of complex product model data.

An outlook concerning the demands placed on this framework can be obtained by anticipating and briefly introducing one of the applications of this framework. In Sec. 7, this framework will be applied in order to investigate the impact of heavy rainfall events on urban regions under the interaction of overground flow and the rain water drainage system. Ultimately, the following question is to be answered: If a heavy rainfall scenario occurs in a city with dense building cover, which parts of the city, its buildings and the equipment within the buildings are affected if the sewer network fails and parts of the city and buildings are not flooded by the rain but by the sewer system, which is unable to cope with the amount of water originating from the parts of the city where the rain is falling.

This setting directly introduces the key demands placed on the framework, which are the following:

Thousands of fully detailed building product models must be organised and access to them has to be provided on multiple levels of detail. This multi-resolution representation of the underlying fully detailed product model data must be available for every single model individually, as there are parts of the computational domain which must be highly resolved and parts which can be coarse or even be neglected. Additionally, the required resolution of individual models must be able to change over the runtime of the framework, depending on which parts of the computational domain are computationally demanding at that moment. The framework must provide this on the fly.

Besides the small-scale product model data, the same demands also apply to large-scale data. The fine detailed terrain specification of the city and the definition of the pipe network within the city must be provided by the framework.

It should be pointed out again that the multi-scale representation of the data is not limited to the geometry. Also, the auxiliary information is covered by the level of detail approach. The roughness of the individual pipe segments of the sewer network is kept, as is the definition of construction level details of building parts such as the information as to whether a flooded element is an electronic device and can be damaged by the water, for example.

Looking at this data basis of the framework it is clear that a hierarchical approach is inevitable. In order to estimate the required performance of the framework, the interfaces of the framework are of interest.

For investigating the behaviour of the fluid flow during the flooding in the city, numerical simulation will be applied and therefore, a mesh - a discretisation of the computational domain - has to be generated. This mesh will easily have a resolution of hundreds of millions of degrees of freedom in order to capture the computational domain accurately. Following that, the need for efficient parallelisation techniques is obvious. Even a hierarchical data structure and efficient algorithms will still end up with a computational load which makes parallelisation strategies inevitable.

In order to not only compute numbers, but also gain an insight about the impact of the flooding on the scenario, efficient postprocessing algorithms are essential. Therefore, a multi-monitor and cave-like visualisation will be presented in order to enable engineers, city planners and specialists to dive into the data and visually investigate the results, adapt model parameters or estimate the impact of the simulated quantities. This multi-tile visualisation of the data set covers not only the geometric representation of the product model data but also the visualisation of the results of the numerical simulation.

Handling large data sets is an active field of research and investigated for various cases. Pocket-size navigation systems perform multi-level access to the maps of a whole continent. GoogleEarth [49] provides interactive visualisation of the terrain and surfaces and provides basic semantic information for buildings, constructions or heritage sites over the whole globe. These approaches rely on distributed processing databases [50], cloud computing [51] or prefetching spatial structures [52]. The proposed application scenario – especially large-scale numerical simulation and multi-monitor visualisation – still needs new concepts. Existing distributed and cloud computing approaches lack the performance required concerning latency, network bandwidth and parallel computing efficiency.

Coming back to the field of handling fully detailed product model data, in the literature there are various approaches to coupling the different scales by building up a common database. This is achieved by extending the schema definition, integrating the different sources via Web Services and (relational) data bases [40, 53, 54]. All existing approaches lack at least one of the formulated demands. Either the approaches achieve a performance which makes the discretisation of the domain not feasible for massively parallel simulations, as the computa-

tional mesh derived for the simulation easily grows to billions of degrees of freedom during runtime. Or the transmission between different scales is not fully guaranteed such that either only coarse representations on the large scale or highly detailed information for small local scales are provided.

In order to cope with all demands with the proposed application, a dual layer hierarchical data structure is presented with its parallelisation strategy and interfaces for coupling numerical simulation, parallel visualisation and postprocessing of the data.

This chapter is organised as follows. Sec. 3.1 introduces the dual layer hierarchical structure which holds the global assembly of all data and the hierarchical product model scale representation.

Sec. 3.2 gives the hybrid parallel parallelisation strategy of the framework which can answer the request to the data set with the right representation at a desired resolution and fast enough. For these requests, a set of interfaces to the framework is presented which will be shown to provide an efficient coupling to different application scenarios. These range from the script-driven exploration of the data set through to the multi-resolution visualisation on semi-immersive environments and the coupling to high-performance grid generation and simulation frameworks.

In Sec. 3.3, the load distribution strategy is introduced. From the input data itself it is not clear which parts of the data set will be requested at a very fine resolution, which at a low resolution and which parts of the framework may not even be part of the computational domain. Therefore, a distribution of the data and thus of the computational load to many parallel processors has been developed which is independent of the specific request to the framework but still shows good scalability.

3.1 Hierarchical Data Representation

Hierarchical concepts are a standard in computer science and generally follow the principle that it may be advantageous when performing an algorithm on a set of raw data to initially structure the data and then use the hierarchy of the data gained. This principle is also referred to as divide et impera or divide and conquer [55, 56]. Of course, one has to invest something before one can earn something, and the investment may be worth it but does not have to be, if for example building up the hierarchy outweighs the gain in hierarchically ordered data.

A simple example is a large store which sells tools and the search for a specific screwdriver. If there is no order to the items for sale, every single item has to be checked to see if it is the screwdriver requested and this has to be done for every order. If the data are sorted, the

location of the screwdriver is separated from the shovels and the hammers. The screwdrivers are ordered into Phillips screwdrivers and flat screwdrivers and the Phillips screwdrivers are sorted by their size. In such a setting it is easy to find a certain screwdriver and it looks like this system is advantageous as the solution is found in a small number of steps. But this only applies if many orders are processed, if only a single order has to be processed, it makes no sense to initially sort the whole store.

Hierarchical data structures are a large field of research, this work makes use of one of the most prominent and frequently used three-dimensional hierarchical data structures - the octree, the two-dimensional version is usually called quadtree. Besides the octree, there are many further data structures such as region trees, R* trees and many others [57, 58]. The octree has been chosen due to its superior properties for the applied parallelisation and domain decomposition techniques. It efficiently provides a hierarchically ordered non-overlapping decomposition. The optimal suitability of the octree will be derived with the applications it is used for in the following.

An octree builds up a hierarchical representation of an object by bisecting the domain along every dimension and performing this bisection recursively for every sub domain which is intersected by the boundary of the object.

Expressed a little more formally, the octree generation can be introduced as follows: Given are a domain Ω, an object $\Gamma \subset \Omega$ for which a hierarchical representation is to be calculated, and a function $f(x) : \Omega \to A$, which assigns every $x \in \Omega$ a value $a \in A$ and A is a finite set. A typical application is to compute $\Gamma^0, \delta\Gamma, \bar{\Gamma}$, i.e. the inner, the boundary and the closure of Γ.

The basic algorithm for generating an octree representation works as follows: The octree is built up as a tree with root node $\omega = \Omega$ by performing the following hierarchical algorithm. If f is constant over ω, this node is a leaf node with value $f(\omega)$, otherwise ω is bisected along every dimension and the algorithm is called recursive for all, in three dimension eight, subsets of ω. These eight subsets or child nodes are called octants. Usually a maximal depth $d_{\max}$ and an undetermined state $s \in A$ are defined in order to limit the depth of the tree. As it is possible that there are parts of the domain on which f is not constant even after infinite bisections, the recursion is stopped after $d_{\max}$ steps and the node is defined as a leaf with the undetermined value s.

As an example, let Ω be the unit square, Γ the unit sphere and f the state of a point as to whether it is inside or outside the domain. From this view, the states of the function f are often referred to as black nodes to mean the inside, white nodes the outside and grey nodes the boundary of the geometry. Fig. 3.1 gives the octree representation for $d_{\max} = 5$.

It is obvious that the exact representation of the sphere will not be achieved due to the axis-parallel bisection approach of the octree. Nevertheless, the octree has a couple of ad-

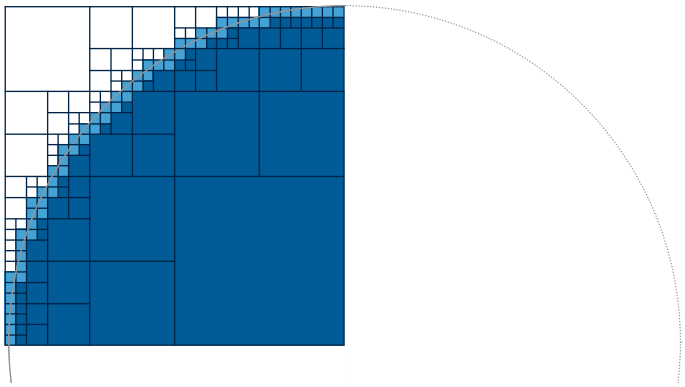

Figure 3.1: The octree representation of a sphere is calculated for a maximal depth $d_{\max} = 5$, the increasing accuracy is adaptive to the boundary.

vantageous properties which will be derived in this work. Two of them can already be seen here. On the one hand the calculation of the volume is simple as it reduces to summing up the volume of black nodes and the volume of a node is directly given by its level in the tree. On the other hand it can be observed that only the boundary and therefore the not determined part of the geometry is refined along the depth of the tree and therefore an adaptive description of the geometry is achieved.

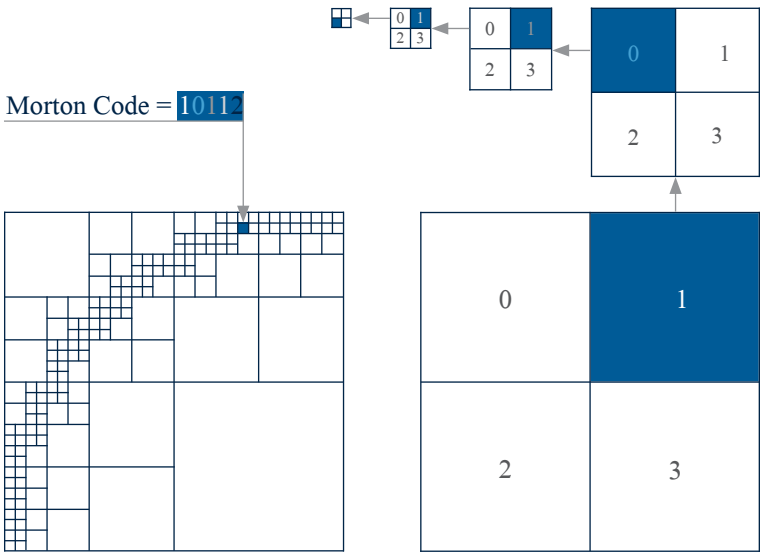

Figure 3.2: The Morton Code of a quadtree, the two-dimensional version of an octree, uses the bisecting nature of the data structure and the order of the child nodes to directly identify the address of an octant.

The coordinates of every single octant can be easily derived by the Morton Code [56] as shown in Fig. 3.2. During the construction of the octree, the ordering of the child nodes for a given parent node is kept constant, which means that for every node it is known which quadrant of the eight octants is covered - in 2D there are four and in 3D eight. In this work

the order top-left→top-right→bottom-left→bottom-right in two dimensions is used and in three dimensions the front octants are followed by the back octants.

3.1.1 Local Scale - Product Model Details

The fully detailed product model data are the finest scale of the data handled by this framework. In order to perform local operations on the data, an octree representation of the data is generated separately for all levels of detail as introduced in Sec. 2.4. Having this hierarchical representation of a single product model at hand, local operations such as efficient visualisation, information exploration and mesh generation for individual objects will be put into practice and presented.

The common file format presented in Sec. 2.5 provides an indexed triangulation for each level of detail, where the indices link to the auxiliary information of each element. A triangular mesh of the product model at level of detail i is denoted as m^i, whereas $(m^i)_j$ describes the individual triangles of the mesh. Based on this, the pseudocode for generating the octree for a single product model is given in Alg. 3.

Algorithm 3 generateOctree

```
Octree octree = generateOctree(Mesh m, Domain d){

        octree.root = generateOctant(m.triangles(), d);
}
Octant o = generateOctant(Triangles tri[], Domain d){

  for (int i=0; i<tri.count(); i++)
    if (tri[i].intersects(d))
      o.triangles().add(tri[i]);

  if (octant.triangles().count() > 0){
    o.type = grey;
    o.children[0] = generateOctant(o.triangles(), d.split(0, 0, 0));
    o.children[1] = generateOctant(o.triangles(), d.split(0, 0, 1));
    o.children[2] = generateOctant(o.triangles(), d.split(0, 1, 0));
    o.children[3] = generateOctant(o.triangles(), d.split(0, 1, 1));
    o.children[4] = generateOctant(o.triangles(), d.split(1, 0, 0));
    o.children[5] = generateOctant(o.triangles(), d.split(1, 0, 1));
    o.children[6] = generateOctant(o.triangles(), d.split(1, 1, 0));
    o.children[7] = generateOctant(o.triangles(), d.split(1, 1, 1));
  }else
    octant.type = white;
}
```

Before generating the second level octree for an individual construction, the domain has to be specified, and is then recursively split in order to achieve the hierarchical representation.

This domain is usually the bounding box of the structure to be approximated. For the second level octrees, this domain is enlarged in order to fit the boundary of the closest octants on the first level octree. The code for generating the coordinates of the enlarged domain is given in Alg. 4 and works as follows.

Algorithm 4 enlargeSecondLevelOctree

```
void enlargeSecondLevelOctree(Octree firstLevel, Octree secondLevel){

  Point min = secondLevel.domain().getMin();
  Point max = secondLevel.domain().getMax();

  Octant minOct = firstLevel.getOctantAt(min);
  Octant maxOct = firstLevel.getOctantAt(max);

  firstLevel.domain().setMin(minOct.getParent().domain().min());
  firstLevel.domain().setMax(maxOct.getParent().domain().max());
}
```

The boundary box of the construction is known from the metafile format. From the maximal depth $d_{\max}$ of the first level octree and its bounding box, the size of the finest octants, the leaf nodes on level $d_{\max}$ can be derived. The edges of the bounding box of the product model are now expressed in the coordinates of the first level octree. This implies directly that the second level octree is not aligned to the bounding box of the product model but to the orientation of the first level octree. The minimal and maximal coordinates of the leaf nodes' parents of the first level octree are now used for generating the second level octree. This means that the computational domain of the second level octree is a multiple of the discretisation of the first level octree as given in Fig. 3.3.

This is the only effect of the global data set assembly on the individual second level octrees and ensures a conforming discretisation of the whole data set across the boundaries of the level of hierarchy.

This also ensures that the Morton Code of a single voxel as seen in Fig. 3.2 from the second level octree can be extended to the whole domain on the global scale. It is therefore possible to directly identify the position of all nodes of second level octree representations with respect to each other and this across the boundaries of the individual product models.

The hierarchical representation can now be computed. The octree holds the vectors of all triangles of the mesh and the mapping between an individual triangle and the auxiliary information. The root node holds the indices of all triangles stored in the octree as it covers the complete model. The recursive construction of the octree starts with a given node, initially with the root node, and iterates over all triangles. Every triangle is checked to see if it intersects the domain of the octant. If this is the case, the index of the triangle is stored in

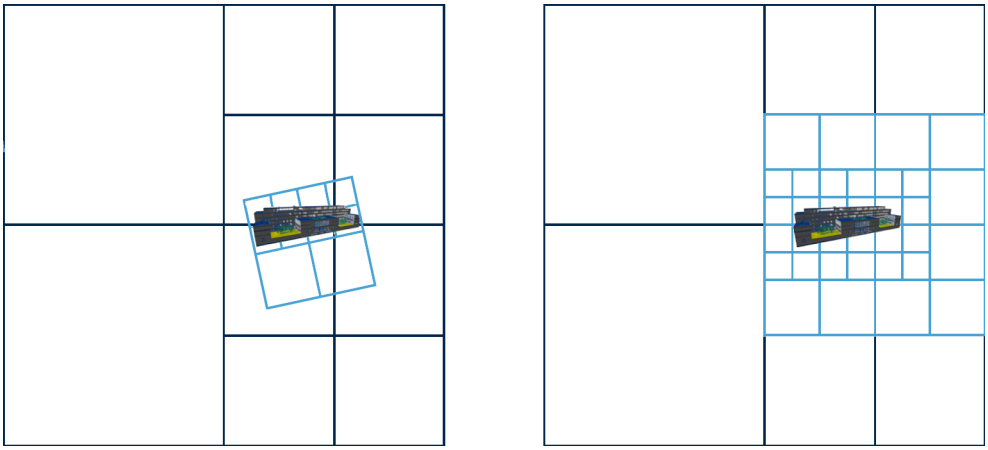

Figure 3.3: Based on an independent dual layer hierarchical data structure, the domain of the initial second level octree in general does not fit the discretisation of the first level tree (left). By increasing the domain of the second layer octree to a multiple of the width of the deepest level of the first level octree, a matching intersection is achieved (right).

the octant. If an octant is not intersected by triangles, it is defined as a leaf or a white node and not further refined. Every node which is intersected by at least one triangle is defined as a grey node and further refined until the predefined maximal length of the tree is reached. As is the case with product models, a triangular mesh is the basis of the octree, no octant can be fully covered by its elements and therefore no black nodes occur in an octree for triangles.

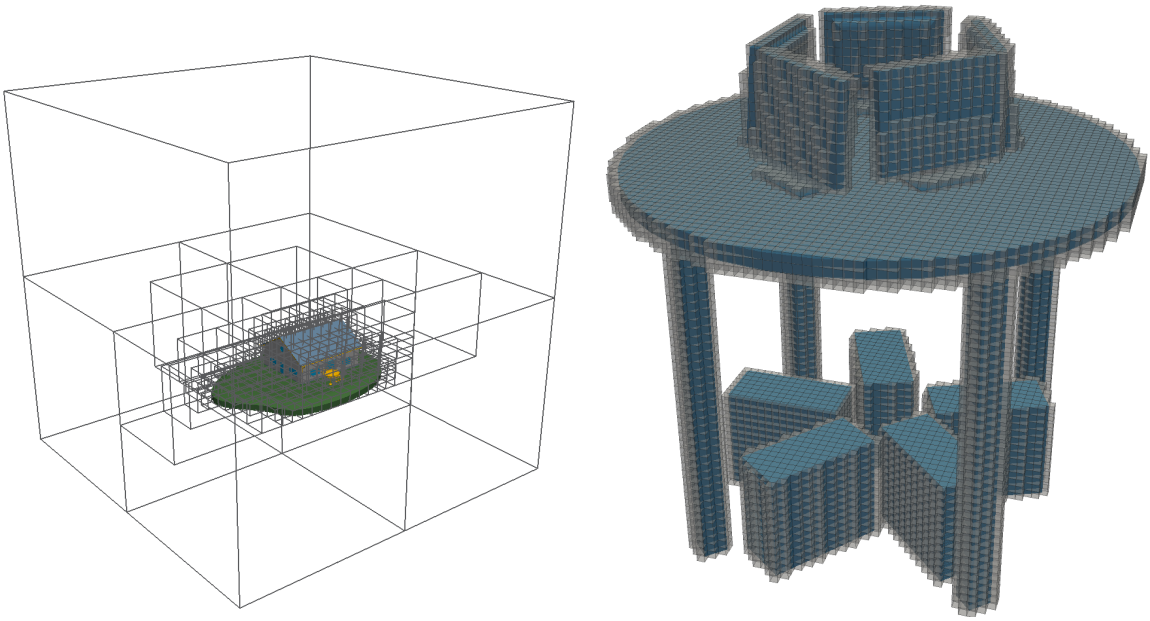

Figure 3.4: For a product model consisting of 84k triangles an octree representation is generated until a depth of 5 (left). Besides the resolution of the geometric description, auxiliary information is also mapped to the individual elements (right).

At this point octree representations for all levels of detail of individual product models are available, see Fig. 3.4. This single octree representation is now completely unknown by its surrounding and only useful for local operations on an individual model and this is also desired. The data set will be organised by evaluating the first level octree introduced in the next chapter. On this first level it is possible to efficiently decompose a request to the framework to the relevant sub-requests to individual product models. These can then be directly evaluated on the structure introduced in this section.

3.1.2 Global Scale - Location Awareness

The key contribution of this work is the handling of large amounts of complex data and the efficient access to and evaluation of these data. Until now, large amounts of complex data have been introduced. These contain fully detailed product model data of constructions and built infrastructure, large scale GIS data such as highly resolved terrain descriptions, and the definition of pipe networks. Furthermore, a city model definition is given which gives the embedding of individual buildings in the urban context. All these data can be assembled on a file or entity level to a data set, this assembly until now describing the semantic and geometric orientation of all individual elements to each other. Put briefly: at the moment there is a tremendous amount of data with the correct relation to each other distributed over individual files. This vast set is now ordered and organised in order to ultimately have a data structure which can quickly and efficiently evaluate almost any requests to these data. Without this ordering, the data would have to be processed one after another and this would result in an unnecessary computational effort which makes its evaluation impossible. And precisely this provides additional insight into the data. The organisation and data fusion presented mean the insight is greater than the sum of possible insights to every single datum.

This organisation is performed as follows. The assembly algorithm presented in Sec. 2.6 gives the location of all constructions and built infrastructure. From this, the bounding boxes and storage information of all product models are available directly. Based on these data, an octree is generated on the bounding boxes of the models only and for every bounding box the only information recorded is the storage path, i.e. the location of the file. The algorithm for generating this octree now relies on the geometric description formed from bounding boxes and is given in Alg. 5.

The special advantage of this data structure is that the octree based on the bounding boxes of the product models can contain data even for thousands of constructions and product models.

Fig. 3.5 shows a snapshot for a couple of buildings with its embedding in the first level octree. In order to be able to perceive the adaptive resolution of the bounding boxes of the individual constructions, the depth of the tree in this visualisation is limited to 4.

Algorithm 5 generateOctree

```
Octree octree = generateOctree(Mesh m, Domain d){

        octree.root = generateOctant(m.quad(), d);
}
Octant o = generateOctant(Quad quad[], Domain d){

  int contained = 0;
  int intersected = 0;
  for (int i=0; i<quad.count(); i++)
    if (d.hasIntersection(quad))
      o.quads().add(quad[i]);
      if (quad[i].contains(d)){
        contained++;
      else
        intersected++;
    }
  }
  if (contained == 0 && intersected == 0)
    o.type = white;
  else if (contained == quad.count())
    o.type = black;
  else{
    o.type = grey;

    o.children[0] = generateOctant(o.quads(), d.split(0, 0, 0));
    o.children[1] = generateOctant(o.quads(), d.split(0, 0, 1));
    o.children[2] = generateOctant(o.quads(), d.split(0, 1, 0));
    o.children[3] = generateOctant(o.quads(), d.split(0, 1, 1));
    o.children[4] = generateOctant(o.quads(), d.split(1, 0, 0));
    o.children[5] = generateOctant(o.quads(), d.split(1, 0, 1));
    o.children[6] = generateOctant(o.quads(), d.split(1, 1, 0));
    o.children[7] = generateOctant(o.quads(), d.split(1, 1, 1));
}
```

Owing to its relevance for the further presentation, the direct link of the fully detailed product models to the first level octree is depicted for a 2D view in Fig. 3.6. Even though the first level octree is based on the bounding boxes and the storage information of the individual models only, access to the fully detailed information is still safeguarded.

3.2 A Scalable Hybrid Parallel Approach

In this section, the hybrid parallelisation of the framework is introduced. Hybrid parallelisation describes the exploitation of concurrent order execution possibilities on the shared memory level with multiple cores in a single machine, and on the distributed memory level over the network of a cluster among the different machines.

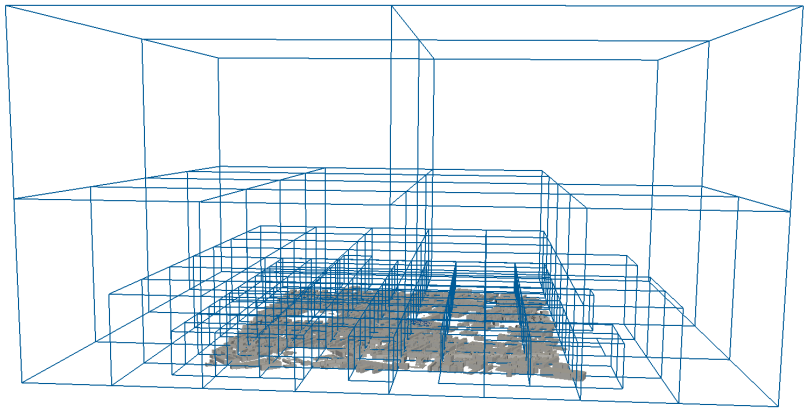

Figure 3.5: By computing an octree based on the bounding boxes of all constructions and built infrastructures, the embedding of the data into the surrounding is structured and the individual models can be related to each other.

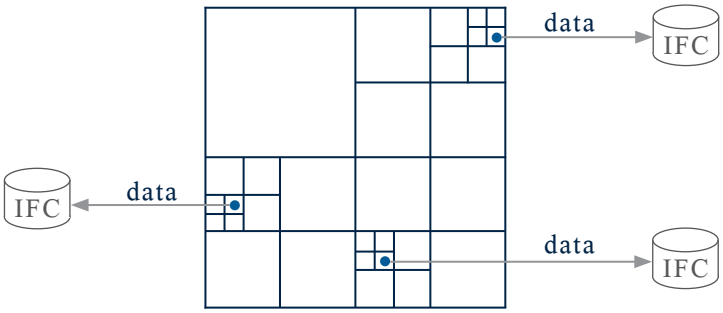

Figure 3.6: The generation of a quad/octree based on the bounding boxes and storage information of all processed models can be computed quickly and efficiently, but still contains the link to the fully detailed information.

As the name says, the different approaches originate from the common or separated address space of the memory of the machines. Shared memory approaches are usually limited to a single computer, where the cores of the CPU have common access to the main memory. There are also approaches for a common address space among multiple machines, such as the recently completed installation of the HLRB2 at Leibniz Rechenzentrum Garching [59], but these approaches have not been followed up due to the communication overhead between the individual computers. Shared memory parallelisation is today usually implemented using the OpenMP standard [60, 61] which provides a tool set for implementing a thread-based execution and synchronisation of tasks. There are further thread-based execution models such as Intel Thread Building Blocks (TBB) [62, 63] which have not been investigated further due to the vendor-specific orientation of the approaches and the universal applicability of the approach presented.

Distributed memory approaches implement the communication of different processors not by accessing a common address space as in shared memory approaches but by explicitly performing the communication between processors by sending and receiving data. The standard for performing distributed memory parallelisation is the Message Passing Interface (MPI) [64, 65, 66]. As MPI itself is a standard for communication and synchronisation, an MPI implementation or distribution kit has to be chosen. Standard implementations are the OpenMPI [67] and MPICH [68] libraries. On high-end installations such as supercomputers, MPI versions tailored to the topology and the specifics of the installation are provided by the vendor and come with the set-up of the machine.

In the following the hybrid parallelisation over the dual layer hierarchical ordering of the data is presented. In the later sections, a whole field of applications of the framework will be introduced but for now, an exemplary question from the field of navigation shall function as an example for the approach of the framework. In navigation and routing applications one of the central questions is the location of spatial features in a city [69, 70]. Therefore, parallel to introducing the parallelisation of the framework, the question will be answered as to how this approach can efficiently deliver the location of specific objects in the city such as the closest phone box to a given position. It is clear that without any ordering of the data, a brute force iteration over the complete data set is not feasible. The approach of this framework will be introduced parallel to answering this question.

3.2.1 Multi-Level Parallelisation

The multi-level parallelisation of the framework follows a hybrid approach with a distributed memory parallelisation over the first layer for the global assembly of the data set and a shared memory parallelisation of the product model scale tasks.

Initially, a set of m product models with the specification of their bounding box, position and orientation in space is available, as introduced in Sec. 2.6. A distribution of the m product models to n MPI processes is then chosen. The investigation of the efficient distribution strategies and impact of the choice of distribution will be discussed in Sec. 3.3.3. Fig. 3.7 gives a mapping of 7368 product models to 5 processors.

Having a mapping of product models to processors at hand, the framework is started on a cluster with n processes. The process with number 0 is defined as the master process and the remaining $n - 1$ processes are the slave processes.

The master process builds up the first layer octree, stores for every product model the slave process it is assigned to - following the distribution strategy given - and sends to each process the storage information of the product models assigned to it.

Figure 3.7: Based on a given mapping of m product models to n processors, the colour indicates the distribution of the models over the processors. All models of a processor are depicted with the same colour.

Each of the n slave processes initially receives the specification of the product models which are mapped to it and generates an octree representation for every individual model.

Coming back to the example mentioned of the navigation to a phone box, the principle of the framework becomes clear. In order to identify the closest phone box to a given position, the master process sends the request to all slaves to identify the closest phone box to this position. Each of the slave processes queries the hierarchical representations of the product models which are assigned to it and identifies the closest phone box. Even on the slave level of processing the hierarchical approach saves computations. As every slave process knows the domain of every individual product model it processes, it starts investigating the hierarchical representation of the individual product models in ascending order of their distance to the given position. As soon as the information is detected, the work of processing the remaining product models can be saved. The master process now collects the local results for the position of the closest phone box from each of the slaves and selects the global result.

The scaling of the framework also now becomes clear. By setting the number of processors n to the number of product models m, every process is assigned to a single product model only. In that sense, a linear speedup, i.e. the perfect parallel efficiency of response time as a function of the number of processors, seems achievable.

Starting from this brief introduction of the hybrid parallelisation approach, the two levels of parallelism will now be introduced in detail over the runtime of the framework. The set-up of the framework is given in Fig. 3.8.

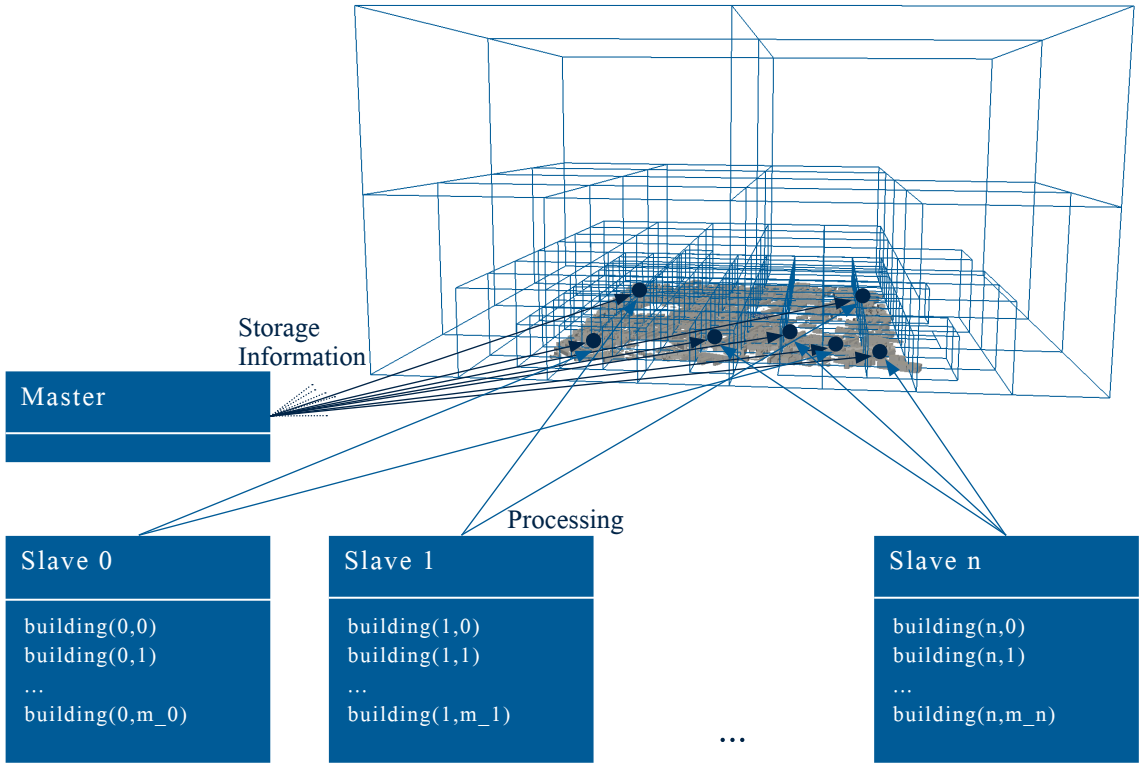

Figure 3.8: The parallelisation of the framework is achieved by a distributed memory approach on the global scale. One master process contains the assembly of the data basis given by the first layer octree and the product models are distributed among the remaining $n-1$ processes. Shared memory parallelisation is then performed over the local product model sets of every single process.

3.2.2 Building Up the Framework

The build-up of the framework and the parallelisation strategy are now given in detail. The execution is started on a cluster with n processes, each process runs on a single machine. The steps for building up the framework are first the setting up of the one process which is the master process, and the $n-1$ slave processes which hold a set of product models each. The master process then builds up the first level octree with the global assembly of the data basis and sends the subset of the product models to each of the slave processes. Each slave process then receives the product models to process and builds up the second level octrees for its product models.

At start up of the n processes, initially every process decides on the basis of its ID whether it is the master process or a slave process. Before defining the MPI communication between

call	description
MPI_Init();	initialise MPI environment
MPI_Comm_rank(rank);	get Id of process
MPI_Comm_size(size);	get number of processes
MPI_send(receiver, type, count, buffer);	send *count* data of *type* stored in *buffer* to *receiver*
MPI_recv(sender, type, count, buffer);	receive *count* data of *type* from *sender* and store it in *buffer*
MPI_Finalize();	close MPI environment

Table 3.1: Basic MPI commands are needed in order to perform the communication between processors by exchanging data via MPI.

the master and the slave processes, the basics of the communication with MPI have to be introduced.

With MPI, the parallelisation is put into practice by starting a set of n processes, usually one process per machine, but multiple processes can also run on a single machine. MPI provides an interface for exchanging data between processes. The only identification available to a processor are its process identifier - the rank - and the total number of running processes - the size. Basically, there is no information about which machine an individual process runs on and also the specification of the process with rank 0 as the master process is a not mandatory convention.

In order to exchange messages between two processes, a send and a receive action with corresponding specifications have to be performed by the processes involved. The specifications for sending and receiving data are the rank of the sender and the receiver, the basic type of data to be sent, the amount of data to be sent. Only if a send and a receive of the corresponding specification are performed are the data transferred between the processes. Besides point to point data exchange between two processors, collective communication among all processors is also available such as broadcasting messages to all processes or collecting information from all processes. The last feature of MPI needed for introducing the parallelisation strategy is the possibility of synchronous and asynchronous communication. Following a synchronous communication pattern, the sender and the receiver stop the execution until the data are transferred, whereas asynchronous communication allows the processes to reserve the memory for the data it receives and proceed with execution. The process can now perform other computations and check whether the data have arrived. The communication can thus be hidden behind computations and there is no blocking of the execution pipeline to slow down the system.

This is only a brief introduction to communication and synchronisation using MPI, Table 3.1 gives the signature of the MPI functionality used. For a detailed introduction to MPI, the reader is referred to [65, 66, 64].

Having this functionality at hand, the distributed parallel build up of the framework can now be introduced and is given in Alg. 6 for the master processing and in Alg. 7 for the slave processing. It should be pointed out that there is only one master process, but $n-1$ slave processes.

Algorithm 6 initMaster

```
void initMaster(Model building[]){

  Vector processModels[];
  for (int i=0; i<building.count(); i++){
    processModels[building[i].processID].add(building[i]);
  }
  for (int i=1; i<processModels.count(); i++){
    int modelCount = processModels[i].models().count();
    MPI_Send(i, int, 1, modelCount);
    MPI_Send(i, byte, sizeof(model) * modelCount, processModels[i]);
  }
  generateFirstLevelOctree()
}
```

Algorithm 7 initSlave

```
void initSlave(Model building[]){

  int countOfModels;
  Model processModels[];

  MPI_Recv(0, int, 1, countOfModels);
  MPI_Recv(0, byte, sizeof(model) * countOfModels, processModels);

  for (int i=1; i<processModels.count(); i++)
    processModels[i].generateSecondLevelOctree()
}
```

The next level of parallelisation is now the level per machine i.e. per process. The master process contains one octree - the first level octree - and the slave processes contain a set of octrees - one second level octree per product model.

The parallelisation of the generation of the octree is performed using shared memory parallelisation and exploits the available cores of the CPU on which the process runs. Alg. 8 gives the queue-based parallelisation.

According to the basic principle, the octree checks recursively for every octant as to whether it is intersected by the approximated object and then splits the child nodes affected. In the literature, there are many approaches for the parallelisation of the octree generation [71, 72, 73]. In this work, parallelisation is implemented by introducing a process queue. This queue

Algorithm 8 generateParallelOctree

```
Octree octree = generateParallelOctree(Mesh m, Domain d){

  Queue octants.push(generateOctants(m.triangles(), d));

  #pragma omp parallel
  {
    while (!octants.isEmpty()){

    #pragma omp critical
    Octant o = octants.pop();

    children = generateOctants(o.triangles, o.domain);
    if (children.count() > 0){
      #pragma omp critical
      {
        for (int i=0; i<8; i++)
          octants.push(children[i]);
      }
    }
  }
}
```

contains the nodes which have to be processed and is initially filled with the root node. Each of the threads of the octree generation is parallelised with accesses to the queue, reads an octant to the process and carries out the intersection check for this individual octant. If the octant has to be refined, the eight child octants are generated and added to the queue as single entries. The access to the queue is protected using a mutual exclusion, which ensures that only one process at a time reads and writes to the queue.

Having the build-up framework at hand, suitable interfaces for a wide set of applications in computational science and engineering are now introduced. These interfaces range from the visualisation of the whole data set and the exploration of the data on various levels of detail to the generation of computational meshes for performing numerical simulations.

3.2.3 Interfacing Multi-Resolution Geometry

In this section, the interface for visualising the whole data set of the framework is introduced. Visualising the complete data set at the finest level of detail is not feasible nor of interest. The whole data set consists of thousands of buildings and constructions and each of them has a geometric representation of hundreds of thousands of primitives. Summing up the primitives to render the visualisation would end up in billions of triangles and this exceeds even most modern hardware installations. Besides that, this approach would also try to render details which are not perceptible.

When exploring the whole data set, the location of the camera or the centre of the investigation is the crucial factor in processing the data. All information close to this point has to be resolved in full detail, all information further away can be coarsened already and information still further away can be represented by a single primitive or even be neglected. This view can also be inspired by having a look at the process of rendering data. In order to render data, the pixels of the visualisation output to the screen have ultimately to be assigned a colour value. When the product model is a long way from the camera it will be projected into a couple of pixels, only a single pixel or not even be visible on the output. The basis for precisely these coarsened representations have already been introduced in Sec. 2.4 as LoDs for product model data.

Having this in mind, the interface for accessing a multi-resolution representation of the data set can be defined as follows:

Given a point P and a definition of the distances $d_i(P)$ over which distance to P a level of detail i is to be applied to the product model definition, the framework provides the geometric representation as a triangular mesh, where each product model is at level of detail i with the distance d and $d_{i-1}(P) \leq d < d_i(P)$.

Algorithm 9 processVisualisation

```
void processVisualisation(float d[],     Point poi){

  calculateLoD(firstLevelOctree.root, d, poi);

  models = firstLevelOctree.colletInvalidModels();

  for (int i=0; i<processors.count(); i++)

    subset = models.selectModelsOfProcess(i);
    sendModelMPItoProcess(i, subset);
}

void calculateLoD(Octant o, float d[], Point poi){
  if (!o.isLeaf())
    for (int i=0; i<8; i++)
      calculateLoD(o.children[i], d, poi);
  else
    for (int i=0; i<o.models.count(); i++){

      LoD tmp = o.models[i].LoD();

      o.models[i].resolveGeometryForDistance(distance, poi);

      if (o.models[i].LoD() != tmp)
        o.models[i].isValid = false;
    }
}
```

The parallel code for the master processing of this request is given in Alg. 9 and works as follows: in the initialisation, the master process has d_i, the scenario specific definition of the distances at which an LoD is to be applied, and the first level octree. Furthermore, every product model leaf node has a flag $isValid$ and this is initialised as $false$.

In the $processVisualisation$ the first level octree is traversed and the correct level of detail is determined for every product model in the tree by recursively calling $calculateLoD$ on the root node. If a node is a leaf node, it calculates the level of detail, otherwise it calls the $calculateLoD$ on its eight child nodes.

The correct LoDs for all product models are now stored in the tree and the mapping is used to determine the slave process on which the product model is defined. Using this mapping, the vector containing the product model identifiers and the LoDs for every single product model are collected per process and sent using MPI to every process analogous to Alg. 6. This vector is restricted to the product models whose flag $isValid$ is set to $false$.

After sending the vector to the slave processes, the master process starts the MPI reception from the slave processes and stores the levels of detail of the data which have been sent. For every LoD the master receives, it sets the $isValid$ flag of the respective product model to $true$ and concatenates all geometric representations received to the multi-layer representation of the whole data basis.

The advantage of introducing the flag $isValid$ is that, usually, only in the first generation of the multi-resolution geometry do all levels of detail have to be calculated and transferred using MPI. When the data are reprocessed and the point P has not changed to any great extent, most of the levels of detail and therefore their geometric representation are still valid and do not have to be recalculated and set.

The slave processing for the multi-resolution representation of the data set starts with receiving the vector with the product models to be processed and their correct levels of detail. To generate the levels of detail of all product models on a single process, shared memory parallelisation can again be performed. As the single product models are independent of each other when performing the LoD calculation, the single calculations can be distributed to parallel tasks. After calculating the geometric representation of every product model, the triangular meshes are collected and sent to the master process.

3.2.4 Interfacing Auxiliary Information

This section gives the access to the auxiliary information of the data set which gives the user of the framework the possibility to explore the data for the fine details of individual construction entities as well. The auxiliary information is all the non-geometric specifications of product model properties as introduced in Sec. 2.1 to 2.3. They range from the definition of the roughness of individual pipes in the sewer network received from GIS sources to the construction detail specifications of the insulation value of windows and doors in a BIM source, for example.

Again, the parallel hierarchical approach of the framework is the key for efficiently accessing the data and this access can be performed by allowing two different methods, depending on the scale of the data exploration. On a local scale, a single product model is investigated and the auxiliary information for a construction detail is requested. On the global scale, the user explores the complete data set, and by clicking on a point in the data set the auxiliary information has to be identified by first finding the product model concerned and then identifying the respective construction detail.

The local-scale access to a construction detail follows the parallel framework introduced in a straightforward way. To access the auxiliary information, the identifiers of the product model and the index in the triangular mesh of the product model are known. On the basis of the identifier of the product model, the master process identifies the slave process on which this specific product model is processed. The master then sends the MPI request for retrieving the specific auxiliary information with the identifiers of the product model and the indexed triangle to the respective slave process. The slave process accesses the respective product model, selects the auxiliary information of the specified construction detail and sends it back to the master process.

Accessing auxiliary information of construction details on the local scale has been straightforward as not only the identifier of the product model is known but also the index of the triangular mesh. As introduced in Sec. 2.5, the metafile format ensures there is direct mapping of the indices of an individual triangle in the mesh representation to its auxiliary information. As this approach exploits the mapping of auxiliary information to geometry, only information with a (triangular) representation are supported. In order to identify the usage of a room, for example, this is not the case. This information can be stored in IFC with an IFCSPACE definition and enables the mapping of the bounding wall or window elements, but already this specification is an ongoing research task [74].

On the global scale, the identifier of the investigated detail is not given directly and only the point the user clicked on is known. This information is also sufficient to identify the product model, the construction detail and therefore the auxiliary information as follows.

When exploring the whole data set on the global scale, only the point P the user clicked on is known together with the directional vector d of its view. The point P and the directional vector d define a ray which intersects the respective construction detail, and therefore the product model is intersected as given in Fig. 3.9.

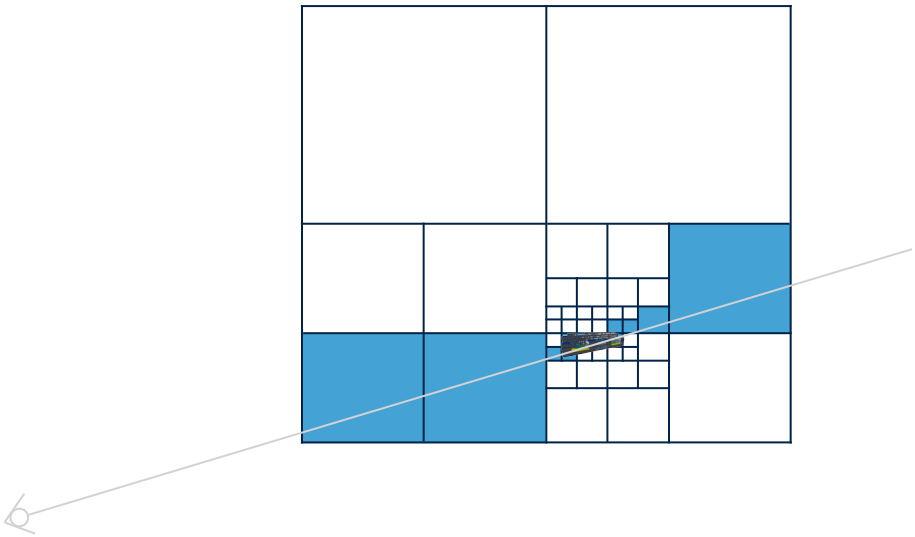

Figure 3.9: The view point and direction of the user define a ray which intersects the requested construction detail and the product model. An intersection test on the first layer octree in the master process identifies the product models intersected, and the test on the product models in the slave processes affected identifies the construction detail.

In order to perform the intersection test, the master process initially recursively checks the first level octree and identifies the octants intersected by the ray starting with the root node. If an octant is not a leaf node, it checks which of its octants are intersected by the ray and calls on the intersection test for the child nodes intersected. In a vector all leaf nodes intersected and therefore all product models intersected are collected. From the list of the intersected product models the slave processes affected are identified using the mapping of the product models to the processors. The master process then sends the request for the auxiliary information with the point P and the directional vector d to every slave process affected and requests the auxiliary information. The slave processes also perform the recursive intersection test on the intersected product models and identify the construction detail requested and therefore the auxiliary information.

Finally, the master process contains the auxiliary information of all construction details intersected by the ray along the view of the user. In general, multiple construction details are projected along d to the same point P. Therefore, the entities whose distance along the directional vector d is negative are deleted as they lie behind the user and only the entity with the minimal (positive) distance along d is selected.

3.2.5 Interfacing Linearised Octree Representations

This section introduces the linearised representation of the octree and the implementation over the dual layer hierarchical data structure of the work presented.

Linearising hierarchical data structures [57] is a well-known technique for storing the tree structure in a simple format such as a stream. Based on this representation, the parent-child relation between the octants and the values of the leaf nodes are available, can be saved to disk and be evaluated. By evaluating the linearisation of two octrees, the intersection of the structures approximated by the octrees can be determined to the accuracy of the resolution depth of the tree. In Sec. 4.2.3 this technique will be used to perform collision detection between constructions and built infrastructure.

Algorithm 10 lineariseOctree

```
char linearisation[] = lineariseOctree(Octree o){

  Stack linearised;

  linearisation = lineariseOctant(linearised, o.root);

  linearisation = linearised.getString()
}
void lineariseOctant(Stack linearised, Octant o){

  linearised.push(o.type);

  if (o.type = gray)
    for (int i=0; i<8; i++)
      lineariseOctant(linearised, o.children[i]);
  else
    linearised.push(o.Id);
}
```

The code for generating the linearisation of an octree is given in Alg. 10 and works as follows. A stream representation of the tree is achieved by traversing the tree in a depth-first manner and pushing the values of the octants to a stack where 0 indicates that an octant is refined and 1 that an octant is not refined. If an octant is not refined, it is a leaf node and the value is pushed to the stack where 0 indicates that the node is a white node, and otherwise the identifier of the product model is pushed to the stack.

It is essential to keep the octants constantly in a fixed order such as left→right, top→bottom, front→back in order to identify the correct relation of the child trees. By doing so, the whole structure of the octree representation can be retrieved and the linearisation as given in Fig. 3.10 contains the full adaptive approximation of the geometry.

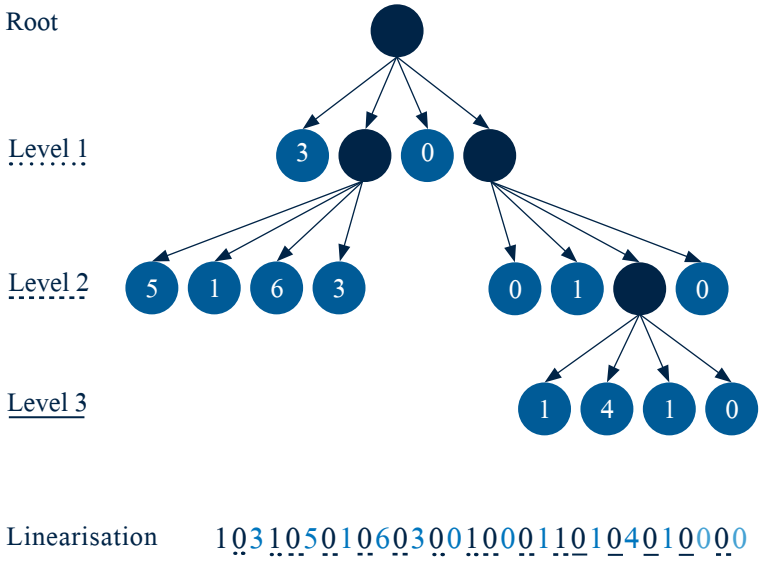

Figure 3.10: The linearisation of an octree is achieved by concatenating the values of the octants in a depth-first manner to an array with the 0/1 information if an octant is a leaf node and followed by the value of the leaf node.

3.2.6 Interfacing Voxel Representation

This section contains the derivation of an equidistant discretisation of the computational domain which is the basis for performing parallel numerical simulations. The focus of the work presented is on handling large sets of complex data and one of the applications is the numerical simulation of physical phenomena. To perform a numerical simulation, the computational domain has to be discretised and to do so, the computational domain has to be separated from the outer parts. Therefore, in the following, an equidistant flag field will be derived which identifies on a regular grid up to a given accuracy for every point whether it is intersected by structure or not. This flag field is also referred to as voxelisation of the computational domain. Based on this field, Sec. 5 provides the coupling to the grid-generation and numerical simulation frameworks.

The generation of the voxelisation can be seen as a request to the parallel framework, in which for a given computational domain

$$D = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$$

and a discretisation width h a matrix $V = V_{ijk}$ with

$$\begin{aligned} i &\in [0..(x_{\max} - x_{\min})/h] \\ j &\in [0..(y_{\max} - y_{\min})/h] \\ k &\in [0..(z_{\max} - z_{\min})/h] \end{aligned}$$

and the voxel $v = v_{ijk}$ with

$$\begin{aligned} v_{ijk} \quad = \quad & [x_{\min} + (x_{\max} - x_{\min}) * i/h, x_{\min} + (x_{\max} - x_{\min}) * (i+1)/h[\times \\ & [y_{\min} + (y_{\max} - y_{\min}) * j/h, y_{\min} + (y_{\max} - y_{\min}) * (j+1)/h[\times \\ & [z_{\min} + (z_{\max} - z_{\min}) * k/h, z_{\min} + (z_{\max} - z_{\min}) * (k+1)/h[\end{aligned}$$

resulting in

$$V_{ijk} = \mathbb{1}_{\{v_{ijk} \text{ is intersected}\}}$$

An example for the derived voxelisation of a given computational domain on the local building scale is depicted in Fig. 3.11.

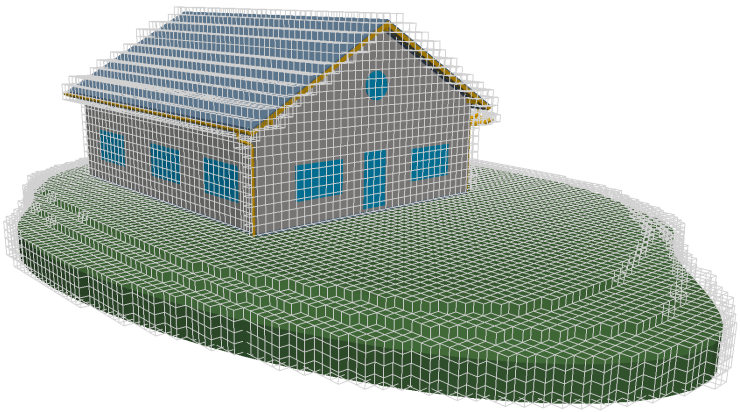

Figure 3.11: A product model on the local building scale is embedded in its voxelisation with a discretisation width of $150 \times 100 \times 100$ voxels. For better visibility, only grey leaf nodes, i.e. voxels, are shown in wire frame mode.

The generation of a voxel representation for a given domain and mesh width directly follows the hierarchical dual layer approach of the framework presented. The master process generates the first level octree to the extent of the computational domain specified for the voxelisation, and the slave processes generate the second level octrees with a mesh conforming to the discretisation of the first level octree. This conforming generation of the first and second level octrees as described in Sec. 3.1.1 is a crucial element for efficiently deriving the

voxelisation. The indices of the corresponding voxels in the flag field can be derived directly from the leaf octants of the second level octrees generated by the slave processes.

For every octant the Morton Code gives the path from the root node to the octant. This address stores the sequence about which parent→child relationship the octants on the path from the root node to the respective octant are. As the octree performs a bisection along every axis, the domain of an octant can be derived this way. The length of the address gives the depth of the octant in the tree. Therefore only the addresses of the grey octants of the second level octrees have to be communicated to the master process. On the basis of these addresses, the master process can generate the flag field requested.

The master process allocates a matrix with the size of the flag field and initialises the flag field to the value 0 - not filled. The master process then sends all slave processes the request to generate the information about their product models for the local flag field. As the first level octree covers precisely the computational domain of the voxelisation requested, only second level octrees which intersect the domain are generated in slave processes. Every slave process recursively collects the addresses of the grey leaf nodes in a vector and sends the addresses back to the master process. From the addresses received from the slave processes the master process derives the indices of the flag field by following the code given in Alg. 11 and sets the corresponding index ranges to 1 - filled.

Algorithm 11 indexRangeOfVoxelAddress

```
int index[] = indexRangeOfVoxelAddress(char[] address){

  Domain d = firstLevelOctree.domain();
  for (int i=0; i<address.length(); i+){

    bit = getBinaryRepresenationOfChar(address[i]);

    d = d.split(X, bit[0] ? lower : upper);
    d = d.split(Y, bit[1] ? lower : upper);
    d = d.split(Z, bit[2] ? lower : upper);
  }
  index = {d.min.x, d.max.x, d.min.y, d.max.y, d.min.z, d.max.z}
}
```

The algorithm presented focuses on a computational domain which spans multiple product models. To generate a local-scale voxelisation of a single product model the procedure can be simplified further. For a single product model voxelisation only one slave process generates a second level octree for the product model of interest. As introduced in Sec. 3.2.5, the linearisation of the octree is derived and the addresses of the octants can be calculated. The rest of the procedure then follows the approach presented even without the communication between the master and a slave process.

It should be pointed out that, from the process of generating the voxelisation, it is clear that a sparse storage scheme [75, 76] is best suited for storing the flag field in an efficient way. In a sparse storage scheme only the values which are not 0 are stored in a vector together with two more vectors, one storing the row indices and one storing the column indices of every row sequence of the data.

Nevertheless, the dense storage of the data set is also kept, as in Sec. 5 this matrix is processed further in a way where the memory dense storage of the data is of interest.

3.2.7 Interfacing Voxel Query

Although a voxel representation of a whole computational domain as given in Sec. 3.2.6 has been generated, an interface for querying the status of a single voxel has also been implemented.

Many grid generation frameworks build up the computational domain in an adaptive way by iteratively querying parts of the domain if they are completely inside the computational domain or completely outside of the domain. Parts which are not completely inside and not completely outside the domain must intersect the boundary and are therefore refined further. Adaptive mesh refinement during the runtime of the framework can also be performed with this functionality by iteratively querying the inside/outside status of parts of the domain.

The Peano [77] framework is of such a type and the coupling of the framework presented will be introduced in Sec. 5.2. The interface for querying the status of a given voxel is given in Alg. 12 and calculates for a given part of the computational domain whether its status is *filled*, *empty* or *not determinable*.

For a given query to a part of the domain, the master process calculates the intersection to the first level octree.

If the intersection is empty, the status of this part can be directly derived as *empty*. If there is an intersection with the bounding box of a product model and the queried part is not covered by this bounding box, all intersected product models are collected and the slave processes are requested to identify the inside/outside status of the requested part.

If all slave processes identify the requested part as being *empty*, the status is set to *empty*, otherwise to *not determinable*. The status *filled* basically cannot occur in the setting of the framework presented as the fine-scale product model data are described using triangular meshes and no set of triangles can cover a voxel completely.

Algorithm 12 queryStatusOfVoxel

```
int status = queryStatusOfVoxel(Voxel o){

  Process p [] = firstLevelOctree.getIntersectedModelsOf(o).processes();

  int stati[];
  for (int i=0; i<p.count; i++){
    stati[i] = getSecondLevelStatus(p[i], o);
  }

  status = stati[0];
  for (int i=1; i<p.count; i++){
    if (status != stati[i]){
      status = notDeterminable;
      return;
    }
  }
}
```

3.2.8 Communication Protocol of the Framework Access

In the previous sections, multiple interfaces for accessing the framework have been introduced. Now the organisation of this access is introduced.

As the global-scale parallelisation follows a distributed memory MPI approach, the communication between processors is performed by sending and receiving messages. As introduced in Sec. 3.2.2, sending data between two processes can only be performed if one process is performing a send and the other process is performing a receive call. Furthermore, the data type, the length of the data and the sender/receiver paring must match. This implies directly that communication using MPI only works if the participating processes know from which and to which process data are to be sent, and what the length and type of these data are. In order to cope with this, a communication protocol between the master and the slave processes has been developed and works as follows.

After the initialisation of all processes, the build-up of the first level octree, the distribution of the product models to the slave processes, and the build-up of the second level octrees, all slave processes perform a blocking receive from the master process for a data set of length 1 and type *char*. This single *char* identifies the specific interfaces and requests to be performed by the slave processes.

For the query of the geometric representation of a product model at a certain level of detail as introduced in Sec. 2.4, the master process initially sends the identifier G to the slave process. The slave process receives the single *char* and calls the *sendGeometry* routine in accordance with the protocol. Afterwards, two integers are sent to the slave process: the Id

of the product model and the level of detail to apply. The slave process now calculates the requested level of detail for the given product model and stores the geometric representation as a triangular mesh in an array. Afterwards, the slave sends the master process one integer which gives the size in bytes of the triangular mesh generated. Finally, a send/receive is performed with the size of the triangular mesh and the data are transferred to the master.

Only by defining a protocol with the exact types and sizes of the data to be exchanged does every processor already know the header of the transfer before it is performed and sending data using MPI is therefore possible. The implementation of the protocol for organising the access to the different interfaces of the framework is given for the master in Alg. 13 and for the slaves in Alg. 14.

Algorithm 13 masterSendRequest

```
void masterSendRequestGeometryRepresentation(Model m){

  Process p = m.process();

  MPI_Send(p.rank, 1, char, 'G');
  MPI_Send(p.rank, 2, int, [m.id, m.LoD]);

  MPI_Recv(p.rank, 1, int, count);
  MPI_Recv(p.rank, byte, sizeof(element)*count, elements);

void masterSendRequestExit(Model m){

  for (each p in process())
        MPI_Send(p.rank, 1, char, 'X');

  exit;
}
```

3.3 Load Distribution

In this section, the load distribution strategy of the framework is introduced. A good load distribution strategy ensures that the computational load during the runtime of the framework is distributed equally over the processes. In the framework presented, the computational load originates from a request to the framework which is split up into sub-requests on the product models handled by the slave processes and then the results of the slave processes are combined by the master processor in order to answer the request. As the master process can only answer the request when all results of the slave processes have been performed, the longest running slave determines the duration for answering the request. Therefore, the requirement to find a good load distribution strategy is equivalent to finding a mapping of product models to slave processes where the computational load on the slave processes for the different requests

Algorithm 14 slaveReceiveRequest

```
void slaveReceiveRequest{

  char action = ''
  while (action != 'X'){

    MPI_Recv(0, 1, char, action);
    select (action)

      case 'G':
        int id, lod
        MPI_Recv(0, 2, int, [id, lod]);
        Mesh elemens[] = models[id].calculate(lod);
        MPI_Send(0, 1, int, elements.count());
        MPI_Send(0, byte, sizeof(element)*elements.count(), elements);
        break;

      case '␣'
        ...
        break;
  }
}
```

to the framework is as equal as possible.

The computational load on a slave process results from the number and the complexity of the product models assigned to a slave process. Whereas the number of product models per slave process could be equally balanced over the processes a priori, the complexity of processing a product model depends on the level of detail it is processed on. As introduced in Sec. 2.4, the size of the geometric representation at the fully detailed LoD4 is an order of magnitude higher than in lower levels of detail. Therefore it is crucial to develop a load distribution strategy which equally distributes the fully detailed product model processing over the slave processes.

This directly implies two aspects. On the one hand, it depends on the specific request to the framework which level of detail is to be applied to a product model and this changes dynamically over the runtime of the framework. On the other hand, the product models which have to be processed on the highest level of detail are locally clustered to a single point. For visualisation purposes this is the viewport.

Therefore, the load distribution strategy can be defined as a mapping which distributes the geometric close assembled product models to different slave processes. More specifically, the distribution of the product models over the slave processes should be equal for any given sub-domain of the whole computational domain. In that sense, the demands placed on the load distribution strategy for the framework presented are complementary to standard strategies

as neighbouring relations have to be prevented and not be preserved. In order to achieve this, two different mappings have been developed and will be introduced in the following.

The load distribution strategy for the framework presented is a mapping $M : i \rightarrow j$ for $i \in [0..m-1]$ and $j \in [1..n-1]$ where m is the number of product models and n is the number of processes of which $n-1$ are slave processes and one process is the master process containing the first level octree and the assembly of the whole data set.

Initially, a probabilistic mapping is introduced which randomly distributes the product models over the slave processes and therefore tends to an equal distribution. Then, a mapping based on a modification of the space-filling curve [78] (SFC) approach is introduced which derives a global distributed order from the local clustered order achieved by SFCs. In the end, the distribution quality achieved is investigated.

3.3.1 Probabilistic Mapping

A probabilistic mapping is generated by distributing every product model to one of the $n-1$ slave processes on the basis of an equally distributed random variable with value between 1 and $n-1$. In order to generate the random distribution of the product models to the slave processes, a vector of independent equally distributed random variables of length m is generated using standard functions.

The distribution of over 7000 product models to 3 slave processes over the domain is given in Fig. 3.12.

3.3.2 Modified Space Filling Curves

The mapping based on the modification of the Lebesgue curve, also known as Z curve, initially generates the SFC order on the first level hierarchical order of the complete data basis and then destructs the locality of the SFC by iteratively distributing the product models to the slave processes following a round-robin scheme. The pattern of the Lebesgue curve is given in Fig. 3.13.

In order to build up the ordering, the first level hierarchical data structure over the whole data set is generated not on the basis of the bounding boxes of the product models but only on the centre points of the product models. Furthermore, not a three-dimensional octree is generated but the two-dimensional quadtree based on the projection of the centre points of the product models along the z-axis as given in Fig. 3.14.

From the quadtree representation of the centre points of the product models, the order following the Lebesgue curve can be derived directly and is given in Alg. 15.

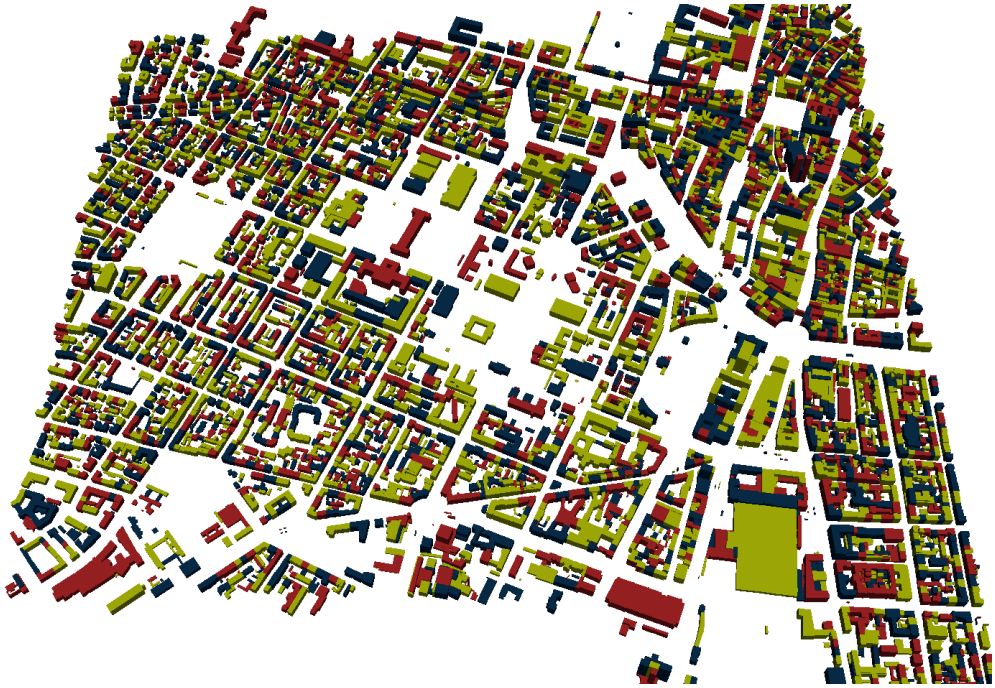

Figure 3.12: Following a probabilistic mapping, over 7000 product models are distributed to 3 slave processes over the domain. The colour value identifies the slave process to which a product model is assigned.

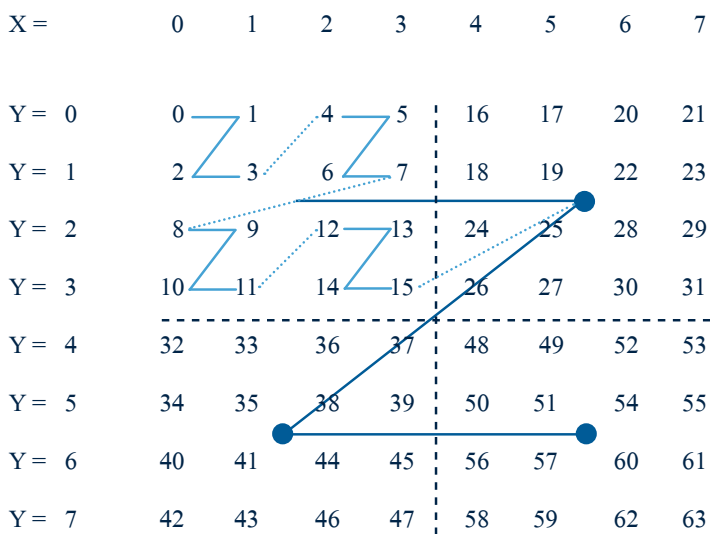

Figure 3.13: The Lebesgue curve orders multi-dimensional data points into a one-dimensional sequence. The order is achieved by recursively sorting points following (in 2D) the pattern top-left→top-right→bottom-left→bottom-right.

The order is derived by recursively adding the nodes of the quadtree following the pattern of the Lebesgue curve starting with the root node. If a node is not a leaf node, its child nodes are added instead of adding the node itself. Now the order derived from the Lebesgue curve

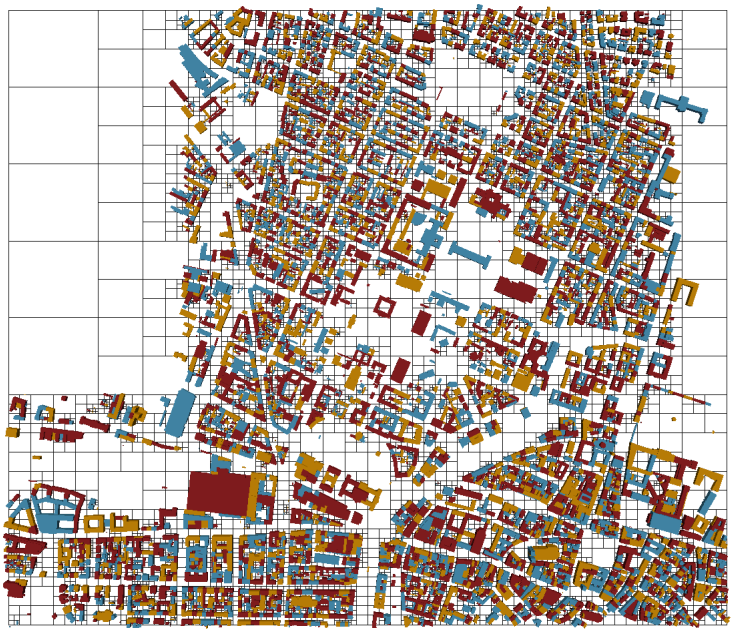

Figure 3.14: The quadtree representation of the product models is generated on the basis of the centre points of their bounding boxes projected along the z-axis.

Algorithm 15 generateZcurveOrder

```
int models[] = generateZcurveOrder(Octree o){

  models = addOctant(o.root);
}

int models[] = addOctant(Octant o){

  if (o.isLeaf()){
    models.add(o.models());
  else
    for (int i=0; i<8; i++)
      models.add(addOctant(o.children[i]);
}
```

is used for generating the mapping to the m processors by distributing them round-robin fashion to the $n-1$ slave processes.

By doing so, the Lebesgue curve ordering of the whole data set is achieved as given in Fig. 3.15 following the classical sequence-based approach and the modified round-robin approach as given in Fig. 3.16.

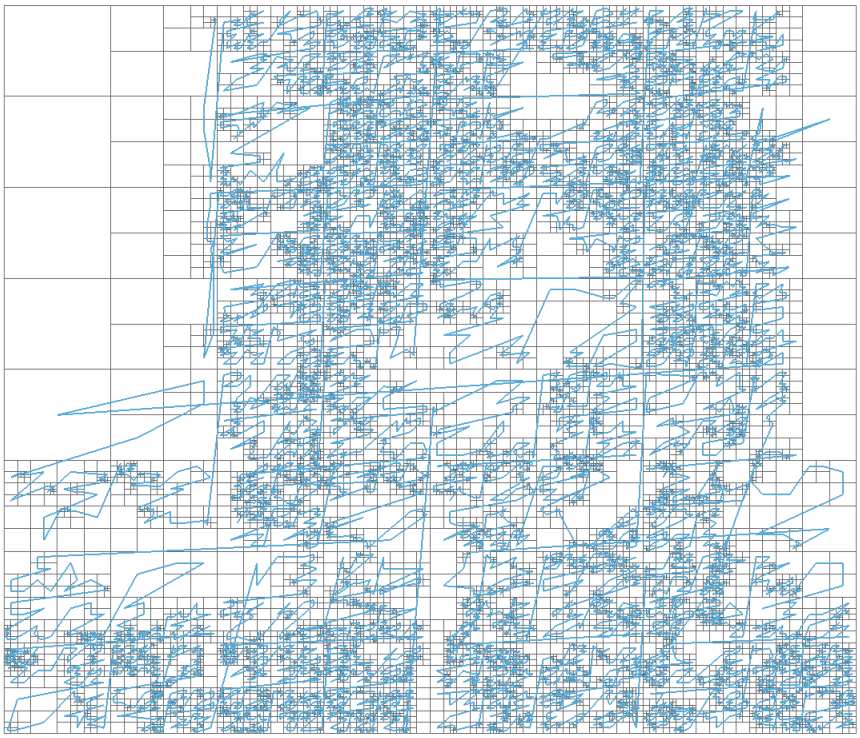

Figure 3.15: The quadtree representation of the product models' centre points is ordered following the pattern of the Lebesgue curve.

Figure 3.16: The quadtree representation of the product models is ordered following the pattern of the Lebesgue curve in the classical sequence-based approach (left) and the modified round-robin approach (right).

3.3.3 Distribution Quality

This section presents the quality of the load-distribution strategies implemented. The three load-distribution strategies investigated are the probabilistic mapping presented in Sec. 3.3.1

LoD	distance	[m]
3		≤ 100
2	> 100	≤ 300
0	> 300	≤ 1200
discard	> 1200	

Table 3.2: Depending on the distance of the product model to the investigation point the appropriate level of detail is applied and more distant constructions are discarded.

and the SFC-based approaches presented in Sec. 3.3.2 following the classical sequence ordering and the modified round-robin distribution of the product models to the slave processes. The augmented city model introduced in Sec. 2.6 is used as a test scenario, the distances for applying the respective LoDs are given in Table 3.2. The test scenario with the LoD application distances depicted is given in Fig. 3.17.

Figure 3.17: The test scenario is depicted together with the application distances for the different LoDs applied. The colour encodes the processor to which a building is mapped.

The distribution quality is a measure of how equally the triangles to be processed by every slave process are distributed. The number of triangles a product model is discretised with results from the level of detail it is processed on and therefore from the distance to the investigation point. As shown in Fig. 3.18, the round-robin ordered Lebesgue curve approach ensures there is a good distribution of highly resolved product models over the processors, not only over the whole computational domain but also locally. The classic Lebesgue curve

ordered approach tends to cluster the highly detailed product models to a single process, depending on the point of investigation. Therefore the round-robin ordered Lebesgue curve approach is used in the work which follows.

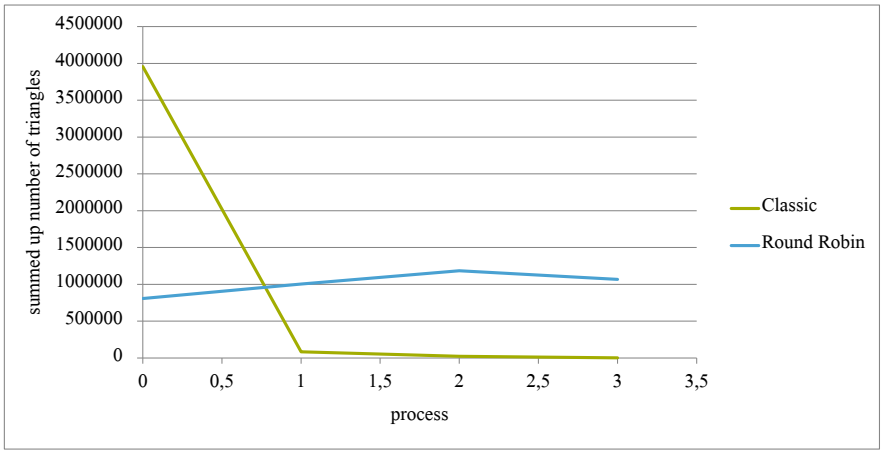

Figure 3.18: The distribution of the highly detailed product models is given as the summed up number of triangles to be processed by every process for the classic Lebesgue curve mapping and the round-robin order, both for the scenario given in Fig. 3.17. The modified round-robin order also distributes the data locally and therefore achieves an almost equal load on the slave processes.

Chapter 4

Visualisation and Exploration

In this chapter, the visualisation and exploration of the hierarchically ordered data is introduced. As the focus of this work is on handling large sets of complex product model data and especially gaining insight from the fused data set and its multi-level information, visualising and allowing exploration and processing approaches for product model data are investigated.

It will be shown that the hierarchical dual layer approach introduced in Sec. 3 gives an efficient evaluable structure to the vast set of raw information. This structure makes the visual exploration from the global scale and the product model scale down to the construction detail scale possible and still provides the access to the auxiliary information. How this feature-rich information is stored differentiates the way complex data is handled in the work presented from that of pure geometry processing. Besides the visualisation, the hierarchical order of the data enables the extension of complex product model scale exploration and evaluation approaches to the global scale without a loss of information depth.

This chapter is organised as follows. In Sec. 4.1.1, the efficient visualisation of the multi-resolution representation of the data basis is presented. In Sec. 4.1.2, the integration of simulation results originating from parallel numerical simulations is presented. In Sec. 4.1.3, the visualisation of multi-resolution data developed is extended to complex parallel visualisation environments such as a CAVE.

Following the efficient visualisation of complex environments, the integration of handheld devices is presented in Sec. 4.2.1 in order to cope with the challenge of human-centred immersive installations. Besides the active navigation using input devices, the indirect navigation due to the motion of the user in the environment is also covered in Sec. 4.2.2. .

Finally, the exploration of the feature-rich product model auxiliary information, and the coupling to and extension of complex product model investigation approaches is presented in Sec. 4.2.3. The potential of hierarchically ordered fully detailed product model definitions for engineering applications is shown by extending a fine-scale spatial query approach to city-

wide data sets and by providing an efficient collision detection approach for supporting the urban construction planning of infrastructure projects.

4.1 Visualisation

The first step in investigating large data sets is visualising them. Therefore, visualisation techniques for complex data are introduced from the algorithmic aspect and their efficient implementation is given. In order to make it possible to dive into the exploration of the processed data, the extension of the visualisation to parallel complex visualisation environments is provided. Thus the discussion and implementation of parallel rendering to CAVE-like systems is given.

4.1.1 Visualisation of Hierarchical Ordered Product Model Data

The challenge of visualising large sets of complex product model data is how to extract the relevant parts at the right resolution and discard parts which are unimportant. Also, the ever-increasing performance of modern Graphics Processing Units (GPUs) means straightforward rendering of the whole data basis introduced in Sec. 2.6 is not possible. Therefore, when visualising a large data set, the invisible parts of it should be discarded and the remaining primitives have to be coarsened to a resolution which corresponds to the resolution of the projection on the output device. As a starting point for the tremendous existing work and results achieved on handling large sets of complex data, the user is referred to [79, 80, 81] and the references therein. In [82] Reichl et al. apply a multi-resolution multi-block grid for visualising large sets of particles interactively. Based on the particle distribution the block distribution is adapted and the particles are resampled.

The main characteristic which distinguishes the setting of the work presented from existing approaches is the unstructured nature of the data. This is further enforced by the dynamically changing information depth a single entity has to be processed on. Most approaches for visualising large data sets focus on structured data. Large data sets such as the results of numerical simulations or image data have a predefined structure of the data points which can be exploited. Either they are defined on a regular grid or on static space partitioning approaches. This is not the case for the data underlying this work.

The visualisation of the fused city model introduced in Sec. 2.6 is implemented using the open source standard OpenGL [83]. OpenGL is on of the standards for visualising scientific data and is platform independent. Therefore it has been chosen instead of standards such as DirectX [84], as OpenGL is generally available in visualisation laboratories.

In order to achieve real time visualisation of such a large data set, the hierarchical ordering

must be exploited on both levels of the data structuring, and the transition between the different scales must be covered.

Ultimately, a visualisation approach is achieved which makes real-time visualisation of the whole data set possible, ranging from a flyover exploration of the whole domain with all scales down to the construction details within buildings.

The greatest distinction of the visualisation technique applied can be derived from the scale or the focus on which the user explores the data. Either the position of the camera is located within one (single) building or the location is on the outside of all buildings. These two cases will be the determination criterion for handling the data and performing the visualisation.

If the camera is located within one building, the visualisation of the whole data set is reduced to visualising this single building. In that case, the hierarchical ordering of the octree will be used to efficiently evaluate the parts of the product model which are in the view frustum.

If the camera is on the outside, the visualisation of all product models will be reduced to the visualisation of their outer shells. For more distant product models even the coarser LoDs will be used for a larger reduction of the primitives to be rendered.

The render distinction between the inside and the outside of the product models neglects to a certain degree the influence of glass, windows and doors. It is clearly possible to see the inside of a product model from a close point on the outside, and from the inside of a product model the surrounding buildings are visible. Both situations can be covered with minimal impact. On the one hand, the outer shell of the product models can also be visualised with the textured surface representation presented in Sec. 2.4.3. On the other hand, the surrounding buildings can be rendered up to a certain distance in order to visualise the neighbouring buildings when exploring the interior of a product model.

The determination of whether the visualisation takes place on the inside of a building or not is performed using the voxel test introduced in Sec. 3.2.7 on the first level octree. Based on the distinction between indoor and outdoor rendering, the two visualisation algorithms will now be introduced in detail.

The visualisation of hierarchical ordered data is introduced in the following sections. Computer games and game engines show a tremendous ongoing development in visualising photo realistic scenes [85, 86]. Therefore the question arises of why this is given within the work presented. Visualisation in computer games uses instancing in order to reuse complex objects many times. This is not feasible for complex building descriptions where each building has to be resolved individually. Furthermore, the navigation possibilities are usually limited to graphs or can be estimated well in advance. Therefore, game engines can drastically reduce the amount of primitives to render and pre-fetch data with a very high hit rate. This is not the case in the proposed application.

Rendering a Single Product Model on the Local Scale

The challenge of rendering a single product model is equivalent to the task of efficiently selecting those primitives which are in the view frustum of the camera from a large set of triangles.

This is achieved by generating the second level octree for approximating the triangular mesh of the product model. The rendering of the triangular mesh is performed using the fixed function pipeline [87] of OpenGL. Triangular elements are rendered using the *glDrawElementRange* as follows. Initially, all triangular elements consisting of the face, vertex, normal and colour information are uploaded to the GPU as an array of structured data. In every render frame, the parts of the array which are rendered now have to be specified. The important point now is how to structure the elements of the triangular mesh in a way that, in every render frame, the identified visible parts of the mesh can be accessed efficiently in the GPU memory and can be pushed to the rendering pipeline.

It turns out that the octree representation gives such an efficient ordering of the triangular mesh. The octree is built up on the basis of the triangular elements of the mesh up to a certain depth. The leaf nodes of the octree now contain the triangles which are intersected by the octant. The visibility check of the triangular elements in the view frustum can now be transferred to evaluating the leaf nodes which intersect the view frustum as depicted in Fig. 4.1.

Besides the advantage that the visibility check does not have to be performed on the whole triangular mesh but on the leaf octants, the octree also gives the linear ordering of the elements. The depth-first order of the leaf nodes and therefore the order of the triangular mesh directly identifies the order of the sequence in the data array to be uploaded to the GPU. The triangular elements of every leaf octant are stored as a consecutive array. The individual arrays are now combined to one array and the offset and the length are stored in the leaf node of the octant.

A hierarchical structure of the triangular data is now achieved and in every rendering frame the octants intersected by the view frustum are identified. The intersected octants give the corresponding parts of the visible triangular mesh in the data array uploaded to the GPU. For these sequences the *glDrawElementRange* is called and follows the fixed function pipeline of OpenGL. Using the fixed function pipeline further acceleration techniques such as back face culling and Z-buffer testing e.g. [88, 89] can be performed automatically.

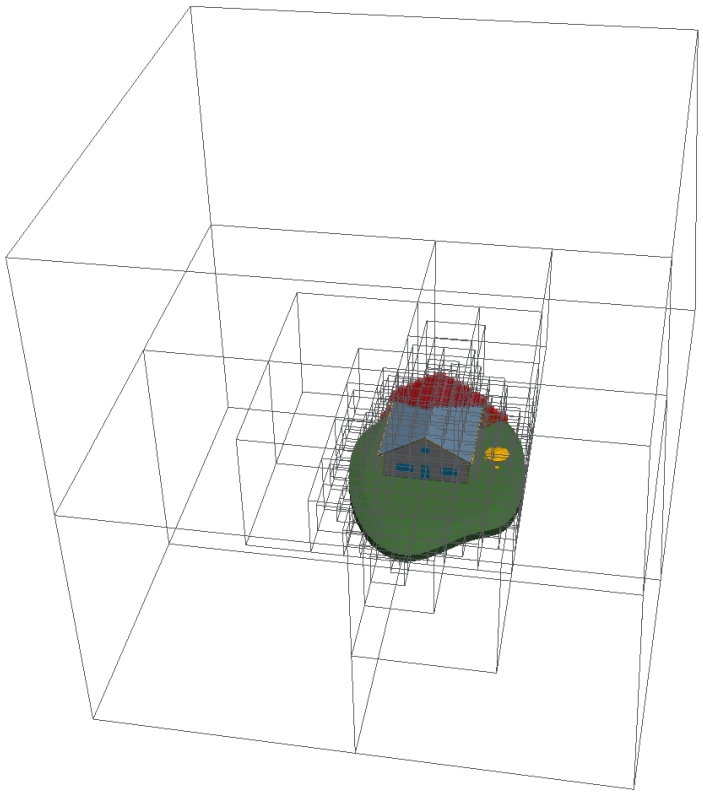

Figure 4.1: Based on an octree representation for a product model the identification of the visible triangles can be transferred to identifying the leaf octants intersected by the view frustum.

Rendering LoD Representations on the Global Scale

Rendering the bird's eye view of a fully detailed city model is achieved by visualising the outer shells of all buildings on different levels of detail depending on their distance to the camera. This is achieved by extending the parallel framework introduced in Sec. 3.2.1 by one further MPI process for visualisation and querying the dual layer hierarchical data structure as given in Fig. 4.2.

In the rendering loop, the visualisation process sends the updated position of the camera to the master process and performs a non-blocking receive to the master process. On the basis of the camera position, the master process identifies the levels of detail to be applied for every single product model and requests from the respective slave processes the update of the geometric representation of the product model with the correct level of detail as introduced in Sec. 3.2.3. After receiving all geometric updates the master process sends the updates one by one to the visualisation process. As soon as the visualisation process receives a geometric representation of a product model it discards the current representation, uploads the representation received to the GPU and performs the rendering of all available product models.

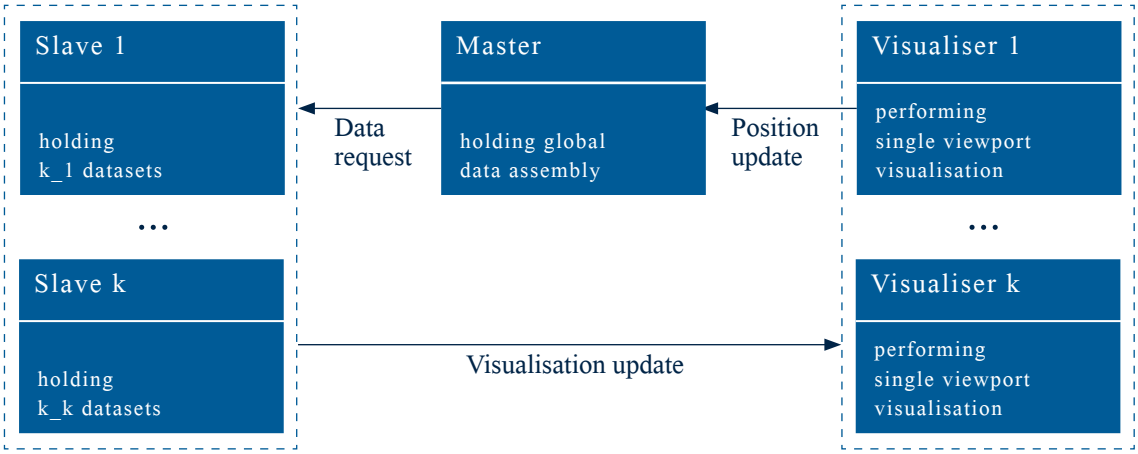

Figure 4.2: The parallel framework is extended by one further process which performs the visualisation. This process asynchronously triggers the adaption of the LoD representation by sending the updated position of the camera to the master processes. The slave processes affected perform the changes in geometry and the master sends the update to the visualisation process.

By doing so, the computational load is completely separated from the visualisation process. The visualisation process holds only a set of geometric representations and sends the updated position of the camera to the master process every n-th frame. The master process provides the necessary LoD updates as to which product model has to be rendered on full detail, which model is coarsened, and which product model is removed from the rendering as it exceeds the distance of the coarsest level of detail.

As the receive of the product models is performed in a non-blocking fashion by the visualisation process, the geometric updates are integrated as soon as they have arrived and the transfer of the data is hidden behind the visualisation process. The visualisation process only updates the data set to be rendered based on the updates received from the master and performs the efficient visualisation of this data set.

In order to save further computations of the GPU, the visualisation process also checks for a product model if it intersects the view frustum as given in Fig. 4.3.

The visibility check is not performed on the whole triangular geometric representation of the product model but limited to checking if a product model's bounding box intersects the view frustum. By doing so, the efficient visualisation of the whole city model is achieved. Fig. 4.4 illustrates the application of different levels of detail to one specific product model on the city-wide scale. With increasing distance, the resolution is decreased and the representation is coarsened. Fig. 4.5 depicts different levels of detail for the embedding of a fully detailed product model to a coarse polygonal description of its surrounding. The hierarchical level of detail approach enables real-time visualisation and a frame rate of at least 60fps is achieved.

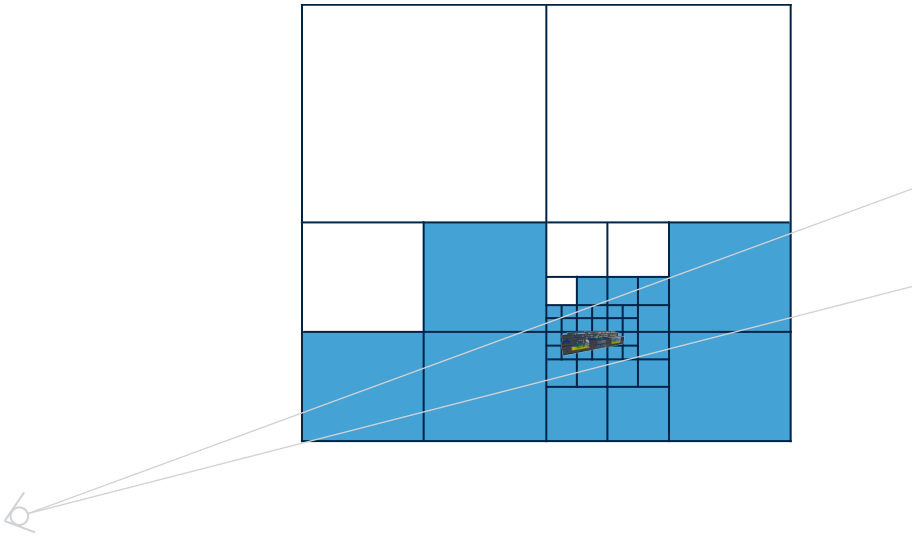

Figure 4.3: The computational load of rendering the outer shells of product model data on the GPU is reduced by limiting the visualisation to product models whose bounding boxes intersect the view frustum.

Figure 4.4: Different levels of detail are visualised for one specific building. With increasing distance, the level of detail gives a coarser representation of the product model.

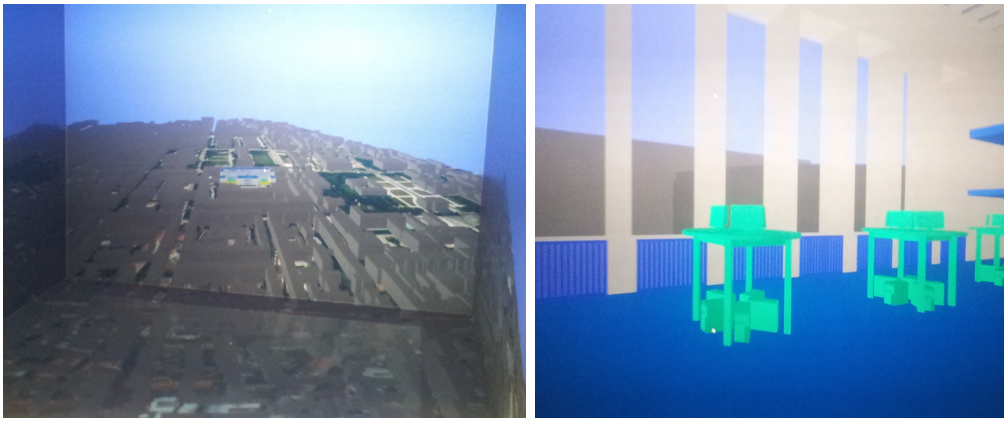

Figure 4.5: Different levels of detail are visualised for the embedding of a fully detailed product model to a coarse polygonal description of its surrounding. The visualisation is given on the city-wide scale (left) and on the product model scale (right).

4.1.2 Integration of Simulation Data

In this section, the visualisation of simulation results originating from parallel numerical simulations is introduced. This is achieved by deriving a triangulation of the evaluation of interest from the simulation results and applying the efficient visualisation technique for triangular meshes introduced in Sec. 4.1.1.

In this work, urban flooding scenarios are investigated using the three-dimensional free surface simulation of fluid flow and therefore the isosurface calculation of the water front in urban regions is investigated. This will be performed by applying the Marching Cubes (MC) algorithm [90] which derives a triangular representation of the isosurface for a given three-dimensional flow field. In the literature there are many further evaluation techniques for visualising quantities of interest in the field of Computational Fluid Dynamics (CFD) [91, 92].

Results from CFD are usually stored as a specification of the three-dimensional velocity vector and scalar quantities such as pressure on a regular grid. In the case of two phase simulations the concentration of one phase is also given as a so-called *alpha* value on the regular grid. For simulation results on axis parallel adaptive grids the regular grid representation can be retrieved by resolving the adaptive parts to the finest resolution in the mesh. For non-structured grids the simulation results can be re-sampled and interpolated to a regular grid.

The Marching Cubes algorithm takes a regular grid of scalar data as its input data and calculates the triangular representation of the isosurface to a given threshold value. The algorithm generates the triangular mesh by combining the triangular representation of the partial isosurface on every voxel of the regular grid.

The isosurface on a single voxel can take one of a limited number of cases depending on the values on the edges of the voxel and the given threshold values. A voxel consists of six faces, eight vertices and eight edges. The scalar quantity for which the isosurface is generated is known on all eight vertices. Therefore it can be evaluated along every edge if the threshold value lies between the values at its vertices. Every edge thus has Boolean information regardless of whether it is intersected by the isosurface or not. From this an eight digit binary status of the voxel is derived which describes the intersection status of its eight edges. These 256 statuses of a voxel can be reduced to 15 different triangulations of a voxel's isosurface due to symmetry. These triangulations are available as a Triangle Lookup Table (TLT) as given in Fig. 4.6.

The generation of the triangular mesh can be trivially parallelised by distributing an equally large subset of the voxels to each of the n processors, generating the triangulation on every subset and combining the results as given in Fig. 4.7. By performing the MC algorithm, isosurface representation of the three-dimensional free surface simulation of fluid flow is achieved as given in Fig. 4.7.

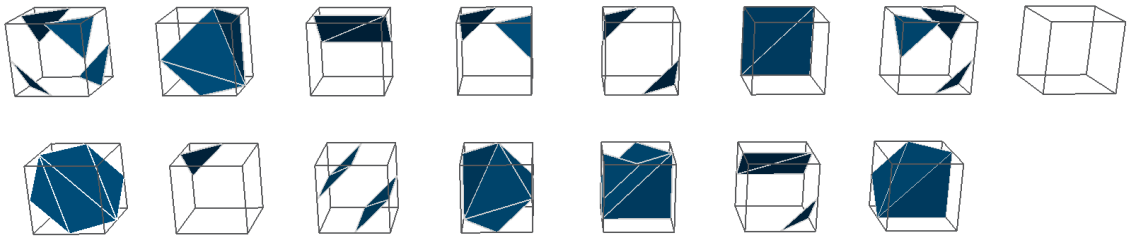

Figure 4.6: For every voxel, the MC algorithm identifies the intersection status of the isosurface along its eight edges. The resulting 256 different cases can be reduced due to symmetry to 15 triangle templates and are available as a lookup table.

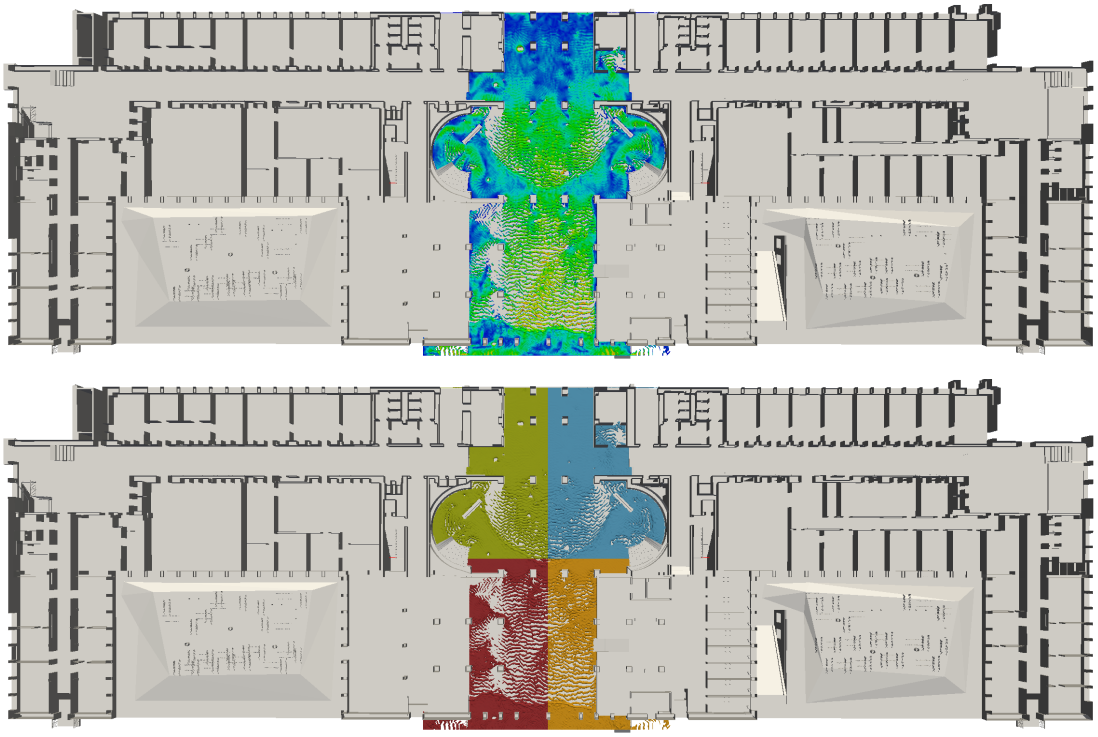

Figure 4.7: For a given flow field the MC algorithm calculates the triangular representation with colour-encoded velocity of the isosurface to a given threshold value (top). In order to parallelise MC, the flow field is split into n - here 4 - equal parts (bottom) and the triangulations of the sub domains are combined to the isosurface of the complete flow field. The colour identifies the processor mapping.

4.1.3 Parallel Visualisation in CAVE-like Environments

In this section, the implementation of the framework presented for parallel multi-screen visualisation environments such as a CAVE is presented. Multi-monitor and CAVE-like installations [93, 94] give the user of a system the possibility to dive into the data and gain insight on the simulations explored and performed which is far beyond the investigation on a standard desktop machine.

In order to perform the visualisation on parallel multi-monitor visualisation installations, the Equalizer [95] SDK has been coupled to the framework presented. Equalizer is a scalable parallel rendering framework and provides generic access to various visualisation applications. It provides an API for implementing the visualisation on a parallel basis and provides the communication and synchronisation approaches for rendering on a parallel multi-monitor installation.

In order to estimate the demands of a parallel rendering framework, a six-sided stereo view CAVE installation shall be investigated. The Cornea@KAUST [96] is such an installation where each of the six sides of the cave is driven by four Sony 4K projectors as shown in Fig. 4.8, two for the overlaying tiles on every side and two for the stereoscopic immersion. In total this sums up to 24 projectors and to a resolution of approximately 200 megapixels. On every frame 24 images have to be generated for the individual projectors and for real-time visualisation 50fps should be achieved. This sums up to a theoretical frame rate of 1000fps to be distributed to the parallel visualisation cluster.

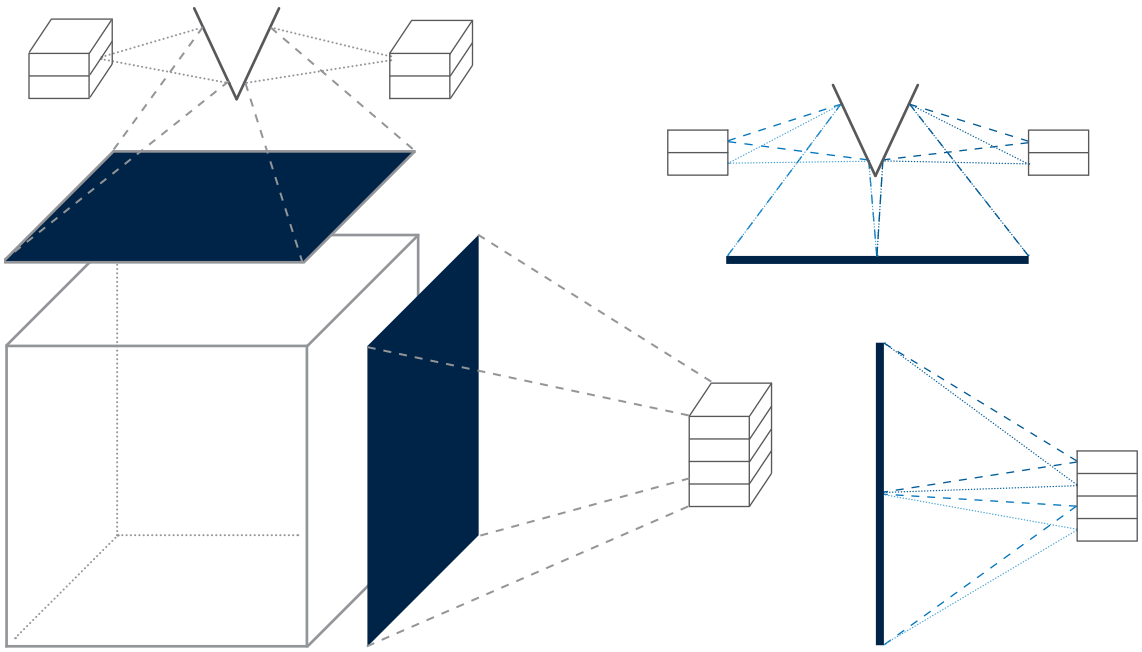

Figure 4.8: The six-sided Corena CAVE@KAUST is driven by four projectors for every side, two for the vertical tiling and two for the stereoscopic immersion.

In order to achieve the required performance, parallel visualisation by integrating the parallel dual layer hierarchical framework to the Equalizer framework has been implemented.

The Equalizer supports general types of multi-monitor installations and can be adapted to almost all visualisation laboratories by configuring the topology of the hardware. Using the configuration start-up parameters every projection plane of the hardware installation is specified by three linearly independent points. By specifying the topology of every projection

plane and adapting the eye separation factor for stereoscopic view, Equalizer identifies the visualisation processes to be started on all rendering clients and adapts the access to the GPUs as given in Fig. 4.9.

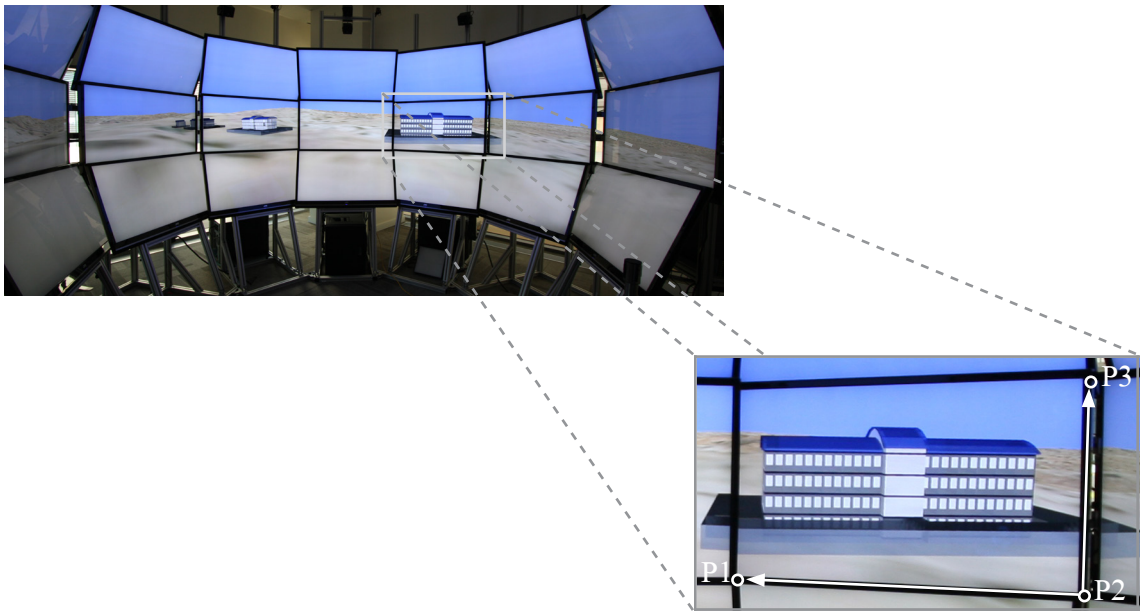

Figure 4.9: Three linearly independent points in every projection plane identify the topology of a multi-monitor installation.

It can now also be seen that the Equalizer framework is ideal for the parallel access framework presented. As every projection plane in the visualisation environment is run by a single parallel process as given in Fig. 4.10 for the Z2@KAUST [97], the visualisation of one frame on 40 monitors, for example, can be transformed to performing the visualisation of the same data set on 40 parallel processors.

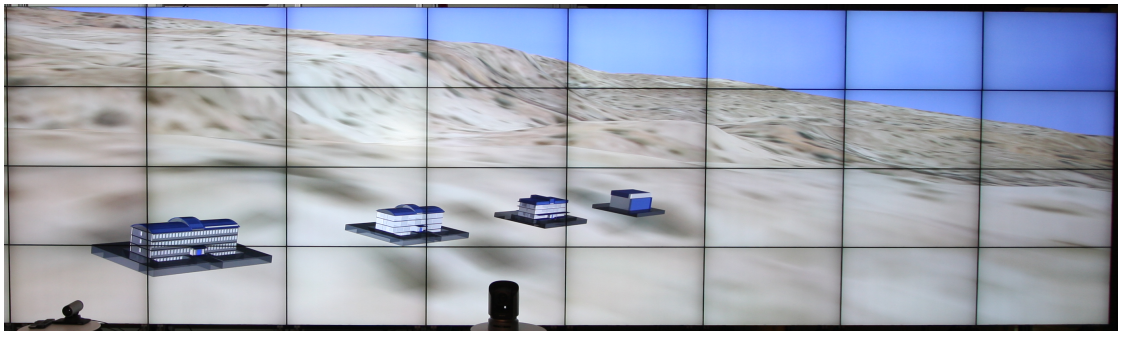

Figure 4.10: The tiled wall installation Z2@KAUST consists of 40 tiled monitors on a 10×4 grid. The rendering on the whole installation is split into parallel rendering of one process for every single monitor.

From the configuration of the topology of the installation, Equalizer derives the transformation of the view and the projection on every monitor. It is therefore sufficient to ren-

der the same data set on every monitor using the visualisation algorithms introduced in Sec. 4.1.1, 4.1.2.

To sum up, using the Equalizer framework makes it possible to integrate multi-plane visualisation environments to the framework presented, if it is possible to provide the visualisation data to n parallel running visualisation processes, one for every plane of the installation.

This parallel visualisation scenario can be implemented using the hierarchical data access framework presented as follows. The parallel execution of the framework given in Sec. 3.2.1 is extended by n instead of 1 visualisation processes and results in the configuration given in Fig. 4.11. One process is defined a priori as the head visualisation process and has additional functionality to send the updated position of the camera to the master process.

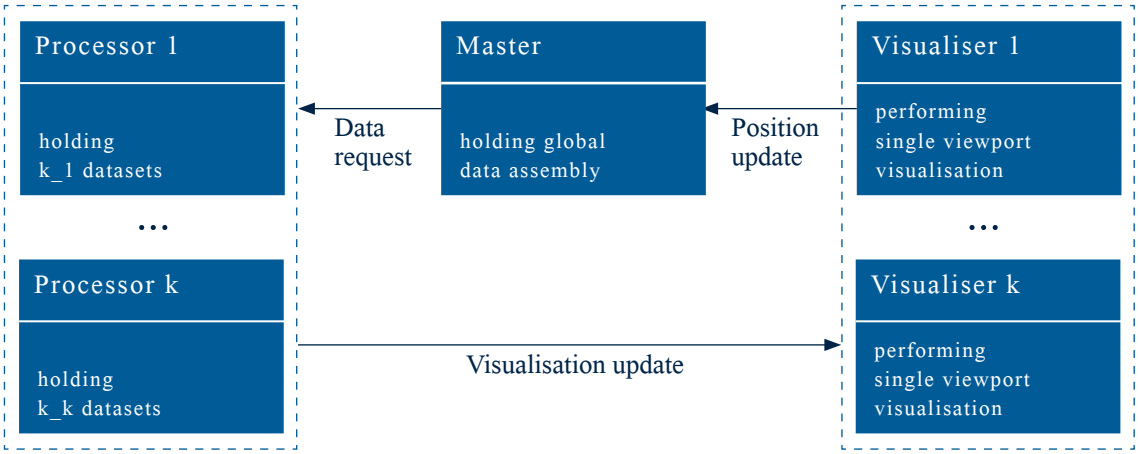

Figure 4.11: The parallel set-up of a framework in a multi-monitor installation consists of additional n visualisation processes, one for each projection plane, where one process additionally sends the camera position updates to the master process.

With this set-up the framework presented has been ported to multiple complex visualisation environments installed at KAUST VizLab such as the Cornea shown in Fig. 4.12 and the NEXcave shown in Fig. 4.13. It has been shown that the general parallel access strategy of the framework allows direct integration of almost any complex visualisation environment.

4.2 Navigation and Data Exploration

Besides the technical challenge of integrating multi-monitor installations, exploiting the potential of immersive environments depends on the intuitive navigation of the exploration and the integration of investigation possibilities. The whole application will only be of value to an engineer or scientist if they are able to interact with the system in an intuitive way and gain insight into the data presented.

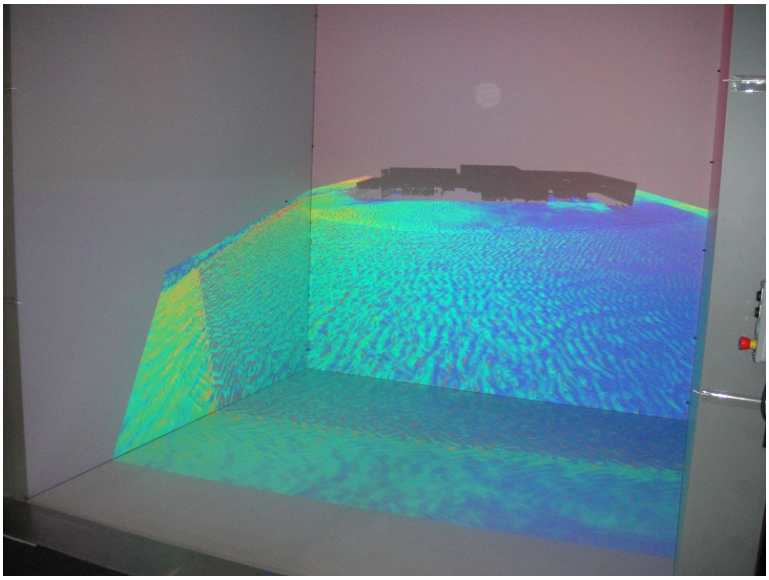

Figure 4.12: On the Cornea@KAUST the visualisation of simulation data is performed by following the parallel visualisation on 24 digital projectors and 6 sides, here with 5 sides due to the opened door of the CAVE.

Figure 4.13: On the NEXcave@KAUST the visualisation of fully detailed product model data on a local scale is performed by rendering the outer shell of the constructions on 21 monitors.

Therefore navigation using handheld devices is introduced in Sec. 4.2.1, since the use of standard keyboard-mouse approaches will focus the user on navigating in the system and not on gaining insight into the data. Besides the navigation in Sec. 4.2.2 the integration of tracked interaction systems is presented. This gives the user the possibility to *grab* and investigate details of the data as desired. The free navigation in and selection of the details of interest makes the multi-resolution and in-depth data access provided by the system presented valuable.

Finally, the extension of complex construction-scale evaluations to the city-wide scale is introduced in Sec. 4.2.3. There are many approaches for investigating data from construction and built infrastructure in detail, these techniques will be ported to a global scale while avoiding the information depth on the fully detailed product model data.

4.2.1 Interacting Handheld Devices

Handheld devices have nowadays become a mature technology [98, 99, 100, 101] and are commonly available to the users. Hardware driven by iOS [102] and Android [103] combines sufficient computing power and various input and sensor techniques in a pocket-size format.

In order to exploit this potential, navigation using the gravity sensor of a handheld device has been developed in order to provide the user with a navigation tool that lets them focus on exploring the data instead of driving the navigation. Besides the standard gravity sensor, the Wireless Lan [104] (WLan) and the Hyper Text Transfer Protocol [105] (HTTP) interfaces provide the technical interfaces needed to navigate in complex visualisation environments.

The structure of handheld navigation is given in Fig. 4.14 and works as follows. The rotational position in the global coordinate system is determined by means of the gravity sensor of the device. The acceleration, turning angle and direction are derived from this position. Using the HTTP protocol, the resulting navigation direction and speed are transferred to the visualisation system and thus replaces the keyboard-mouse-based navigation with an intuitive steering.

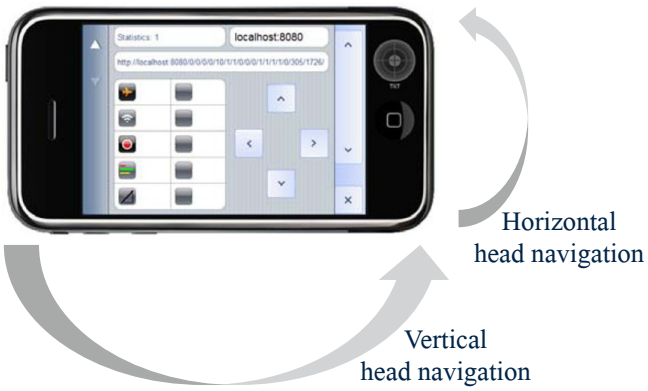

Figure 4.14: The gravity sensor of a mobile device is used to determine the desired direction, turning angle and speed of navigating through the data.

The navigation commands derived from the gravity sensor and the interface are now serialised to a command string. The serialised command contains the communication address and port of the framework, the rotation angles and speed of navigation, binary parameters such as wire frame rendering and the checksum and counter for deserialising the request.

In order to perform the communication between the handheld device and the framework a communication protocol has been implemented as given in Fig 4.15. The framework runs an HTTP server using the Boost::Asio [106] component supporting asynchronous processing of incoming HTTP packets. As soon as a serialised command has been transferred to the framework it is processed in the render loop. In order to cope with the possible imperfect transfer, the validity of the HTTP request is checked against the received checksum, invalid requests being discarded.

Requests are only processed in an ascending counter order. In the given scenario the transfer frequency has been set to 20Hz. As it turned out that individual requests are not received in the order they are sent from the handheld device, the requests have been assigned a time stamp by the counter. As soon as a request of a certain counter is processed, older requests are defined to be outdated and not carried out.

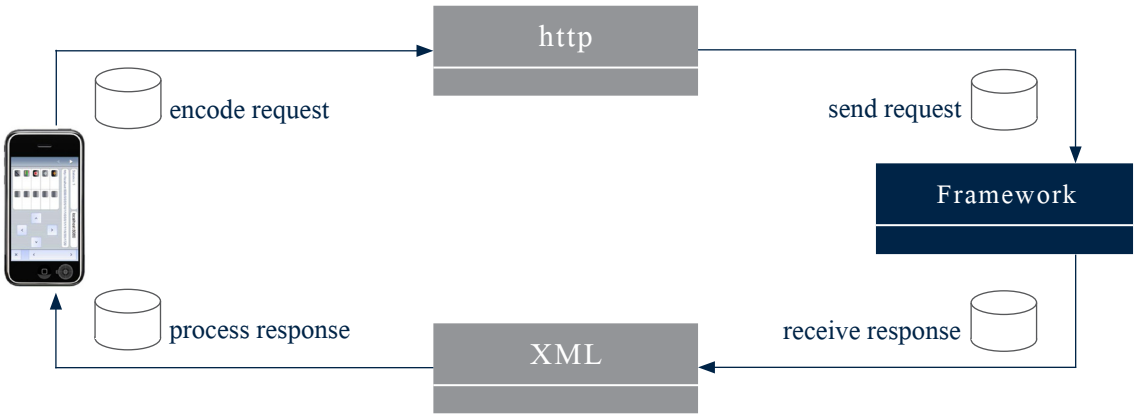

Figure 4.15: The communication protocol between the handheld device and the framework transfers the serialised request to the framework. Besides the navigation parameters, the checksum for evaluating the correct transfer of the broadcast and the counter for evaluating the order of the requests are also transferred.

Ultimately, a navigation interface in complex visualisation environments is achieved using handheld devices as shown in Fig. 4.16 which replaces the unsuitable use of keyboard-mouse navigation for such installations and uses standard techniques such as WLAN and HTTP which are generally available in visualisation laboratories.

4.2.2 User Tracking and Tracked Interaction

In the previous sections the extension of the visualisation to multi-monitor immersive environments and the navigation within these installations has been introduced. At this point the concept presented allows the user to explore the data and navigate through them in a way that lets them dive into it.

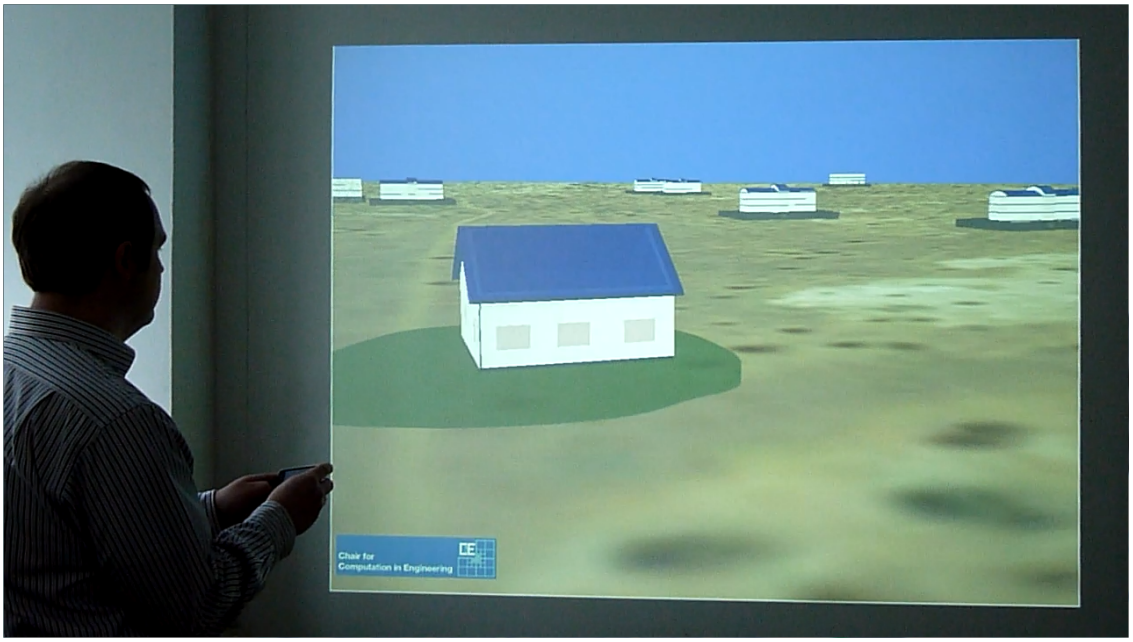

Figure 4.16: Handheld devices provide the user with a navigation interface which lets them focus on exploring the data set instead of handling the input device.

In this section, the focus shifts to the fact that, in immersive environments, users move their whole body, their arms and their head and have the freedom to navigate and interact with all these components. Technically this freedom can be formulated as the demand to determine the position and the direction of different parts within the immersive system. This goal is achieved by incorporating tracking systems [107].

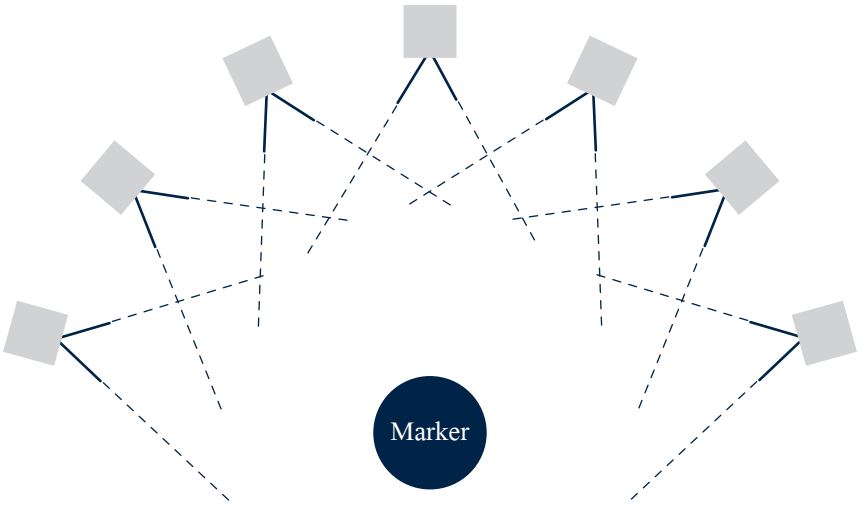

Figure 4.17: In a tracked immersive installation the marker's position and direction is derived by recording the interior of the installation with multiple cameras for different viewports. From the differing images of the reflecting parts of the marker the position and direction of the marker and therefore the part it is attached to can be derived using image processing and linear calculus.

A tracking system consists of a tracked device, the so-called marker, and the tracking cameras as depicted in Fig. 4.17. The marker consists of multiple linearly independent aligned reflecting spheres attached to a structure and a set of cameras which record the interior of the immersive environment. Owing to the different reflecting properties of the spheres and the linearly independent overlapping view directions of the tracking cameras the position and direction of the marker can be determined.

Having the position and direction of the marker at hand, two basic demands for immersive installations can be fulfilled. On the one hand, the user moves in the immersive installation and therefore their viewport in the visualisation changes. This is covered by wearing a hat with a marker attached to it which is tracked by the installation. The viewport is derived from the position and view direction of the user's head and the correct visualisation is computed as shown for the NEXcave@KAUST in Fig. 4.18.

Figure 4.18: The position and direction of the user's view is derived from the tracked marker mounted on their head. From this information the viewport and thus the projection onto the different planes of the CAVE are derived, here for the NEXcave@KAUST.

On the other hand, the users have their hands free and should be given the possibility to point at entities. The mouse interaction is therefore replaced by a tracked pointing device as shown in Fig. 4.19.

The implementation of the tracked interaction of the framework has been performed using the Dtrack [108] installation of VizLab@KAUST. Dtrack supports multiple tracked devices and provides the position and direction of the tracked markers using a separate image processing client. Owing to the integration of the Equalizer framework as introduced in Sec. 4.1.3 tracked user movement and tracked scene interaction could be achieved in order to allow the user a seamless interaction with and insight from the data.

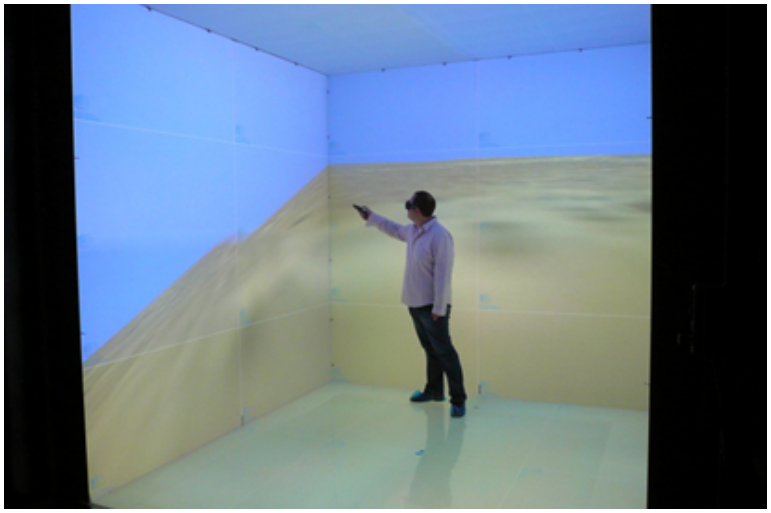

Figure 4.19: A tracked pointing device replaces the impractical mouse interaction in immersive installations and lets the user freely point at the visualised items, here shown in Cornea@KAUST.

4.2.3 Product Model Investigation and Engineering Applications.

In this section, the product model exploration features and the integration of complex construction scale investigation applications are introduced. From the multi-monitor visualisation and immersive navigation approach introduced in the previous sections a tool is provided which lets the user visually explore the geometric representation of the multiple resolutions of the data basis.

Initially, the access of the auxiliary information is presented. The user has thus the possibility to navigate through the whole data set on a city-wide scale, focus on the exploration of a single construction and then explore the construction detail information such as the insulation type of a window or the IP address assigned to a network port as given in Fig. 4.20.

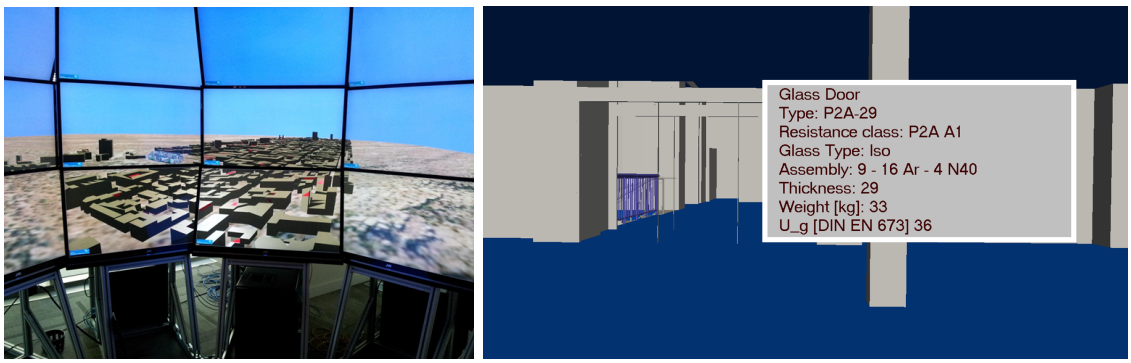

Figure 4.20: The navigation of the whole data set lets the user explore on a city-wide scale (left). By navigating to the construction details of a product model the full information depth such as the material parameters of a door installation is achieved (right).

Accessing auxiliary information follows the dual layer access concept of the framework presented. The position and the direction of the device pointing to a construction detail are derived from the integration of tracked pointing devices introduced in Sec. 4.2.2. The geometric information of the ray pointing from the input device to the screen is initially evaluated over the first level octree in order to determine the constructions intersected by the ray and then to extract the construction details on the second level octree as introduced in Sec. 3.2.4.

This approach now lets the user evaluate the whole data basis on all scales, from the global scale of the whole data set down to the local scale of construction details. This approach therefore distinguishes the work presented from pure geometry handling and visualisation approaches.

Since the approach adopted in this work can bridge the information linkage gap between large-scale city models, which give the embedding of a whole urban region to its surrounding, and Building Information Models, the extension of BIM scale applications to the whole city scale is introduced in the following. The applications introduced cover spatial queries to product model details and the support of the construction planning of large-scale structures.

Spatial Extension of the Structured Query Language

In this section, the global evaluation of spatial queries for product model details is introduced. This enables questions of the following type to be answered. *What is the capacity regarding hospital beds within 50km? Where is the closest fire extinguisher?* There is in-depth research on evaluating individual product models originating from BIM and extending this to GIS sources [109, 110].

In the work presented this approach is extended to taking into account the spatial relation of all product models to each other. By doing so, the evaluation of the construction details over the whole domain becomes achievable and queries as given in Alg. 16 become feasible. In this query, all buildings are identified which have a central air conditioning system installed and whose floor space exceeds 1000sqft within a diameter of 50.000ft

Algorithm 16 Pseudocode for a query combining spatial and IFC criteria

```
SELECT  *
FROM    building b
WHERE   b.IFC_has_AC = TRUE
AND     SUM(b.IFC_floor_space) > 1000
AND     DIST(my_position - b.center) < 50.000
```

The processing of such queries is performed by exploiting the hierarchical structure of the framework as follows. The master process identifies all product models fulfilling the global

distance criterion and from that the respective slave processes. The master process sends these slave processes the request to evaluate the product model scale part of the query. The resulting record set is collected by the master and the complete query is processed.

The order set for processing spatial queries has been restricted to distance-based operators on the global scale and selective operators on the product model scale, but could be extended by combining the existing work to the spatial query of product models discovered in [111].

Urban Construction Planning and Investigation

The final application for evaluating product model details over their global embedding comes from the field of urban construction planning and focuses on the planning of large-scale infrastructure projects. In Fig. 4.21 this question is targeted by evaluating the possible intersection of the sewer network with existing constructions in the city.

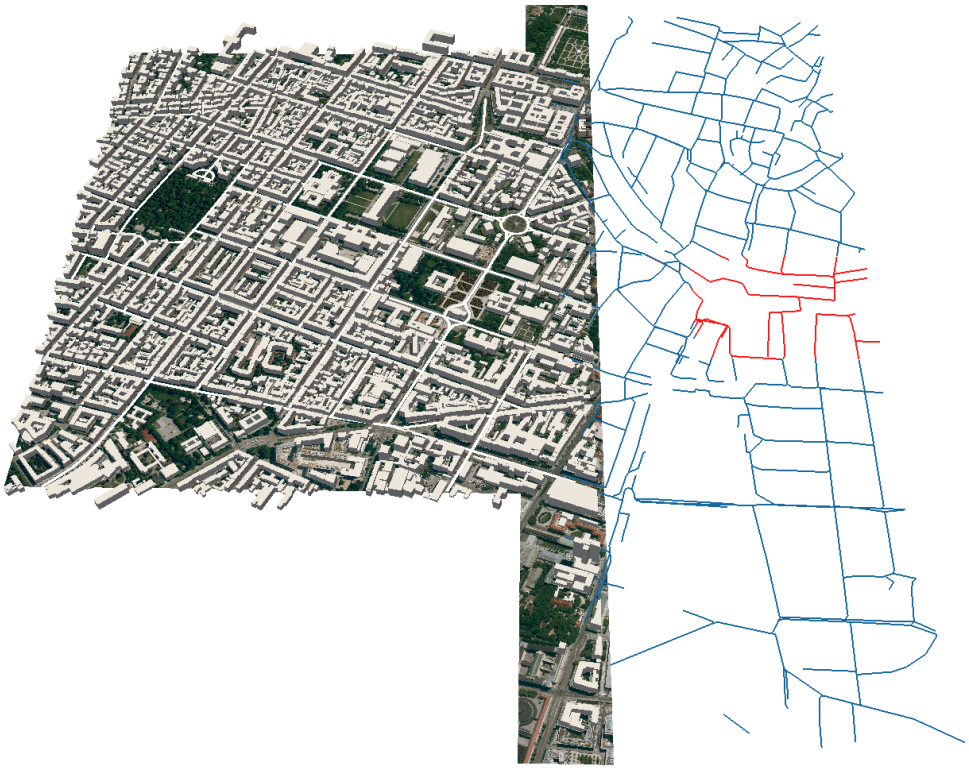

Figure 4.21: The evaluation of possible intersections of a sewer network with existing constructions in the city is performed by deriving the hierarchical representation of both configurations.

Following the approach of the framework presented this question can be answered using an octree evaluation called multiplexing [112]. As introduced in Sec. 3.2.5 the tree structure of an octree can be linearised in a depth-first manner. This linearisation of two octrees is now concatenated level-wise with the logical *AND* operator. The resulting stream of the operation now determines the intersection of non-white parts of the two octrees as given in Fig. 4.22.

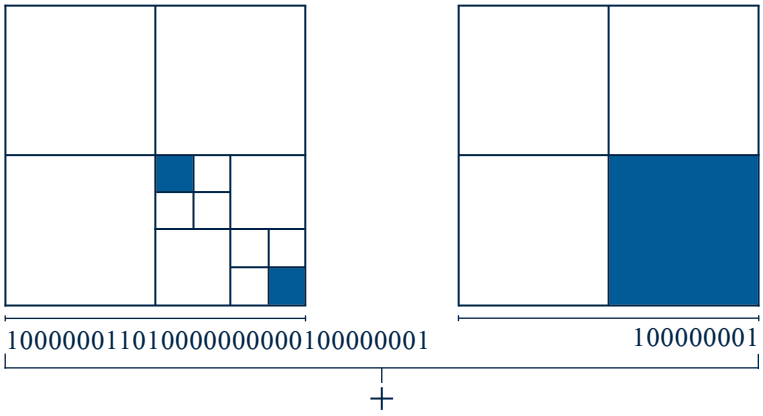

Figure 4.22: In order to determine the intersection of non-white parts of the two octrees their linearisation is concatenated level-wise with the logical *AND* operator.

This algorithm is now applied to the existing and the planned constructions as follows. The existing constructions and built infrastructure of the city are resolved up to the accuracy of their bounding boxes by the first level octree. A separate octree is now generated for the newly planned construction. By performing multiplexing of both the octrees the intersection of the planned construction and the existing city data is determined. All intersected parts of the first level octree directly provide the product models of the existing constructions intersected. Octrees are now generated for both the bounding boxes of the intersected constructions and with the same domain for the planned construction. These octrees are now pairwise multiplexed against each other and the possible intersections provide the fine-scale intersections of the planned and the existing constructions over the whole domain.

This approach is based on comparing octree representations. Fig. 4.21 shows the intersection of the sewer network and a planned construction. As the sewer network is composed of fine pipe segments which mainly follow a horizontal orientation, the octrees have to be highly resolved – in this case up to level 13. This induces high computational effort which can be tackled by performing a multi-level intersection test. Such an approach can determine the possible intersections of the compared constructions up to a certain resolution depth and proceed in an iterative way for the identified sub-domains.

Chapter 5

Coupling Grid-Generation and Simulation Frameworks

In this chapter, the coupling of the data access framework to three parallel grid-generation and numerical simulation frameworks is introduced. It will be shown that the parallel hierarchical data access concept presented in this work directly integrates into existing grid-generation and simulations frameworks. Three different interface and coupling approaches follow. Coupling to third-party frameworks has been developed and implemented due to a variety of reasons.

Having a parallel-access framework at hand provides the data basis for engineering-relevant applications. Pure data do not provide insight without sophisticated applications based on these data. Furthermore, this framework enables researchers with a focus which is not on handling complex data but on the efficient numerical simulation of physical phenomena to incorporate the data handled by this work.

Moreover, the demands of modern hardware installations and problem-solving paradigms exceed the capabilities of a single heroic research approach. Focusing on all steps of the simulation pipeline is simply not possible and therefore the investigation of interfaces between the different steps of the simulation pipeline and specialisation of different groups on these steps is imperative. Most parallel solving environments these days are tailored to a single application and one specific hardware installation. The development of generalised grid-generation and simulation frameworks, involving new software development paradigms and code structuring, may overcome these boundaries and lead to applications whose lifetime does not end together with the respective hardware installation to which they are tailored.

Ultimately, coupling this work to existing approaches and research in progress shows the applicability of this approach and proves the general interoperability of the framework presented. Furthermore, the coupled frameworks bring a set of solvers, problem-solving scenarios and evaluation techniques with them and therefore leverage the value of this work.

This chapter is organised as follows. In Sec. 5.1 the Sierpinski [113, 114] framework is coupled which solves two-dimensional hyperbolic problems on full adaptive triangular grids. In Sec. 5.2 the Peano [77] framework is coupled, a grid-generation framework of three-dimensional adaptive Cartesian grids. Finally, in Sec. 5.3, the Open Source Field Operation and Manipulation (OpenFOAM) [115, 116] package is coupled, a package with a large base of developers and applications from science and industry. With the integration of OpenFOAM a large community working on all fields of computational science and their scientific impact is integrated.

In this chapter, the focus is solely on integrating the grid-generation capabilities. The applications and solving of specific numerical simulation questions will be introduced in Sec. 6, 7. This separation is inspired by the distinct evaluation of date handling, grid generation and numerical simulation. It underlines the different possible integration points of the framework presented in the steps of the simulation pipeline.

5.1 Coupling the Sierpinski Framework

In this section, the coupling of the hierarchical data access framework presented to the Sierpinski framework is introduced. The Sierpinski framework is a *Generic Framework for Full-Adaptive 2D-Discontinuous Galerkin Methods based on Sierpinski Space Filling Curve*. It is an open source software package which uses dynamically changing grids to solve hyperbolic equations in two dimensions.

The grid generation is performed on the basis of triangular elements and implements full adaptivity, including mesh refinement based on runtime adaptivity criteria. he order of the elements is kept using the Sierpinski SFC over the spacetree representation of the computational domain. A novel parallelisation and load balancing strategy is implemented on the basis of the splitting of the spacetree along the Sierpinski curve. Further properties of the Sierpinski framework are the interactive simulation in an OpenGL environment, hybrid parallelisation with a shared memory parallelisation using OpenMP and Intel TBB and distributed memory parallelisation using MPI.

Various solvers can be integrated into the framework utilising an object-oriented kernel-based approach. Besides a Riemann solver, the GeoClaw [117] package has also been integrated in order to perform Tsunami simulations. Special focus is put on the cluster-based simulation and load balancing using data migration in hybrid parallelisation scenarios.

The fluid flow simulations follow a Discontinuous Galerkin [118, 119] approach in order to solve the two–dimensional Shallow Water Equations with Euler time-stepping and constant bathymetry. In Fig. 5.1 the simulation of a radial dam break using the Sierpinksi framework is given.

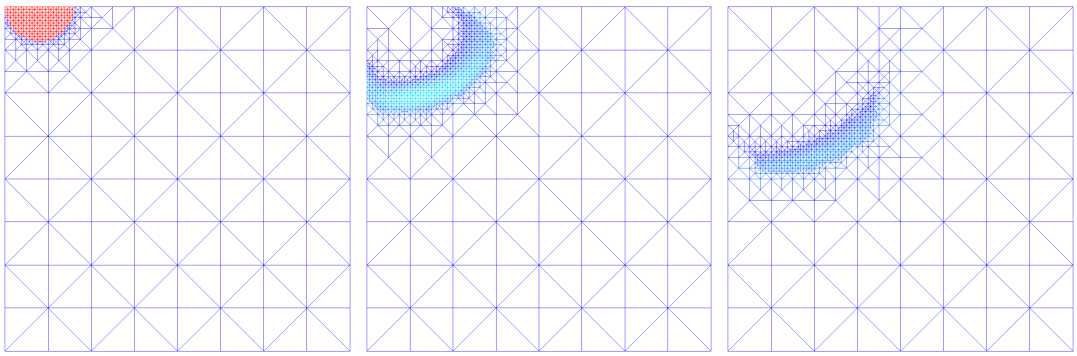

Figure 5.1: In radial dam break scenarios for the two-dimensional SWE a reservoir of fluid at rest is initially applied by the boundary conditions. The initial resolution of the triangular mesh is 16×16 triangles, the resolution depth is limited to 6 levels. The solution at the time-step iteration is given together with the refined mesh and the colour-encoded height.

The coupling of the Sierpinski to the framework presented can be established by specifying the bathymetry, i.e. the elevation of the ground over which the fluid flow is simulated. As the framework presented contains a fully detailed three-dimensional data basis, the two-dimensional mesh has to be derived from the three-dimensional data and this will be introduced in the following. The resulting two-dimensional bathymetry mesh is stored using netCDF [120], a machine-independent data format for array-oriented scientific data. Sierpinski directly supports netCDF so the coupling is established by exporting data from the framework presented to a valid input file for the Sierpinski framework.

5.1.1 Derivation of the Bathymetry Discretisation

In order to derive a two-dimensional bathymetry discretisation for a specified computational domain, the three-dimensional description handled by the framework presented has to be dimensionally reduced. This projection is not loss free in general as it is obvious that not all three-dimensional information can be expressed in two dimensions. For pure surface descriptions such as the terrain information derived from GIS as introduced in Sec. 2.3.1, the two-dimensional mesh can be derived directly as a mesh in the plane with the height of the terrain at every data point. For complex data such as buildings and constructions the height value of every point in the plane is not unique. When the doors or windows are open, for example, it is necessary to specify which value should be the height value. This can be the height of the door sill or the height of the roof or values at different storeys. The interpretation of the height depends on the specific scenario of interest. Adjusted definitions lead to different meshes and therefore to different results for the numerical simulation which follows.

The three-dimensional solid/fluid discretisation can be generated for a computational domain and a mesh resolution specified by the user as presented in Sec. 3.2.6. Fig. 5.2 gives the voxelisation derived on the global scale and on the local scale of a building.

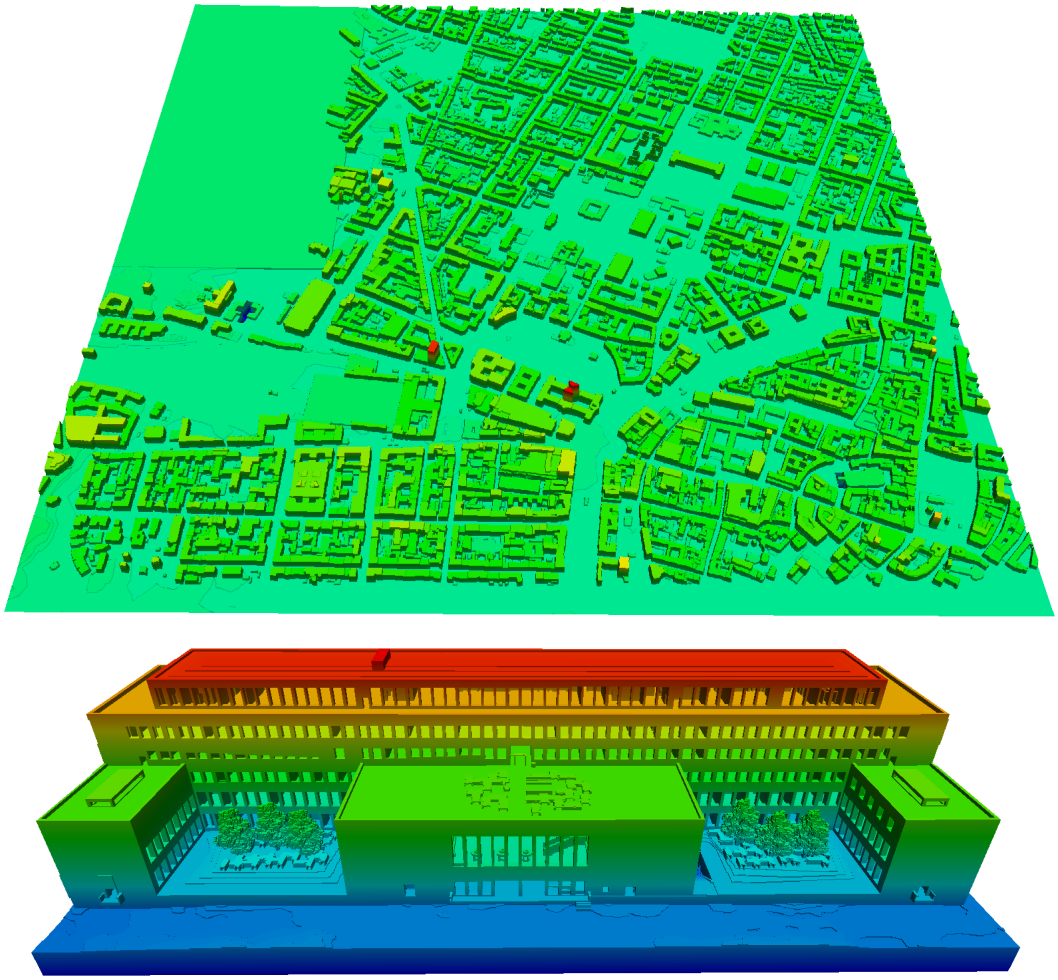

Figure 5.2: The three-dimensional voxelisation of the data basis can be derived on the global scale (top) and on the local scale for a single building (bottom). The height is encoded in the colour.

Based on this three-dimensional discretisation of the solid and the fluid parts of the domain, the two-dimensional definition of the bathymetry is derived for two different scenarios. The code for deriving *maximal* two-dimensional bathymetry mesh from the three-dimensional solid/fluid discretisation is given in Alg. 17.

The implementation given works as follows. The height of the mesh is defined as the highest solid value of the data set at that point of the plane. This would be the case for a Tsunami simulation of the fluid flow where the focus of the investigation is on evaluating the global behaviour of the flow, and the flow through the buildings and constructions is neglected. The formula for the two-dimensional bathymetry mesh $B = B_{ij}$ derived from a three-dimensional voxelisation $V = V_{ijk}$ is given as

Algorithm 17 derive2dMeshAtMaximalElevation

```
Matrix B = derive2dMeshAtMaximalElevation(Matrix V){

  for (int i=0; i<V.lengtX(); i++)
    for (int j=0; j<V.lengtY(); j++)
      for (int k=0; k<V.lengtZ(); k++)
        if (V(i,j,k) == solid)
          B(i,j) = k;
}
```

$$B_{ij} = \min_k \left\{ V_{ijk} | V_{ij\hat{k}} = fluid \quad \forall \hat{k} > k \right\}$$

In Fig. 5.3 the two-dimensional mesh derived for the local scale three-dimensional mesh depicted in Fig. 5.2 is given.

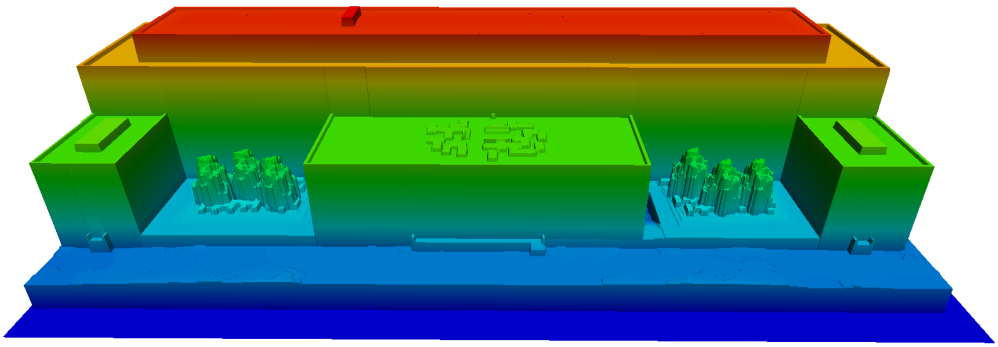

Figure 5.3: From the three-dimensional voxelisation of the computational domain the two-dimensional bathymetry mesh is derived on the local scale for a single construction. The height value at every data point is derived as the maximal solid value at that point in the plane.

In the second case, the height of the mesh is defined as the value below the first fluid voxel of the data set at that point of the plane. This would be the case for an urban heavy rain scenario where the focus of the investigation is on evaluating the flow through the buildings, especially open doors or entrances to underground car-parks. The impact of flooding will only be covered for the first opening of a construction in terms of height. The code for deriving two-dimensional bathymetry mesh from the three-dimensional solid/fluid discretisation is given in Alg. 18.

The formula for the two-dimensional bathymetry mesh $B = B_{ij}$ derived from a three-dimensional voxelisation $V = V_{ijk}$ is given as

$$B_{ij} = \max_k \left\{ V_{ijk} | V_{ij\hat{k}} = solid \quad \forall \hat{k} \leq k \right\}$$

Algorithm 18 derive2dMeshAtFirstFluid

```
function B = derive2dMeshAtFirstFluid(matrixV){

  for (int i=0; i<V.lengtX(); i++){
    for (int j=0; j<V.lengtY(); j++){

      bool found = false;
      for (int k=0; k<V.lengtZ() && !found; k++){
        if (V(i,j,k) == fluid){
          found = true;
          B(i,j) = k-1;
        }
      }
    }
  }
}
```

In Fig. 5.4 the two-dimensional mesh derived for the local scale three-dimensional mesh depicted in Fig 5.2 is given.

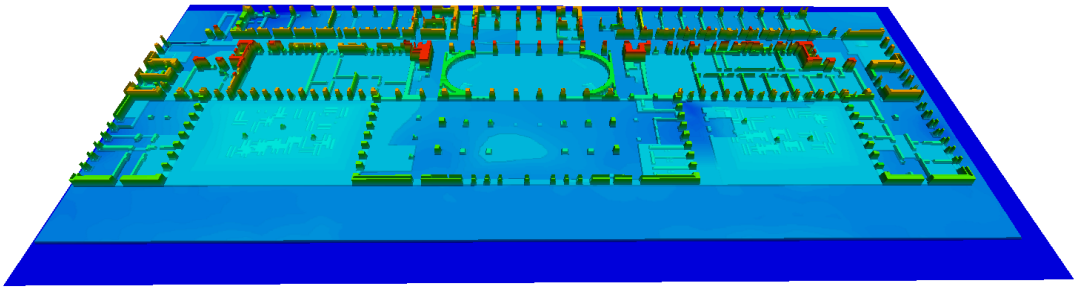

Figure 5.4: From the thee-dimensional voxelisation of the computational domain the two-dimensional bathymetry mesh is derived on the local scale for a single construction. The height value at every data point is derived as the value below the first fluid voxel at that point in the plane.

With the algorithm given a two-dimensional bathymetry mesh can be derived from the three-dimensional voxelisation of the computational domain. Two of the many ways for defining the height value of a point in the plane have been given. This definition has to be adapted to the respective simulation scenario.

In Fig. 5.2 and 5.3, 5.4, the bathymetry meshes derived are given over the full computational domain including the surface and the building description. This induces steep gradients on the intersection of the surface and walls or pillars, for example. These steep gradients have to be treated by the numerical scheme. There are different approaches such as removing the buildings from the computational domain and treating the interface as a boundary condition. These approaches in the scenario of urban flooding focus on investigating the behaviour of the water front in the streets and neglect the impact to the buildings. In Sec. 7.1 these

approaches are investigated, and this section presents the derivation of the two-dimensional mesh for coupling the work presented to SWE frameworks.

For a valid numerical simulation the definition of a computational mesh is only one part. Furthermore, boundary conditions, initial conditions, material parameters and simulation parameters are required. For boundary conditions not only the discretised definition has to be provided but also the model of the imposed behaviour. For the simulation of a flooding following the SWE, material parameters such as the roughness or the Strickler value can be derived from GIS sources as introduced in Sec. 2.3.1. The parts of the mesh which form the boundary have to be derived from the extend of the computational domain and the interface to buildings and constructions. The model which is imposed by the boundary conditions has to be chosen carefully depending on the specific simulation scenario. If the simplification is valid, that only the flow around buildings is investigated, for the boundary of the buildings reflecting behaviour could be assumed. Furthermore, the flooding of a building could be modelled with an outflow condition, which again has to be adapted to the specific scenario. The definition of initial conditions in general follows real-world data and the expertise of engineers. For a flooding induced by an exceeding river, values of measuring stations should be available. For a rain induced flooding, measurements from meteorological stations have to be investigated. The proposed framework can provide a basis but obviously many questions are left open.

5.2 Coupling the Peano Framework

The Peano framework is an open source framework for solving Partial Differential Equations on adaptive Cartesian grids. Based on the spacetree concept it provides a problem-solving environment which brings almost any geometry descriptions together with the interface for solving PDEs [77].

The classical octree concept can be extended to the generalised spacetree approach. Any discretisation using a spacetree can be expressed as a hierarchical order of Cartesian grids. By following this approach the iteration over adaptive Cartesian grids is performed as the element-wise traversal over the spacetree hierarchy following the Peano SFC. This structure provides node-wise and element-wise access to the derived grid and is therefore the basis for solving PDEs adaptively.

The Peano framework features dynamic adaptivity in space and time. The geometric multi-scale representation of the domain is directly achieved by the underlying hierarchical concept. Owing to the efficient implementation it comes with low memory requirements and cache-aware behaviour.

return	function	parameters
bool	geometry::isCompletelyOutsideNotInverted	(Vector x, Vector resolution)
bool	geometry::isOutsideClosedDomainNotInverted	(Vector x)
bool	geometry::isCompletelyInsideNotInverted	(Vector x, Vector resolution)

Table 5.1: The Peano geometry interface queries the status of the spacetree nodes during the build up and the traversal of the grid.

The parallelisation strategy of Peano yields a hybrid approach, enabling distributed memory parallelisation using MPI and shared memory parallelisation based on OpenMP and Intel TBB. Load balancing is performed using multiple implemented approaches as shown in [121].

As Peano does not only provide a matrix-free multi-grid solver for CFD applications but is a generally applicable framework for adaptive mesh refinement (AMR) based grid generation and traversal, coupling Peano to the work presented opens the door for various existing and future research projects. Besides the aforementioned adaptive solution of the three-dimensional Navier-Stokes equations as published in [122], research groups of the Peano team are working on the solution to engineering-relevant topics in the field of solid mechanics, Tsunami simulation or the Lattice-Boltzman method (LBM), to name but a few.

The access to the data basis provided by the present work is integrated into the Peano framework via a slim and efficient interface. During the build-up of the spacetree, Peano queries the underlying geometry for the status of the spacetree nodes in the computational domain. In the refinement process there is a check of every node as to whether it is completely inside or completely outside the computational domain. Nodes which are not completely inside nor completely outside the domain must belong to the boundary and are refined further. Following this approach, Peano provides a geometry interface which couples the complete framework via the implementation of the request if a given cube is completely inside or completely outside the domain. This request can be processed efficiently by the hierarchical data access concept presented in this work and will be introduced in the following.

5.2.1 Interfacing the Geometry Description

The coupling of the framework presented to Peano is performed by implementing the geometry interface of Peano. During the build up and the refinement of the Cartesian grid Peano has to decide whether a node of the spacetree is in the computational domain or not. If it is part of the computational domain it is created as a grid cell, otherwise not. This inside/outside check is the only link between Peano and the underlying shape of the computational domain. The interface of the methods for implementing the geometry interface of Peano are given in Table 5.1.

For the implementation of the interface the case for performing a CFD analysis based on

the data access to the framework presented is now introduced. In order to derive the fluid mesh, the computational domain is the whole domain excluding the parts covered by solid bodies, i.e. the construction, built infrastructure and the terrain data. The implementation is only given for the case of deriving a fluid mesh. Other applications, such as a structural analysis, would require parts of the complement of the fluid mesh, but can be easily derived by Boolean operations.

The implementation of the interface is performed by using the voxel query interface presented in Sec. 3.2.7. In order to access the voxel query interface, Peano is started in an additional MPI process together with the framework's master process and the $n-1$ slave process holding the product models. By doing so, Peano runs in a parallel environment together with the framework presented as given in Fig. 5.5. This concept was already used in Sec. 4.1.3 for the coupling of the visualisation as a separate process to the framework, and is a general concept of this work. By adding additional processes to the framework the data access can be queried by other applications.

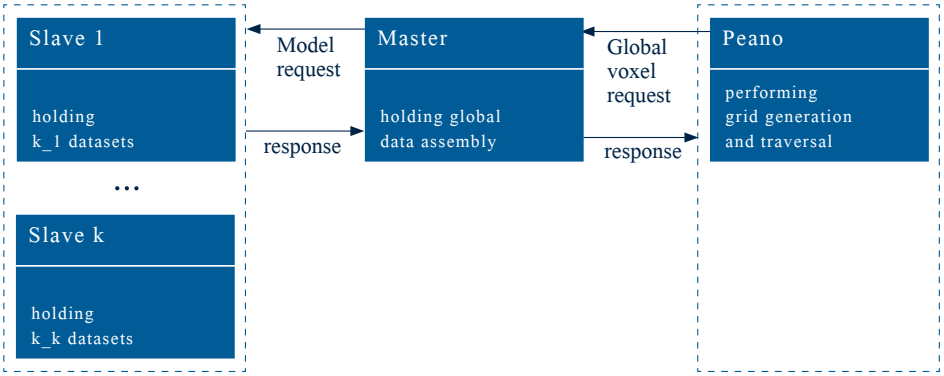

Figure 5.5: In order to access the voxel query interface, Peano is started in an additional MPI process, together with the framework's master process and the $n-1$ slave process holding the product models.

In Fig. 5.5, the framework is started with one MPI process for Peano. As Peano itself provides parallel access to the geometry interface, the number of Peano processes can be increased accordingly and multiple processes can perform read access to the voxel query interface. The code for the implementation of the Peano geometry interface is given in Alg. 19.

For the method $isCompletelyOutsideNotInverted$ the voxel spanned by $x-resolution/2$ and $x+resolution/2$ is checked to see if it is intersected only by solid parts of the computational domain by using the voxel query presented in Sec. 3.2.7. For the method $isOutsideClosed-DomainNotInverted$ the point x is checked to see if it is contained in the first layer octree and therefore in the computational domain. For the method $isCompletelyInsideNotInverted$ the voxel spanned by $x-resolution/2$ and $x+resolution/2$ is checked to see if it is intersected only by fluid parts of the computational domain. Fig. 5.6 gives the return values of the interface for a specific domain.

Algorithm 19 The Peano geometry interface queries the status of the spacetree's nodes during the build up and the traversal of the grid.

```
bool Construction::isCompletelyOutsideNotInverted
        ( Vector x, Vector resolution ) {

        return framework.voxelQuery(x, resolution/2, solid);
}

bool Construction::isOutsideClosedDomainNotInverted
        ( Vector x ) {

        return !framework.globalOctree.contains(x);
}

bool Construction::isCompletelyInsideNotInverted
        ( Vector x, Vector resolution ) {

        return framework.voxelQuery(x, resolution/2, fluid);
}
```

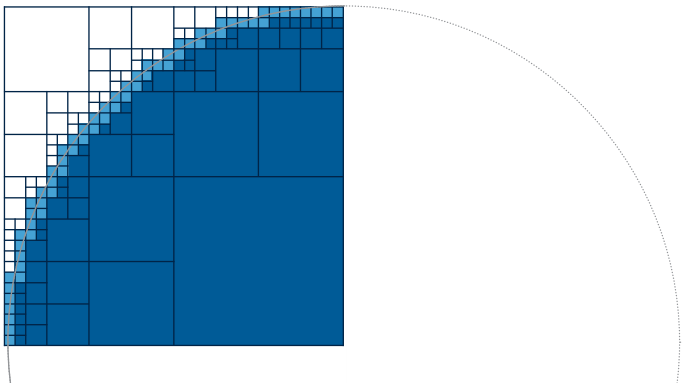

Figure 5.6: The Peano interface identifies the computational domain of a sphere as follows. The white octants give the outside of the domain where *isCompletelyOutsideNotInverted* returns *true*, the dark blue octants form the inner part and here *isCompletelyInsideNotInverted* returns *true*. The light blue depicted octants cover the boundary of the domain and there both methods *isCompletelyOutsideNotInverted* and *isCompletelyInsideNotInverted* return $false$. This triggers further refinement of the interface.

By implementing this slim and efficient interface, the Peano framework is coupled to the hierarchically ordered data of the work presented. Fig. 5.7 shows a sample for the fluid mesh on a local scale for the main building of TUM.

It should not be denied that this approach induces strong simplifications. The application of boundary conditions in general will be driven by the geometric and semantic information of buildings and their construction details. As the decomposition with this approach is driven by

the global structure of the domain, the product model data which specify a certain boundary condition may be spread over multiple processors. In order to cope with this, either the decomposition strategy has to ensure that one product models is mapped to a single process or additional communication among the processors has to be introduced which in general will be expensive. Furthermore, the generated mesh is based on axis-parallel elements and therefore non-boundary conforming. Fig. 5.6 highlights how serious the errors induced by this simplification are. The area of a sphere is approximated by an octree with first-order accuracy only. For the arc-length there is not even convergence. The approximated boundary delivers the same wrong estimation, independent of the resolution-depth of the tree. These simplifications do not only propagate to the advanced application of boundary conditions but also to the evaluation of the simulated quantities. The pressure distribution over the boundary layer of a cylinder, meshed with an octree, is not evaluable directly over the attaching faces and cells. This is still an active field of research in Fluid Structure Interaction (FSI). The adequate and efficient handling of non-boundary conforming meshes and efficient application of boundary conditions can be tackled by immersed boundary methods [123, 124], which has not been implemented in this dissertation project but is referred to explicitly.

5.3 Coupling OpenFOAM

The Open Source Field Operation and Manipulation simulation tool OpenFOAM is an open source software package developed in C++ with the main focus on Computational Fluid Dynamics (CFD). Besides CFD, OpenFOAM also provides various further application and investigation scenarios such as electrodynamics or combustion processes, for example.

The main advantage of coupling the work presented to OpenFOAM is the fact that it has a bright scientific community working on solvers for new applications, increasing the efficiency and accuracy of existing implementations, and validated and verified numerical solvers exist for a wide scope of applications, especially on fluid flow. Furthermore, OpenFOAM provides one common geometry and discretisation interface which is initially independent of the solver to be applied. By implementing this interface in the framework presented almost any scenario on the data basis can be expressed as input data for OpenFOAM. Starting from that discretisation the user can choose the application of interest and prepare numerical simulations on the data.

It should be pointed out that the experience of engineers is still needed in order to perform reliable numerical simulations and to interpret the correctness of the results achieved. The direct coupling of the framework presented to OpenFOAM does not provide a one click simulation pipeline for numerical analysis but it gives engineers a tool at hand to focus on their expertise and leverage their work.

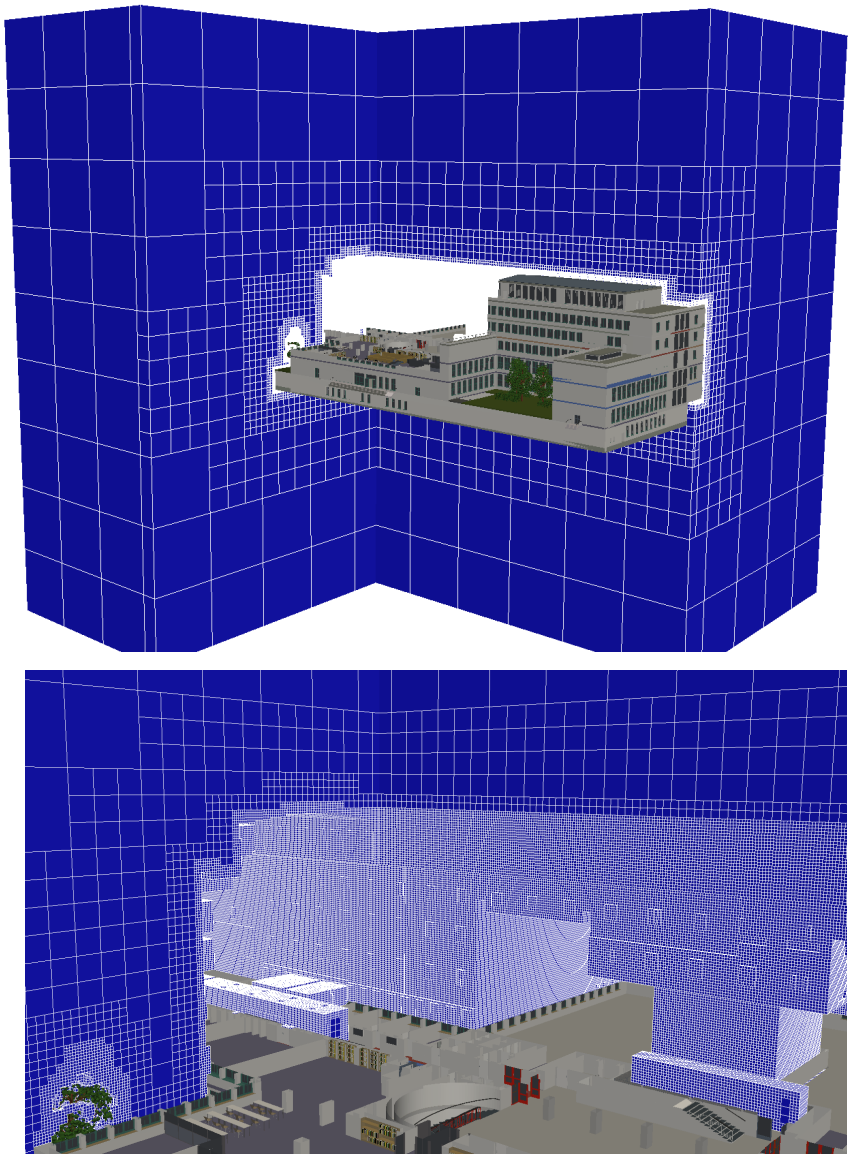

Figure 5.7: The fluid mesh is built up by the Peano framework and adaptively refined at the boundary of the domain. The inside/outside check for building up the grid is performed over the dual layer hierarchical data structure of the work presented.

This section is organised as follows. Sec. 5.3.1 introduces the preprocessing of the solid/fluid information of the computational domain which was performed. In Sec. 5.3.2 the derivation of the mesh in OpenFOAM format is presented. In Sec. 5.3.3 the possibility of performing mesh generation for OpenFOAM in parallel is introduced for achieving a decomposition of the computational domain.

5.3.1 Preprocessing the Solid/Fluid Discretisation

Based on the extent and the desired accuracy of the resolution specified by the user, a solid/fluid discretisation of the computational domain can be achieved by using the voxel representation interface introduced in Sec. 3.2.6.

Based on this discretisation the OpenFOAM mesh can be derived after preprocessing the solid/fluid field acquired. This field is not yet valid as a computational mesh for fluid flow as it does not cover the computational domain exactly. This can be seen from the fact that a highly resolved solid/fluid discretisation of a building will also identify the inner parts of rooms as fluid, even if all windows and doors are closed. Even if the area within the room is a fluid domain, it is not connected to the outside and therefore a completely separate domain. Multiple separate domains are not supported by OpenFOAM and therefore have to be removed in order to derive the computational domain of interest.

The derivation of the computational domain of interest will be performed by requesting the user to specify one point within the domain for which they wish to generate the mesh. Applying a flood fill algorithm will derive the part of the domain which is connected and contains this so-called seed point. Both parts are valid computational domains, but as they are not connected they do not influence each other and are to be distinguished. This procedure also saves computational costs as complex parts of the geometries at the boundary of the separate domains are not resolved and do not have to be handled by the simulation. There are approaches [125] which describe how to identify the inner and outer parts of an octree resolved structure without the computationally expensive flood fill algorithm. The flood fill algorithm is applied in this work in order to derive a minimally connected domain.

The code for applying the flood fill algorithm to a given solid/fluid matrix is given in Alg. 20 and works as follows.

The algorithm identifies the connected parts of the domain by recursively checking all - in three dimensions six - neighbours of a given point if they are fluid cells, starting with the seed point. If a point is fluid the algorithm is called for its eight neighbours. All fluid points are marked and as soon as the recursion ends the marked points define the connected computational domain containing the seed point. In the given implementation the recursion is broken up in order to avoid a call stack overflow. In the worst case such as a channel of diameter one, the recursion depth is the full size of the matrix dimension. The recursion is prevented by using a queue for the cells to be processed. The queue is initialised with the seed point. As long as the queue is not empty a point is removed from the queue and its neighbours are checked to see if they are part of the computational domain or not. All points passing this test are pushed to the queue. The algorithm ends as soon as the queue is empty.

Algorithm 20 fillMatrix

```
void fillMatrix(Point seed){

queue toProcess.push_back( seed_point);
while (toProcess.size() > 0){

  Point tmp = to_process.back();
  to_process.pop_back();
  if (x < hX-1) fillPoint(x+1, y  , z  );
  if (x > 0)    fillPoint(x-1, y  , z  );
  if (y < hY-1) fillPoint(x  , y+1, z  );
  if (y > 0)    fillPoint(x  , y-1, z  );
  if (z < hZ-1) fillPoint(x  , y  , z+1);
  if (z > 0)    fillPoint(x  , y  , z-1);
}
void fillPoint(Point p){
  if (!p.visited()){
    p.setVisited();
    to_process.push_back(point);
  }
}
```

In Fig. 5.8 the resulting matrix is visualised for a solid/fluid matrix on the local scale of a building. The seed point is specified on the outside and therefore the separate fluid domain on the inside of the room is removed.

This preprocessed solid/fluid matrix is now the basis for generating the OpenFOAM mesh. It describes the one connected computational domain containing the seed point. The interpretation of this solid/fluid matrix as a hex-mesh immediately gives the definition of the cells, faces, edges and points with their connectivity and the boundary required for a computational mesh. The derivation of the OpenFOAM mesh is given in the following section.

5.3.2 Mesh Generation

In this section, the computational mesh in OpenFOAM format is derived from the preprocessed solid/fluid discretisation presented in Sec. 5.3.1. This discretisation is specified as a three-dimensional Boolean matrix with extent $n_x \times n_y \times n_z$, describing the solid/fluid status of every voxel of side length h in the domain spanned by the vectors $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$ with $i_{\max} = i_{\min} + n_i \cdot h$.

This Boolean matrix with its extent and discretisation width can also be interpreted as a formulation of the fluid mesh with hexahedral elements. This formulation is now processed in the OpenFOAM mesh format. Fig. 5.9 shows a two element hexahedral mesh including the numbering of the cells, elements, faces and vertices.

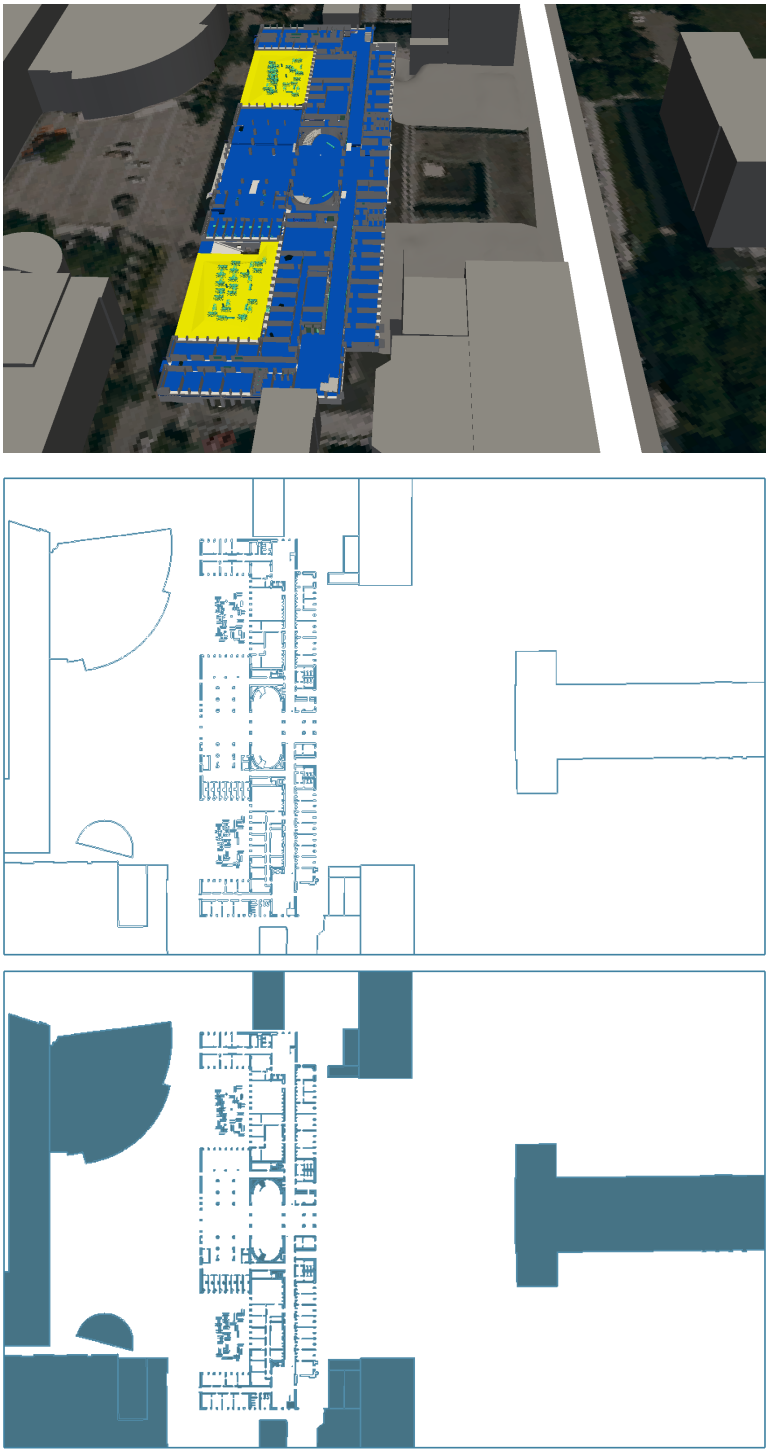

Figure 5.8: For the original domain (top) the voxelisation is derived. Fluid voxels are depicted white, solid voxels are blue; this discretisation is calculated in two ways. Without performing the fill algorithm (middle), fluid parts which are not connected to the seed region are also identified as computational domain. These parts are removed by performing the fill algorithm (bottom).

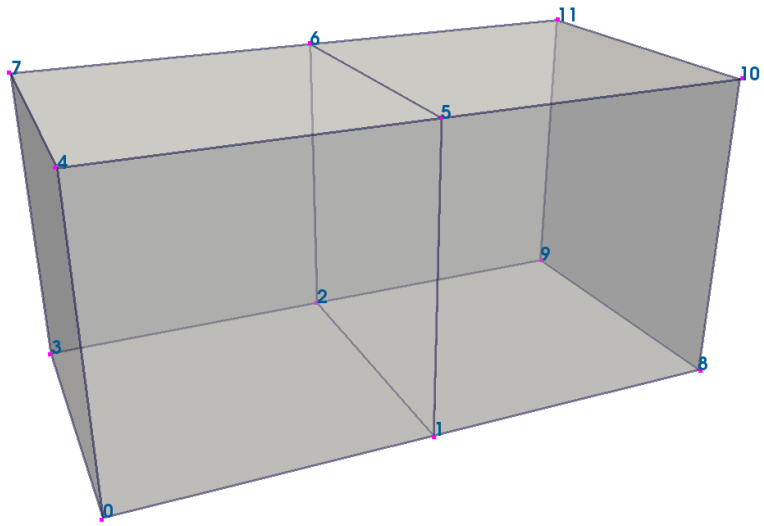

Figure 5.9: An OpenFOAM mesh of two hexahedral elements consists of 11 faces, 20 edges and 12 vertices.

The mesh given in Fig. 5.9 also covers the different types of faces. The face spanned by vertices $[1, 5, 6, 2]$ is an interior face, the remaining faces are boundary faces. The type of condition can be specified for every boundary face. The complete mesh specification of the two hexahedral mesh elements consists, as does every OpenFOAM mesh, of five different files which store the mesh and these are given for *points* in Sec. A.4.1, *faces* A.4.2, *boundary* A.4.3, *owner* A.4.4, *neighbour* A.4.5.

Before introducing the five files and the code for their generation in detail, the upper limits of their storage requirement are given. A single hexahedral element consists of 1 cell, 6 faces and 8 vertices. For the solid/fluid discretisation of extent $n_x \times n_y \times n_z$ the following upper limits can therefore be found.

- $n_x \cdot n_y \cdot n_z$ cells
- $6 \cdot n_x \cdot n_y \cdot n_z$ faces
- $(n_x + 1) \cdot (n_y + 1) \cdot (n_z + 1)$ vertices

These are not sharp upper bounds as the common faces are neglected. Nevertheless, in order not to expensively extend the memory allocation for the data during runtime these values are used in order to allocate the maximal main memory required.

Generation of the Mesh

The mesh is generated by iterating over the entries of the solid/fluid discretisation and inline filling the arrays of the cells, faces and vertices. The key for storing only the relevant parts of the mesh lies in building up a linearised index and ordering the elements. The linearised index defines an injective mapping from the three-dimensional coordinates of the matrix to a linear sequence. The ordering of the elements provides a surjective function over the indices. This bijective mapping finally is the basis for writing the mesh specification.

In order to generate the mesh as given in Alg. 21 the algorithm iterates over the solid/fluid matrix with dimension $hX \times hY \times hZ$ and processes fluid cells only.

Algorithm 21 iterateAllMeshCells

```
void iterateAllMeshCells(int hX, int hY, int hZ){
  for (int z=0; z<hZ; z++)
    for (int y=0; y<hY; y++)
      for (int x=0; x<hX; x++)
        if (isFluid(x, y, z))
          processCell(x, y, z);
}
```

In the *processCell* method every cell is now stored together with its vertices and faces as given in Alg. 22.

Algorithm 22 Pseudocode for a query combining spatial and IFC criteria

```
void processSingleCell(int x, int y, int z){

  processPointsOfCell(x, y, z);
  processCell(x, y, z);
  processFacesOfCell(x, y, z);
}
```

In the *processPointsOfCell* method the eight vertices are now stored with their linearised index $i = \hat{x} + (hX+1) * \hat{y} + (hX+1) * (hY+1) * \hat{z}$ with $\hat{i} = [i, i+1]$. By defining this index, the coordinates of the vertices can be stored in a linear array. Additionally, this linearised index is identical for the same vertex processed by different cells. This ensures that a unique vertex is only stored once. In the *processCell* method, the cells are stored to the array with their mapping to the ascending index. In the *processFacesOfCell* method the faces of the cell are now stored again using the linearised index. In this method the check for the different boundary types is also performed. Indices with a value of 0 or $i_{\max} - 1$ in one of the three dimensions are at the boundary and therefore stored as boundary faces. Indices which have neighbouring non-fluid voxels save the corresponding face in that direction as a boundary

too. After processing all elements of the matrices the arrays contain the definition of the complete computational mesh.

This approach for coupling the work presented to OpenFOAM produces non-boundary conforming meshes just as the coupling to Peano. Therefore, it is explicitly referred to the explained limitations and open questions formulated at the end of Sec. 5.2.

5.3.3 Parallel Mesh Generation and Domain Decomposition

In Sec. 5.3.1 the derivation of the mesh in OpenFOAM format has been introduced using serial processing. This chapter describes the parallelisation of the mesh generation together with the domain decomposition. OpenFOAM provides parallel numerical simulation following a domain decomposition approach in which the computational domain is split up into n non-overlapping boundary matching sub-domains. The parallel simulation is performed on each sub-domain in a distributed MPI process. The synchronisation of the simulation is performed by specifying a special face type of neighbouring elements of different sub-domains.

In Sec. 5.3.2 the *boundary* and *inner* face types have been introduced and this is now extended by the *processor* face type. Processor faces are defined pair-wise, both describing the same face on the boundary between two elements on different sub-domains and therefore on different processors. By doing so, OpenFOAM can identify the synchronisation pattern and contains the corresponding ghost layers. The identification of matching *processor* faces is performed by encoding the IDs of the participating processors in the name. The naming convention follows the pattern *proc_i_to_proc_j* in order to identify the communication between the processes with the ID i and j. A sample for this process mapping for a computational domain of two processors is given in Fig. 5.10 together with the corresponding definition in the face list.

This slim interface for identifying matching processor faces can now be used in order to parallelise the mesh generation. By doing so, the build up of the complete OpenFOAM mesh can be prevented. OpenFOAM meshes can easily grow in ASCII storage to file sizes of hundreds of gigabytes for complex scenarios such as the urban flooding presented in Sec. 7.

Parallel mesh generation can be performed for the preprocessed mesh derived in Sec. 5.3.1 as follows. The matrix is distributed into n non-overlapping sub-matrices, each describing a connected section of the system matrix. These sub-matrices are indexed with a unique element of the sequence $[0, 1, .., n-1]$. Mesh generation is now performed on each of the sub-matrices independently according to the algorithm introduced in Sec. 5.3.2. The resulting n meshes are now still completely independent of each other. The decomposition of the domain is performed on the shape of the global system matrix. Therefore, individual constructions are intersected by the boundary of the sub-domains and have to be processed on multiple

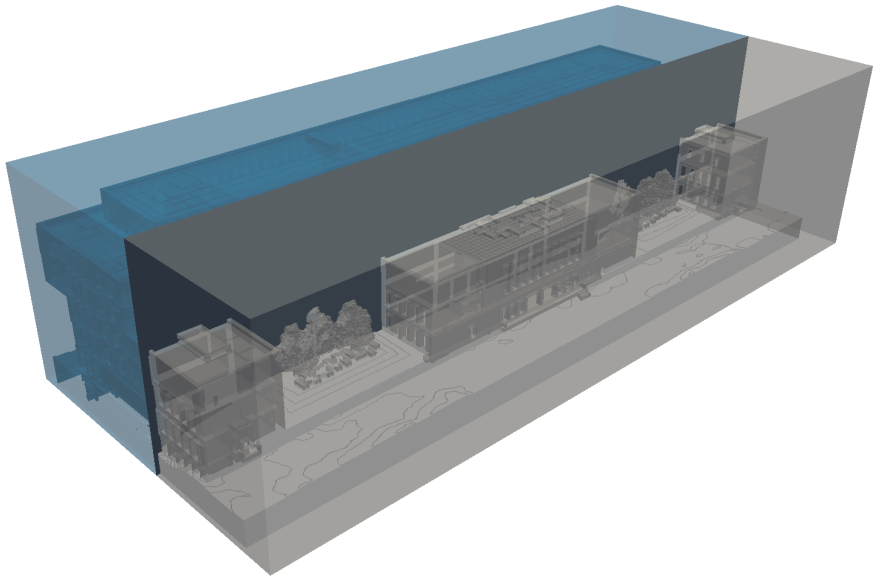

Figure 5.10: The two sub-domains of a computational domain on the local building scale are coloured according to their processor location. The *processor* faces are highlighted and interface the communication between the processors.

processors. This induces additional computational load, in return a uniform pattern of the domain decomposition is achieved. From the sequence to which the single sub-matrices are assigned, the neighbouring sub-domains and their index in the sequence are derived. Finally, the boundary faces of the independent sub-domain meshes are updated. Boundary elements which are also on the boundary of the computational domain are left as *boundary* faces, boundary elements of the faces which are on the interface of two processors are updated to *processor* faces with the identification of the two processors following the naming convention. Fig. 5.11 shows a block-wise decomposition of $1000 \times 1000 \times 50$ sub-domains for 4 processors.

Decomposed meshes are used often due to the simple mapping of the grid points to the processor block. Nevertheless, a block decomposition does not necessarily give an optimal distribution. In an optimal decomposition not only the computational work originating from the number of grid points is balanced, but the communication of the ghost layers of the processor faces is also minimal.

OpenFOAM provides a set of domain decomposition approaches for initial meshes spanning the whole computational domain. One of these approaches is the *Scotch* [126, 127] decomposition, a graph-partitioning algorithm producing fill-reducing orderings and usually better meshes than alternative domain decomposition strategies, see [128]. Fig. 5.12 shows the decomposition of a mesh of resolution $1000 \times 1000 \times 50$ distributed to 8 processors following the *Scotch* strategy.

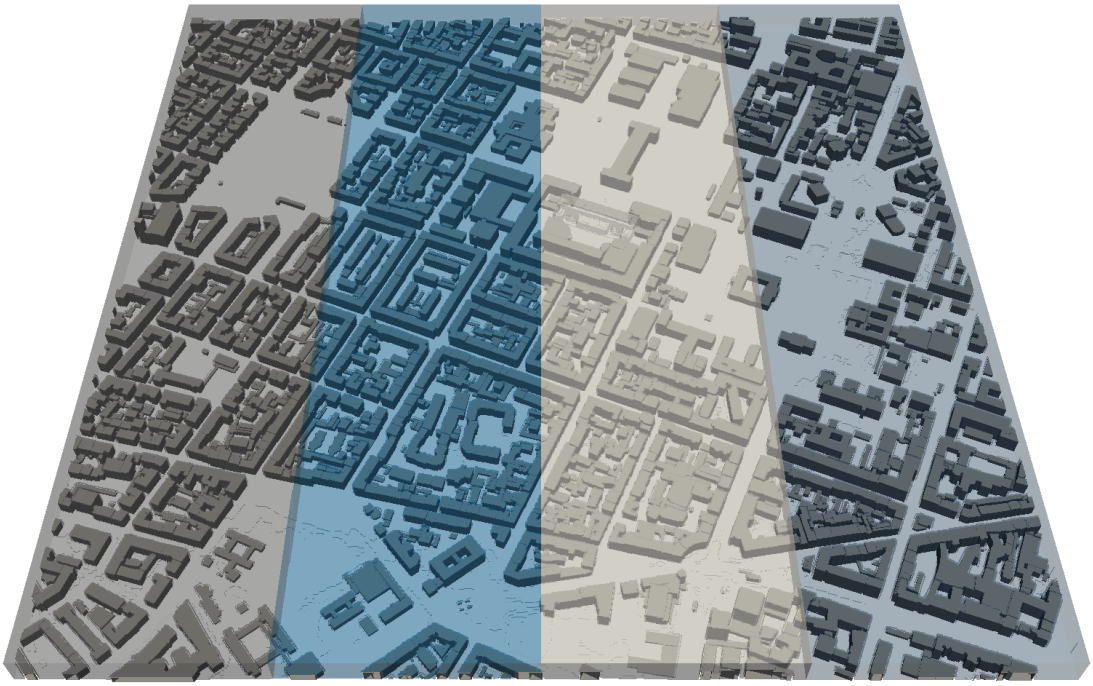

Figure 5.11: The decomposition of a $1000 \times 1000 \times 50$ global-scale domain is given for 4 processors following a block approach. The mapping of the different sub-domains to the processors is encoded in the colour.

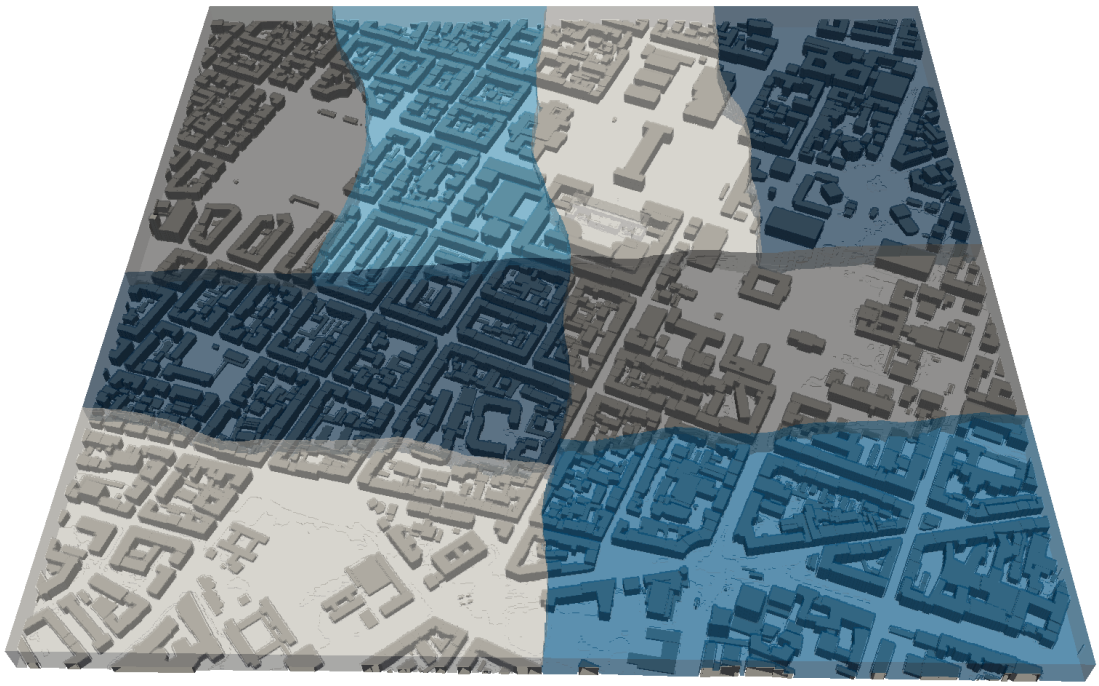

Figure 5.12: The decomposition of a $1000 \times 1000 \times 50$ global-scale domain is given for 8 processors following the *Scotch* approach. The mapping of the different sub-domains to the processors is encoded in the colour.

Chapter 6

Multi-Resolution Parallel Numerical Simulation

This chapter introduces the parallel numerical simulation of physical phenomena on the multi-resolution data basis provided by the framework presented. It will be shown that the provided interfaces of this framework give users a tool at hand which lets them specify the extent and the resolution of the discretisation of the computational domain and then perform the parallel numerical simulation of physical phenomena. The setting up of physically reliable simulation scenarios, investigating the results obtained and adapting model parameters needs to be performed by engineers, but the basis for this can be provided by the work presented. With the application scenarios presented the specialists can focus on their expertise and their work is supported. This chapter is organised as follows.

Sec. 6.1 introduces the application of different CFD solvers. Based on the specification of the user, the computational mesh is derived in line with the approaches presented in Sec. 5.3. With this interface the user can perform parallel numerical simulations at multiple resolutions, freely adjustable by the user.

In Sec. 6.2 two approaches will be introduced which enable the multi-resolution extension of the simulation approach. First of all, the projection of coarse simulation results into the initial conditions of more finely resolved resolution will be presented. This allows a hierarchically ordered investigation of the domain of interest to be performed with increasing accuracy.

The coupling of different physical phenomena will then be introduced. As a use case, the propagation of large-scale dimensionally reduced simulation results to the initial condition of fully dimensionally resolved simulation schemes will be presented.

Sec. 6.3 introduces the efficient investigation of the numerical simulation results achieved beyond the visualisation of the quantities. As the framework presented provides the complete

link between all scales of the data, the impact of the simulated quantities on the underlying product model specification can be explored in full detail. This makes it feasible to use highly resolved urban flooding scenarios to investigate which rooms of buildings and which technical installations are affected. Finally, in Sec. 6.4, the performance of the applied solvers on a cluster computing installation is investigated.

6.1 Fluid Flow Simulation

In this chapter, the application of four different PDE solvers from the field of Computational Fluid Dynamics is introduced. All simulations rely on the different grid-generation and simulation frameworks presented in Sec. 5. The domain and the discretisation accuracy are defined by the user and therefore the computational mesh is generated accordingly.

6.1.1 Three-Dimensional Potential Flow

The simulation of three-dimensional inviscid, frictionless and irrotational flow

$$\nabla \cdot \vec{v} = 0$$

$$\Delta p = 0$$

is performed by utilising the OpenFOAM solver *potentialFoam* [129, 130]. Potential flow describes the simplified external flow around bodies, viscous effects on the boundary layer are neglected. Fig. 6.1 gives the potential flow through a building on a domain of discretisation of $450 \times 200 \times 275$ voxels. The boundary conditions applied describe an inflow with a constant velocity of 1m/s on the left side and a free outflow on the right side.

6.1.2 Two-Dimensional Shallow Water Equation

The two-dimensional Shallow-Water-Equations [131, 132] (SWE)

$$\begin{aligned}
\frac{\partial h}{\partial t} + \frac{\partial (uh)}{\partial x} + \frac{\partial (vh)}{\partial y} &= 0 \\
\frac{\partial (uh)}{\partial t} + \frac{\partial \left(u^2 h + \frac{1}{2} g h^2\right)}{\partial x} + \frac{\partial (uvh)}{\partial y} &= 0 \\
\frac{\partial (vh)}{\partial t} + \frac{\partial (uvh)}{\partial x} + \frac{\partial \left(v^2 h + \frac{1}{2} g h^2\right)}{\partial y} &= 0
\end{aligned}$$

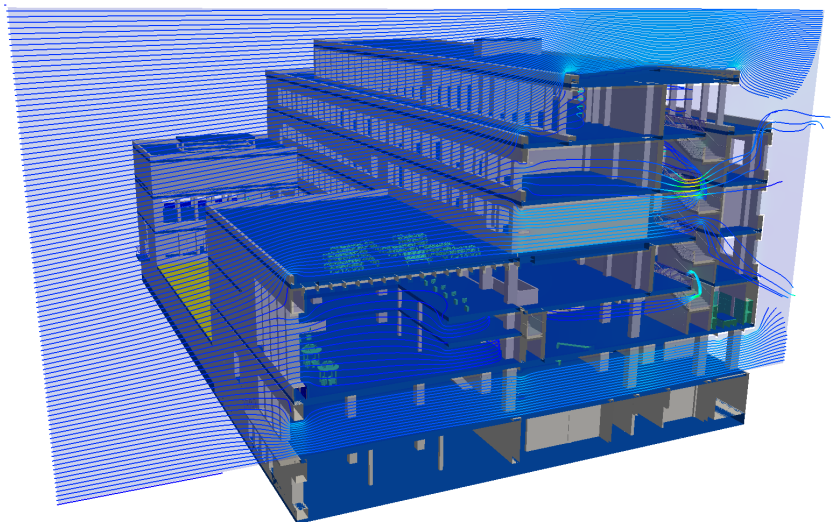

Figure 6.1: The potential flow through a building's product model is discretised over $450 \times 200 \times 275$ voxels. The boundary conditions applied describe an inflow of constant velocity on the left and a free outflow on the right side of the domain.

are solved by utilising the OpenFOAM solver *shallowFoam* developed by Kreuzinger + Manhart Turbulenz GmbH and the Chair of Hydromechanics at Technische Universität München. This solver uses a Finite Volume (FV) discretisation and Euler time stepping, incorporates wetting and drying of the computational domain and supports extended force descriptions. Fig. 6.2 shows the flow through the river bed of the Dornbirner Ache [38] with an extent of 7.1km by 8.8km. The domain is discretised with an adaptive triangular mesh of over 100k triangles. The boundary conditions applied describe a constant inflow on the left-hand side of the river bed and a free outflow on the right-hand side.

6.1.3 Three-dimensional Incompressible Navier-Stokes Equation

The simulation of three-dimensional single-phase flow is performed by utilising the Peano CFD solver [122] developed at the Chair of Scientific Computing in Computer Science at Technische Universität München. The Peano CFD package solves the incompressible Navier-Stokes equations [131] consisting of the continuity equation $\nabla \cdot \vec{v} = 0$ and the momentum equation for the i-th component:

$$\frac{\partial v_i}{\partial t} + \nabla \cdot (v_i \vec{v}) = \nabla \cdot (\nu \nabla v_i) - \frac{1}{\rho} \nabla \cdot (p \vec{e_i}) + b_i$$

where $\nu = \mu / \rho$ is called the kinematic viscosity. In Fig. 6.3 the fluid flow around a building is visualised using stream lines with colour-encoded velocity. This simulation neglects turbulence and pretends unrealistically high viscosity. Therefore, this result has to bee seen as a feasibility study of the coupling instead of a physically reliable result.

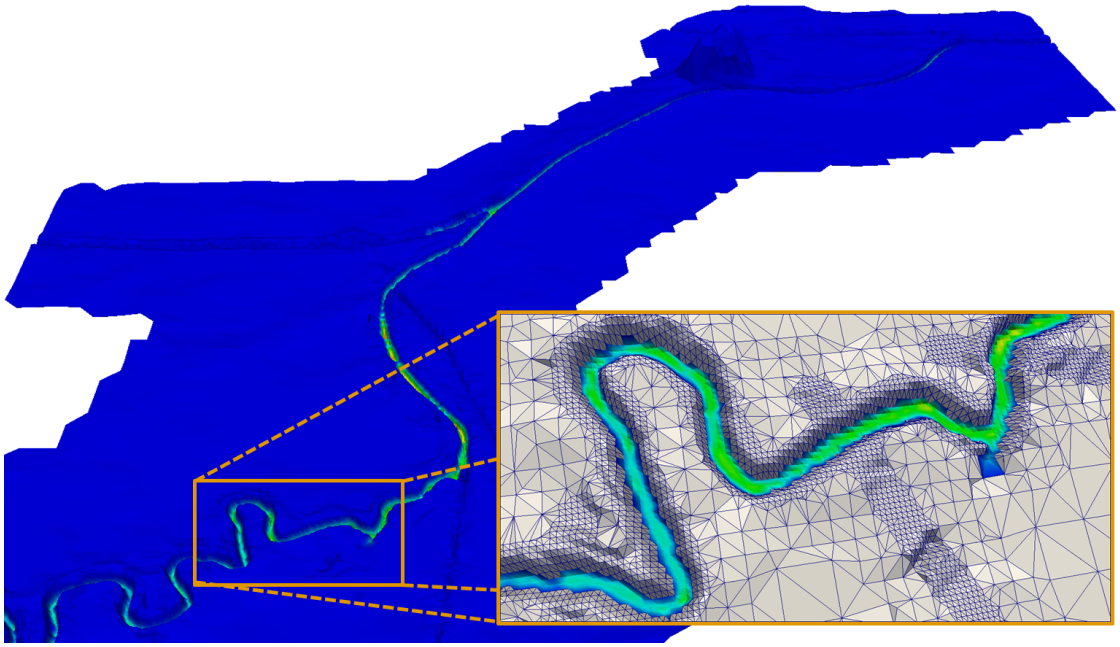

Figure 6.2: The flow of the Dornbirner Ache in Austria is simulated using the Shallow-Water-Equations. The boundary conditions applied imply a constant inflow of the river on the left-hand side and a free outflow on the right-hand side. The extruded surface of the fluid is visualised by colour encoding the velocity of the flow.

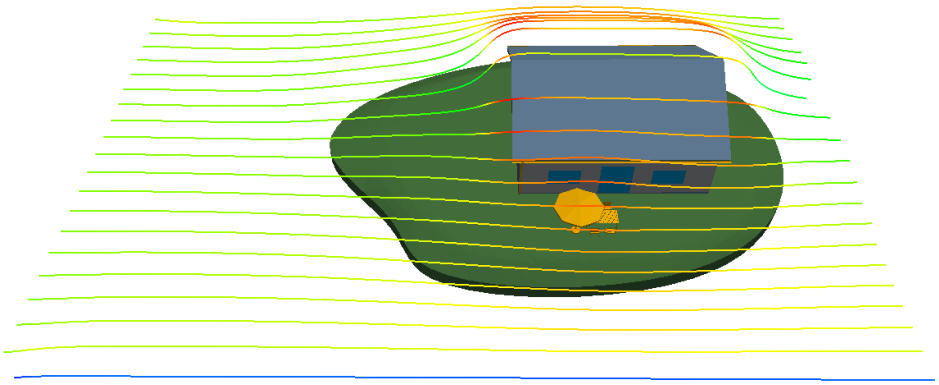

Figure 6.3: The three-dimensional Navier-Stokes equations describe the flow of Newtonian fluids. The flow around a building is simulated for a computational domain of $150 \times 100 \times 100$ voxels. The boundary conditions applied imply a constant velocity of 1m/s on the left side and free outflow on the right.

6.1.4 Three-dimensional Free Surface Simulation

The simulation of three-dimensional free surface flow is performed by utilising the OpenFOAM solver *interFoam*. The nomenclature follows [133]. For a given domain $\Omega \subset \mathbb{R}^3$ with a given boundary Γ, a uniform Cartesian grid discretisation $(\Omega_i)_{i \in I}$ is given. In order to per-

form a flooding simulation, the three-dimensional incompressible Navier-Stokes-Equations consisting of the continuity equation $\nabla \cdot \vec{v} = 0$ and the momentum equation for the i-th component:

$$\frac{\partial v_i}{\partial t} + \nabla \cdot (v_i \vec{v}) = \nabla \cdot (\nu \nabla v_i) - \frac{1}{\rho} \nabla \cdot (p \vec{e_i}) + b_i$$

where $\nu = \mu/\rho$ is called the kinematic viscosity are applied. The equations are solved on the grid $(\Omega_i)_{i \in I}$, following a Finite Volume (FV) approach. The interface between the gas-phase region $\mathcal{R}_g$ and the liquid-phase region $\mathcal{R}_l$ is tracked by a Volume-of-Fluid (VoF) approach. Therefore, an indicator function

$$\mathbb{1}(x,t) = \begin{cases} 1 & x \in \mathcal{R}_l \text{ at time } t \\ 0 & x \in \mathcal{R}_g \text{ at time } t \end{cases}$$

is defined in order to identify whether a point x at time t is in the liquid-phase or in the gas-phase region. The integral over all cells (Ω_i)

$$\gamma_i = \gamma(x_i, t) = \frac{1}{|\Omega_i|} \int_{\Omega_i} \mathbb{1}(x,t) dV$$

gives the liquid fraction field.

Besides the VoF method, there are a variety of approaches for tracking the boundary of multi-phase flow. The Level Set Method [134] (LSM) uses a level set function which takes negative values in the one phase and positive values in the other phase. The zero level of the level set function thus identifies the boundary of the two phases. The LSM has additionally the advantage that geometric shapes can be used for identification. For a detailed introduction to free surface approaches, refer to [135, 136, 137].

In Fig 6.4, the free surface simulation is performed on the city-wide scale for a domain with a resolution of $1000 \times 1000 \times 50$ cells. The boundary conditions applied describe free outflow on the four x/y-normal boundaries. The initial conditions describe a partially flooded domain with fluid at rest, representing the amount of water flooding the city.

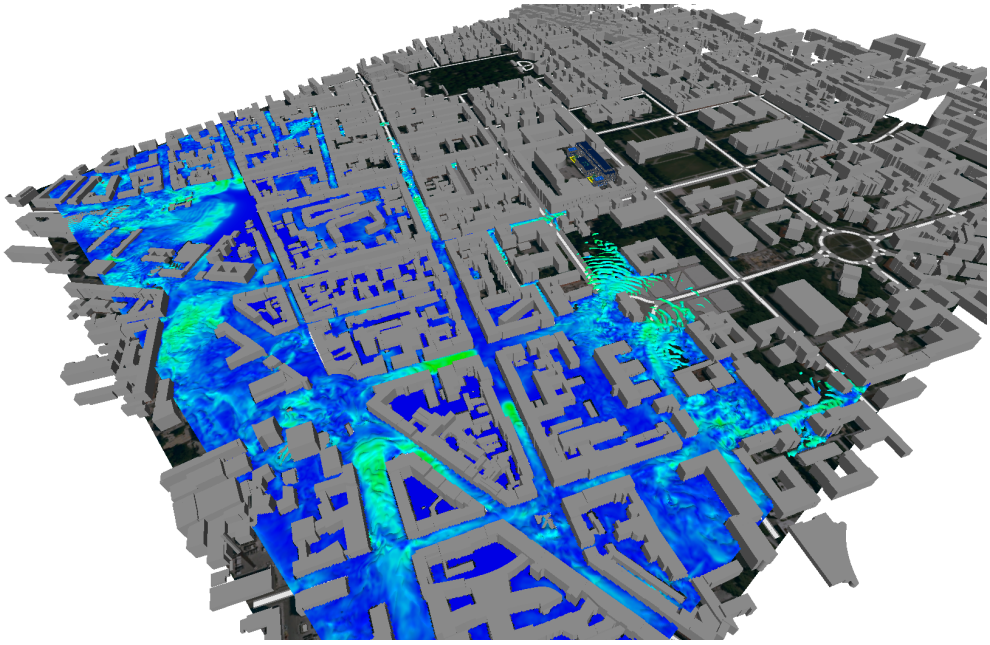

Figure 6.4: The simulation of urban flooding is performed as a three-dimensional free surface simulation. The surface of the fluid is visualised with colour encoded velocities. The computational domain is discretised with a grid of $1000 \times 1000 \times 50$ cells. The boundary conditions applied describe a free outflow on the x/y normal boundaries, the amount of water is imposed by initial conditions for the fluid at rest.

6.2 Resolution Refinement

In this section, the adaptivity of the parallel numerical simulation is introduced. The approach presented is adaptive in the sense that, in a one-directional way, the result of a simulation is used as an initial condition for the simulation of a refined subset.

This approach applies reflecting, potential or transient boundary conditions to the finer resolved computational domain. Obviously, this is a scenario-specific modelling question and limits the achieved results to small time intervals. The application of transient boundary conditions following a derived flow profile is known as passive adaptivity approach [137, 131, 138] and can be extended to even more computational expensive active adaptivity approaches, where the coupling at the boundary of the two domains is solved explicitly.

Furthermore, the term adaptivity here has to be seen different from numerical simulations in which the mesh resolution is locally refined due to geometry or fluid-driven criteria. This approach has been introduced by the coupling to the Peano framework in Sec. 5.2 or can be derived as at least a geometry-adaptive mesh from Sec. 3.2.5. Therefore, the term 'adaption' instead of 'adaptivity' will be used in the following. This approach focuses on a sequence of simulations where each simulation is defined on a subset of the preceding computational

domain. The subsequent computational domain has a smaller extent and finer resolution. The result of the preceding simulation is projected as the initial condition onto the mesh of the simulation.

A good example is the flooding of an urban region. Starting with the whole extent of a city, performing three-dimensional free surface simulation of a heavy rain scenario is computationally very expensive and not suitable for a construction-level resolution.

Therefore, the initial simulation can be performed using the dimensionally reduced SWE. As the SWE are a two-dimensional formulation their application is also feasible at very high resolutions and gives an approximation of the flow behaviour on the scale of the whole city. As introduced in Sec. 5.1.1, SWE do not cover the three-dimensional case as is important for buildings and constructions. Objects such as multiple storeys cannot be covered by a two-dimensional mesh such as is needed for simulating the SWE. The engineer decides on the basis of the results of the two-dimensional SWE simulation which buildings or parts of the simulation he is interested in. These parts can then be initialised with a fully three-dimensional free surface simulation in order to investigate the impact of the flooding on the interior of the building. Based on an initial simulation of the whole building and the results achieved thereby, the resolution adapted approach can then allow the user to decide to refine a particular part of the building even further. Such a simulation can finally be used to investigate the impact of the flooding on individual construction details such as electronic equipment.

It should not be denied that this adaption approach also induces errors, of course. A more finely resolved domain generally includes parts of the computational domain which have not been resolved on a coarser level due to the imperfect mesh. Only estimated initial conditions are available for these values. Furthermore, the global effect may get lost with the adaption approach. If, for example, the refinement covers a computational domain where the water immediately approaching a building is included, but a water front which approaches the building in the next time steps is excluded, this effect will be lost completely by applying potential or reflecting boundary conditions.

6.2.1 Resolution Adaption

The formulation in this section follows the Finite Volume (FV) discretised multi-phase simulation with the nomenclature introduced in Sec. 6.1.4. Resolution adaption increases the accuracy of a numerical simulation by performing a sequence of successively refined simulations where the results on the coarse scales are projected as initial condition to the fine scales.

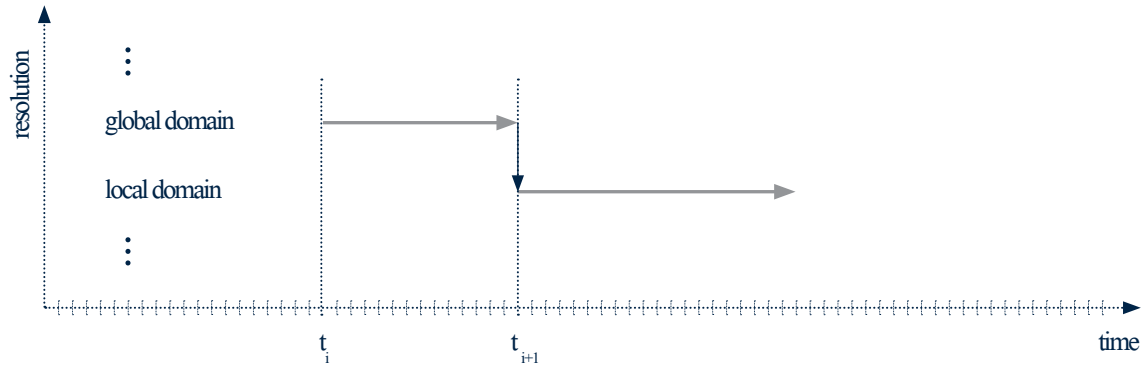

Figure 6.5: Starting at time step t_i, coarse grid computations are performed until t_{i+1}, followed by a projection from the global to the local domain (vertical arrow). The coarse grid simulation is stopped at t_{i+1}.

Fig. 6.5 shows the adaption approach for one step of a refinement. The simulation on the global domain that is performed on the computational domain (Ω_i) and starts at t_i, is stopped at a predefined time t_{i+1}. The simulation values that were achieved on the global domain at time step t_{i+1} are now used as initial condition for the refined simulation on the local domain. The computational domain of the local domain is a subset $(K_i)_{i\in J} \subset (\Omega_i)_{i\in I}$ with $J \subset I$ of the global domain and every cell (K_i) is refined with cells (S_{ij}) with $\bigcup_j (S_{ij}) = (K_i)$ and $(S_{ij}) \cap (S_{ik}) = \emptyset$ for $j \neq k$.

Given the solution $u_i = (\vec{v}_i, p_i, \gamma_i)$ on the global domain $(\Omega_i)_{i\in I}$ at time t_{i+1}, the refined simulation on the local domain (S_{ij}) is initialised with values $\tilde{u}_{ij} = u_i$.

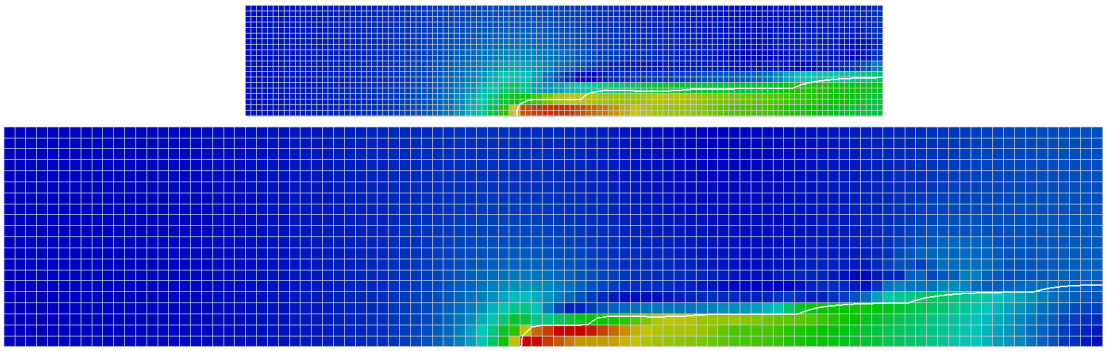

Figure 6.6: A two-phase simulation is performed on the global domain (bottom) and on the refined subset of the local domain (top). The velocity is colour encoded and the interface is visualised as the contour of the phase field γ. The mesh generation based on the representation of the data basis with octrees of different depth achieves conforming meshes over the different scales. This conformity enables the direct propagation of simulation results to the matching cells of a refined simulation.

Fig. 6.6 shows two conforming computational domains with an adapted resolution that doubles in every dimension. The values of the coarser simulation are projected onto the – in this

case four – corresponding voxels of the finer resolution. Cells of the coarse mesh therefore do not intersect but completely contain cells of the finer mesh and vice versa, cells of the finer mesh are mapped to exactly one cell of the coarser mesh.

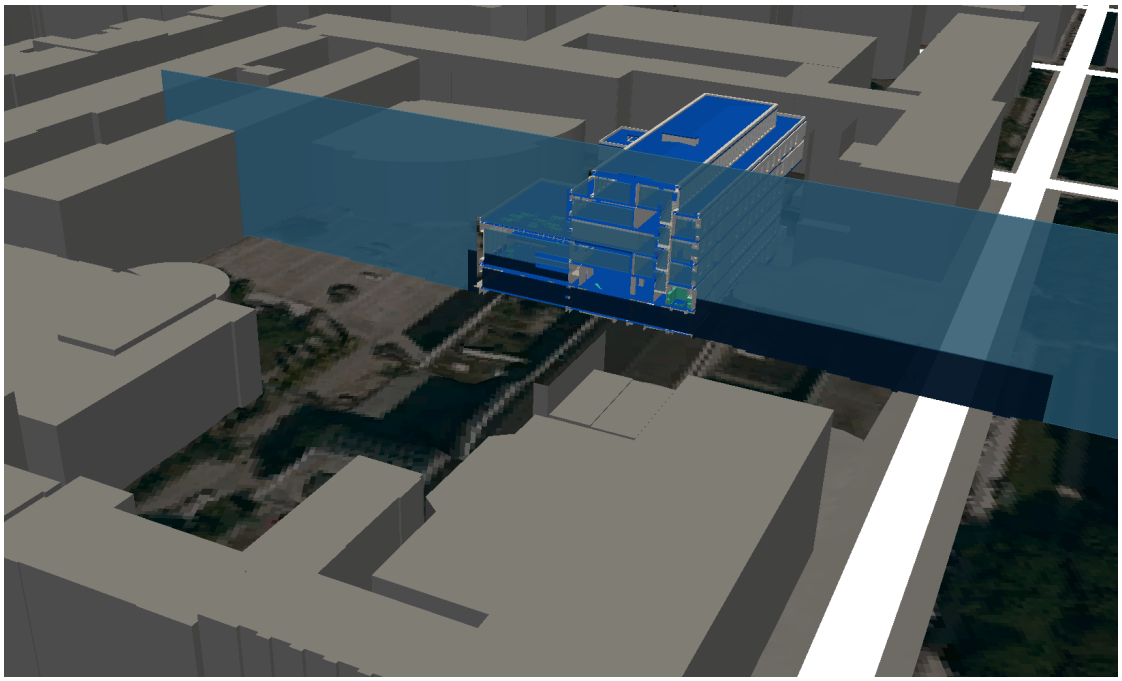

Figure 6.7: The two dimensional validation area is a slice through the computational domain. The global domain is coloured in light blue and the local domain is coloured in dark blue.

Special focus has to be put on the boundary conditions applied. In this case, reflecting, potential or transient boundary conditions are implemented at the inflow and reflecting boundary conditions at the outflow. Anyhow, the refined computational domain (K_i) has to be chosen carefully, i. e. it has to be chosen "slightly" larger than the local domain of interest, thus the flow profile can evolve without showing artefacts due to the boundary conditions. It is inevitable, that the applicability of this approach has to be investigated for every simulation case, as zooming to a subset of the computational domain and applying boundary conditions changes the characteristics of the flow regime and the physics simulated.

If the fluid flow on the local domain is mainly driven by the flow over the boundary, this approach will give reasonable results only for very small time ranges (if it does at all). Even if the flow on the local domain of interest is covered well on the extended local domain, still the influence of the boundary conditions will propagate through the computational domain, and limits a reliable analysis to small time steps.

A test case has been set up which brings an insight to the error introduced by the boundary conditions as well as to the time range that the refined simulation can be expected to deliver acceptable results. Fig. 6.7 shows the two dimensional validation domain. The global domain

covers a slice through the investigated building and the surrounding area, the local domain covers the entrance and main lobby area of that building.

The global domain $[-300; -30] \times [100; 30]$ has a resolution of 800×120 voxels, the local domain $[-200; -30] \times [-50; -5]$ has a resolution of 300×50 voxels. In order to estimate the influence of the boundary conditions only, both domains are resolved with the same mesh width $h = 0.5$. The initial conditions on the global domain impose a flooding with a fixed amount of water. The time step for projecting the simulation values to the local domain is chosen at $t = 10$ seconds and given in Fig. 6.8.

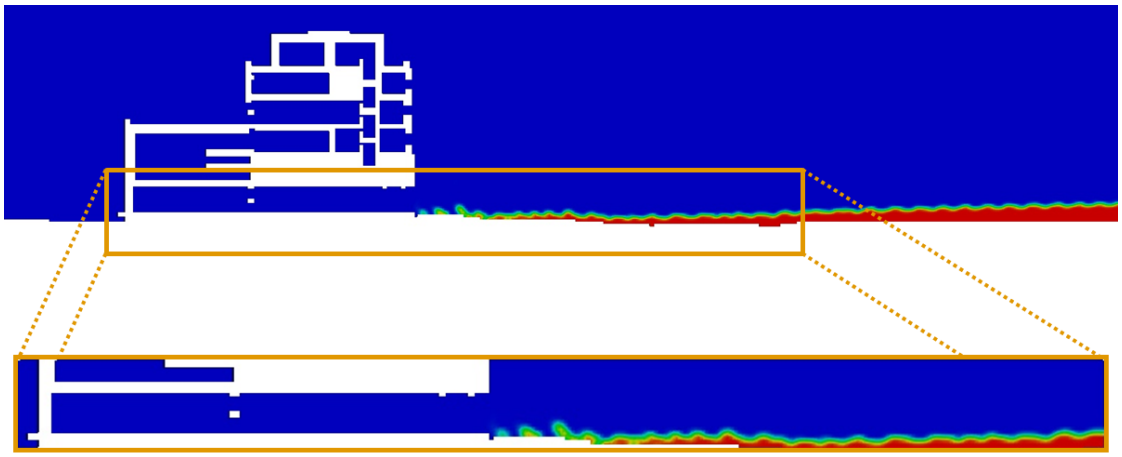

Figure 6.8: At time step $t = 10$ seconds, the simulation results achieved at the global domain are projected as initial conditions to the local domain.

Fig. 6.9 shows the two simulations on the global and the local domain, both at time step $t = 12.5$ seconds and with reflecting, potential and transient boundary conditions applied on the local domain. Both simulations run in parallel. For reflecting and potential boundary conditions, both simulations are not coupled any more after projecting the global domain simulation results as initial conditions to the local domain once at time step $t = 10$ seconds. For potential boundary conditions, the values for the pressure and the velocity are fixed at time step $t = 10$ seconds. For transient boundary conditions, the simulation on the global domain is performed for the time interval $t = [10..20]$ seconds. The achieved flow profile is evaluated over the boundary of the local domain and the values for the pressure and the velocity are interpolated. A visual comparison shows promising results for the evolution of the water front, but it also shows already clearly the influence of the reflecting and potential boundary conditions at the rear part of the wave.

Fig. 6.10 gives the results for the two simulations at time step $t = 15$ seconds, where still promising results for capturing the characteristics of the wave front could be achieved. The effect of the reflecting boundary conditions are at this time step already obvious by a visual inspection at the rear part of the wave.

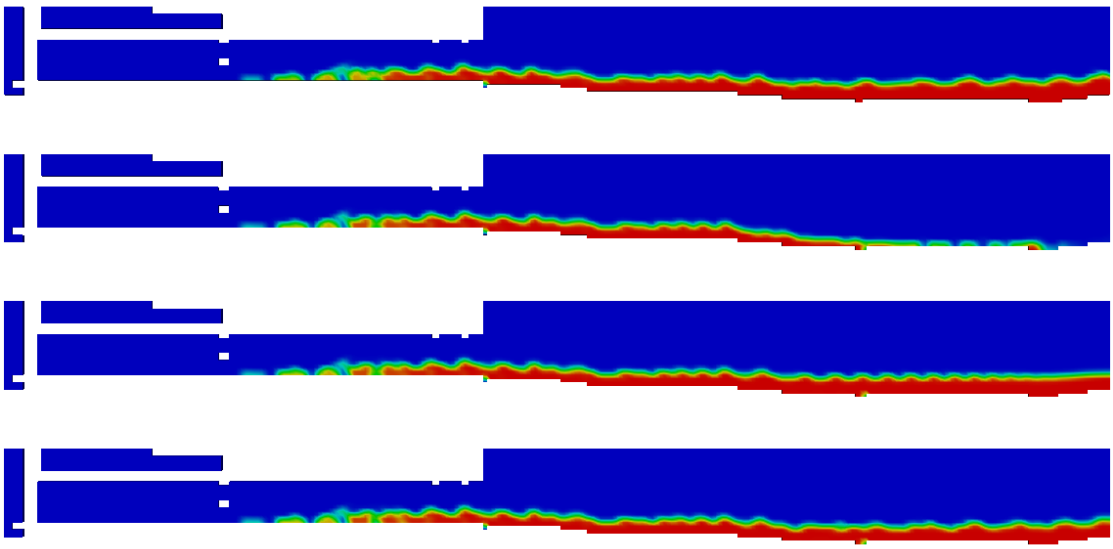

Figure 6.9: The simulation on the global domain (top) is compared to the results on the local domain incorporating reflecting, potential and transient (f.t.t.b) boundary conditions. For the given results here at time step $t = 12.5$ seconds, a visual comparison shows promising results for the evolution of the water front but shows clearly the influence of the reflecting and potential boundary conditions at the rear part of the wave.

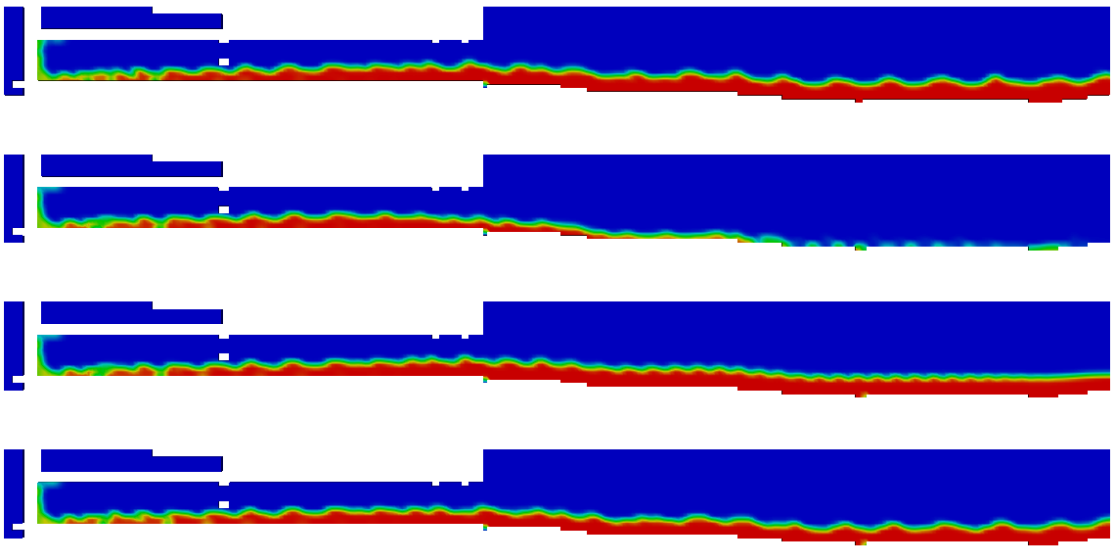

Figure 6.10: Time step $t = 15$ seconds still shows promising results for capturing the characteristics of the wave front. The effects of the reflecting and potential boundary conditions are obvious by a visual inspection at the rear part of the wave.

A better insight to the influence of the boundary conditions over time gives the analysis of the mismatch between the two simulations on the global and on the local domain. Therefore, this criterion is defined as the difference of the amount of water over the x-axis $d(x, t)$ defined

as

$$d(x_i, t) = \frac{1}{n_i} \sum_{j=1}^{n} |\gamma_{i,j} - \tilde{\gamma}_{i,j}|$$

where n_i denotes the number of voxels in vertical direction. Fig. 6.11 highlights the difference of the water height over the x-axis of the local domain for time steps $t = 12.5$ and $t = 15$ seconds.

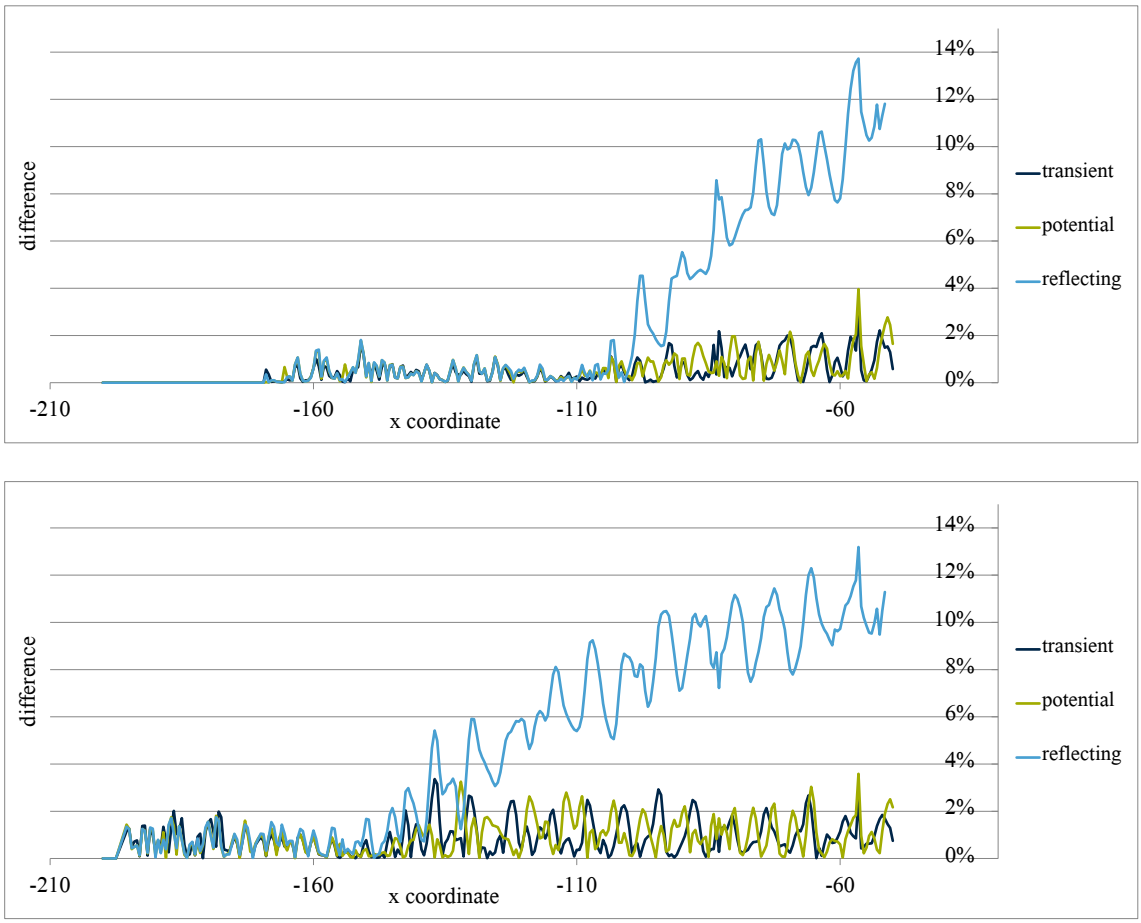

Figure 6.11: The difference $d(x, t)$ of the water height, relative to the height of the computational domain, over the x-axis of the local domain is given for time steps $t = 12.5$ (top) and $t = 15$ (bottom) seconds. It shows the propagation of the errors induced by the reflecting boundary conditions.

Based on this investigation, the acceptable accuracy of the refinement approach with reflecting boundary conditions for short and with potential and transient boundary conditions for medium-sized time intervals is assumed in this precise simulation scenario under certain conditions. The local domain of interest has to be extended to a sufficiently large domain which must be able to capture the upstream as well as the downstream behaviour of the flow in an appropriate manner. Hence, the possible time range depends on the characteristics of

the flow and the size of the enlarged local domain.

An approach with transient boundary conditions comes with much higher computational costs than the reflecting or potential boundary conditions. On the one hand, the global simulation has to be performed over longer time ranges for the derivation of the flow profile, and on the other hand, the boundary conditions have to be adapted in the time steps of the simulation on the local domain accordingly.

6.3 Postprocessing - Propagation to Product Model Data

In this chapter, the postprocessing for highly resolved parallel numerical simulations on product model data beyond graphics is introduced. After performing parallel numerical simulations as introduced in Sec. 6.1 the quantities of interest are available over the computational grid. In Sec. 4.1.2 algorithms for the graphical investigation of the results have been introduced. With these, the results are usually investigated with streamlines, isosurfaces, slice planes, etc. [81].

In the context of the work now presented, the full linkage over the different scales of the hierarchically ordered data can be used for a much more thorough investigation. It will be shown that the numerical simulation results can be mapped to the construction details of buildings and built infrastructure in order to answer question such as: *How long have the concrete walls been in contact with water during the flooding? Has the maintenance room with the electronic installation been flooded? Which was the maximal power the water exerted on a certain window?*

The key for this insight is the mapping between the mesh generation introduced in Sec. 5.3.2 and the hierarchical dual layer access concept introduced in Sec. 3.1. As depicted in Fig. 6.12, a boundary face of the computational mesh is a face of a leaf octant of a product model. This means that the fluid domain is not separated from the structural representation of the product model data. The opposite is the case. The boundary faces of the computational mesh are a sharp interface to the finest construction entities of the underlying product model data and this link is maintained by the hierarchical approach presented.

A typical work flow could look as follows. On the basis of a two-dimensional SWE simulation a heavy rain fall scenario is simulated over the domain of interest. From these global-scale results the engineer decides to investigate a certain region further with a highly resolved free surface simulation. This simulation is initialised with the results of the global simulation and provides the results on a local building scale. The user now queries all electronic devices using the spatial query approach or directly selects a construction detail as introduced in Sec. 4.2.3. From the hierarchical order of the data the common faces of the octree representation derived in Sec. 3.1.1 can be identified with the boundary faces of the generated computational mesh

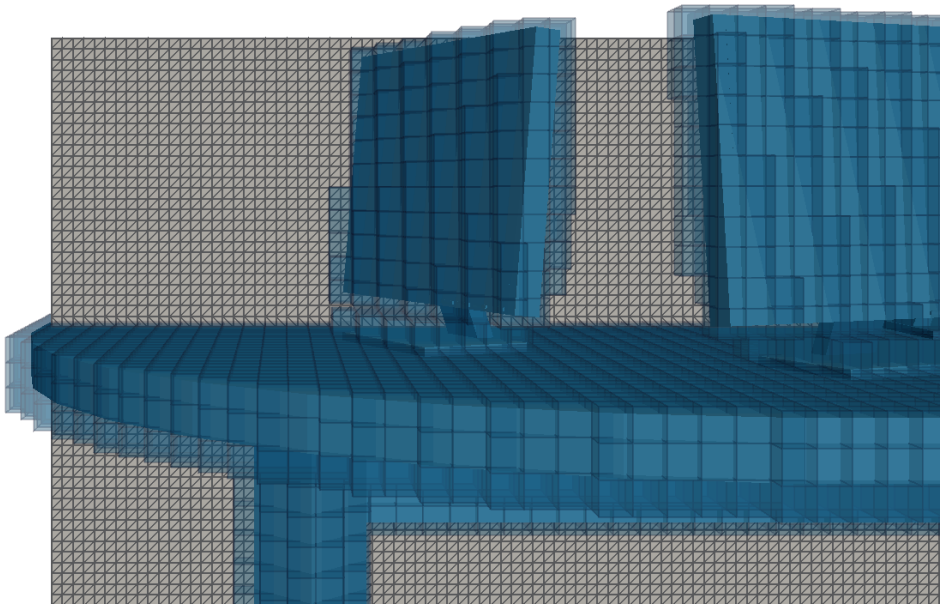

Figure 6.12: The computational mesh of the fluid flow simulation can be mapped to the construction details of the product model data. The boundary faces of the fluid mesh are shared with the octree representation of the construction detail. This mapping gives the linkage between simulation results and construction details affected.

as introduced in Sec. 5.3.2.

By doing so, the user obtains the impact of the fluid flow on the construction details. By extending this investigation from the results of a single time step to the series of results over the whole simulation, flooding profiles against a wall or the velocity profile on a window are derived. It should be pointed out again that this approach assumes the mapping of an investigated entity to its geometric representation. In order to estimate the flooding of a room, for example, this relation between the information and geometry is not given directly and has to be established as described in Sec. 3.2.4. Furthermore, the non-boundary conforming discretisation requires special focus as described at the end of Sec. 5.2.

The results of this investigation have now been implemented for the finest details of the product model data. The mapping of the simulation results to the construction details can also be seen as an enrichment of the product model data. The impact of the fluid flow can be stored for the construction details as attributes of the measured quantities during the simulation. This fuses the numerical results with the product model data for that specific simulation. Starting from this, the level of detail formulations and the hierarchical order also apply to the data set augmented with the simulation results or, to be more specific, their influence on the product model data.

6.4 Performance Results

Finally, this chapter presents the performance results for the parallel numerical simulation. They are investigated in terms of a strong speedup over the parallel processes of the simulation. The strong speedup calculates the performance as the runtime over the number of processors using a constant input size, i.e. a computational domain of constant size. The weak speedup, in contrast, describes the performance as the runtime over the number of processors where the input size is proportional to the number of processors.

Fig. 6.4 presents the scenario for the performance measurement using the *interFoam* OpenFOAM package for simulating building scale flooding. The domain of $1000 \times 1000 \times 50$ voxels is decomposed using a block structure for 1 to 64 processors.

The measurement is performed on the *Sandstorm* cluster, the complete specification of the machine is given in Sec. A.2. Through the initial conditions 304k of the 29million fluid voxels are filled with water. The simulation is performed for 1 second of real time. The complete specification of the OpenFOAM cases simulation parameters is given in Sec. A.5.

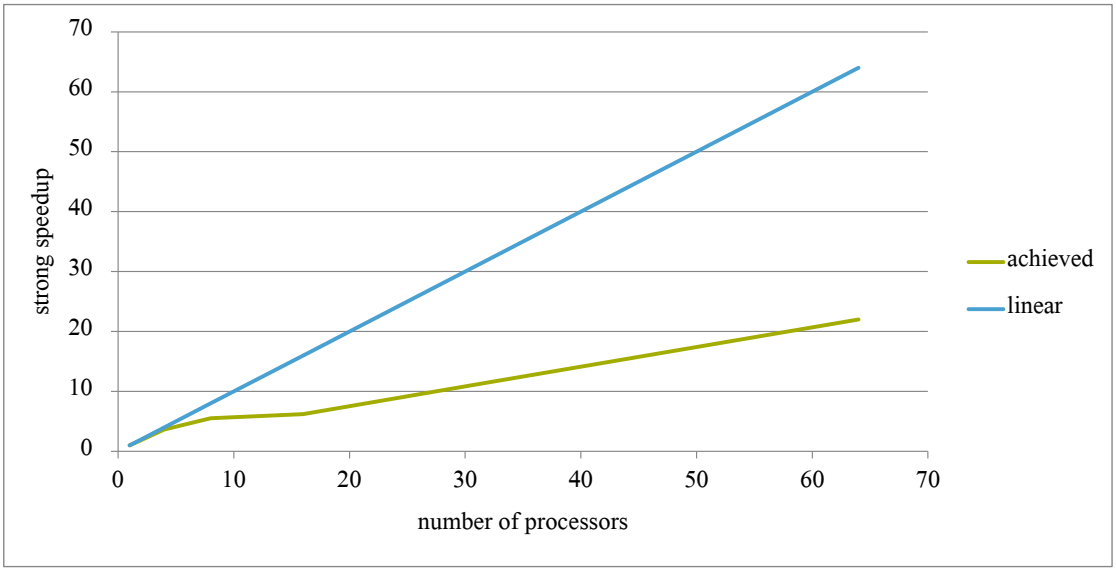

Figure 6.13: The measurement of the strong speedup for a computational domain of $1000 \times 1000 \times 50$ voxels for the *interFoam* OpenFOAM solver is carried out over 1 to 64 processors.

The scaling in Fig. 6.13 shows the results expected for the given architecture and hardware. As soon as the number of processors exceeds the directly accessible area of the main memory, a first drop in parallel efficiency occurs. By exceeding 16 processors, the performance of the standard network interfaces suffers a further drop in parallel efficiency. From that level of efficiency the speedup is achieved up to the full 64 processors of the machine. Obviously, the sustained speedup is not optimal and the parallelisation strategy has to be subject to

further research. Recent parallel installations come with a high number of nodes and a parallel efficiency of 35% is unacceptable for 64 parallel processors. Nevertheless, the proposed framework has not been optimised to an individual installation and the interfaces are of general type for different computing architectures. Therefore, the results can be seen as acceptable but extendible.

Chapter 7

Urban Flooding Simulation with Pipe Network Interaction

Urban flooding simulations are of great and increasing interest. The European floods in 2013 and 2002 are examples of the impact on regions, cities, buildings, constructions and, first and foremost, the people affected by them.

In this chapter, an urban flooding simulation including the interaction of pipe network flow originating from the sewer system is investigated. This application integrates the interfaces and approaches of this work into an investigation relevant to engineering and shows the potential of this approach. Furthermore, it can be seen as a blue print for applying the work presented to emerging questions. The hierarchical ordering of large data sets from construction and built infrastructure and the efficient parallel access to it provides the basis for a broad field of applications in engineering and computational science. This chapter shows the applicability of the approach and points the way to many future investigations based on this framework.

The investigation of urban flooding on a highly resolved multi-resolution data basis by performing parallel numerical simulations gives engineers, city planners and crisis management groups a tool to make decisions on a well-founded basis. By having such a tool at hand, decisions can be made to divert water to unpopulated areas, a forecast of the impact can be given, existing installations can be evaluated, or new measures for the future can be planned, to name just a few.

This point of view evidences the interdisciplinary approach needed for tackling the challenges of the future. Providing this kind of a tool requires the knowledge of researchers from many fields. The detailed description of the terrain, the surface and pipe networks requires the knowledge of geoinformation specialists. Constructions and built infrastructure with their detailed product modelling require specialists from Building Information Modelling. In order

to perform numerical simulations of highly resolved urban flooding scenarios, modelling specialists are needed to develop accurate models for the principles of flow in these cases. The discretisation and solving of these models for large-scale scenarios and accurate simulations requires the involvement of computational scientists from the field of scientific and high performance computing. The interpretation, the validation and the derivation of the impact of the results achieved require specialists from hydromechanics and hydraulic engineering. In short, the challenges of the future demand that the different disciplines team up and find a common language. The work presented here and this chapter in particular have also to be seen in the context of this vision. It shows the way for integrating different disciplines and bringing knowledge together. On this path, urban flooding for a heavy rain scenario where the water drainage system collapses is investigated. In this scenario, heavy rain floods a specific part of a city. The large amount of water puts hydrostatic pressure on the sewers of the drainage system in this region and causes the sewer system to flood parts of the city where no rain is falling as well. The interaction of the pipe flow through the drainage system and the surface flow will be simulated in order to gain insight about the effect on buildings, not only the outside, but also to the equipment and installations inside. Both sources of the flow, the overground surface flow and the flow in the pipe network, have to be investigated individually as well as how they interplay.

This chapter is organised as follows. In Sec. 7.1, an investigation of existing research approaches is given. Starting from existing coupled simulation, a holistic approach for pipe network and surface flow is introduced in Sec. 7.2. In Sec. 7.3, this approach is put into practice for a highly resolved configuration of Munich city centre.

7.1 Existing Work

In this chapter, existing literature on the investigation of urban surface flow, flow in water drainage systems, run-off of surfaces and roof structures, as well as their interaction is reviewed. The investigation and simulation of this behaviour is of practical engineering relevance [139]. In May 2013, the Institution of Civil Engineers (ICE) held the ICE Flooding 2013 [140] in order to bring together scientists and researchers to develop flood resilient communities.

There are a variety of approaches for the one-dimensional treatment of pipe network flow and the flow of streets. Urban surface flow is usually simulated using two-dimensional treatments. The coupling is introduced for 1D, 1D/2D and 2D formulations with separate treatment of the behaviour and the integration to holistic models.

In [141] Mignot et al. investigate the flood of Richelieu in Nimes, France in 1988 by applying the two-dimensional SWE with an explicit second order scheme. In [142] Fewtrell et al.

introduce a flood simulation for a hypothetical flooding of Greenfields, Glasgow. As they assume that the practical investigation of flood scenarios is limited to standard desktop PCs they investigate the efficiency of a two-dimensional storage cell-type approach on the basis of GIS digital elevation models. In order to show a time efficient approach with low hardware requirements, Chen et al. introduce an adapted flat water model in [143]. In [144] the scenario of Glasgow is investigated for six different hydraulic models, all simulating the surface flow for the 1.0×0.4km spanning site which occurred in July 2002. Also, in [145] Shahapure et al. focus on the derivation of the elevation plan for GIS-based input data for Navi Mumbai, in Maharashtra. They introduce a finite element method (FEM) based approach in order to solve the mass balance equation for the overland flow and the diffusion wave form of the Saint Vernant equations for the storm water flow. In [146] Bates et al apply a one-dimensional SWE approach to a two-dimensional storage cell inundation in order to decouple the two coordinates.

The preface of the special edition of the *Journal of Hydrology* [147] on urban hydrology introduces several papers on the areas rainfall on the city scale, modelling of rainfall run-off, and interaction of the flow on the surface and in the pipe network. In [148] Schmitt et al. present a small case study with formulated demands on the data and verification for one-dimensional drainage systems. In [149] Carr et al. focus on the effects of the degree of resolution of a two-dimensional approach. They state that two-dimensional and highly resolved approaches are necessary in order to make cross and secondary flows become observable. In [150] Mark et al. compare the one-dimensional treatment of the interaction between the pipe network, the open channel flow in streets, and areas with fluid at rest. In [151] Bolle et al. focus on the investigation of sewer network and river flow behaviour. Starting from the different interaction and influences the two flow regimes have on each other they propose an integrated treatment of the underlying schemes instead of treating them separately. In [152] Barnard et al. compare the linking of packages for the one-dimensional formulation of the storm and waste water systems with the two-dimensional formulation of the free surface flow. It is applied by Fairfield City Council located in the state of New South Wales, Australia, among others. UK [153] In [154] Kouyi et al. focus on a one-dimensional linking between run-off of heavy rainfall and flow originating from river and sewer networks. They state that they have captured the characteristics of gauged flow well with the calibration of only the proportional loss of the model parameters. In [155] Gray presents a one- and a two-dimensional coupled approach for multiple gauged events of hydrological processes originating from the run-off from surfaces, the flow in the streets, and the input from the house roofs.

It is shown that one-dimensional formulations for the flow in water drainage systems and two-dimensional treatment of surface flow provide reliable results. In this work, a full three-dimensional treatment of the data set is introduced in a monolithic approach in order to also investigate the impact on constructions and built infrastructure. The impact of a collapsing water drainage system and the flooding of an urban region resulting from the water of the

pipe network is thus investigated. Owing to the three-dimensional multi-scale formulation, the impact on constructions and the installations within can be investigated together with the local scale among individual buildings and the global flow behaviour on the city scale.

7.2 A Holistic Pipe Network and Surface Flow Approach

In this section, the holistic approach for simulating urban flooding scenarios originating from heavy rain events with pipe network and surface flow interaction is introduced. The demands on such a simulation are as follows.

- The terrain surface of the city and the geometric representation including constructions and built infrastructure are combined in one model.
- The pipe network of the urban drainage system is directly integrated into that model.
- The numerical simulation applied covers both the three-dimensional free surface flow of the overground water and the one-dimensional pipe network flow in the sewer system.
- The interplay between the overground and pipe network flow is covered.
- A multi-resolution approach ensures adaptive refinement from a city-wide scale to the local building scale including the construction details in the interior of the buildings.
- A parallelisation strategy for sufficiently large computer clusters ensures feasible simulation runtime of such a large problem.
- The postprocessing of the simulation results reveals the global flow behaviour on the city scale and also the impact on constructions and built infrastructure.

The ability of the work presented to cover these demands has been shown for some of these points. The fusion of the different data from GIS and BIM for the terrain specification, the constructions and built infrastructure, and the sewer system's pipe network have been introduced in Sec. 2.6. The parallel handling and efficient access to the created database have been introduced in Sec. 3. An interface for generating sufficiently large meshes was introduced in Sec. 5.3. The resolution adapted approach was introduced in Sec. 6.2.1 and a multi-scale postprocessing with the evaluation of fluid flow results and their impact on construction details was introduced in Sec. 6.3.

What is still missing and is introduced in the following sections is the ability to perform both the overground free surface and the drainage system pipe network flow in one model. It must be shown that the characteristics of one-dimensional pipe network flow are given by a free surface flow solver applied to a three-dimensional computation domain. Furthermore, the

interplay of fluid exchange on the transition from the pipe network to the free surface flow must be investigated.

It will be shown that, with the integration of the *interFoam* package, the algorithms presented in this work fulfill these demands. Before presenting the results of the simulation performed, the unanswered questions concerning the surface flow and the pipe network flow including their interplay are discussed in the following sections.

7.2.1 Surface Flow

The investigation of surface flow by incorporating the *interFoam* package has been extensively investigated in [133]. *interFoam* has been validated and verified against standard benchmarks such as breaking dam scenarios. In order to apply the solver for simulating urban flooding scenarios, the city model of Munich city centre has been derived with the outer shell description of constructions and built infrastructure.

The model spans an area of 2km by 2km. It contains over 3000 buildings and the fluid mesh is resolved with $1000 \times 1000 \times 50$ voxels. The amount of water is applied by specifying the inner conditions as the respective amount of water at rest as given in Fig 7.1. Boundary conditions enforce free outflow of the domain and laminar flow is assumed. The time step width is dynamically adapted. The detailed specification of the simulation case is given in Sec. A.5.

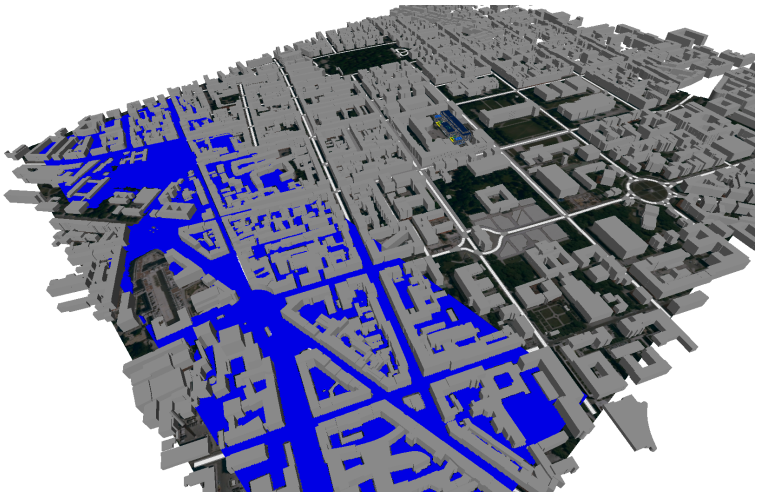

Figure 7.1: The amount of water is applied by specifying the initial conditions as the respective amount of water at rest.

The simulation is performed on the *Sandstorm* Cluster described in Sec. A.2 with a domain decomposition to 48 processors. The simulation is performed for 60 seconds in real time and the results are given in Fig. 7.2.

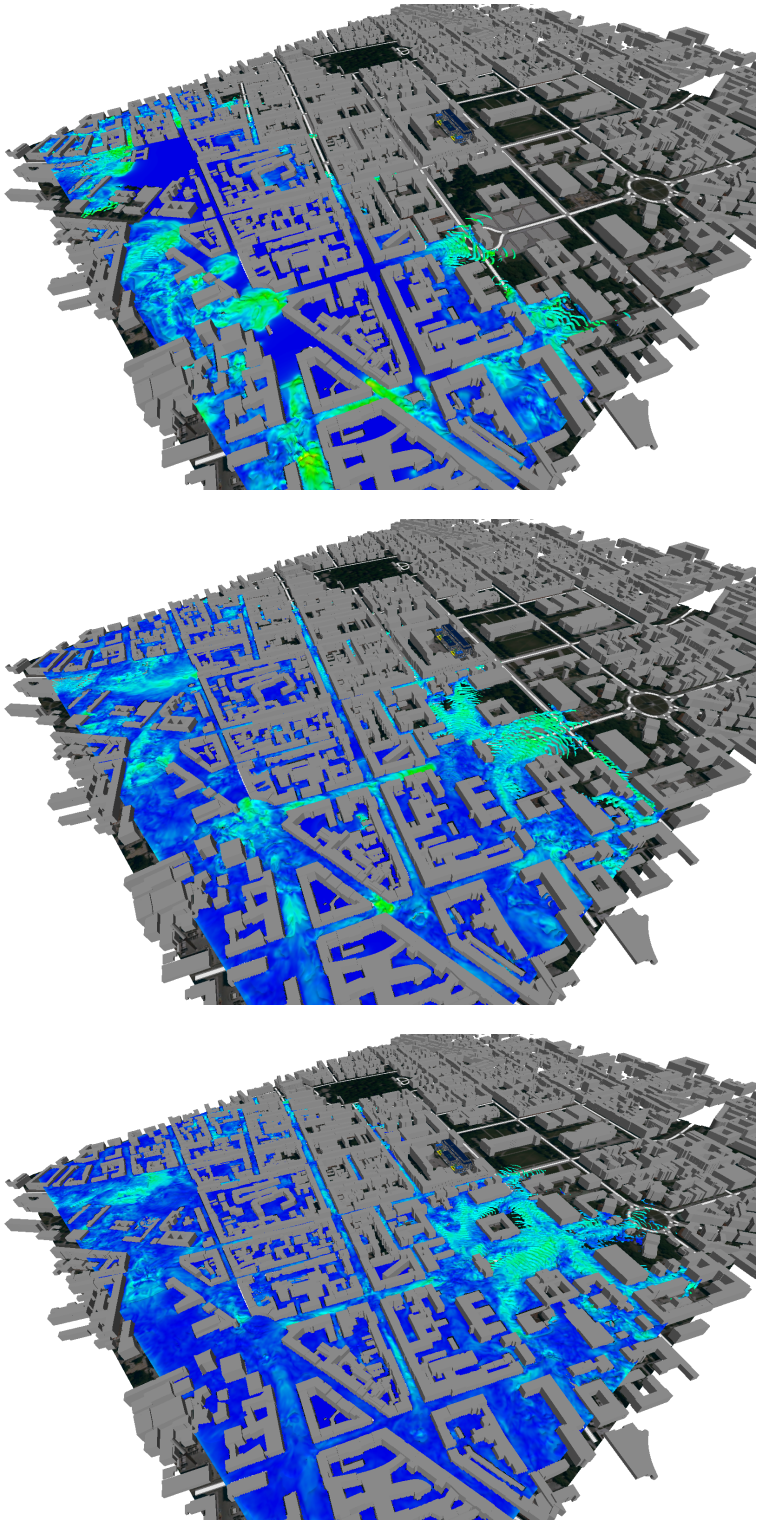

Figure 7.2: The flooding of the detailed city model is visualised with the contour of free surface simulation and colour encoded velocities.

These results are very plausible. Initial conditions apply a partial flooding of the city with a fluid at rest and as the flow evolves, the flooding propagates along the streets. The water follows the lower parts of the domain and the main flow progresses around buildings accordingly. For surface flow phenomena a valid simulation can be expected from the inspection of the results and the investigations performed in the literature showing the validation and verification of *interFoam* for benchmark tests.

7.2.2 Pipe Network Flow

In this section, a holistic approach for simulating one-dimensional pipe network flow, three-dimensional free surface flow as well as their interaction is developed. This will be applied in order to simulate the flooding of a city due to the collapsing drainage system within one single computational model. In order to achieve this, the following procedure will be followed. Initially, the computational mesh for the terrain and the buildings is derived. Afterwards, the segment-wise axis-aligned discretisation of the pipe network is added to the mesh. Therefore it has to be shown that a three-dimensional free surface flow solver is capable of capturing one-dimensional pipe flow characteristics embedded in a three-dimensional computational domain. By doing so, the computational model of the surface flooding can be extended by pipe-network characteristics and this is achieved by a straight-forward and minimal extension of the mesh with the pipe-network discretisation.

For this approach further aspects have to be investigated. The discretisation of the pipe-network segments follows the coordinate axes; a straight segment will be split up into two perpendicular axis-aligned segments. Furthermore, the resolution of the computational mesh in general will be significantly coarser than the diameter of the pipe segments. Additionally, the voxel-representation induces a rectangular instead of a round shape of the cross section. One approach for tackling these limitations is to make use of the bi-directional linkage between the voxel-discretisation and the underlying geometry. The size of the representing voxels can be modified for every single pipe segment individually. In order to capture a realistic behaviour, the pressure drop and the flow velocity following the correct shape of every pipe segment have to be determined and from that the diameter of the discretised approximation has to be adapted accordingly.

In order to investigate this, a benchmark has been set up for which the characteristics of flow are known. Two basins are coupled via a single pipe. The two basins are filled with water to different levels, the pipe is also filled with water. At the beginning of the simulation all fluids are at rest. This scenario matches the known setting for the behaviour of pipe flow between two basins of different hydraulic heads. As is known from the basics of hydromechanics for Newtonian incompressible one-dimensional flow in a pipe [131], the pressure line of the pipe is linear between the hydraulic head values at the two basins. It thus follows that the velocity

in the pipe is constant over the length.

This scenario is implemented in a three-dimensional benchmark, incorporating one-dimensional pipe flow. As given in Fig. 7.3, the computational mesh for the two basins and the connecting pipe is resolved with $100 \times 100 \times 50$ voxels. The crucial point is now that the pipe is resolved in the direction orthogonal to the flow with one voxel only.

Figure 7.3: A three-dimensional benchmark of two basins connected with a pipe implies that resolving the pipe with a single voxel in the orthogonal directions of the flow imposes the characteristic of one-dimensional flow.

This degenerated resolution of the pipe forces OpenFOAM to perform one-dimensional simulation of the fluid flow in the pipe, embedded to a three-dimensional domain. As OpenFOAM follows a strictly three-dimensional simulation approach, the specification of boundary conditions enables simulations of lower dimensions. For a two-dimensional simulation OpenFOAM expects a three-dimensional grid with a fixed resolution of 1 in one dimension. The boundary conditions applied along this dimension enforce two-dimensional simulation of the degenerated three-dimensional domain. One-dimensional treatment is achieved by performing boundary conditions accordingly along a second dimension.

From this investigation the results of the embedded one-dimensional flow for the three-dimensional basin test can now be investigated. Fig. 7.4 shows the pressure line over the pipe length. As expected, a pressure drop occurs where the pipe enters the basins. For reasons of completeness the values of the velocity orthogonal to the pipe length are shown in Fig. 7.5 and are neglected.

Finally, it can be stated that by integrating dimensionally reduced pipe discretisation into a three-dimensional domain, basic one-dimensional pipe flow can be evaluated using *interFoam*. It should be pointed out clearly that this is for laminar flow the case only; turbulent flow regimes are not covered by this approach. In order to capture the influence of the effects introduced by pipe roughness in the turbulent case, a more advanced treatment for capturing the pressure drop induced by the pipes has to be performed.

The discretisation of the pipe with one voxel diameter also follows directly from the approach of this framework. The data for pipe networks of sewer systems in general are a lower dimensionality formulation. As derived from the GIS sources in Sec. 2.3.2, the specification

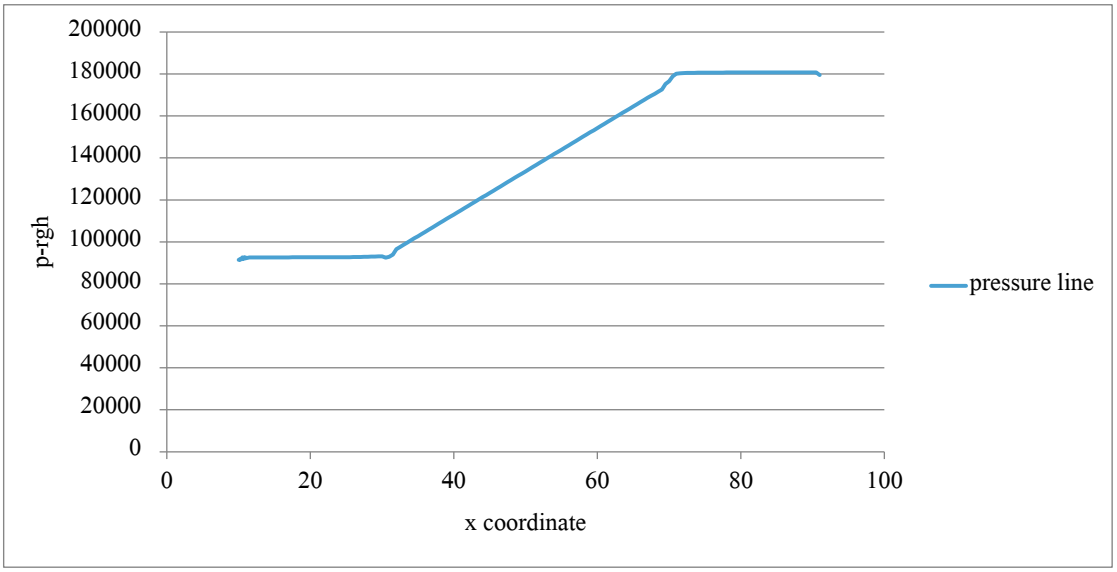

Figure 7.4: The pressure line over the axis along the flow in the direction of the pipe shows that the one voxel wide discretisation imposes one-dimensional pipe flow in the three-dimensional domain.

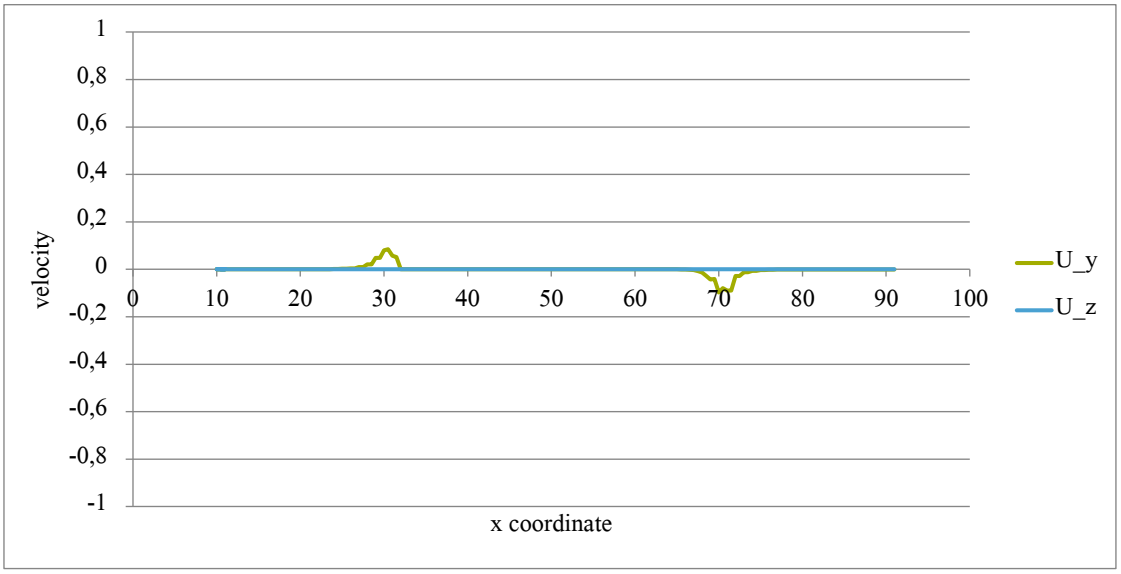

Figure 7.5: The velocities orthogonal to the flow in the pipe are depicted but neglected due to the one-dimensional interpretation.

of the Munich sewer system is available as a graph. From this formulation, a one-dimensional discretisation of the pipe is already performed because of the octree representation of the linear pipes derived for the mesh generation.

One aspect which should be mentioned clearly is the influence of the pipe diameter on the time-step size of the simulation. As the maximal time-step size for a numerical simulation according to the Courant-Friedrichs-Lewy [131] condition is limited by the minimal mesh

size, having small voxels in a computation also enforces small time steps. On the one hand, there are local time-stepping approaches [156, 157, 158] which focus on tackling precisely this challenge; on the other hand, in the simulation applied, the diameter of the pipes is in the same range of accuracy as the domain resolution. As the computational domain is resolved with a fine resolution to cover the geometry which matches the resolution of the pipe diameter, the step-size is not further limited by introducing one-dimensional pipe segments.

7.2.3 Interplay of Surface and Pipe Network Flow

In this section, the interplay of surface and pipe network flow is investigated or, to be more specific, the question as to which principles the inflow from the surface to the pipe network follows and vice versa is addressed.

As investigated by existing approaches in Sec. 7.1, both cases should be examined differently. Whereas the exit of the pipe network flow to the surface and the confluence of different surface flows is covered by free surface flow formulations such as *interFoam*, the inflow from the surface to the pipe network is a highly complex phenomenon and depends on the target scenario.

For urban flooding events originating from heavy rain scenarios, the velocity of the surface flow tends to be lower and the hydraulic head over the sewers is utilised as a pressure boundary condition of the sewer network. For urban flooding events originating from flooding rivers, the velocity and amount of water impacting on the urban region is usually so high that the impact of the sewer network on the global behaviour is neglected.

The investigation of the exact inflow of surface flow to a manhole has been examined in [159]. In their work, the authors derive a highly resolved discretisation of a single manhole and validate their results of the numerical simulation with a 1:1 laboratory test. Their convergence studies show that the exact geometry has to be resolved accurately in order to cover the correct behaviour. From these investigations the asymptotic inflow behaviour of surface flow to the manholes is derived. A direct integration of the exact inflow behaviour of surface flow to pipe network manholes cannot be achieved with a monolithic approach for an urban flow investigation if only because of the mesh accuracy required. Nevertheless, in the scenario investigated the behaviour is not driven mainly by the inflow of the surface flow to the sewer system but by the flooding of the city due to the collapsing drainage system.

Therefore, the good results shown with the *interFoam* package for the pipe network flow in combination with the free surface flow overground justify the application of the approach for the given scenario.

7.3 The Scenario

This chapter now presents the application scenario of the framework presented. It covers an urban flooding simulation of a heavy rainfall event where surface flow and pipe network flow are both investigated including their interplay.

In this scenario, a certain region of the city is affected by heavy rainfall. This causes hydrostatic pressure on a part of the rain water drainage system and the sewers in that area. Because of this hydrostatic pressure, the flow in the pipe network evolves and floods parts of the city where no rain is falling. The evolution of the flow in the pipe network and on the surface is investigated and the flooded areas of the city and affected buildings are evaluated.

7.3.1 Data Basis and Discretisation

The data basis covers the city centre of Munich including the terrain description, the modelling of buildings, and the pipe network as derived in Sec. 2.6. Following the approach introduced in this work, the model is discretised with a resolution of up to $1800 \times 1650 \times 75 = 222.75$million voxels. Fig. 7.6 shows the contour of the boundary of the computational domain both for the global and for the local scale.

7.3.2 Boundary Conditions

In order to impose the scenario of a collapsing rain water drainage system, the sewer network is given a constant hydrostatic pressure at one of the inlets. This is achieved by placing a large reservoir of water over one of the inlets as shown in Fig. 7.7. Furthermore, the width of this inlet and the attaching pipe is 5 meters and the boundary condition is set to slip. By doing so, a non-physical start-up phase is imposed in order cope with the limitations of the computational model and let the characteristics of the collapsing drainage system evolve. The pipe network system is assumed to be already flooded.

The domain boundary conditions are set to free outflow. The complete specification of the parameters of the simulation is given in Sec. A.5.

7.3.3 Results

The simulation follows the OpenFOAM package *interFoam*, the definition of the simulation case is given in Sec. A.5. The simulation is performed with a resolution of up to 222.75million voxels of the computational domain. In Fig. 7.8 the results of the pipe network and the surface flow are depicted. For better visibility the constructions are removed. In Fig. 7.9 the results

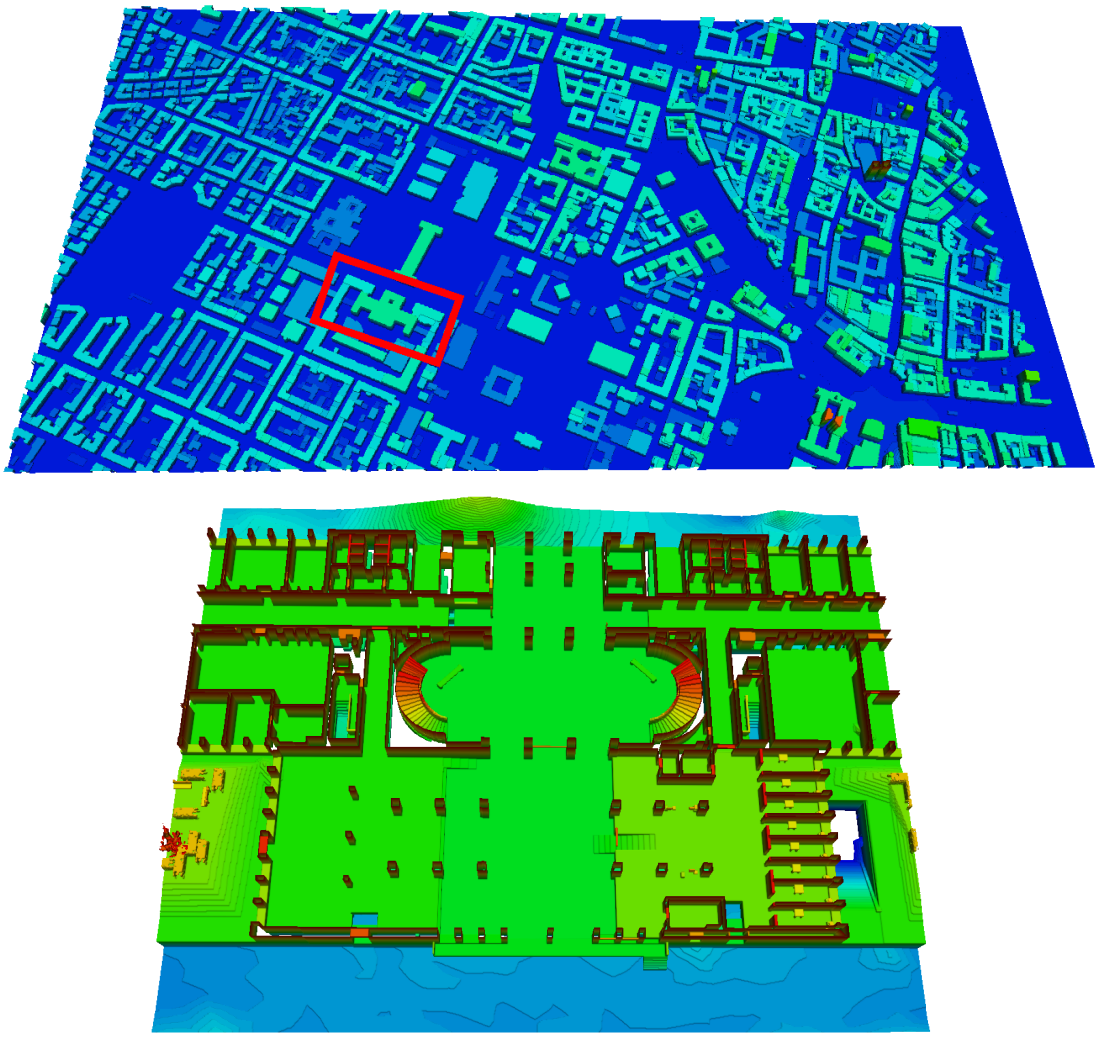

Figure 7.6: The computational domain is depicted by its 0.5 contour for the global scale (top) and for the local building scale (bottom). The relative height of the surface is colour encoded.

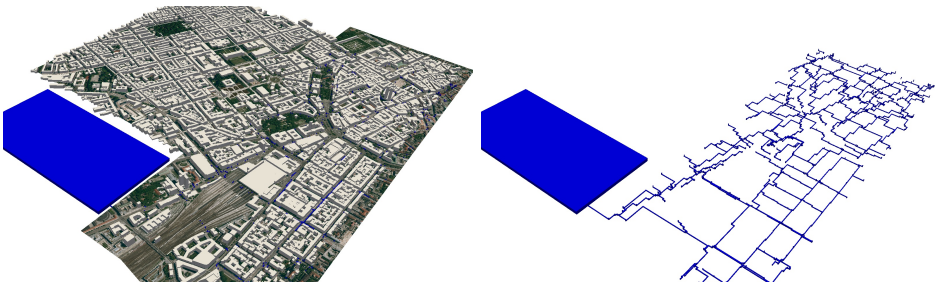

Figure 7.7: The collapsing drainage water system is modelled by placing a constant hydrostatic pressure on the inlet of the network. The constant pressure is imposed by placing a sufficiently large reservoir on the inlet. As an initial condition, the pipe network is already flooded.

of the surface flow are depicted with visualised constructions in order to show the flooding of the city due to the collapsing water drainage system. A visual inspection of the simulation results including the canal flow below the complex city structure shows very plausible results.

In Fig. 7.10, the first adapted resolution refinement is given for the flow behaviour on the local building-scale simulation. In Fig. 7.11 the second adapted resolution refinement is given for the projection of the local building-scale flow behaviour on the construction detail scale. All simulations have been performed on the *Sandstorm* cluster, the complete specification of the machine is given in Sec. A.2. The simulation of the pipe network and the surface flow consumed over 5000 core-hours, the adapted simulations of the local building-scale flow behaviour consumed approximately 400 core-hours each.

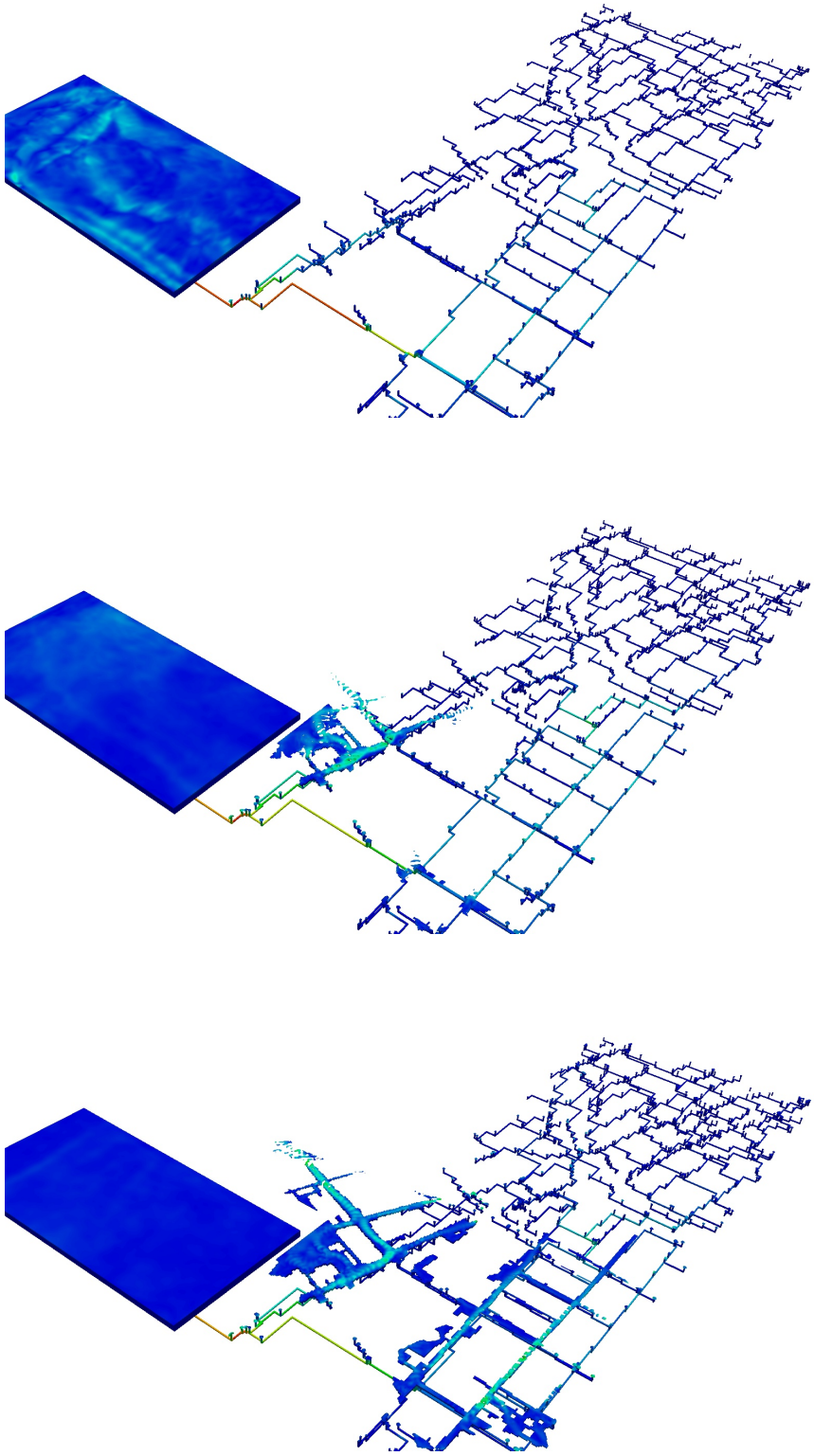

Figure 7.8: The results of the pipe network and the surface flow are visualised without the constructions for better visibility.

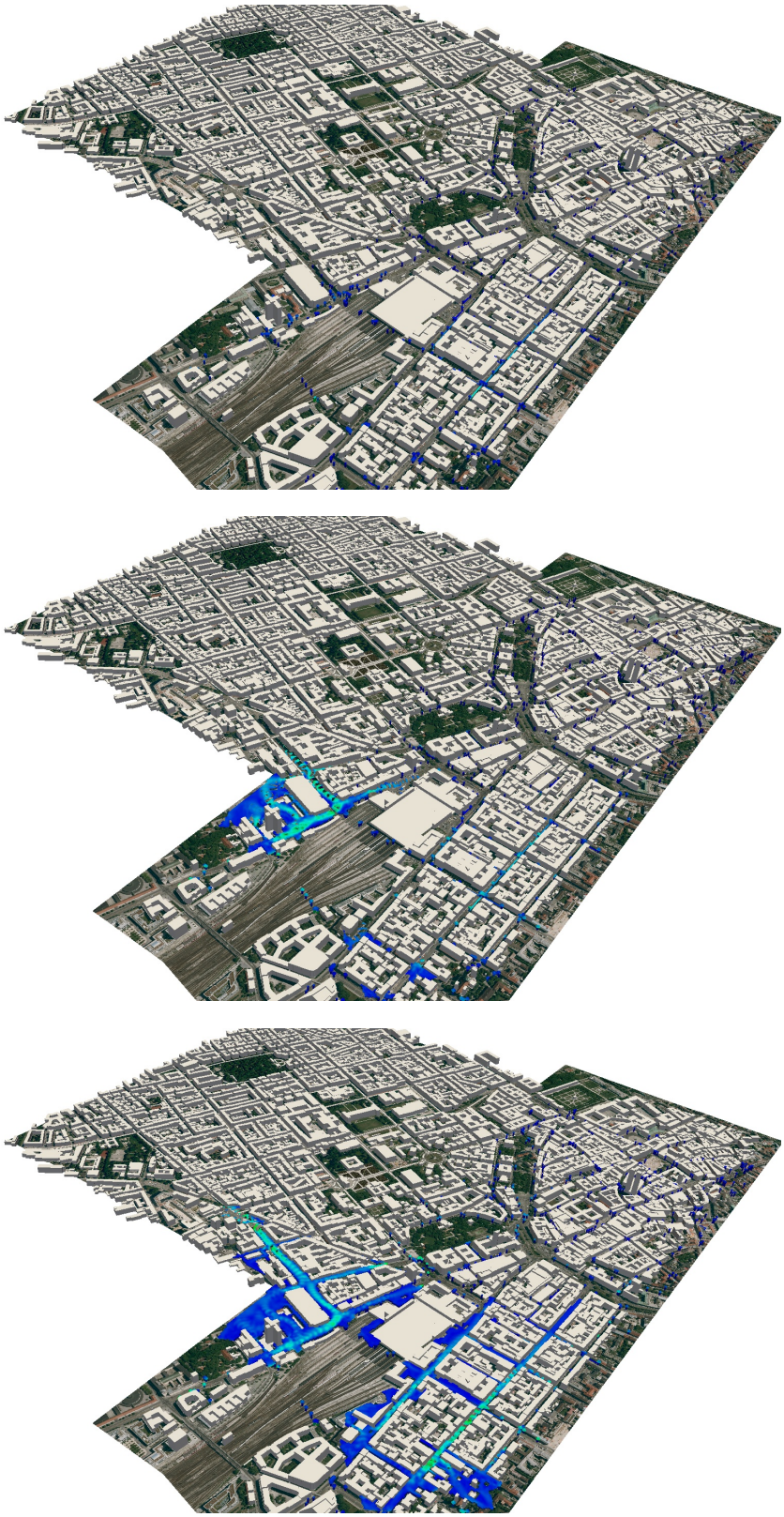

Figure 7.9: The surface flow is depicted with visualised constructions in order to show the flooding of the city due to the collapsing water drainage system.

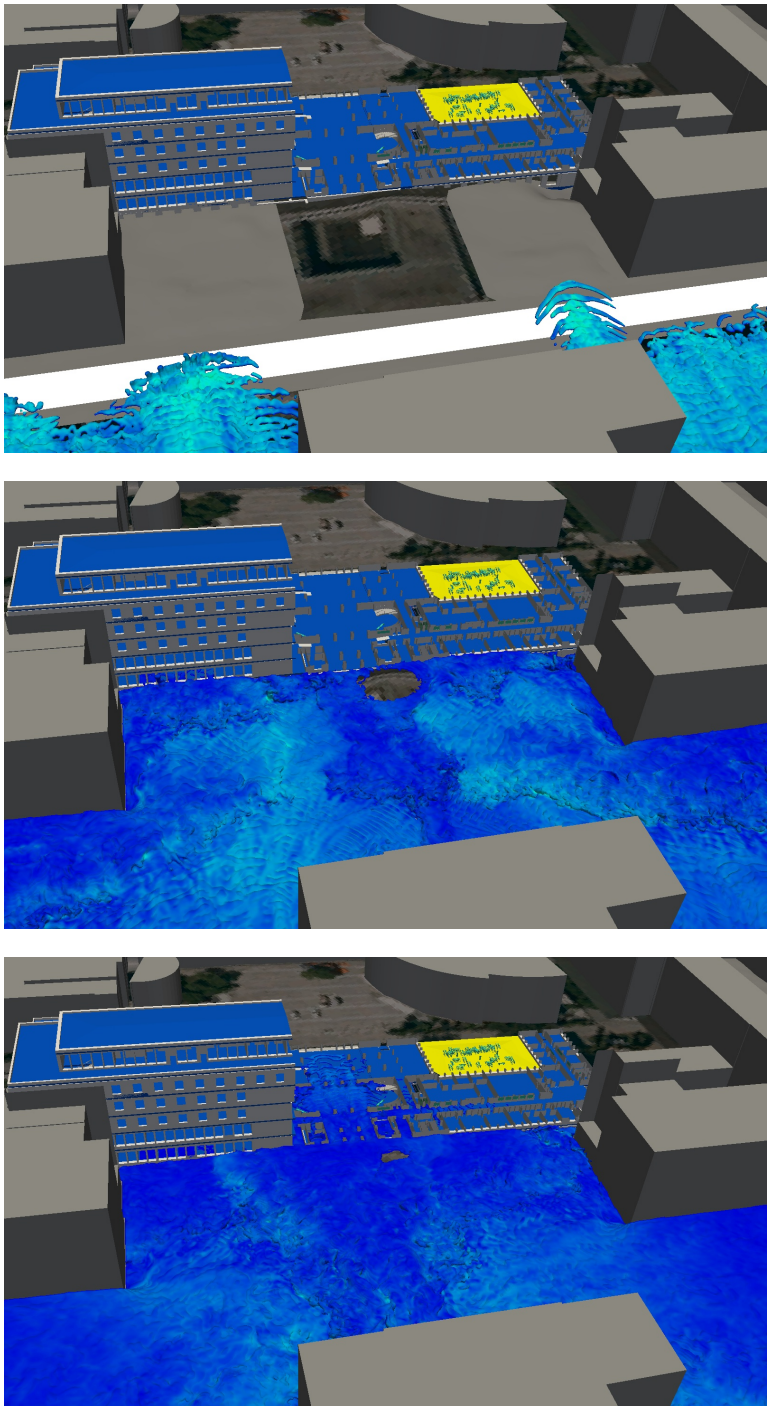

Figure 7.10: The first adapted resolution refinement gives the flow behaviour on the local building-scale simulation.

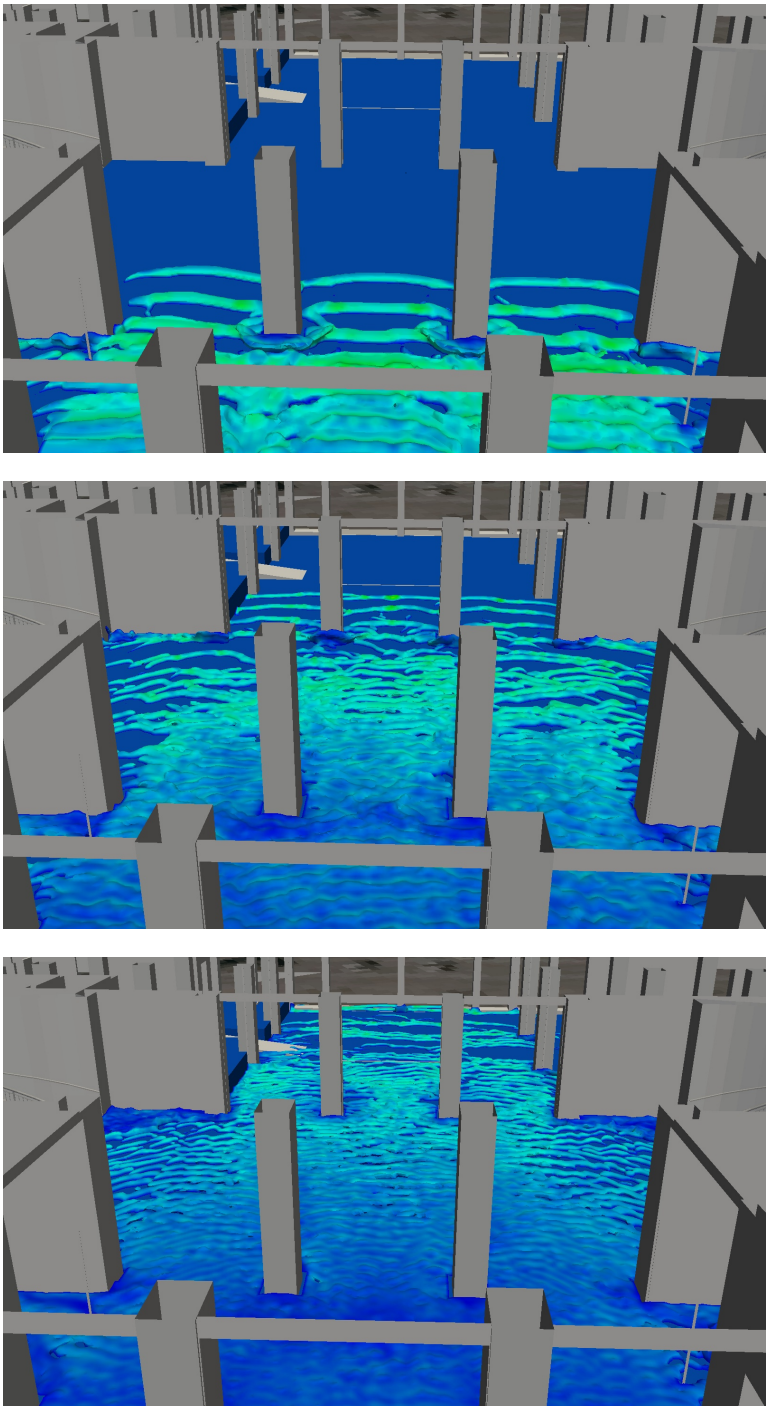

Figure 7.11: The second adapted resolution refinement projects the local building-scale flow behaviour on the construction detail scale as the initial conditions of the simulation.

Chapter 8

Conclusion and Outlook

In this work, techniques from computational science and engineering were applied to the approaches of civil engineering in order to make a contribution to getting a step closer in answering the demanding questions and challenges of our cities.

A parallel data access framework with clearly defined and proven interfaces to all parts of the simulation pipeline such as preprocessing, numerical simulation and postprocessing was developed. This framework is capable of storing, handling and providing access to large amounts of highly detailed data from constructions, built infrastructure, geographical data and infrastructure networks.

The strict usage of parallelisation techniques and efficient algorithms was necessary and introduced in detail. These techniques made it possible to create a framework which can provide access to multi-resolution representations of large data sets fast enough to put into practice a complete exploration and simulation pipeline, which ranges from a multi-monitor or CAVE-based real-time visualisation to multi-scale simulation scenarios such as flooding of urban regions.

Finally, the application of an urban flooding simulation with the incorporation of pipe network interaction was given. The capabilities of the framework allow the coupling of the different simulations to be handled efficiently and insight is gained over all scales from the global flow behaviour to the impact on single construction entities.

This work is an interdisciplinary approach which focuses on integrating efficient techniques from the fields of civil engineering, hydromechanics, visualisation and scientific computing as depicted in Fig. 8.2. This approach also gives rise to the outlook and potential for future work, especially the applications from fluid flow and hydromechanics. The complete in-depth analysis of fluid phenomena cannot be treated in such a work, nor does it claim to do so. The focus of this work was to show the possible links to engineering approaches and provide a technical breakthrough for these techniques.

Many questions point the way for future research such as the integration of turbulence models into the fluid flow simulations performed. The geographical data provide the land usage description, just as the building product models describe the surface properties of the construction details or the roughness values of the pipes of the drainage system.

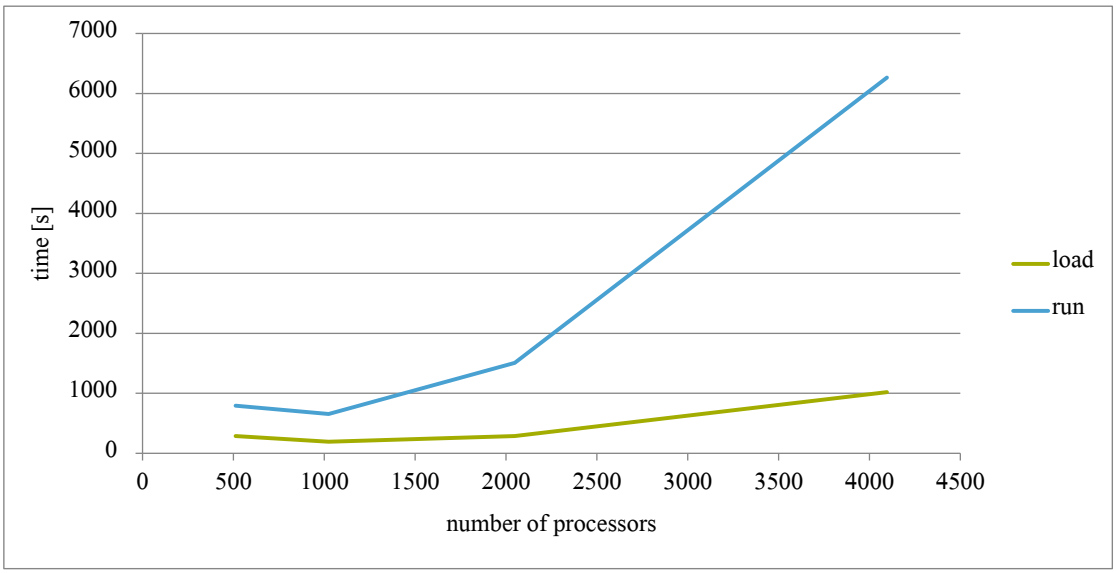

Figure 8.1: The load time of the computational mesh and the run time of the numerical simulation as function of the number of processors on the Shaheen@KAUST supercomputer shows the demand for advanced grid generation and load balancing techniques.

Another starting point is the performance of the numerical simulations applied on supercomputers. Fig. 8.1 gives the load time of the mesh and the runtime of the numerical simulation as function of the number of processors on the Shaheen@KAUST A.3 for the free surface simulation as introduced in Sec. 7, two points are obvious. First, the file-based mesh distribution of the domain decomposition levels off for higher numbers of processors as all data have to be loaded from the same parallel file system. Second, load distribution is applied in a preprocessing step and no load balancing is performed during simulation. The computational load does not only depend on the number of cells per processor but also on fluid flow driven criteria, such as the handling of the free surface in this case. Thus, the computational load is not equally distributed over the processors during the simulation because of the communication, the runtime even increases for high numbers of processors. The results show that there is still room for improving the efficiency of parallel numerical simulations and adapting new solvers of fluid phenomena. Moreover, the extension to computational solid mechanics should be targeted, as fine construction and material parameters are available from the fully detailed product model descriptions.

Furthermore, the mesh generation in the work introduced is performed on axis parallel Cartesian grids. The underlying product model data give a triangular representation of the ge-

ometry. Owing to the axis parallelism of the mesh, this geometry is never resolved to full accuracy. By extending the mesh generation on the level of the leaf nodes to the exact intersection with the primitives covered, boundary conforming meshes could be generated. Alternatively, numerical approaches incorporating immersed boundary methods should be investigated in order to cope with the imperfect mesh generation. Furthermore, the geometric data basis of the triangular representation can be extended to more advanced descriptions such as high order or NURBS descriptions.

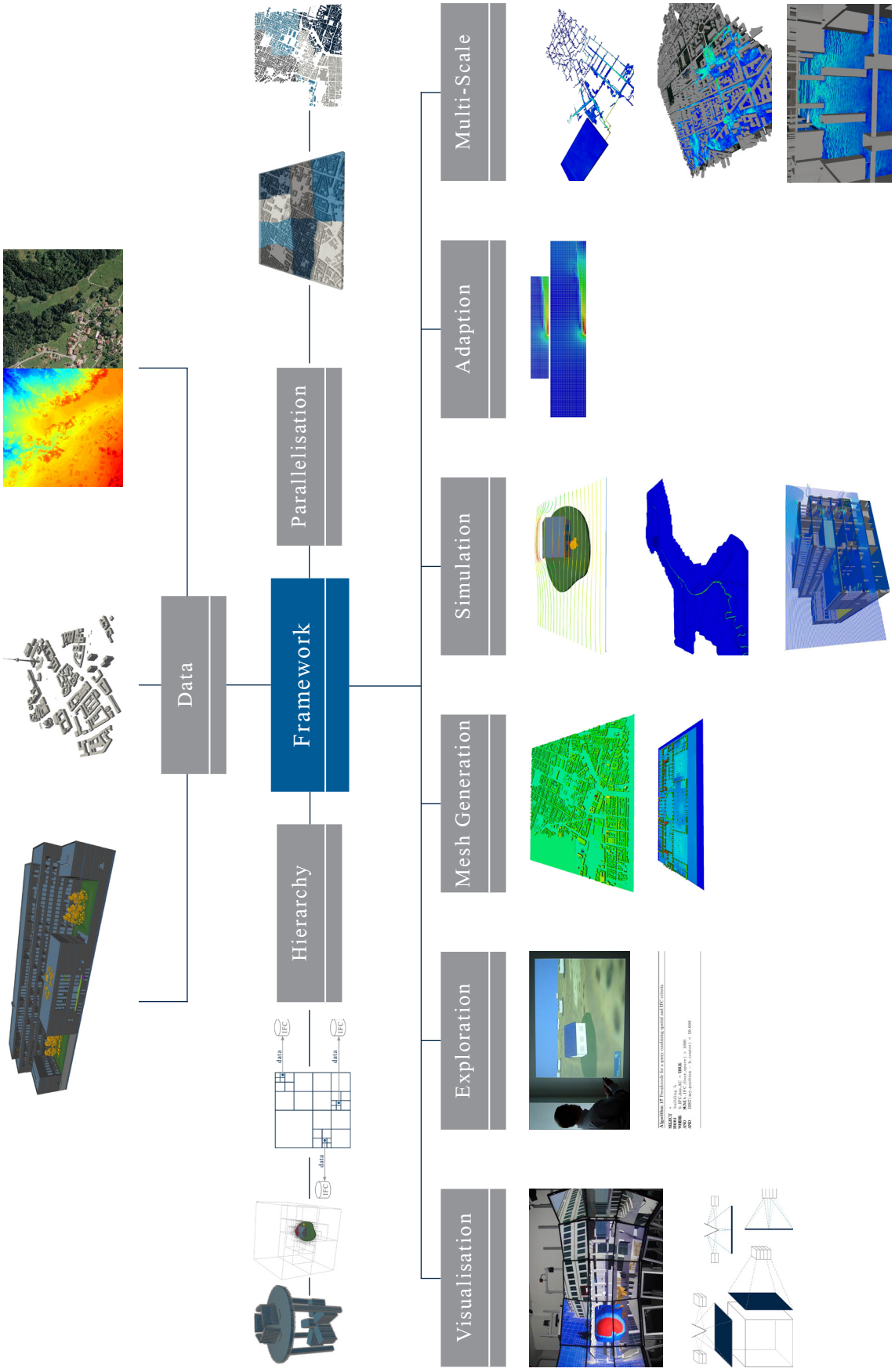

Figure 8.2: This work is an interdisciplinary approach which focuses on integrating efficient techniques from the fields of civil engineering, hydromechanics, visualisation and scientific computing.

Appendix A

Appendix

A.1 Data Fusion and Set Augmentation

What is crucial for the work presented is the assembly of a large data set which covers a domain of sufficient size and where all data introduced from the GIS to the BIM scale are organised. Highly detailed product models of buildings are available but only as an individual model of one specific building without the assembly in its neighbourhood. City models are available but the level of detail for individual buildings is only available at LoD1 for many and at LoD2 for specific regions, see [160]. Approaches exist for extending GIS by BIM specifications [40] and building up highly detailed city models [161], but there is still no dataset available which is fully detailed over the scales of a city and its constructions and built infrastructure.

Therefore an algorithm for enriching a city model with fully detailed product models has been developed and will be introduced in the following. This algorithm starts from a city model - here the model of the city centre of Munich [160] - identifies the individual buildings in this model and replaces each of them by that product model out of a candidate list which is most similar to the LoD1 description of the building. Replacing in this setting means that the coarse representation of the building is removed from the city model and the product model identified is processed instead as an additional individual model.

By doing so, the spatial relation of the individual buildings, which is given by the city model but lacks the details of the individual buildings, is fused with the highly detailed information of the building models.

Unfortunately, it is clear that product model descriptions of the buildings are not available for all buildings in the city centre of Munich or any other city model. Therefore, for buildings whose product models are not available, the algorithm chooses a replacement from a sufficiently large candidate set of synthesised product models.

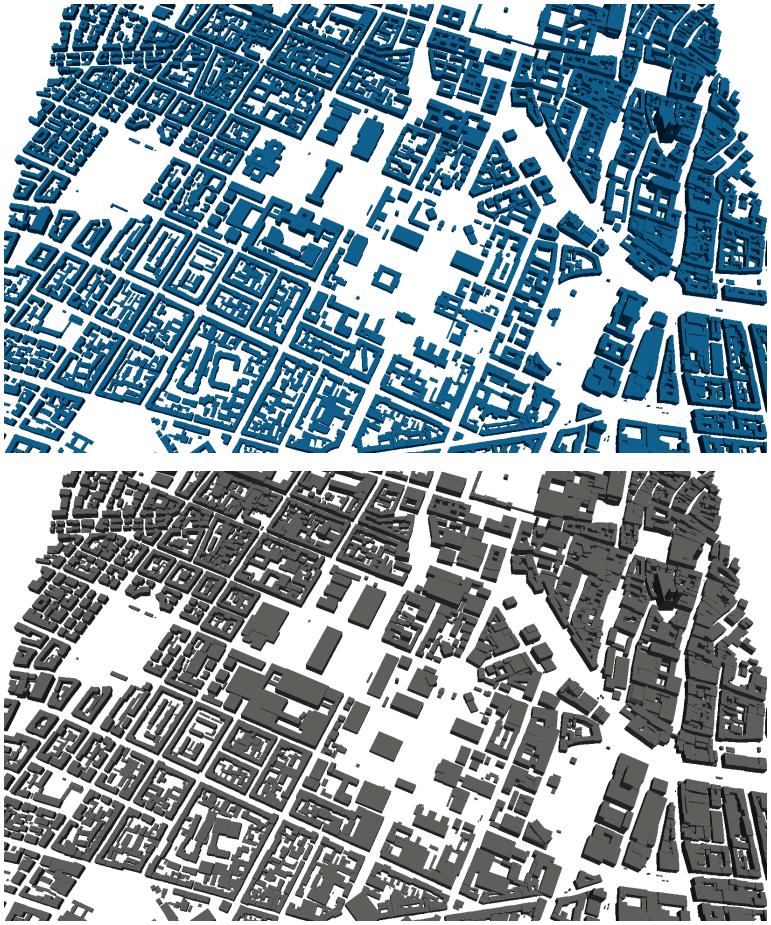

Figure A.1: The city model of city centre Munich spans an area of 2.8km by 2.3km. For the single polygonal building descriptions (top) the aligned bounding boxes are calculated (bottom).

The algorithm is given in Alg. 23 and works as follows: For every polygonal description of a building in the city model the aligned bounding box is calculated. On the basis of the aligned bounding box, see Fig. A.1, the most similar product model out of the candidate list is chosen. If the product model is similar enough to the description derived from the city model, it is replaced. Eventually a data set of the city centre of Munich with fully detailed construction descriptions is achieved.

Many algorithms are known for calculating the aligned bounding box [162, 163]. In the approach presented, the polygons out of which the geometry is defined are collected and their vertices are stored in a set. The bounding box is derived by rotating all vertices of the polygon for which it is computed around the z- axis with the angles $\alpha \in \left[0..\frac{\pi}{2}\right]$. The axis-aligned bounding box is calculated for each of the rotated sets of points. The angle $\tilde{\alpha}$ for which the volume of the bounding box is minimal identifies the angle of the non-axis aligned bounding box. Owing to symmetry it is sufficient to restrict this to the angles between in $\left[0..\frac{\pi}{2}\right]$.

Algorithm 23 augmentPolygonWithBuilding

```
void augmentPolygonWithBuilding(Object polygon,
        Object buildingCandidates[]){

 float rotationAngle;
 float projected2dArea;
 for (float angle = 0; angle<pi/2; angle+= .1){
  float tmpArea = polygon.rotateZ(angle).2DprojectionZ().area();
  if (tmpArea < projected2dArea){
   rotationAngle = angle;
   polygon.2dArea = tmpArea;
  }
 }

 Object replaceBuilding;
 float similarityMatch;
 for (each e in buildingCandidates){
  if (similarity(polygon.2dArea,polygon, e) > similarityMatch){
   replaceBuilding = e;
   similarityMatch = similarity(polygon, e);
  }
 }
 if (similarityMatch.isGood())
  polygon.augmentWith(replaceBuilding);
}
```

A decision now has to be made for every building as to which product model it is replaced with. This is done by checking the similarity of the polygonal description with every product model of the candidate list. Some basic requirements of this similarity criterion are as follows: If the exact product model for a building is in the candidate list, it should be chosen. The closer the length and width of the polygonal description and the product model are to each other, the more similar they are. As it is clear that an optimal replacement is not available for every building, the following assumptions are made in order to obtain a high replacement rate and therefore a complex assembly of the models. A product model can be scaled up to a certain factor $\pm\beta\%$ in order to fit the polygonal description. This scaling can also be done unevenly along the three axes even though this leads to slightly distorted geometries. If a building is already used as a replacement for many buildings, another not that similar building should be chosen which is not used that often. Using these criteria, where it is admitted that there are many reasonable additional criteria conceivable, the following similarity measure of an aligned bounding box and a product model is defined:

$$sim(poly, bim) = \big((poly.width - bim.width)^2 + (poly.width - bim.width)^2 + f(bim.count)\big)$$

Fig. A.2 depicts the polygonal description of a building, its aligned bounding box and the fully detailed product model chosen.

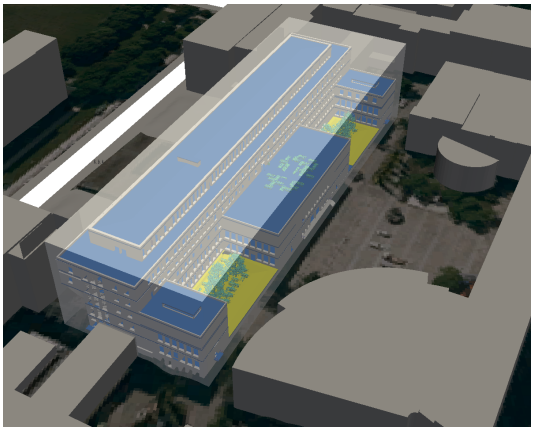

Figure A.2: Based on the polygonal description of a building, the aligned bounding box is derived and replaced by the product model which is most similar.

It should be mentioned that having a sufficiently large set of product models available is not self evident. The Chair for Computation in Engineering and the Chair of Computational Modelling and Simulation at Technische Universität München conduct research projects in order to derive high quality product model definitions. In Finland and Singapore, for example, it is required to provide the IFC description of public buildings.

Eventually a highly detailed urban data set is achieved. It is fused with the definition of the pipe network of the sewer system in the city and results in the test data set given in Fig. A.3.

With this data set at hand, the main part of this work can be introduced in the following: Handling large sets of complex data. The evaluation and investigation of the correctness and feasibility of the proposed concepts and algorithms would not have been possible without this work.

A.2 CiE Sandstorm Cluster

http://www.cie.bgu.tum.de

The Sandstorm Cluster is a small 4-node cluster installed in August 2012 at the Chair for Computation in Engineering at TUM. Each node contains two Intel Xeon E5-2690 CPUs running at 2.9GHz and 192GB RAM. The network interconnect is based on a one-to-one 1 Gigabit Ethernet connection from every node to every node and a 1 Gigabit connection to an external switch. The sustained Linpack performance http://www.top500.org/project/linpack of the system is 1.3 TFLOPS. Hyper-threading capabilities have been deactivated.

Figure A.3: For Munich city centre, a city model data set is created which fuses the specification of the terrain with its texture, the building assembly and the underground water drainage system.

A.3 Shaheen

A.3.1 KAUST Blue Gene/P 'Shaheen'

http://www.hpc.kaust.edu.sa

Shaheen is a 16 rack IBM Blue Gene/P supercomputer, each node is equipped with four 32-bit, 850 MHz PowerPC-450 cores and 4GB DDR memory. On aggregate, Shaheen has 65,536 processing cores and 64TB of memory. Shaheen provides a three-dimensional point-to-point Blue Gene/P torus network for general-purpose IPC. Each torus link can transmit up to 425 MBps in each direction, for a total of 5.1GBps bidirectional bandwidth per node.

A.4 OpenFOAM Mesh Specification

A.4.1 points

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    location    "constant/polyMesh";
    object      points;
}

12
(
    (-300 -220 100)
    (-100 -220 100)
    (-100 -20 100)
    (-300 -20 100)
    (-300 -220 300)
    (-100 -220 300)
    (-100 -20 300)
    (-300 -20 300)
    (100 -220 100)
    (100 -20 100)
    (100 -220 300)
    (100 -20 300)
)
```

A.4.2 faces

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       faceList;
    location    "constant/polyMesh";
    object      faces;
}

11
(
            4(1 2 6 5)
            4(0 4 7 3)
```

```
        4(0 1 5 4)
        4(1 8 10 5)
        4(0 3 2 1)
        4(1 2 9 8)
        4(8 9 11 10)
        4(3 7 6 2)
        4(2 6 11 9)
        4(4 5 6 7)
        4(5 10 11 6)
)
```

A.4.3 boundary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      polyBoundaryMesh;
}

6
(
    leftWall
    {
        type            wall;
        nFaces          1;
        startFace       1;
    }
    frontWall
    {
        type            wall;
        nFaces          2;
        startFace       2;
    }
    lowerWall
    {
        type            wall;
        nFaces          2;
        startFace       4;
    }
    rightWall
    {
        type            wall;
```

```
        nFaces          1;
        startFace       6;
    }
    backWall
    {
        type            wall;
        nFaces          2;
        startFace       7;
    }
    atmosphere
    {
        type            patch;
        nFaces          2;
        startFace       9;
    }
)
```

A.4.4 owner

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       labelList;
    note        "nPoints: 12 nCells: 2 nFaces: 11 nInternalFaces: 1";
    location    "constant/polyMesh";
    object      owner;
}

11
(
0
0
0
1
0
1
1
0
1
0
1
)
```

A.4.5 neighbour

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       labelList;
    note        "nPoints: 12 nCells: 2 nFaces: 11 nInternalFaces: 1";
    location    "constant/polyMesh";
    object      neighbour;
}

1
(
1
)
```

A.5 OpenFOAM Case Definition

A.5.1 system/controlDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime 1;

deltaT          .01;

writeControl    adjustableRunTime;

writeInterval     .2;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;
```

```
adjustTimeStep   yes;

maxCo            0.5;
maxAlphaCo       0.5;

maxDeltaT        1;


// ************************************************************************* //
```

A.5.2 system/decomposeParDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 1; //2 4 8 16 32 64

method          simple;

simpleCoeffs
{
    n               (1 1 1); //2 4 8 16 32 64
    delta           0.001;
}

hierarchicalCoeffs
{
    n               ( 1 1 1 );
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;
```

```
roots           ( );


// ************************************************************************* //
```

A.5.3 system/fvSchemes

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default          Euler;
}

gradSchemes
{
    default          Gauss linear;
}

divSchemes
{
    div(rho*phi,U)  Gauss limitedLinearV 1;
    div(phi,alpha)  Gauss vanLeer;
    div(phirb,alpha) Gauss interfaceCompression;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
}
```

```
snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p_rgh;
    pcorr;
    alpha1;
}


// ************************************************************************* //
```

A.5.4 system/fvSolution

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-10;
        relTol          0;
    }

    p_rgh
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-07;
        relTol          0.05;
    }
```

```
    p_rghFinal
    {
        \$p_rgh;
        tolerance       1e-07;
        relTol          0;
    }

    U
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     3;
    nNonOrthogonalCorrectors 0;
    nAlphaCorr      1;
    nAlphaSubCycles 2;
    cAlpha          1;
}


// ************************************************************************* //
```

A.5.5 setFieldsDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha1 0
```

```
);

regions
(
    boxToCell
    {
        box (-1400 -900 -45) (-800  1100 -9);
        fieldValues
        (
            volScalarFieldValue alpha1 1
        );
    }
);


// ************************************************************************* //
```

List of Figures

2.1 IFC product model of the main building of TUM [16]. 7

2.2 CityGML model of Berlin, Germany [22]. 9

2.3 Realistic terrain representation is achieved by mapping the conforming texture to a surface definition given on a (uniform) grid. 10

2.4 The pipe network of the sewer system in the city centre of Munich, Germany for a domain of approximately 2 square kilometres. 11

2.5 Two raster data sets describing a surface (top) and the corresponding texture (bottom), both with a stepwise down sampling for $k = [1, 200, 800]$ (f.l.t.r.). The dataset is provided by the Chair for Geoinformatics at Technische Universität München. 13

2.6 A discretised product model representation of an IFC file stores the list of indexed faces, which give the fully detailed geometry of the model. For every face, the identifier gives the mapping to the corresponding building element. Every building element is stored with the full set of its auxiliary information. 14

2.7 The fully detailed geometric representation of a building product model is coarsened by the set of bounding boxes for each individual building element. All bounding boxes of the same type of element are grouped and approximated by their bounding boxes. The bounding box of all groups of element types gives the bounding box of the product model (f.t.t.b). 16

2.8 After identifying the outer shell of a fully detailed product model, the complement of the shell with respect to the product model gives the inner structure of the construction. 17

2.9 The indexed mesh of a product model is rendered with colour-encoded identifiers of the triangles. The frame buffer of the scene rendered contains the colour value of the visible triangle for every pixel and therefore the information of all visible triangles for that view. 18

2.10 The rotation of the product model is discretised over the two rotation angles. As for every single rotation step the visible triangles are identified and added to the triangles already found, the number of triangles increases. 19

2.11 On a fully detailed BIM model (left) the outer shell is generated using the GPU-based algorithm for identifying the visible triangles forming the outside. In order to gain an insight into the reduction of about 90% from 84578 to 6686 triangles, the wire frame representation is given (right). 20

2.12 In order to derive a textured representation of a product model, the complete model is approximated by a simplified geometric representation such as its bounding box or the convex hull. 21

2.13 The bounding box of a product model identifies the - in this case six - projection planes for the generation of the textures. The projection of the view along the normal of the face is computed for every face and all textures are combined to the representation of the model. 21

3.1 The octree representation of a sphere is calculated for a maximal depth $d_{\max} = 5$, the increasing accuracy is adaptive to the boundary. 31

3.2 The Morton Code of a quadtree, the two-dimensional version of an octree, uses the bisecting nature of the data structure and the order of the child nodes to directly identify the address of an octant. 31

3.3 Based on an independent dual layer hierarchical data structure, the domain of the initial second level octree in general does not fit the discretisation of the first level tree (left). By increasing the domain of the second layer octree to a multiple of the width of the deepest level of the first level octree, a matching intersection is achieved (right). 34

3.4 For a product model consisting of 84k triangles an octree representation is generated until a depth of 5 (left). Besides the resolution of the geometric description, auxiliary information is also mapped to the individual elements (right). 34

3.5 By computing an octree based on the bounding boxes of all constructions and built infrastructures, the embedding of the data into the surrounding is structured and the individual models can be related to each other. 37

3.6 The generation of a quad/octree based on the bounding boxes and storage information of all processed models can be computed quickly and efficiently, but still contains the link to the fully detailed information. 37

3.7 Based on a given mapping of m product models to n processors, the colour indicates the distribution of the models over the processors. All models of a processor are depicted with the same colour. 39

3.8 The parallelisation of the framework is achieved by a distributed memory approach on the global scale. One master process contains the assembly of the data basis given by the first layer octree and the product models are distributed among the remaining $n-1$ processes. Shared memory parallelisation is then performed over the local product model sets of every single process. 40

3.9 The view point and direction of the user define a ray which intersects the requested construction detail and the product model. An intersection test on the first layer octree in the master process identifies the product models intersected, and the test on the product models in the slave processes affected identifies the construction detail. 47

3.10 The linearisation of an octree is achieved by concatenating the values of the octants in a depth-first manner to an array with the 0/1 information if an octant is a leaf node and followed by the value of the leaf node. 49

3.11 A product model on the local building scale is embedded in its voxelisation with a discretisation width of $150 \times 100 \times 100$ voxels. For better visibility, only grey leaf nodes, i.e. voxels, are shown in wire frame mode. 50

3.12 Following a probabilistic mapping, over 7000 product models are distributed to 3 slave processes over the domain. The colour value identifies the slave process to which a product model is assigned. 57

3.13 The Lebesgue curve orders multi-dimensional data points into a one-dimensional sequence. The order is achieved by recursively sorting points following (in 2D) the pattern top-left→top-right→bottom-left→bottom-right. 57

3.14 The quadtree representation of the product models is generated on the basis of the centre points of their bounding boxes projected along the z-axis. 58

3.15 The quadtree representation of the product models' centre points is ordered following the pattern of the Lebesgue curve. 59

3.16 The quadtree representation of the product models is ordered following the pattern of the Lebesgue curve in the classical sequence-based approach (left) and the modified round-robin approach (right). 59

3.17 The test scenario is depicted together with the application distances for the different LoDs applied. The colour encodes the processor to which a building is mapped. 60

3.18 The distribution of the highly detailed product models is given as the summed up number of triangles to be processed by every process for the classic Lebesgue curve mapping and the round-robin order, both for the scenario given in Fig. 3.17. The modified round-robin order also distributes the data locally and therefore achieves an almost equal load on the slave processes. 61

4.1 Based on an octree representation for a product model the identification of the visible triangles can be transferred to identifying the leaf octants intersected by the view frustum. 67

4.2 The parallel framework is extended by one further process which performs the visualisation. This process asynchronously triggers the adaption of the LoD representation by sending the updated position of the camera to the master processes. The slave processes affected perform the changes in geometry and the master sends the update to the visualisation process. 68

4.3 The computational load of rendering the outer shells of product model data on the GPU is reduced by limiting the visualisation to product models whose bounding boxes intersect the view frustum. 69

4.4 Different levels of detail are visualised for one specific building. With increasing distance, the level of detail gives a coarser representation of the product model. 69

4.5 Different levels of detail are visualised for the embedding of a fully detailed product model to a coarse polygonal description of its surrounding. The visualisation is given on the city-wide scale (left) and on the product model scale (right). 69

4.6 For every voxel, the MC algorithm identifies the intersection status of the isosurface along its eight edges. The resulting 256 different cases can be reduced due to symmetry to 15 triangle templates and are available as a lookup table. 71

4.7 For a given flow field the MC algorithm calculates the triangular representation with colour-encoded velocity of the isosurface to a given threshold value (top). In order to parallelise MC, the flow field is split into n - here 4 - equal parts (bottom) and the triangulations of the sub domains are combined to the isosurface of the complete flow field. The colour identifies the processor mapping. 71

4.8 The six-sided Corena CAVE@KAUST is driven by four projectors for every side, two for the vertical tiling and two for the stereoscopic immersion. . . . 72

4.9 Three linearly independent points in every projection plane identify the topology of a multi-monitor installation. 73

4.10 The tiled wall installation Z2@KAUST consists of 40 tiled monitors on a 10×4 grid. The rendering on the whole installation is split into parallel rendering of one process for every single monitor. 73

4.11 The parallel set-up of a framework in a multi-monitor installation consists of additional n visualisation processes, one for each projection plane, where one process additionally sends the camera position updates to the master process. 74

4.12 On the Cornea@KAUST the visualisation of simulation data is performed by following the parallel visualisation on 24 digital projectors and 6 sides, here with 5 sides due to the opened door of the CAVE. 75

4.13 On the NEXcave@KAUST the visualisation of fully detailed product model data on a local scale is performed by rendering the outer shell of the constructions on 21 monitors. 75

4.14 The gravity sensor of a mobile device is used to determine the desired direction, turning angle and speed of navigating through the data. 76

4.15 The communication protocol between the handheld device and the framework transfers the serialised request to the framework. Besides the navigation parameters, the checksum for evaluating the correct transfer of the broadcast and the counter for evaluating the order of the requests are also transferred. . 77

4.16 Handheld devices provide the user with a navigation interface which lets them focus on exploring the data set instead of handling the input device. 78

4.17 In a tracked immersive installation the marker's position and direction is derived by recording the interior of the installation with multiple cameras for different viewports. From the differing images of the reflecting parts of the marker the position and direction of the marker and therefore the part it is attached to can be derived using image processing and linear calculus. 78

4.18 The position and direction of the user's view is derived from the tracked marker mounted on their head. From this information the viewport and thus the projection onto the different planes of the CAVE are derived, here for the NEXcave@KAUST. 79

4.19 A tracked pointing device replaces the impractical mouse interaction in immersive installations and lets the user freely point at the visualised items, here shown in Cornea@KAUST. 80

4.20 The navigation of the whole data set lets the user explore on a city-wide scale (left). By navigating to the construction details of a product model the full information depth such as the material parameters of a door installation is achieved (right). 80

4.21 The evaluation of possible intersections of a sewer network with existing constructions in the city is performed by deriving the hierarchical representation of both configurations. 82

4.22 In order to determine the intersection of non-white parts of the two octrees their linearisation is concatenated level-wise with the logical *AND* operator. . 83

5.1 In radial dam break scenarios for the two-dimensional SWE a reservoir of fluid at rest is initially applied by the boundary conditions. The initial resolution of the triangular mesh is 16×16 triangles, the resolution depth is limited to 6 levels. The solution at the time-step iteration is given together with the refined mesh and the colour-encoded height. 87

5.2 The three-dimensional voxelisation of the data basis can be derived on the global scale (top) and on the local scale for a single building (bottom). The height is encoded in the colour. 88

5.3 From the three-dimensional voxelisation of the computational domain the two-dimensional bathymetry mesh is derived on the local scale for a single construction. The height value at every data point is derived as the maximal solid value at that point in the plane. 89

5.4 From the thee-dimensional voxelisation of the computational domain the two-dimensional bathymetry mesh is derived on the local scale for a single construction. The height value at every data point is derived as the value below the first fluid voxel at that point in the plane. 90

5.5 In order to access the voxel query interface, Peano is started in an additional MPI process, together with the framework's master process and the $n-1$ slave process holding the product models. 93

5.6 The Peano interface identifies the computational domain of a sphere as follows. The white octants give the outside of the domain where *isCompletelyOutsideNotInverted* returns *true*, the dark blue octants form the inner part and here *isCompletelyInsideNotInverted* returns *true*. The light blue depicted octants cover the boundary of the domain and there both methods *isCompletelyOutsideNotInverted* and *isCompletelyInsideNotInverted* return *false*. This triggers further refinement of the interface. 94

5.7 The fluid mesh is built up by the Peano framework and adaptively refined at the boundary of the domain. The inside/outside check for building up the grid is performed over the dual layer hierarchical data structure of the work presented. 96

5.8 For the original domain (top) the voxelisation is derived. Fluid voxels are depicted white, solid voxels are blue; this discretisation is calculated in two ways. Without performing the fill algorithm (middle), fluid parts which are not connected to the seed region are also identified as computational domain. These parts are removed by performing the fill algorithm (bottom). 99

5.9 An OpenFOAM mesh of two hexahedral elements consists of 11 faces, 20 edges and 12 vertices. 100

5.10 The two sub-domains of a computational domain on the local building scale are coloured according to their processor location. The *processor* faces are highlighted and interface the communication between the processors. 103

5.11 The decomposition of a $1000 \times 1000 \times 50$ global-scale domain is given for 4 processors following a block approach. The mapping of the different sub-domains to the processors is encoded in the colour. 104

5.12 The decomposition of a $1000 \times 1000 \times 50$ global-scale domain is given for 8 processors following the *Scotch* approach. The mapping of the different sub-domains to the processors is encoded in the colour. 104

6.1 The potential flow through a building's product model is discretised over $450 \times 200 \times 275$ voxels. The boundary conditions applied describe an inflow of constant velocity on the left and a free outflow on the right side of the domain. 107

6.2 The flow of the Dornbirner Ache in Austria is simulated using the Shallow-Water-Equations. The boundary conditions applied imply a constant inflow of the river on the left-hand side and a free outflow on the right-hand side. The extruded surface of the fluid is visualised by colour encoding the velocity of the flow. 108

6.3 The three-dimensional Navier-Stokes equations describe the flow of Newtonian fluids. The flow around a building is simulated for a computational domain of $150 \times 100 \times 100$ voxels. The boundary conditions applied imply a constant velocity of 1m/s on the left side and free outflow on the right. 108

6.4 The simulation of urban flooding is performed as a three-dimensional free surface simulation. The surface of the fluid is visualised with colour encoded velocities. The computational domain is discretised with a grid of $1000 \times 1000 \times 50$ cells. The boundary conditions applied describe a free outflow on the x/y normal boundaries, the amount of water is imposed by initial conditions for the fluid at rest. 110

6.5 Starting at time step t_i, coarse grid computations are performed until t_{i+1}, followed by a projection from the global to the local domain (vertical arrow). The coarse grid simulation is stopped at t_{i+1}. 112

6.6 A two-phase simulation is performed on the global domain (bottom) and on the refined subset of the local domain (top). The velocity is colour encoded and the interface is visualised as the contour of the phase field γ. The mesh generation based on the representation of the data basis with octrees of different depth achieves conforming meshes over the different scales. This conformity enables the direct propagation of simulation results to the matching cells of a refined simulation. 112

6.7 The two dimensional validation area is a slice through the computational domain. The global domain is coloured in light blue and the local domain is coloured in dark blue. 113

6.8 At time step $t = 10$ seconds, the simulation results achieved at the global domain are projected as initial conditions to the local domain. 114

6.9 The simulation on the global domain (top) is compared to the results on the local domain incorporating reflecting, potential and transient (f.t.t.b) boundary conditions. For the given results here at time step $t = 12.5$ seconds, a visual comparison shows promising results for the evolution of the water front but shows clearly the influence of the reflecting and potential boundary conditions at the rear part of the wave. 115

6.10 Time step $t = 15$ seconds still shows promising results for capturing the characteristics of the wave front. The effects of the reflecting and potential boundary conditions are obvious by a visual inspection at the rear part of the wave. . . 115

6.11 The difference $d(x, t)$ of the water height, relative to the height of the computational domain, over the x-axis of the local domain is given for time steps $t = 12.5$ (top) and $t = 15$ (bottom) seconds. It shows the propagation of the errors induced by the reflecting boundary conditions. 116

6.12 The computational mesh of the fluid flow simulation can be mapped to the construction details of the product model data. The boundary faces of the fluid mesh are shared with the octree representation of the construction detail. This mapping gives the linkage between simulation results and construction details affected. 118

6.13 The measurement of the strong speedup for a computational domain of $1000 \times 1000 \times 50$ voxels for the *interFoam* OpenFOAM solver is carried out over 1 to 64 processors. 119

7.1 The amount of water is applied by specifying the initial conditions as the respective amount of water at rest. 125

7.2 The flooding of the detailed city model is visualised with the contour of free surface simulation and colour encoded velocities. 126

7.3 A three-dimensional benchmark of two basins connected with a pipe implies that resolving the pipe with a single voxel in the orthogonal directions of the flow imposes the characteristic of one-dimensional flow. 128

7.4 The pressure line over the axis along the flow in the direction of the pipe shows that the one voxel wide discretisation imposes one-dimensional pipe flow in the three-dimensional domain. 129

7.5 The velocities orthogonal to the flow in the pipe are depicted but neglected due to the one-dimensional interpretation. 129

7.6 The computational domain is depicted by its 0.5 contour for the global scale (top) and for the local building scale (bottom). The relative height of the surface is colour encoded. 132

7.7 The collapsing drainage water system is modelled by placing a constant hydrostatic pressure on the inlet of the network. The constant pressure is imposed by placing a sufficiently large reservoir on the inlet. As an initial condition, the pipe network is already flooded. 132

7.8 The results of the pipe network and the surface flow are visualised without the constructions for better visibility. 134

7.9 The surface flow is depicted with visualised constructions in order to show the flooding of the city due to the collapsing water drainage system. 135

7.10 The first adapted resolution refinement gives the flow behaviour on the local building-scale simulation. 136

7.11 The second adapted resolution refinement projects the local building-scale flow behaviour on the construction detail scale as the initial conditions of the simulation. 137

8.1 The load time of the computational mesh and the run time of the numerical simulation as function of the number of processors on the Shaheen@KAUST supercomputer shows the demand for advanced grid generation and load balancing techniques. 140

8.2 This work is an interdisciplinary approach which focuses on integrating efficient techniques from the fields of civil engineering, hydromechanics, visualisation and scientific computing. 142

A.1 The city model of city centre Munich spans an area of 2.8km by 2.3km. For the single polygonal building descriptions (top) the aligned bounding boxes are calculated (bottom). 144

A.2 Based on the polygonal description of a building, the aligned bounding box is derived and replaced by the product model which is most similar. 146

A.3 For Munich city centre, a city model data set is created which fuses the specification of the terrain with its texture, the building assembly and the underground water drainage system. 147

List of Tables

2.1 By performing the stepwise level of detail coarsification presented, the number of triangles to be processed is reduced iteratively. 17

2.2 By performing outer shell calculation of fully detailed product models, a reduction of up to 90%can be achieved. 20

3.1 Basic MPI commands are needed in order to perform the communication between processors by exchanging data via MPI. 41

3.2 Depending on the distance of the product model to the investigation point the appropriate level of detail is applied and more distant constructions are discarded. 60

5.1 The Peano geometry interface queries the status of the spacetree nodes during the build up and the traversal of the grid. 92

Bibliography

[1] G. Lee, R. Sacks, and C. M. Eastman. Specifying parametric building object behavior (BOB) for a building information modeling system. *Automation in Construction*, 15:19, 2006.

[2] C. M. Eastman. BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors. *Wiley*, 2008.

[3] R. Howard and B.-C. Björk. Building information modelling - experts' views on standardisation and industry deployment. *Advanced Engineering Informatics*, pages 271–280, 2008.

[4] C. McGraw-Hill. Building information modeling (BIM) - transforming design and construction to achieve greater industry productivity. *SmartMarket Report*, page 48, 2008.

[5] C. McGraw-Hill. The business value of BIM - getting building information modeling to the bottom line. *SmartMarket Report*, page 52, 2009.

[6] T. M. Froese. The impact of emerging information technology on project management for construction. *Automation in Construction*, 19:531–538, 2010.

[7] C. McGraw-Hill. The business value of BIM in europe. *SmartMarket Report*, page 52, 2010.

[8] M. P. Gallaher and A. C. O'Connor. Cost analysis of inadequate interoperability in the U.S. capital facilities industry. *Proceedings of the CIB W78 2012: 29th International Conference, Beirut, Lebanon, 17-19 October*, 2004.

[9] The American Institute of Architects. Integrated project delivery: A guide. *The American Institute of Architects*, 1:62, 2007.

[10] Y. Ji, A. Borrmann, J. Beetz, and M. Obergriesser. Exchange of parametric bridge models using a neutral data format. *Journal of Computing in Civil Engineering*, 27:593–606, 2013.

[11] M. Obergriesser, T. Euringer, A. Borrmann, and E. Rank. Integration of geotechnical design and analysis processes using a parametric and 3D-model based approach. *Proc. of*

the ASCE International Workshop on Computing in Civil Engineering, Miami, Florida, USA, 2011, page 8, 2011.

[12] A. Borrmann and K. Heine. Zukunft und Vergangenheit – 4D-Modellierung als Werkzeug für die Bauplanung und die baugeschichtliche Forschung. *Tagungsband Hand-High III - Von Handaufmass bis High Tech, Cottbus, Germany, 2010*, page 8, 2010.

[13] S. Daum and A. Borrmann. Definition and implementation of temporal operators for a 4D query language. *Proc. of the ASCE International Workshop on Computing in Civil Engineering, Los Angeles, CA, USA, 2013*, page 8, 2013.

[14] BuildingSMART. *http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release*, Last accessed: 2014-01-22.

[15] *STEP Application Handbook: ISO 10303*. SCRA, 2006.

[16] P. Steger. Erstellen eines 3D-Modells mit ArchiCAD. *Bachelorthesis - Chair for Computation in Engineering (TUM)*, 2009.

[17] A. Monteiro and J.P. Martins. BIM modeling for contractors- improving model takeoffs. *Proc. of the CIB W078 29th International Conference On Applications Of IT In The AEC Industry*, page 10, 2012.

[18] A. Monteiro and J. P. Martins. A survey on modeling guidelines for quantity takeoff-oriented BIM-based design. *Automation in Construction*, 35:238–253, 2013.

[19] A. Borrmann, T.H. Kolbe, A. Donaubauer, H. Steuer, and J.R. Jubierre. Transferring multi-scale approaches from 3D city modeling to IFC-based tunnel modeling. *Proc. of the 3DGeoInfo*, 2013.

[20] Y. Ji, A. Borrmann, J. Beetz, and M. Obergrießer. Exchange of parametric bridge models using a neutral data format. *ASCE Journal of Computing in Civil Engineering*, Exchange of Parametric Bridge Models using a Neutral Data Format:DOI: 10.1061/(ASCE)CP.1943–5487.0000286, 2013.

[21] T. Pazlar and Z. Turk. Interoperability in practice: geometric data exchance using the IFC standard. *ITcon – Special Issue Case studies of BIM use*, 13:362–380, 2008.

[22] Berlin business location center. *http://www.businesslocationcenter.de/de/wirtschaftsatlas-berlin*, Last accessed: 2014-01-22.

[23] M. Kada. The 3D Berlin project. *Photogrammetric Week 2009*, pages 331–340, 2009.

[24] Ordnance survey. *http://www.ordnancesurvey.co.uk*, Last accessed: 2014-04-03.

[25] T. H. Kolbe, G. Gröger, and L. Plümer. CityGML – interoperable access to 3D city models. *In Proceedings of the First International Symposium on Geo- Information for Disaster Management (Delft, Netherlands, March 21-23, 2005)*, page 16, 2005.

[26] T.H. Kolbe. CityGML–3D geospatial and semantic modelling of urban structures. *Presentation on the GITA/OGC Emerging Technologies Summit in Washington on*, page 38, 2007.

[27] G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele. OGC City Geography Markup Language (CityGML) encoding standard, v2.0. *http://www.opengeospatial.org/standards/citygml*, Last acccessed: 2014-01-22.

[28] OGC Keyhole Markup Language. *http://www.opengeospatial.org/standards/kml/*, Last acccessed: 2014-01-22, 2011.

[29] CityGML virtual 3D city models. *http://www.citygml.org*, Last accessed: 2014-01-22.

[30] B. Mao. *Visualisation and Generalisation of 3D City Models*. PhD thesis, Royal Institute of Technology, 2011.

[31] E. Sadek, A. Jamaludin, B. Rosdi, and M. Kadzim. The design and development of a virtual 3D city model. *http://www.hitl.washington.edu/people/bdc/virtualcities.pdf*, Last accessed: 2014-01-22.

[32] Y.Ban. ViSuCity-a visual sustainable city planning tool. *Report, URI:urn:nbn:se:kth:diva-89533*, 2008.

[33] L. Tang, C. Chen, H. Huang, and K. Lin. Research on HLA-based forest fire fighting simulation. *Proc. of the 9th AGILE Conference on Geographic Information Science*, 2006.

[34] T. H. Kolbe. CityGML tutorial. *Institute for Geodesy and Geoinformation Science, Technische Universität Berlin*, page 103, 2007.

[35] P. Bolstad. *GIS Fundamentals: A First Text on Geographic Information System*. 2005.

[36] A. K. W. Yeung and C. P. Lo. Concepts and techniques of geographic information systems. *Upper Saddle River, NJ*, 2002.

[37] N. Schuurman. *A short introduction*. Malden, 2004.

[38] Das Land Voralberg im Internet. *http://www.vorarlberg.at/*, Last accessed: 30.01.2014.

[39] Utah AGRC. *http://gis.utah.gov/*, Last accessed: 2014-01-22.

[40] R. de Laat and L. van Berlo. Integration of BIM and GIS: The development of the CityGML GeoBIM extension. In *Advances in 3D Geo-Information Sciences*, pages 211–225. Springer, 2011.

[41] H. Steuer, A. Donaubauer, T.H. Kolbe, M. Flurl, R.-P. Mundani, and E. Rank. Planning inner-city-railway-tracks: Dynamic integration of geospatial web services in a collaborative multi-scale geometry modelling environment. *Proc. of the Workshop on Intelligent Computing in Civil Engineering (EG-ICE)*, 2013.

[42] B. Hagedorn, M. Trapp, T. Glander, and J. Dollner. Towards an indoor level-of-detail model for route visualization. In *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on*, pages 692–697. IEEE, 2009.

[43] C. Dick, J. Krüger, and R. Westermann. GPU-aware hybrid terrain rendering. pages 3–10, 2010.

[44] C. Dick, J. Krüger, and R. Westermann. GPU ray-casting for scalable terrain rendering. In *Proceedings of Eurographics 2009 - Areas Papers*, pages 43–50, 2009.

[45] TNO Building and Construction. IFC Engine. *http://www.ifcbrowser.com/*, Last accessed: 2014-01-22.

[46] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. *Rendering Techniques 2001, Springer*, pages 115–125, 2001.

[47] Wirtschaftsatlas Berlin. *www.3d-stadtmodell-berlin.de*, Last accessed: 2014-01-28.

[48] D. P. Luebke. Level of detail for 3D graphics. *Morgan Kaufmann*, 2003.

[49] Google Earth. *http://earth.google.de/*, Last accessed: 2014-03-14.

[50] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, , and D. Woodford. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems*, 31(3), 2013.

[51] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[52] M.-J. Kraak and F. Ormeling. Cartography: visualization of spatial data. *Guilford Press*, 2011.

[53] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, J. Dickinson, R.Thomas, A. Pardasani, and H. Xue. Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. *Advanced Engineering Informatics*, 24(2):196–207, 2010.

[54] B. Akinci, H. Karimi, A. Pradhan, C.-C. Wu, and G. Fichtl. CAD and GIS interoperability through semantic web services. *CAD and GIS Integration*, page 199, 2010.

[55] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *Cambridge: MIT press*, 2, 2001.

[56] G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. *Technical Report, Ottawa, Canada: IBM Ltd.*, 1966.

[57] H. Samet. Spatial data structures. *Addison-Wesley*, 1995.

[58] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[59] Leibniz-Rechenzentrum. *http://www.lrz.de/services/compute/museum/hlrb2/*, Last accessed: 2014-01-28, 2013.

[60] OpenMP. The OpenMP API specification for parallel programming. *http://openmp.org/*, Last accessed: 2014-01-28.

[61] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[62] J. Reinders. *Intel Threading Building Blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, Inc., 2010.

[63] A. Kukanov and M. J. Voss. The foundations for scalable multi-core software in Intel Threading Building Blocks. *Intel Technology Journal*, 11(4), 2007.

[64] The Message Passing Interface (MPI) standard. *http://www.mcs.anl.gov/research/projects/mpi/*, Last accessed: 2014-01-28.

[65] W. D. Gropp, E. L. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. the MIT Press, 1999.

[66] W. D. Gropp, E. L. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[67] Open MPI: Open source high performance computing. *http://www.open-mpi.org/*, Last accessed: 2014-01-28.

[68] MPICH. *http://www.mpich.org/*, Last accessed: 2014-01-28.

[69] A. Dieberger and A. U. Frank. A city metaphor to support navigation in complex information spaces. *Journal of Visual Languages & Computing*, 9(6):597–622, 1998.

[70] J. Paay and J. Kjeldskov. Understanding and modelling built environments for mobile guide interface design. *Behaviour & Information Technology*, 24(1):21–35, 2005.

[71] T. Tu, D. R. O'Hallaron, and O. Ghattas. Scalable parallel octree meshing for terascale applications. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, 2005.

[72] H. Sundar, R. S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008.

[73] C. Burstedde, L. C. Wilcox, and O. Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[74] R. J. Hitchcock and J. Wong. Transforming IFC architectural view BIMs for energy simulation. *Proceedings of Building Simulation*, 2011.

[75] S. Pissanetzky. *Sparse matrix technology*. Academic Press London, 1984.

[76] Y. Saad. *Numerical methods for large eigenvalue problems*, volume 158. SIAM, 1992.

[77] T. Weinzierl. A framework for parallel PDE solvers on multiscale adaptive cartesian grids. *Verlag Dr. Hut*, 2009.

[78] H. Sagan. *Space-filling curves*, volume 18. Springer-Verlag New York, 1994.

[79] R. Fraedrich, J. Schneider, and R. Westermann. Exploring the millennium run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1251–1258, 2009.

[80] D. Keim, G. Andrienko, F. Gennady, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. *Springer*, 2008.

[81] C. D. Hansen and C. R. Johnson. *The visualization handbook*. Elsevier, 2005.

[82] F. Reichl, M. Treib, and R. Westermann. Visualization of big SPH simulations via compressed octree grids. *Proceedings of IEEE Big Data*, 2013.

[83] OpenGL the industry's foundation for high performance graphics. *http://www.opengl.org/*, Last accessed: 2014-03-04.

[84] F. D. Luna. *Introduction to 3D game programming with DirectX 10*. Jones & Bartlett Publishers, 2008.

[85] D. H. Eberly. 3D game engine design: a practical approach to real-time computer graphics. *CRC Press*, 2006.

[86] E. Lengyel. Mathematics for 3d game programming and computer graphics. *Cengage Learning*, 2012.

[87] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[88] H.-J. Bungartz, M. Griebel, and C. Zenger. *Einführung in die Computergraphik: Grundlagen, Geometrische Modellierung, Algorithmen*. Vieweg+Teubner Verlag, 2002.

[89] J. D. Foley, A. Van Dam, and S. K. Feiner. Computer graphics: Principles and practice. 2009.

[90] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[91] T. A. DeFanti and M. D. Brown. Visualization in scientific computing. *Advances in Computers*, 33:247–305, 1991.

[92] G. M. Nielson, H. Hagen, and H. Müller. Scientific visualization. *Institute of Electrical & Electronics Engineers*, 1997.

[93] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142. ACM, 1993.

[94] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, 1992.

[95] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *Visualization and Computer Graphics, IEEE Transactions on*, 15(3):436–452, 2009.

[96] CORNEA. *http://kvl.kaust.edu.sa/Pages/CORNEA.aspx*, Last accessed: 2014-01-28.

[97] KAUST Visualization Core Lab. *http://kvl.kaust.edu.sa/Pages/Showcase.aspx*, Last accessed: 2014-01-28.

[98] F. Zhou, H. B.-L. Duh, and M. Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE Computer Society, 2008.

[99] D. Wagner, T. Pintaric, F. Ledermann, and D. Schmalstieg. Towards massively multi-user augmented reality on handheld devices. In *Pervasive Computing*, pages 208–219. Springer, 2005.

[100] J. Baus, A. Krüger, and W. Wahlster. A resource-adaptive mobile navigation system. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 15–22. ACM, 2002.

[101] D. W. F. Van Krevelen and R. Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2):1, 2010.

[102] iOS 7. *http://www.apple.com/ios/*, Last accessed: 2014-01-28.

[103] Android. *http://www.android.com/*, Last accessed: 2014-01-28.

[104] 802.11. *IEEE*.

[105] Network Working Group. *RFC 2616*.

[106] C. M. Kohlhoff. Boost::Asio: a cross-platform C++ library for network and low-level I/O programming. *Report*, 2010.

[107] S. Blackrnan and A. House. Design and analysis of modern tracking systems. *Boston, MA: Artech House*, 1999.

[108] ART. *http://www.ar-tracking.com*, Last accessed: 2014-03-04.

[109] A. Borrmann and E. Rank. Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics*, 23(1):32–44, 2009.

[110] A. Borrmann. From GIS to BIM and back again–a spatial query language for 3D building models and 3D city models. In *5th international 3D geoinfo conference, Berlin*, 2010.

[111] A. Borrmann, S. Schraufstetter, and E. Rank. Implementing metric operators of a spatial query language for 3D building models: octree and B-Rep approaches. *Journal of Computing in Civil Engineering*, 23(1):34–46, 2009.

[112] R.-P. Mundani, H.-J. Bungartz, E. Rank, R. Romberg, and A. Niggl. Efficient algorithms for octree-based geometric modelling. *In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing, Civil-Comp Press, 2003*, 2003.

[113] M. Schreiber, H.-J. Bungartz, and M. Bader. Shared memory parallelization of fully-adaptive simulations using a dynamic tree-split and -join approach. *Proc. of the IEEE International Conference on High Performance Computing (HiPC)*, 2012.

[114] M. Schreiber, T. Weinzierl, and H.-J. Bungartz. Cluster optimization and parallelization of simulations with dynamically adaptive grids. In F. Wolf, B. Mohr, and D. an Mey, editors, *Euro-Par 2013*, volume 8097 of *Lecture Notes in Computer Science*, pages 484–496, Berlin Heidelberg, 2013. Springer-Verlag.

[115] H. Jasak, A. Jemcov, and Z. Tukovic. OpenFOAM: A C++ library for complex physics simulations. In *International Workshop on Coupled Methods in Numerical Dynamics, IUC, Dubrovnik, Croatia*, pages 1–20, 2007.

[116] OpenFOAM. *http://www.openfoam.com/*, Last accessed: 2014-01-28.

[117] M. J. Berger, D. L. George, R. J. LeVeque, and K. T. Mandli. The GeoClaw software for depth-averaged flows with adaptive refinement. *Advances in Water Resources*, 34(9):1195–1206, 2011.

[118] V. Aizinger. A discontinuous Galerkin method for two-dimensional flow and transport in shallow water. *Advances in Water Resources*, 25:67–84, 2002.

[119] J.-F. Remacle, S. S. Frazo, X. Li, and M. S. Shephard. An adaptive discretization of shallow-water equations based on discontinuous Galerkin methods. *International Journal for Numerical Methods in Fluids*, 52(8), 2006.

[120] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[121] T. Weinzierl. Dynamic load balancing of spacetree grids. *Proc. of the International Conference on Parallel Computing - ParCo*, September 2013.

[122] T. Neckel. *The PDE framework Peano: an environment for efficient flow simulations.* Verlag Dr. Hut, 2009.

[123] R. Mittal and G. Iaccarino. Immersed boundary method. *Annual Review Fluid Mechanics*, 37:239–260, 2005.

[124] C. Peskin. The immersed boundary method. *Acta Numerica*, 11:1–39, 2002.

[125] R.-P. Mundani. Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben. *Shaker*, 2006.

[126] F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. *Proc. of the Scalable High-Performance Computing Conference (SHPCC), IEEE Press*, pages 486–493, 1994.

[127] M. Wang, Y. Tang, X. Guo, and X. Ren. Performance analysis of the graph-partitioning algorithms used in OpenFOAM. *Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference on*, pages 99–104, 2012.

[128] N. Selvakkumaran and G. Karypis. Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(3):504–517, 2006.

[129] potentialFoam. *http://openfoamwiki.net/index.php/PotentialFoam*, Last accessed: 2014-03-03.

[130] J. Virbulis, , and J. Seņņikovs. Transient modelling of groundwater dynamics in the Baltic Artesian Basin. *Groundwater in Sedimentary Basins*, page 15, 2012.

[131] C. Hirsch. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*, volume 1. Butterworth-Heinemann, 2007.

[132] Yuri N. Skiba and Denis M. Filatov. On splitting-based mass and total energy conserving arbitrary order shallow-water schemes. *Numerical methods for partial differential equations*, 1992.

[133] S. S. Deshpande, L. Anumolu, and M. F. Trujillo. Evaluating the performance of the two-phase flow solver interFoam. *Computational Science & Discovery*, 5(1):014016, 2012.

[134] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[135] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 8:2182, 1965.

[136] C. W. Hirt and B. D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.

[137] J. H. Ferziger and M. Perić. *Computational methods for fluid dynamics*, volume 3. Springer Berlin, 1996.

[138] M. Schäfer. Computational engineering – introduction to numerical methods. *Springer-Verlag*, 10.1007/3-540-30686-2, 2006.

[139] TEAMPLAN HOLDING AG.
Überflutungsnachweis mit gekoppeltem Oberflächenmodell.
http://www.fischer-teamplan.de/fachbereiche/abwasserableitung/generalentwaesserung/ueberflutungsnachweis-mit-gekoppeltem-oberflaechenmodell/, Last accessed: 2014-03-19, 2014.

[140] D. Balmforth and R. Benyon. Developing flood resilient communities. In *Proc. of the ICE Flooding 2013*, 2013.

[141] E. Mignot, A. Paquier, and S. Haider. Modeling floods in a dense urban area using 2D shallow water equations. *Journal of Hydrology*, 327(1):186–199, 2006.

[142] T. J. Fewtrell, P. D. Bates, M. Horritt, and N. M. Hunter. Evaluating the effect of scale in flood inundation modelling in urban environments. *Hydrological Processes*, 22(26):5107–5118, 2008.

[143] J. Chen, A. A. Hill, and L. D. Urbano. A GIS-based model for urban flood inundation. *Journal of Hydrology*, 373(1):184–192, 2009.

[144] N. M. Hunter, P. D. Bates, S. Neelz, G. Pender, I. Villanueva, N.G. Wright, D. Liang, R.A. Falconer, B. Lin, and S. Waller. Benchmarking 2D hydraulic models for urban flooding. *Proceedings of the ICE-Water Management*, 161(1):13–30, 2008.

[145] S. S. Shahapure, T. I. Eldho, and E.P. Rao. Coastal urban flood simulation using FEM, GIS and remote sensing. *Water resources management*, 24(13):3615–3640, 2010.

[146] P. D. Bates, M. S. Horritt, and T. J. Fewtrell. A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling. *Journal of Hydrology*, 387(1):33–45, 2010.

[147] Introduction to the special issue on urban hydrology. *Journal of Hydrology*, pages 163–165, 2004.

[148] T. G. Schmitt, M. Thomas, and N. Ettrich. Analysis and modeling of flooding in urban drainage systems. *Journal of Hydrology*, 299(3):300–311, 2004.

[149] R. S. Carr and G. P. Smith. Linking of 2D and pipe hydraulic models at fine spatial scales. In *7th International conference on urban drainage modelling and the 4th international conference on water sensitive urban design, Melbourne, Australia*, 2006.

[150] O. Mark, S. Weesakul, C. Apirumanekul, S. B. Aroonnet, and S. Djordjević. Potential and limitations of 1D modelling of urban flooding. *Journal of Hydrology*, 299(3):284–299, 2004.

[151] A. Bolle, A. Demuynck, R. Bouteligier, S. Bosch, A. Verwey, and J. Berlamont. Hydraulic modelling of the two-directional interaction between sewer and river systems. In *7th International Conference on Urban Drainage Modelling and the 4th International Conference on Water Sensitive Urban Design; Book of Proceedings*, page 896. Monash University, 2006.

[152] T. E. Barnard, A. W. Kuch, G. R. Thompson, S. Mudaliar, and B. C. Phillips. Evolution of an integrated 1D/2D modeling package for urban drainage. *Contemporary Modeling of Urban Water Systems, Monograph*, 15, 2007.

[153] J. Leandro, A. S. Chen, S. Djordjević, and D. A. Savić. Comparison of 1D/1D and 1D/2D coupled (sewer/surface) hydraulic models for urban flood simulation. *Journal of hydraulic engineering*, 135(6):495–504, 2009.

[154] G. L. Kouyi, D. Fraisse, N. Rivière, V. Guinot, and B. Chocat. 1D modelling of the interactions between heavy rainfall-runoff in urban area and flooding flows from sewer network and river. *Proc. of the 11th International Conference on Urban Drainage*, 2008.

[155] N. D. S. Domingo, A. Refsgaard, O. Mark, and B. Paludan. Flood analysis in mixed-urban areas reflecting interactions with the complete water cycle through coupled hydrologic–hydraulic modelling. *Water Science & Technology*, 62(6):1386–1392, 2010.

[156] M. Dumbser, M. Käser, and E. F. Toro. An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes–V. Local time stepping and p-adaptivity. *Geophysical Journal International*, 171(2):695–717, 2007.

[157] C. Fumeaux, D. Baumann, P. Leuchtmann, and R. Vahldieck. A generalized local time-step scheme for efficient FVTD simulations in strongly inhomogeneous meshes. *Microwave Theory and Techniques, IEEE Transactions on*, 52(3):1067–1076, 2004.

[158] A. J. Crossley, N. G. Wright, and C. D. Whitlow. Local time stepping for modeling open channel flows. *Journal of Hydraulic Engineering*, 129(6):455–462, 2003.

[159] S. Djordjević, A. J. Saul, G. R. Tabor, J. Blanksby, I. Galambos, I. Sabtu, G. Sailor, et al. Experimental and numerical investigation of interactions between above and below ground drainage systems. In *12th International Conference on Urban Drainage*, volume 1999, pages 10–15, 2011.

[160] Bayern. Bayerische Vermessungsverwaltung. *http://vermessung.bayern.de*, Last accessed: 2014-01-22.

[161] J. Stoter, G. Vosselman, J. Goos, S. Zlatanova, E. Verbree, and R. Klooster. Towards a national 3D spatial data infrastructure: case of The Netherlands. *Photogrammetrie-Fernerkundung-Geoinformation*, 6, 2011.

[162] S. Ding, M. A. Mannan, and A. N. Poo. Oriented bounding box and octree based global interference detection in 5-axis machining of free-form surfaces. *Computer-Aided Design*, 36(13):1281–1294, 2004.

[163] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996.