Distributed and Parallel Data Management to Support Geo-Scientific Simulation Implementations

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für
Bauingenieur-, Geo- und Umweltwissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION

von

Dipl.-Systemwiss. Markus Wilhelm Jahn

aus Erfurt (Thüringen)

Tag der mündlichen Prüfung: Dezember 15, 2021

Referent: Martin Breunig

Korreferenten: Mulhim Al Doori, Markus Ulrich

Karlsruhe

August 2022

Acknowledgement

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) with the projects "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) and "Topologically consistent modelling in a spatio-temporal context with moving objects in the city model" (project number: 326802322). The Thuringian state office for land management and geo-information (TLBG) is thanked for their public geo-data repository. Arivaldo Leão de Amorim and his working group are thanked for modelling and providing a city model of a historical part of the city Salvador, Brazil. Patrick Erik Bradley, Mulhim Al Doori and Martin Breunig are thanked for fruitful discussions and help.

This work is dedicated to my grandparents Margot and Wolfgang Beck for the support throughout my studies of applied system sciences and to my wife Julia and to my daughter Mathilda who have been so patient while I was absent-minded to draw up this work.

Abstract

Collaboration between scientists is science in action. It is a difficult task to find a common language within and between scientific domains. One solution is to use abstractions. General system science, as a scientific abstraction layer, is the science of modelling any phenomena, systems whose behaviours are analysed in order to get a deeper understanding of the phenomena of interest. Applied system scientists model, analyse and simulate real world phenomena which may intersect multiple scientific domains.

The goal of computational science is to support any scientific work by providing applicable computational technology together with data management and processing concepts. Expert tools tend to focus on specific scientific domains for economical, technical or some other reasons, on the cost of collaboration and abstraction cross-domain. This results in a challenging collaboration with and between experts of different scientific domains, which hinders the scientific workflow. The data mining for large scale simulations and the model generation is a complex process where scientists of different expertises need to work closely together in a cooperative process. Most of the data are based on estimations and interpretations of geo-scientific data from different kinds of sources including photogrammetry, remote sensing, geodesy, volunteered geographic information, internet sources (e.g. twitter) and many more.

It turns out that there are a few deficits which hinder the development of spatial and spatio-temporal distributed data mining. The first deficit is produced by the technical developments of parallel and distributed data management itself. Due to bandwidth restriction, massive data transfer is not possible. The second deficit is due to the traditions of mapping within geography. Many sub-disciplines of geography focus on the creation and analysis of 2D maps for simplification reasons. The geometry cores and coordinate systems of common geo-information systems are often based on those 2D maps and their reference systems. The step towards the development of coordinate systems tailored to computer arithmetic with suitable precision at large scales has not been fully taken, yet. This includes the steps towards efficient 3D (spatial) or even 4D (spatio-temporal) data models at global scale and the transitions between those spaces. The third deficit is the missing integration of geo-information concepts such as the management of 2D/3D (spatial) and 3D/4D (spatio-temporal) data access through suitable spatial and spatio-temporal access methods into modern parallel and distributed data management technologies.

The background for this work was provided by the DFG Projects “3D Data and model management for the geo-sciences with special consideration of topology and time”

(project number: 94512227) and “Topologically consistent modelling in a spatio-temporal context with moving objects in the city model” (project number: 326802322). The goals of the projects are to establish a data management model which is able to manage topology in the context of geo-sciences and city models, which includes 4D (spatio-temporal) data. The proposed data management concept relies on abstract mathematical models in terms of geometry, arithmetic, and topology. This dissertation extends the research to distributed data management concepts to support simulation implementations in order to address the collaboration gab between applied system scientists and specialized scientists who depend on distributed geo-information. Due to the technical developments of parallel and distributed data management, it is possible to advance this collaboration.

A collaboration concept is developed and discussed, which is based on interconnected data extraction and transformation pipelines. This results in a proposed data management concept. The proposed data management concept is introduced bit by bit. The spatial model is based on simplicial complexes, and the spatio-temporal model is based on the Polthier and Rumpf model [53] for moving and morphing simplicial complexes. Geometrically induced topology and the term of topological consistency plays a major role for some types of simulation implementations. But there are other topological relations which need to be managed by the proposed data management concept also. The set of relation types together with the spatial and spatio-temporal model shape an abstract data management model which is based on the *Property Graph Model* and the *Feature Model* of ISO 19109. The development of spatial, spatio-temporal and topological access methods which are able to manage a *Property Graph* in parallel and distributed environments completes the proposed data management concept.

The proposed data management concept is implemented and fused with *DB4GeO* [14]. *DB4GeO* is a service-based geo-spatio-temporal object-oriented research database architecture of the working group of Martin Breunig [14]. The object core of *DB4GeO* is completely redesigned. New node operations are developed in order to build the *Property Graph* with the set of special relations and the properties of the underlying geometries. The node operations include a special algorithm which is able to triangulate a set of d-dimensional simplicial complexes into a set of disjoint $(d + 1)$-dimensional simplicial complexes. The geometry core of *DB4GeO* is based on double precision. The core is revised and extent by new algorithms for triangulation, difference, and equality calculations in order to be able to implement the node operations. Furthermore, two new access methods are implemented, one scalable p-adic space-filling curve index and one topological access method. All the new implementations are explained in detail, evaluated by experiments based on different field data and finally discussed. This proposed data management concept in combination with the described collaboration concept may enhance the collaboration between experts of different scientific domains who rely not only on spatial or spatio-temporal topologically consistent data when put into practise.

Zusammenfassung

Es ist eine schwierige Aufgabe, innerhalb und zwischen den wissenschaftlichen Domänen eine gemeinsame Sprache zu finden. Eine Lösung ist die Verwendung von Abstraktionen. Allgemeine Systemwissenschaft als wissenschaftliche Abstraktionsschicht ist die Wissenschaft der Modellierung beliebiger Phänomene, Systeme, deren Verhalten analysiert werden kann, um ein tieferes Verständnis über diese Phänomene zu erlangen. Angewandte Systemwissenschaftler modellieren, analysieren und simulieren Phänomene der Wirklichkeit, die mehrere wissenschaftliche Domänen umfassen können.

Das Hauptziel der *Computational Science* ist es, jede wissenschaftliche Arbeit durch die Bereitstellung anwendbarer Computertechnologie in Verbindung mit Datenmanagement- und Verarbeitungskonzepten zu unterstützen. Expertenwerkzeuge neigen dazu, sich aus wirtschaftlichen, technischen oder anderen Gründen auf bestimmte wissenschaftliche Bereiche zu konzentrieren, auf Kosten der Zusammenarbeit und bereichsübergreifender Abstraktion. Daraus resultiert eine schwierige Zusammenarbeit mit und zwischen Experten unterschiedlicher Wissenschaftsbereiche, die den wissenschaftlichen Arbeitsablauf behindert. Das *Data Mining* für groß angelegte Simulationen und die Modellgenerierung ist ein komplexer Prozess, bei dem Wissenschaftler unterschiedlicher Fachrichtungen in einem kooperativen Prozess eng zusammenarbeiten müssen. Die meisten Daten basieren auf Schätzungen und Interpretationen geowissenschaftlicher Daten aus verschiedenen Quellen, darunter Photogrammetrie, Fernerkundung, Geodäsie, freiwillig bereitgestellte geografische Informationen, Internetquellen (z. B. *Twitter*) und viele mehr.

Es zeigt sich, dass es einige Defizite gibt, welche die Entwicklung von verteiltem *Data Mining* behindern, welches räumliche oder raum-zeitliche Geoinformationen beinhaltet. Aufgrund der Bandbreitenbeschränkung ist eine massive Datenübertragung nicht möglich. Des Weiteren konzentrieren sich viele Teildisziplinen der Geographie aus Vereinfachungsgründen auf die Erstellung und Analyse von 2D-Karten. Die Geometriekerne und Koordinatensysteme gängiger Geoinformationssysteme basieren häufig auf diesen 2D-Karten und ihren Bezugssystemen. Der Schritt zur Entwicklung von auf Computerarithmetik zugeschnittenen Koordinatensystemen mit geeigneter Genauigkeit in großen Maßstäben ist noch nicht vollständig getan. Dazu gehören Schritte hin zu effizienten 3D (räumlich) oder sogar 4D (raum-zeitlich) Datenmodellen im globalen Maßstab und die Übergänge zwischen diesen Räumen. Außerdem fehlt es an Integration von Geoinformationskonzepten wie der Verwaltung von 2D/3D (räumlich) und 3D/4D (raum-zeitlich) Datenzugriffen durch geeignete räumliche und raum-zeitliche Zugriffsverfahren in moderne parallele und verteilte Datenhaltungstechnologien.

Den Hintergrund für diese Arbeit lieferten die DFG-Projekte “3D-Daten- und Modellmanagement für die Geowissenschaften unter besonderer Berücksichtigung von Topologie und Zeit” (Projektnummer: 94512227) und “Topologisch konsistente Modellierung im raum-zeitlichen Kontext mit bewegten Objekten im Stadtmodell” (Projektnummer: 326802322). Die Ziele der Projekte sind die Etablierung eines Datenmanagementmodells, das in der Lage ist, Topologien im Kontext der Geowissenschaften und Stadtmodellen zu verwalten, die auch 4D (raum-zeitliche) Daten beinhalten können. Das vorgeschlagene Datenmanagementkonzept basiert auf abstrakten mathematischen Modellen in Bezug auf Geometrie, Arithmetik und Topologie. Diese Dissertation erweitert die Forschung auf verteilte und parallel Datenmanagementkonzepte zur Unterstützung von Simulationsimplementierungen, um die Kooperationslücke zwischen angewandten Systemwissenschaftlern und spezialisierten Wissenschaftlern zu schließen.

Es wird ein Kollaborationskonzept entwickelt und diskutiert. Daraus ergibt sich ein vorgeschlagenes Datenmanagementkonzept. Die geometrisch induzierte Topologie und der Begriff der topologischen Konsistenz spielen für einige Arten von Simulationsimplementierungen eine große Rolle. Es gibt jedoch auch andere topologische Beziehungen, die durch das vorgeschlagene Datenverwaltungskonzept verwaltet werden müssen. Die Menge der Relationstypen bildet zusammen mit dem räumlichen und raum-zeitlichen Modellen ein abstraktes Datenmanagementmodell, das auf dem *Property Graph Model* und dem *Feature Model* der ISO 19109 basiert. Die Entwicklung von räumlichen, raumzeitlichen und topologischen Zugriffsmethoden, die in der Lage sind, einen *Property Graph* in parallelen und verteilten Umgebungen zu verwalten, vervollständigen das vorgeschlagene Datenmanagementkonzept.

Das vorgeschlagene Datenmanagementkonzept ist implementiert und mit *DB4GeO* [14] fusioniert. *DB4GeO* ist eine servicebasierte, objektorientierte Forschungsdatenbankarchitektur für räumliche und raum-zeitliche Geodaten der Arbeitsgruppe von Martin Breunig [14]. Der Objektkern von *DB4GeO* ist komplett neu gestaltet. Neue Knotenoperationen können den *Property Graph* mit der Menge der speziellen Beziehungen und den Eigenschaften der zugrunde liegenden Geometrien erstellen. Die Knotenoperationen beinhalten einen speziellen Algorithmus, der in der Lage ist, eine Menge von d-dimensionalen simplizialen Komplexen in eine Menge von disjunkten $(d + 1)$-dimensionalen simplizialen Komplexen zu triangulieren. Der Geometriekern von *DB4GeO* basiert auf “double” Genauigkeit. Dieser Kern ist überarbeitet und um neue Algorithmen für Triangulations-, Differenz- und Gleichheitsberechnungen erweitert, um die Knotenoperationen implementieren zu können. Darüber hinaus sind zwei neue Zugriffsverfahren implementiert, ein skalierbarer p-adischer Index basierend auf raumfüllenden Kurven und ein topologisches Zugriffsverfahren. Alle neuen Implementierungen werden ausführlich erklärt, durch Experimente auf Basis unterschiedlicher Felddaten evaluiert und abschließend diskutiert. Die praktische Umsetzung des vorgeschlagenen Kollaborationskonzepts in Kombination mit dem beschriebenen Datenmanagementkonzept vermag die Zusammenarbeit zwischen Experten verschiedener wissenschaftlicher Domänen zu verbessern, welche nicht nur auf räumlich (oder raum-zeitlich) topologisch konsistente Geodaten angewiesen sind.

Contents

List of Figures

List of Tables

1 Introduction

The variety of geo-scientific data and information reaches from photogrammetry and remote sensing over volunteered geographic information (VGI) or non-volunteered geographic information to all kinds of calculated combinations of those data to create knowledge by techniques which are subject of the so-called “data mining”.

The following paragraph is published in [30] beforehand:

Computational science has rapidly developed over the past years. New technologies provide the storage and the processing of large amounts of daily produced structured and unstructured data with more or less high user interaction. Those technologies may also be used to handle spatial or spatio-temporal archives and analytics, especially concerning geo-scientific data. The five big V’s defining *Big Data* (value, variety, veracity, velocity and volume) suit to geo-scientific data and their common use cases. Those data sets consist of large amounts of information concerning moving, morphing, topology and the trend of attributes of the given geo-scientific objects (high volume). Geo-scientific data are structured, processed and analysed by scientists of a broad variety of different expertise, working tightly together to generate value of measured data (value). Because of the number of scientists, their different use cases and missing standards, geo-scientific data could be seen as relatively heterogeneous (variety). In any case, the data needs to be analysed quickly (velocity) and needs to be trustful (veracity). It is to mention that all geo-scientific information is subject to uncertainty [26]. So when dealing with veracity of geo-scientific data, we have to deal with uncertainty information about the data.

Computational geometry relies on computational finite numerics. The combination of models of different scales using double or integer arithmetic is not a trivial task. It is only possible to manage small scaled models on a global scale if the used coordinate data types provide sufficient precision. This may lead to data types which need to use numerous bits to provide the necessary precision. It is a challenge to manage geo-scientific data on a global scale on a multi-database query processing architecture with the support of spatial, spatio-temporal or topological access methods with sufficient geo-scientific data precision and uncertainty information.

The following enumeration and the next paragraph summarizes the challenges pointed out by Martin Breunig, Mulhim Al Doori and Patrick Bradley in [30] which motivated this work:

1. Consistent storage and efficient retrieval of big geo-spatial data will significantly improve today's usability of geo-database management systems
2. Geo-spatio-temporal or in general *nd*-topology models are very useful to automatically check the consistency of polytope models
3. Due to the complexity and the large scale of GIS and remote sensing data, it is desirable to identify and analyse geographic objects when designing complex distributed systems
4. The availability of a wide range of analysis methods facilitates the success of a data mining task, however this relies heavily on the data and GIS experts ability to configure the appropriate selected algorithm
5. Differing backgrounds of GIS experts and variability of computing skills can raise another challenge

Another area that will benefit from big data research progress is the issue of scalability and the ability to exploit big distributed data sets of images that do not fit into memory. The need to rethink current data analysis algorithms is evident and will impose more pressure on data analysts as graphical data sets grow exponentially. This will also affect the ability to deal with data imprecision, evaluate and correct errors in graphical raw data or segmentation data. This gives rise to the need to define sets of robust algorithms capable of incorporating data errors and imprecisions by defining the appropriate methodology to evaluate and correct errors or imprecisions in data and therefore on knowledge. This methodology will need to show significant success in combining all the information available on studied areas regardless of their media to enhance that data analysis process and therefore reflect positively on knowledge generation and management.

Simulation analytics are also important to create knowledge. On the one hand, simulations play a major role in understanding complex systems and, on the other hand, in predicting anything what is of interest. If scientists may want to collaborate on a large scale, it is easy to understand that the data management to handle these complex structured and unstructured data must provide ways to manage data as complex as each participating scientists may want. The data structure needs to be abstract enough to give each scientist freedom to integrate its models, and precise enough to handle typical geo-scientific data and information. The challenge is to bring scientists of different backgrounds together without constraining their creativity. Complex and specialized standards and specifications force scientists to design models only on the bases of the standard or specification. Figure 1.1 illustrates the connection of the three different

domains where this work is settled, the pyramid to create wisdom and how complex queries can be which involve all the three different domains.

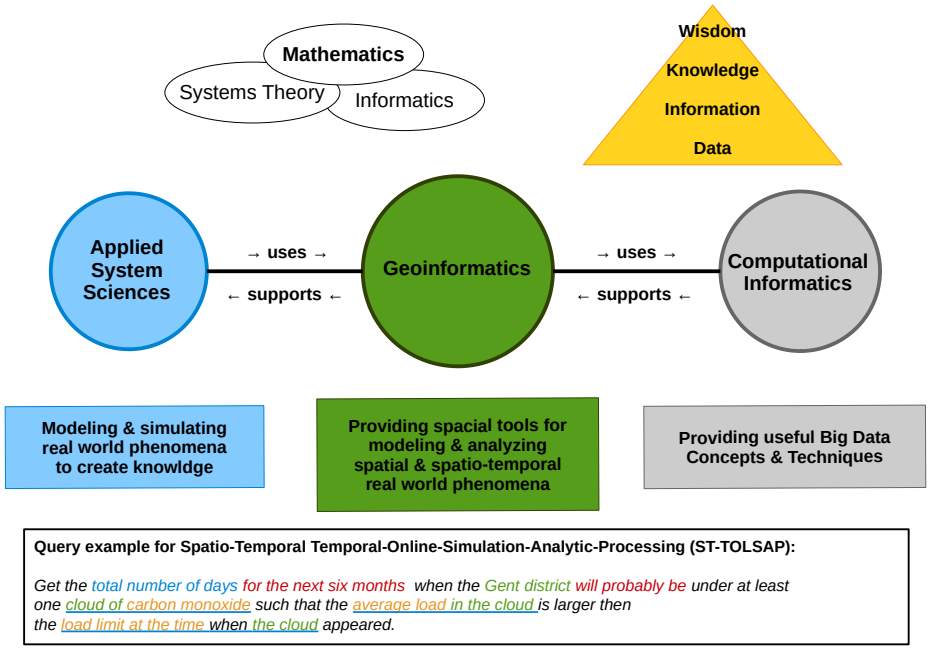


Figure 1.1: Big picture

The following four paragraphs are published in [30] beforehand:

Traditional workflows depend on data transfer by streaming the data to some analysing unit. Sensors record some entities in the first step and stream them for further modelling or analytics. Present research topics address edge-based computing as a modern way by pre-analysing and filtering relevant data by sensor-networks to reduce data transfer. A second step is the modelling of the geo-scientific object in form of a 3d representation in combination with thematic attributes. Further, monitoring and modelling leads to moving and morphing multidimensional representations of the geo-scientific object of interest. To run analytics on the generated data as the third step, data usually needs to be transferred to some GIS or analysing tool. Obviously, data transfer is a bottleneck when dealing with distributed and parallel data. Processing analytics also overloads the hardware capacities of the scientists' laboratories.

In the case of simulations, e.g. finite element method, scientists usually write their own simulation applications or use specialized software products. Simulation results are written to some persistent memory to copy from for further analytics and visualization. Even only for the visualization of the simulation results, the data transfer between

main memory and the graphics memory turns out to be a bottleneck when dealing with large simulation results. In-situ solutions help to solve that problem by visualizing and analysing the data side by side to the running simulation in real-time, concurrent, also on high performance computing clusters [58]. Scientists are able to see results immediately while running the simulation. Time-consuming simulations do not need to run until the end if real-time results show strange behaviours.

Modern workflows use DBMS's (database management systems) to store the representations and integrate some analytic intelligence into the DBMS. Some analysing work and even modelling different representations can be done faster by the "intelligent" geo-scientific DBMS itself, running centralized on more powerful servers than the workstations of the scientists. Furthermore, DBMS's are built to select, query and analyse large amounts of data efficiently where else file based solutions tend to fail. Multi-user support and privacy controls are further advantages of DBMS's. But parallel multidimensional DBMS's for geo-scientifically use are still research topics nowadays and therefore not commonly used. Achieving global scientific modelling and analytics on distributed and parallel geo-scientific data leads to high performance computing and its distributed/parallel data processing power on clusters with new ways of data management such as NoSQL-DB (Not-Only-Standard-Query-Language-Database) support.

Virtualization, cloud-based software products and services will be established to outsource the hardware into data centres or data warehouses which are built to archive distributed and parallel data and solve analytics centralized in a parallel multi-user environment. Programming paradigms such as vectorization, parallel algorithms, data streaming algorithms [44], calculations on GPGPUs (General-Purpose-Computation-on-Graphics-Processing-Unit), the map-reduce programming model or test- and behavior-driven-development show new ways of developing efficient analytic software products for the environmental, economical or any other science community with different skills in computer sciences or programming to solve their tasks. To bring all of those modern technologies, programming paradigms and scientists together, new workflows have to be developed to ease the way of cooperative working. From the geo-informatic point of view, this is one of the challenges to supporting the efficient production of added value to geo-scientific data.

1.1 Goals

Distributed data mining is a common task applied system scientists or analysts need to solve in collaboration. This work serves the support of distributed data mining of geo-scientific data to produce and deliver the input data of arbitrary geo-analytics and -simulations in respect to data-transfer reduction, since the bottleneck of modern distributed and parallel environments is IO bandwidth. A collaboration concept needs to be worked out.

A first set of sub-goals are summarized in the following paragraph, which is published in [30] beforehand:

The goals are to introduce a sound mathematical approach for a topologically consistent geo-spatio-temporal model based on the concept of the *incidence graph*, to create a general topological model for geo-scientific data and to present a redesigned version of *DB4GeO*, the service-based geo-spatio-temporal database architecture of the working group of Martin Breunig [14], which integrates those concepts and models, on the way to efficient distributed and parallel management of massive geo-spatial data.

Another sub-goal is stated in the following paragraph, which is published in [31] beforehand:

To simulate environmental processes, noise, flooding in cities as well as the behaviour of buildings and infrastructure, 'watertight' volumetric models are a measuring prerequisite. They ensure topologically consistent 3D models and allow the definition of proper topological operations. However, in many existing city or other geo-information models, topologically unchecked boundary representations are used to store spatial entities. In order to obtain consistent topological models, including their 'fillings', in this work, a triangulation combined with overlay and path-finding methods should be presented by climbing up the dimension, beginning with the wireframe model.

This pre-processing example shall be applied to some test data in order to serve as an experiment to evaluate the redesigned version of *DB4GeO*.

Furthermore, the load-balancing within one high performance cluster or the management of distributed data for distributed data mining can be enhanced by spatial, spatio-temporal or topological access methods. The sub-goals are to provide and evaluate implementations of space filling curve indices and topological access methods in order to analyse the behaviour of those access methods.

The goals of the beforehand published paper [8] concerning the space filling curve index were summarized by Patrick Erik Bradley as follows:

> It is not our aim to find the 'best' p-adic Gray-Hilbert curve for a particular problem relying on indexing methods. The emphasis on this article is more to combine space-filling curves with a tree structure in a high-dimensional and p-adic setting, and to study these p-adic Gray-Hilbert curves from a theoretical point of view and to substantiate this study by some experimental results on some data sets. In particular, we feel that the relationship between the distribution of data and the local properties of p-adic space-filling curves encoding the data deserves much further exploration.

My part and therefore a sub-goal of this work, was to find the implementation of a fixed and a scalable p-adic space-filling curve index based on a tree structured access method which preserves the topological structure of p-adic space-filling curves when traversing

the trees and print the leaf nodes. This includes an evaluation of the implemented p-adic space-filling curve indices by the use of multidimensional field data.

All the previously defined goals need to be supported by the implementation of suitable importers for the different kinds of data sets and exporters to use the results for further processing or visualization.

1.2 Outline

The following Chapter 2, about rudiments, is divided into three sections. The first section introduces *Systems theory*. It includes typical workflows and the used spatial and spatio-temporal data structures. The second section of the rudiments introduces general distributed and parallel concepts of the computer science. The third section introduces typical geo-scientific data structures. The chapter closes with an overview of related organizations, standards, and technologies.

Chapter 3 starts with a problem description and the resulting requirements. In the following, a basic collaboration concept and a proposed data management concept are described. The chapter closes with the implementation steps to be taken in order to realize the theoretical concepts.

Chapter 4 discusses the implementations of the proposed data management concept in detail. The algorithms are presented, whereby using the philosophy of the *Property Graph Model*. Examples to illustrate the algorithms are given. This chapter also includes the description of new access method implementations which may be used for the management of distributed data or for controlled data distribution (e.g. load-balancing).

Chapter 5 provides experiments which are performed on data-sets from different sources. The heavy influence of double precision arithmetic on the results, in particular the positional and angular precision, is discussed for the implemented geometry core. The two new access methods are evaluated, also.

Chapter 6 discusses the results of the previous experiments and closes with an outlook of my future research on the way to support the pre-processing and processing of geo-analytics and -simulations in parallel and distributed system environments.

2 Rudiments

The following sections summarize rudiments across the three main fields which are connected to the goals of this work.

The first section introduces the work and workflow of system scientists, which use simulations to get a deeper understanding of what holds the world together.

The second section gives an overview of concepts which computer science offers these days to support data mining and processing in distributed and parallel environments.

The third section provides information about the tools and paradigms which are used within geoinformatics as a bridge between computational sciences and geography, cartography or any other geo-sciences (e.g. simulation models on geo-data or -information).

The fourth section introduces related work and research which apply to the goal of this work.

2.1 Simulation models

This section introduces simulations and how they help to understand more parts of the real world. The focus lies on simulation types, which may include spatial data. Most of this section is based on the lectures of my studies of applied system science in Osnabrück, especially the lecture notes of *Equation based models I and II* [45]. This introduction shows only the tip of the iceberg. But it is important to understand the abstract level of system science in order to understand the artistic freedom of simulations and why it is important to save this freedom. Furthermore, the presented workflow illustrates the data-flows when dealing with simulations.

2.1.1 Applied system science

To understand the use of simulations, it is necessary to understand the focus of general *Systems theory*. One definition of general *Systems theory* was made by the biologist Bertalanffy 1977 [45]:

> Scientific investigation and theory of systems in various branches of science (Physics, chemistry, biology, psychology, social sciences, economics, etc.), where general *Systems theory* summarizes those principles that apply to all or defined subclasses of systems.

Therefore, general system science may combine the subjects of those branches of science into one model, which may allow the investigation of more complex queries.

According to Stachowiak, models which help to answer certain queries have three characteristics [45]:

1. Characteristic of illustration: Models are illustrations of originals, where originals are natural or artificial objects.
2. Characteristic of simplification: A model does not reflect all the properties of the original. It is a simplified representation of a section of reality, which is determined by the task (model purpose).
3. Characteristic of subjectification: Models are only for a certain group of people who are familiar with the rules of the game and only with regard to certain executable operations and within certain time intervals. Once a more perfect model of the same original is created, the old model becomes worthless. Through this continued chain of constant improvement of the models or "images" of the real, it may appear as a way of thinking that one day an objective picture of reality is reached.

Furthermore, Stachowiak summarizes the tasks (model purposes) into four classes [45]:

1. Functionality: Models are made to fulfil certain functions. They are often preferred over their original because they are simpler and easier to be handled.
2. Simulation: Operations are to be carried out and tested on the model, which cannot be carried out on the original object itself, or only with great difficulty. The model replaces the original on a trial basis.
3. Explanation: The model is supposed to explain certain phenomena or the behaviour of objects.
4. Prediction: Sometimes models must also be able to make predictions about the future behaviour of the objects.

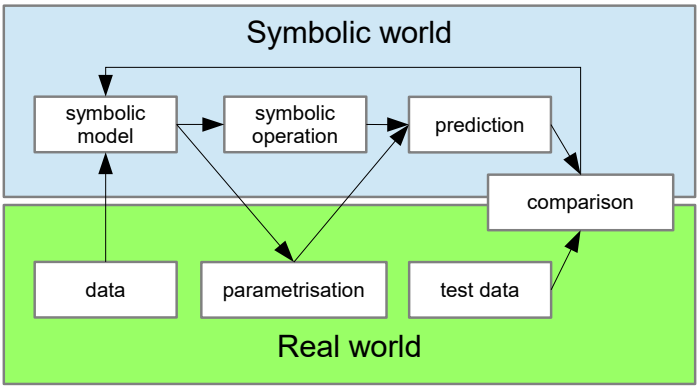


Figure 2.1: Methodology of applied system science [45].

The process of scientific work is shown in Figure 2.1.

Simulation models are one class of four different classes within the mathematical modelling, according to Wissel. He describes them in the year 1989 as follows [45]:

> The modelled systems should be mapped as realistically as possible. The complexity of real systems makes the models very complicated and extensive, so that they can only be solved on the computer. The goal of such models is to test the effect of various interventions on this image of the system. Simulations serve as a substitute for experiments, and it is hoped to use them for the creation of forecasts and to be able to give pointers for the system management. Difficulties are caused by the time-consuming and costly acquisition of large data sets. In addition, the statistically based uncertainty increases with the number of parameters to be determined from the same data set, i.e. with increasing complexity of the model. An optimal level of complexity for a model can be found. Furthermore, complex computer models are in principle unsuitable to provide an understanding of the functional relationships.

The task did not change since 30 years of public computer technology. It is still easy to see that there are as many models as people can think of and that each of those people have difficulties to process their models when large spatio-temporal data is involved. Having this in mind, it is desirable to increase geo-information transfer while reducing the geo-data transfer and to provide processing power to the people from the geo-informatics point of view. Therefore, this chapter concentrates on models which use spatial or spatio-temporal data to find out how spatial or spatio-temporal data integrates into models.

A few steps need to be made when creating a mathematical model [45]:

1. describing the problem
2. identifying the purpose of the model
3. defining the borders of the system
4. verbally describing the model
5. extracting the elements of the system (with their dimension)
6. extracting the structure of the system
7. creating a diagram which shows the effects of each element to each element (effect graph)
8. set up the functional relations
9. quantification
10. creating a simulation diagram
11. developing the simulation executable
12. validation (structure, behaviour, empirical, application)
13. apply to the purpose (prediction, analysing equilibrium, attractors, repellers, stability, different scenarios etc.)

It is to mention that the process repeats from step 4 if the results of step 12 are not satisfying. However, this general approach to create mathematical models to describe real world phenomena applies to various branches of science.

Steps 5, 6 and the final creation of the effect graph in step 7 are the most import steps which define the dimension and the involved geometry types. As mentioned before, reducing the complexity will reduce the uncertainty. As an example, black box models may be complex enough when the purpose lies on predicting the structure of age in a certain population. The geo-data would integrate into the model only as retrieving the box as a shape and retrieving its population in a vector of age classes (population pyramid).

There could be models which focus on the spatial differences of age pyramids, like comparing Germany and France, since the results have been spatially linked. But this is not part of the model itself. If the model integrates dependencies of one population pyramid to another spatially "neighbouring" population pyramid (in some kind of diffusion process), the whole model needs a lot more information (consistent topology, quality of intersections etc.) from its geometry model than before. The whole simulation model needs even more information from a spatio-temporal model (spatio-temporal consistent topology, spatio-temporal quality of intersections etc.) if the model integrates moving and morphing shapes.

The following sections introduce different simulation types and their spatial or spatio-temporal input dependencies in order to find similarities or abstract layers which they may have in common. This will help to design geo-data structures which represent the input or data sets where the input can be extracted from.

The result of steps 8 is a mathematical model which can be analysed. Step 9 and 12 can involve geo-data if any parameter or state variable depends on geo-data or -information. Both steps create geo-data or -information transfer. Step 10 and 11 will be discussed in Section 2.1.4.

2.1.2 System of partial differential equations

Many simulation models numerically solve a system of differential equations. Systems of ordinary differential equations are usually used to model environmental systems, chemical systems, biological systems, economic systems and many more. The internal differential equations may be simple or complex related to each other (linear or non-linear). The system of differential equations shows simple or complex behaviour. The behaviour may change under certain initial configurations (Bifurcations).

Partial differential equations add spatial dependencies in terms of reaction-diffusion-advection equations. A system of those partial differential equations is used to model any reaction and spatial transfer of any entity in dependence of any other entity in space and time. An example is given by the Navier-Stokes equations, which model the motion of viscous fluid substances. As one of seven millennium prize problems with one million dollar reward, the equations have not been solved analytically, yet. Simulations are the only way to numerically approximate the solutions. Another application can be the exposure of substances which interact locally and follow some diffusion rules or are transported along an advection field. Other systems of partial differential equations may model population dynamics in space and time. Each population may be coupled by some predator prey term, may follow some advection field or even diffuses by the state of other species (bio-diffusion). The result shows waves, chaos, Turing patterns and so on. The application is rich, but the solving is not trivial.

The simplest form of input for spatial discretization is a Cartesian grid (or mesh) where the real numbers dx define the spacing between each grid (or mesh) point in each spatial dimension. Having for example three populations in a box of 50 km times 10 km times 1 km and a discretization of $dx = 1m = 0.001km$ the simulation would need $1.5x10^{12}$ real numbers to measure the mass or concentration of each population on each grid (or mesh) point with a resolution of $50000x10000x1000$. This also means, that it would need $1.5x10^{12}$ processes which all do the same calculations, more or less with different values. Cartesian grids (or meshes) belong to the group of structured grids because the topology is relatively simple, since each grid point is identifiable by an *id* (e.g. defined by lexicographically ordered points) and because of the Cartesian metric, the position of each grid (or mesh) point is analytically calculable by their *id* if at least the position

of one grid (or mesh) point is known together with the axis orientations. The spatial discretization needs to be small enough to keep the error in acceptable ranges, which will raise the computational effort.

Other types of so-called structured grids (or meshes) may also be used. Each of those grids (or meshes) define a different base shape for their elements. In case of Cartesian discretization, the base shape is a unit cube. Other definitions may use different spacings along each spatial dimension, are based on curves, tetrahedra etc. All the structured grids have in common that the elements have the same connectivity to their neighbours, a regular connectivity. But the diffusion and advection equations need to include the spatial distortions at every grid (or mesh) point if the grid is not a Cartesian grid (or mesh). These models are also known as regular grids or voxel models.

Unstructured grids (or meshes) involve irregular topologies. The base shapes for the elements are connected in various ways, as needed. Typical examples include simplicial networks. Each base shape may be topologically connected to a number of neighbours.

Structured and unstructured grids (or meshes) are used as input for finite difference, finite volume and finite element methods. Those three numerical methods solve a system of partial differential equations. Which spatial discretization is used as input depends on the chosen simulation model. Which simulation model is used depends on the model for the real world phenomenon of interest.

The temporal discretization depends on the numerical solver. The Euler method is the simplest method to use, which adds the changes from present time to the next time to each state variable by a given time step size through the system of ordinary or partial differential equations. This is a linear approximation. The choice of a time step size dt, the temporal discretization, will imply that slopes are missed. The numerical error will increase step by step. Other methods dynamically evaluate the system of ordinary or partial differential equation time to find the best choice for each step, and may also take already calculated steps into account. This reduces the numerical error.

Smoothed Particle Hydrodynamics (SPH) or in general the Lagrangian simulation models or particle-based modelling are grid (or mesh) free simulations. The idea is to calculate the position of a particle by the properties of itself and all or just a part of distant neighbouring particles. Since more distant particles have less influence, there is a maximal distance where the influence is zero due to computational arithmetic. The error shrinks the more particles are simulated, but this will raise the computational effort. For too close particles, a smoothing distance is often used to ensure robustness of the algorithms. This type of simulation model may also be seen as rule-based simulation model, which will be described in the next section.

2.1.3 Rules instead of equations

Rule-based simulation models are different from simulation models which try to approximate states of systems of ordinary or partial differential equations, only. The idea is to connect elements by a set of rules which will be performed each time step for each element.

Cellular automata are rule-based simulation models. The application reaches from biological systems over physical systems to chemical systems and many more. The idea is to use a grid (or mesh) and to define a set of rules and discrete states for the grid (or mesh) elements. The set of rules changes the state for each element. The rules use the states of a number of neighbouring elements and the state of the element itself to calculate the new state of this element. There is no error involved, which accumulates over time.

The basic characteristics of a cellular automaton can be described as follows [45]:

1. Its dynamic takes place in space and time.
2. Its space is a discrete set of elements of a structured grid (or mesh).
3. Each of these elements has only a discrete number of possible states.
4. The states change in discrete time steps.
5. All elements are identical and behave according to the same development rules.
6. The development of an element depends only on it own condition and that of the locally surrounding neighbouring elements.

One of the popular simulation models of the kind is the "game of life" simulation model. It is defined on a 2D Cartesian grid (or mesh). The results show moving and morphing shapes on the 2D Cartesian grid only by the following simple definitions [45]:

1. Each element is topologically connected to 8 neighbours (Moore neighbourhood).
2. The elements are only in one of two states (alive or dead)
3. The birth of an element can happen if itself is dead and will happen if it has three living neighbouring elements.
4. The element drops dead on loneliness if it is alive and has less than two neighbours which are alive
5. The element drops dead on overpopulation if it is alive, and it has more than 4 neighbours which are alive

A more general simulation model is achieved by agent based modelling (AGM). Agent based modelling is used in various disciplines, also. Many complex systems are modelled and analysed using statistics, especially in sociology.

Each agent belongs to a class of agents which share the same set of rules. The rules change the state of an agent by analysing its own state and the state of agents it is topologically connected to. So each class of agents needs to set up rules for every other class in the model. The topological connection may be static or dynamic, geometrically induced or by any other model which is able to topologically connect agents. The geometrical induced topology may be calculated by overlay or by ranges (by euclidean distance, accumulated path weights in topological spaces etc.).

2.1.4 Simulation processing

Simulation processing is as complex as the models behind the scene. The easiest way to run simulations is:

1. Find the atomic elements where the set of rules/equations needs to be processed, together with their topology (to be found in the set of rules/equations).
2. Put the states of topologically connected elements, the dependent elements, into a data structure which allows the hardware to retrieve the information from topologically connected elements as fast as possible.
3. If the hardware is able to run multithreaded applications, create a pool of threads which is able to process each element by one thread. If the hardware is only able to run single-threaded applications, create one process which iterates over each element when processing one simulation step.
4. Run each thread for the desired number of steps and write the results synchronized into a data structure where each thread is able to retrieve the necessary information for the next simulation step as fast as possible. Write the results or parts of it for analysis to persistent storages at the end of each simulation step, if needed.

Step 1 depends on the mathematical model. Once this step is done, the simulation diagram shows what needs to be implemented.

Step 2 leads to suspect which difficulties are involved. One part is to search data from a variety of sources in a variety of formats and of various spatial extent. Another part is to transform the data into a special structure from those multiple sources. The third part is to transfer the data (import) or to transfer the simulation code (deploy) to the hardware which is able to execute the code and carries that special data structure. All of these parts need to be supported by the used geo-information technologies, since spatial simulation processing may be seen as a special geo-information technology. A closer look will be given in Section 2.3.

Step 2 also includes the restructuring of the data into a data structure which suits to the topology of the simulation elements and the hardware. If the topology is rather simple, regular, like in form of structured grids and the topology does not change over time, it needs algorithms which generate this structured data. This is also known as discretization or mesh generation. If the topology is irregular and changes over time, this data structure is much more complex and will be discussed in Section 2.3.

Step 3 illustrates that there are always two complete data sets needed to calculate one simulation step, since the present simulation step depends on the results of the last simulation step. Furthermore, the synchronization of each thread and the writing of results to persistent storages for further analysis may be needed. These results may consist of the entire spatial model per time step. In order to spatio-temporally analyse the results, the spatial results should be put together into one topologically consistent spatio-temporal model. A closer look on those models will also be given in Section 2.3.

2.1.5 Relevance for the work

The goal of this section was to summarize the work and workflow of system scientists. It is important to know what kind of spatial data sets are required by each of the different simulation types. The spatial and spatio-temporal data sets reach from structured to unstructured grids or meshes. It is also of great interest how simulation models are created and how they are used and analysed. Regular topology or a discretization of the space at regular points helps fast simulation processing based on arrays. But unstructured grids or meshes are able to represent any shape which might be of interest. This knowledge enables computational scientists, especially geo-information technologists, to design the appropriate geo-information technology which is able to support the needs of system scientists.

2.2 Distributed and Parallel Data Management

The next sections will discuss fundamental concepts of computer sciences described by common computer science literature (e.g. [57], [22], [48], [21]) to be used for spatial, spatio-temporal or topological distributed and parallel data management which may be used to support simulation analytics. The presented rudiments are meant to give only an overview and present only the tip of the iceberg concerning this fast developing science.

ITO (information technology outsourcing) is a major concept which transfers any data storages or data processing units to an external information technology provider. The benefits of ITO are mainly the support of those providers which are experts in the field of information technology. The providers ensure security and performance by contract with calculable cost. If an institution managed its own information technology, it would need additional room, electricity, information technology, the computer scientists who manage all of this etc. Therefore, ITO is a form of collaboration and in the field of cloud and grid computing it could be the only affordable way for some institutions or in private which are not able to spend the start-up or running costs of cloud and grid computing. There are also ecological benefits which arise by sharing information technology.

As an example, the provider may only provide computational power by virtual machines. These virtual machines may be configured as needed. Another example would be to use cloud services like *Office365* or *Overleaf* as applications which are ready to use. Useful tools are developed on every level.

The following two paragraphs are published in [30] beforehand:

Two main online processing paradigms are used for distributed and parallel data management. The first focuses on the efficient transaction processing known as OLTP (online transaction processing) and the second on analysing the data known as OLAP (online analytic processing).

While modern OLTP-applications may focus on real time processing of sensor data in sensor-networks, OLAP-applications focus on complex analytics of structured data, usually done in data warehouses. Both online processing paradigms hinder each other when working on the same data stock.

The ETL (extract transform load) is a process which imports data from a number of sources, transforms them into an appropriate structure and exports it to a target system. The sources and sinks may be a combination of files, databases, or applications. ETL's are used to fed data warehouses. Therefore, an ETL is able to connect OLTP with OLAP and enables data analytics which usually consists of collecting, cleansing, organizing, storing, analysing and governing data [22].

2.2.1 Data Management

The following two paragraphs are published in [30] beforehand:

Traditional DBMS (database management systems) are R-DBMS (Relational-DBMS), O-DBMS (Object-DBMS) or the hybrid, OR-DBMS (Object-Relational-DBMS). R- and OR-DBMS use table definitions to manage the data. Each column of a table defines an attribute of a record (row). While in R-DBMS only basic attribute types are supported, in OR-DBMS each attribute may be defined by a more complex object type known from OOP (Object-Oriented Programming). Both DBMS types use SQL (Structured Query Language) for communication. O-DBMS manage objects in a number of different ways and use an OQL (Object Query Language). While an R-DBMS is not flexible enough to manage more complex data types, an O-DBMS lacks standardized communication and efficient data management due to its complex object type definitions. However, relational data management in the form of tables can be parallelized and distributed in various ways. Therefore, the R-DBMS model is still widely used and efficient in various fields which deal with structured basic data types. NewSQL Systems are R-DBMS's which are used for OLTP.

NoSQL Systems are DMS's (data management systems) focusing on the management of semi-structured or unstructured data. They do not use SQL for communication. This is also the reason for the naming convention. Four groups are to mention. These are Key-Value-Stores, Extensible-Record-Stores, Document-Stores and Graph-DBs. Each system differs in data-quality and -quantity. While Key-Value-Stores manage huge quantities of data, data itself has less structure. On the other hand, Graph-DBs manage less data quantities but have more structure. Key-Value-Stores manage their data by a key. This key is unique across the whole system. The key is linked to a value. This value can be any type of data. Retrieving data is only possible through the key. It is a schema free data management, but the simplest and therefore easiest to handle when dealing with distribution or parallelization. Extensible-Record-Stores manage the data by a flexible table structure. Unlike in relational DBMS, each record is assigned to multiple column families, which results in different attribute types per record. Each record has a unique key. Document-Stores focus on text-based semi-structured documents like XML or JSON. Documents are managed by keys. Attributes of the documents can be indexed to speed up queries containing those attributes. Graph-DBs connect entities via relations to store topological information. In terms of a *Property-Graph-Model*, each node (entity) has a number of basic attributes. Each node can be connected via relations. Each relation may also have a number of basic attributes. Graph-DBs provide algorithms to analyse the structure of the graph.

Access methods are part of a database to optimize the calculations of database queries. A database query asks for information within a database and uses a language like SQL or OQL. Access methods organize the data in a way such that efficient calculation of information based on the managed entities of the database is possible. The structures which organize the data are usually based on unique identifications/keys, hash codes or

trees (e.g. balanced trees). If a key is a direct reference to the data, then using a key is the fastest way of retrieving data.

Since the memory address of the data has usually no meaningful connection to the contained data, the meaningful connection needs to be mapped. An example are keys within relational databases. Each column of a table may be used as a key if the data inside the column is unique. If the data inside the column is not unique, one needs to add another column to the key. The key becomes a two-dimensional key, a so-called concatenated index. If each $2d$-key is not unique, one has to add another column, forming a $3d$-key and so forth. The table gets sorted starting by the first column of the key. If the first column of the key is not unique, the second column of the key is used to sort that part of the table and so forth. Any structure or algorithm which can find data in a sorted list efficiently (which means: not linearly) can be used to query the data (e.g. B-Tree). Hash tables are also used to index data. A hash function generates a hash code (the key) by transforming the data which should be part of the key. It happens that some data may be transformed into the same hash code. If this happens, the data is added to a set which can be accessed by the code. The set is usually called a bucket in terms of hash codes. Each bucket is a subset of the data collection and can be queried more easily. Therefore, keys help to distribute data for fast retrieval.

2.2.2 Parallel Processing

The following three paragraphs are published in [30] beforehand:

Multithreaded environments enable applications to run parts of its code in parallel. In contrast to single-threaded environments, it yields the ability to speed up the application. The hardware is able to run multiple threads in parallel by the use of CPUs (central processing units) or GPUs (graphical processing unit). GPUs are designed to run code in parallel, which historically was used to run only 3D calculations as fast as possible. The quality of modern GPUs is often measured in how many cores are able to run which type of code. The type of core is designed for special purposes (cuda cores, tensor cores etc.) which are able to calculate special task fast. CPUs, on the other hand, are historically designed to run sequential code but may have multiple cores, too. CPUs generally consist of much fewer cores than GPUs. However, a computer may consist of single or multiple CPUs and single or multiple GPUs, depending on its purpose. A data server usually consists of multiple CPUs to handle multiple user interactions efficiently. A visualization server consists of multiple GPUs to render larger scenes. Any combination is conceivable.

A Cluster is usually made of high speed connected racks consisting of high speed connected blades which are basic computers with an HDD (Hard Disk Drive) or a SSD (Solid State Drive), RAM (Random Access Memory) and a CPU (Central Processing Unit) in case of a Shared-Nothing-System. In case of a Shared-Disk-System, the blades

share a number of HDDs or SSDs. High efficiency is expected by Shared-Nothing-Systems if the data is well distributed such that every blade has average work load for nearly all transaction/process types. Shared-Disk-Systems are not as dependent on data distribution as Shared-Nothing-Systems, but synchronization efforts could slow down the system. In-Memory grids use only RAM as main storage to reduce read and write operation times.

While distributed DBS's (database systems) suit best for distributed allies being part of one workflow, parallel DBS's are made for massive OLTP where several DBS servers host a copy of the same DBS to provide massive multi-user usage or for OLAP to solve one complex analytic query in parallel on one large data set distributed on a cluster as mentioned in common computer science literature.

Sharding is a major concept for data distribution across a cluster. Sharding splits the data collection into multiple parts. Each part is managed by one sub-node and can be processed autonomously. The major measurement to evaluate parallelization is expressed by the *Speed-Up*. The *Speed-Up* depending on the number of involved processors n is defined as [57]:

Definition 2.2.1.

$$\text{Speed-Up}(n) = \frac{\text{sequential time}}{\text{parallel time using } n \text{ processors}}$$

If the *Speed-Up* is below one, the parallel implementation takes more time, which is caused by *Start-Up / Termination / Synchronization* costs. Usually a *Speed-Up* equal to n is expected, but the relation is typically not linear and may raise above n if the *Input / Output* costs are reduced due to parallelization or may drop below n due to *Start-Up / Termination / Synchronization* costs.

The *Amdahl's law* integrates the fact that some applications are only partly parallelizable. It is defined as follows [57]:

Definition 2.2.2 (Amdahl's law)**.** *Let F_{opt} be the ratio of parallel processes relative to all processes and S_{opt} the* Speed-Up *of the parallelization. The overall* Speed-Up *can be calculated by:*

$$\text{Speed-Up} = \frac{1}{1 - F_{opt} - \frac{F_{opt}}{S_{opt}}}$$

For $\lim_{S_{opt} \to \infty}$ of *Speed-Up* from Definition 2.2.2 follows that the *Speed-Up* is bounded by $\textit{Speed-Up}_{max} = \frac{1}{1-F_{opt}}$.

Another interesting measure is the *Batch Scale Up*. The *Batch Scale Up* compares the *Speed-Up* of using n processors on n data units with using only one processor on one data unit. A value of one is desirable. The *Gustafson law* takes the dependency on growing data units into account [57]:

Definition 2.2.3 (Gustafson's law)**.** *Let n be the number of data units and the number of processors. Let s be the ratio of sequential processes relative to all processes. Let p be the ratio of parallel processes relative to all processes. With*

$$f = \frac{s}{s+p}$$

the Scalable Speed-Up *can be calculated by:*

$$\text{Speed-Up}(n) = \frac{s + n \times p}{s+p} = n - f \times (n-1)$$

Gustafson's law presupposes that using n processors will also be alongside with using them on n data units. Furthermore, it does not declare any dependencies which may be caused by *Input / Output* costs or *Start Up / Termination / Synchronization* costs [57].

2.2.3 Edge, Fog and Cloud Computing

The previously described systems are centralized solutions to run multiple applications, multiple DBS's as OLTP system or parts of one application, in one environment, using one DBS as OLAP system. Centralized solutions are cloud solutions which demand adequate data transfer connections. If the data transfer or the multi-user interaction get too high, the system cannot handle the workload even if it could process the data or multi-user interaction.

As mentioned before, distributed DBS's (database systems) suit best for distributed allies being part of one workflow. Content Delivery Networks are another example of distributed processing. In order to ensure the availability of content, worldwide server clusters are spread geographically and work tightly together to process any multiple user interaction. The system manages the distribution of the data by analysing the demand from the multiple user interaction. Queries are routed in dependence of geographical distance, connection speed and workload.

Input / Output costs between clusters are higher than the internal *Input / Output* costs of a cluster due to the different IO bandwidths. Even if parallel data readers and writers inside a cluster are involved which use different persistent storages as shared nothing system for *Input / Output* parallelization to reduce internal *Input / Output* costs, the *Input / Output* costs of persistent storages are higher than dealing with RAM (Random Access Memory) only. Therefore, reducing data transfer on any level is a sensitive key issue to speed up any application.

Concepts are needed which reduces data transfer. In-Situ processing is a concept which demands the processing of data where it is stored. Other concepts are edge- and fog-based networks. It is possible that entire networks pre-filter or pre-process certain information before it is sent to several archiving clusters for real-time processing in OLTP or directly to processing clusters for OLAP. Fog based solutions integrate this solution by processing pre-filtered or pre-processed data on OLTP or OLAP systems, furthermore. Pipelines can be set up which extracts and transforms data to create fewer data to be transported to the next processing unit, be the target an OLTP or OLAP system.

Focus of Cloud Computing:

1. Reduce local computing by efficient cluster computing in appropriate data warehouses through ITO (information technology outsourcing).

Focus of Edge and Fog computing:

1. Reduce data transfer across the network by outsourcing pre-filtering and pre-processing data-transformation steps, which reduce the size of the transferred data.

Edge, fog and cloud computing use heterogeneous DBS's. Concepts are needed to ensure efficient orchestration, modularity, interoperability, programmability, multiple application support and the support for different mixes of computation, networking, and storage. A global schema helps to unify the conceptual view of different sources and make data distribution more transparent. The ETL (extract transform load) process is usually used for data integration into data warehouses which use parallel databases for OLAP purposes. The ETL uses a global schema which supports the analysing purposes of the OLAP system.

Another data integration concept based on a global schema are *Mediator-Wrapper* architectures [57]. The *Mediator* takes queries and routs it to the *Wrapper* for the source. The *Wrapper* translates the query into a query which the source understands. The results are sent back to the *Mediator*, which integrates the data into the sink system. The application of this concept is limited to use for large data transfer [57]. The advantage of a *Mediator-Wrapper* architecture to use for a multi-database query processing architecture are summarized in [48] as follows:

> First, the specialized components of the architecture allow the various concerns of different kinds of users to be handled separately. Second, mediators typically specialize in a related set of component databases with "similar" data, and thus export schema and semantics related to a particular domain. The specialization of the components leads to a flexible and extensible distributed system. In particular, it allows seamless integration of different data stored in very different components, ranging from full-fledged relational DBMSs to simple files.

Schema integration and transformation is a complex task when dealing with complex global standards like *CityGML* (City Geography Markup Language) or *IFC* (industry foundation classes). Alternative concepts are provided by *peer-to-peer* architectures or *linked-data* concepts. *Peer-to-peer* architectures link the data between each node by a local schema or interface. *Linked-data* concepts may also use ontology concepts to link the data. With the help of query languages like *SPARQL* complex queries can be formulated which integrate the different data sources. Knowledge about the structure/ontology of the sources is required to formulate the queries [57].

2.2.4 Programming paradigms

The following sections introduce some basic programming paradigms for parallel and distributed communication. Most of them are implemented using Java or have a close relation to Java.

2.2.4.1 Service Oriented Architecture

Service Oriented Architecture (SOA) is a paradigm that allows a certain division of labour. Individual tasks are grouped according to topics and made available as services. Services can communicate with one another (machine-to-machine) via standardized communication interfaces (web services). More complex services delegate their tasks to more specialized services. Web services as implementation of a Remote Procedure Call (e.g. CORBA, RMI) may have high complexity due to a lot of freedom which leads to difficulties in distribution but high usage for parallelization since high performance clusters are their own ecosystem. Web services with Representational State Transfer (REST) simplifies communication with four basic operations (get, put, post and delete) on abstract resources.

Web services with Remote Procedure Call as Remote Method Invocation (Java):

1. Let Service Interface implement Remote Interface
2. Let Service Class implement Service Interface
3. Let Service Class extend UnicastRemoteObject Class
4. Start Server and bind Service Object on Registry
5. Start Client and lookup Service Object on Registry and use it

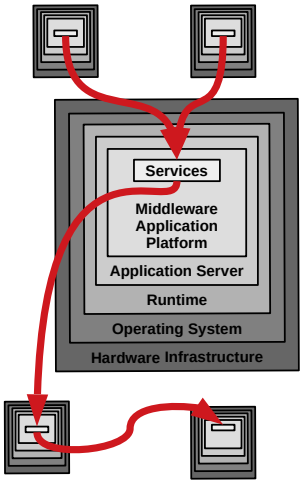


Figure 2.2: Data transfer using Service Oriented Architecture

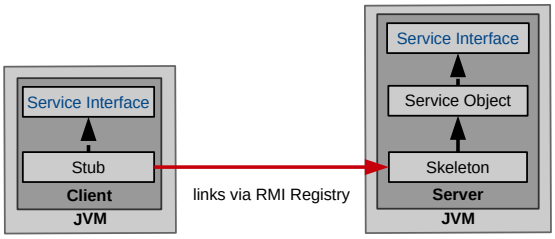


Figure 2.3: Java based Service Oriented Architecture

2.2.4.2 Microservice Architecture

As mentioned before, web services can get relatively complex that its administration will be a difficult task. The philosophy of microservices is to split up the web service into smaller pieces, which communicate with each other over API composition or through Command Query Responsibility Segregation (CQRS).

Over the past years, container management platforms like Kubernets or Docker established the support of:

1. auto-scalability
2. rescource allocation
3. replication
4. load balancing

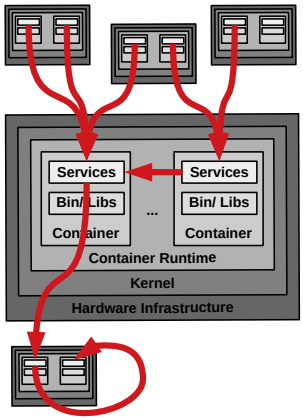


Figure 2.4: Data transfer using Microservices

"Containering" is a form of virtualization to encapsulate a piece of software inside an image which can be run virtually on any host system, which controls any permissions concerning hardware usage. In terms of microservice architectures, those container management platforms provide the ability to run a single web service by creating a container which runs the web service. This also implies that microservices do not run on the same data stock, but they can connect to a shared database.

2.2.4.3 The Streaming Architecture Kafka

In the following is a collection of helpful information taken from the Kafka documentation [35]:

> A streaming platform has three key capabilities:
>
> - Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
> - Store streams of records in a fault-tolerant durable way.
> - Process streams of records as they occur.
>
> Applications:
>
> - Building real-time streaming data pipelines that reliably get data between systems or applications
> - Building real-time streaming applications that transform or react to the streams of data

Concepts:

- Runs as a cluster on one or more servers that can span multiple datacenters.
- Cluster stores streams of records in categories called topics.
- Each record consists of a key, a value, and a timestamp.

Core APIs:

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

2.2.4.4 The Graph Computing Framework TinkerPop

In the following is a collection of helpful information taken from the TinkerPop documentation [65]:

> Apache TinkerPop™ is an open source Graph Computing Framework. Within itself, TinkerPop represents a large collection of capabilities and technologies and, in its wider ecosystem, an additionally extended world of third-party contributed graph libraries and systems.
>
> While TinkerPop holds some roots in Java, and thus, languages bound to the Java Virtual Machine (JVM), it long ago branched out into other languages such as Python, Javascript, .NET, and others. To compound upon that diversity, it is also seeing extensive support from different graph systems which have chosen TinkerPop as their standard method for allowing users to interface with their graph. Moreover, the graph systems themselves are not only separated by OLTP and OLAP style workloads, but also by their implementation patterns, which range everywhere from being an embedded graph system to a cloud-only graph. One might even find diversity parallel to Gremlin if considering other graph query languages.

Similar to computing in general, graph computing makes a distinction between structure (graph) and process (traversal). The structure of the graph is the data model defined by a vertex/edge/property topology. The processing of the graph is the means by which the structure is analyzed. The typical form of graph processing is called a traversal.

TinkerPop's role in graph computing is to provide the appropriate interfaces for graph providers and users to interact with graphs over their structure and process. When a graph system implements the TinkerPop structure and process APIs, their technology is considered TinkerPop-enabled and becomes nearly indistinguishable from any other TinkerPop-enabled graph system save for their respective time and space complexity.

TinkerPop provides two primary means of interacting with a graph: online transaction processing (OLTP) and online analytical processing (OLAP). OLTP-based graph systems allow the user to query the graph in real-time. However, typically, real-time performance is only possible when a local traversal is enacted. A local traversal is one that starts at a particular vertex (or small set of vertices) and touches a small set of connected vertices (by any arbitrary path of arbitrary length). In short, OLTP queries interact with a limited set of data and respond on the order of milliseconds or seconds. On the other hand, with OLAP graph processing, the entire graph is processed and thus, every vertex and edge is analyzed (some times more than once for iterative, recursive algorithms). Due to the amount of data being processed, the results are typically not returned in real-time and for massive graphs (i.e. graphs represented across a cluster of machines), results can take on the order of minutes or hours.

With Gremlin OLTP, the graph is walked by moving from vertex-to-vertex via incident edges. With Gremlin OLAP, all vertices are provided a VertexProgram. The programs send messages to one another with the topological structure of the graph acting as the communication network (though random message passing possible). In many respects, the messages passed are like the OLTP traversers moving from vertex-to-vertex. However, all messages are moving independent of one another, in parallel. Once a vertex program is finished computing, TinkerPop's OLAP engine supports any number MapReduce jobs over the resultant graph.

2.2.5 Relevance for the work

The following two paragraphs are published in [30] beforehand:

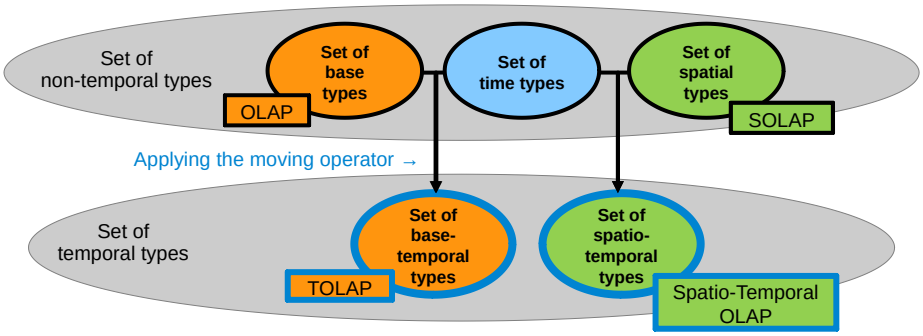
Both, OLTP and OLAP are of great importance for geo-spatio-temporal DMS (data-management-systems). While modern OLTP-applications may focus on real time processing of sensor data in sensor-networks, OLAP-applications focus on complex analytics of structured data, usually done in data warehouses. Both online processing paradigms hinder each other when working on the same datastock. Data warehousing can be differentiated by SOLAP (spatial OLAP, GIS interaction with OLAP), TOLAP (temporal OLAP, evolution of dimension instances available through the definition of temporal dimensions in OLAP), S-TOLAP (spatial TOLAP, GIS interaction with TOLAP), ST-OLAP (spatio-temporal OLAP, OLAP capabilities on spatio-temporal data-structures) and finally ST-TOLAP (spatio-temporal TOLAP, TOLAP capabilities combined with spatio-temporal data-structures) [66].

Further investigations need to be done concerning the speed-up and scale-up of parallel processing in ST-TOLAP and the physical or virtual data integration. Some operations on specific spatio-temporal data distributions might result in too large intra-communication or synchronization overheads. Therefore, how the data is being distributed, the load-balancing and sharding, has impact on the processing time of spatio-temporal queries. For example, if algorithms depend on the local neighbourhood of spatial objects the local neighbourhoods need to be accessible within one node/blade as far as possible to reduce intra-communication (e.g. simulations, interpolations etc.). As a second example, if algorithms depend on distributed local neighbourhoods of spatial objects the local neighbourhoods should be distributed across the cluster such that every node/blade has nearly the same work load to query or calculate all pieces (distributed R-Tree for searching in parallel or editing etc.). In both cases, reading data from other nodes will turn into a massive communication between the blades for certain tasks within a Shared-Nothing-System or synchronization efforts within a Shared-Disk-System. A good topological structure of the geo-scientific data will help the controlled distribution.

Figure 2.5 shows an example query for ST-TOLAP. This example shows how complex a query for ST-TOLAP can be. In connection with the last chapter, it is to remark that besides the OLAP clusters for cloud computing, the OLTP solutions and edge- or fog-computing may be applied.

The definitions of the different kinds of OLAP can be applied to OLTP also as SOLTP (spatial online transaction processing), TOLTP (temporal online transaction processing), S-TOLTP (spatial temporal online transaction processing), ST-OLTP (spatio-temporal online transaction processing) and ST-TOLTP (spatio-temporal temporal online transaction processing).

The diversity of DBMS's showed basic data management concepts which may be used by some data providers. Parallel processing may speed up any application under certain constraints. One major issue is the availability of data to each process. Edge and fog



Query example for ST-TOLAP:

Get the total number of days when the Gent district has been under at least one cloud of carbon monoxide such that the average load in the cloud is larger then the load limit at the time when the cloud appeared.

Figure 2.5: The combination of data types for ST-TOLAP and an example query.

computing has been invented to reduce data transfer. But those concepts need to be supported by efficient programming paradigms to create efficient communication within the network. The previously described programming paradigms show that API's and frameworks are the major communication concepts when digging deeper into database programming and their communication. Therefore, the support of ETL or the alternatives (*peer-to-peer* architectures or *linked-data* concepts) is done through efficient API's and frameworks.

2.3 Geo-Scientific Data Management

File based geo-information systems and data transfer are still common. The data within the files may be text or binary. If the structure of the file is unknown, special software is needed to open those files. If the structure of the file is known, it is possible to implement importers which extract and transform the data into the data structure needed. Modern data providers use databases which provide services for data extraction. It is possible to write importers if a database driver / interface of the database system exists. The use of databases is still not a standard in various fields including field data acquisition in geography, remote sensing and geodesy. Geo-information science used to focus on the development of GIS (geographic information systems). GIS's were traditionally based in 2D space. There is still a lack of supporting 3D or 4D space to manage spatial or spatio-temporal data.

The following paragraph is published in [30] beforehand:

A brief discussion about distributed and parallel data in conjunction with GIS is given in [26]. A generic model for spatio-temporal data is presented in [47]. In [11] a topological model for data without restriction on dimension (including spatial or temporal) is presented based on the notion of Alexandrov topology.

The first subsection introduces incidence graphs and topological consistence. The following subsections introduce basic spatial and spatio-temporal data models and structures which find application within the geo-information sciences in order to manage and process spatial and spatio-temporal data.

2.3.1 Incidence Graph and Topological consistence

This section was written by Patrick Erik Bradley as mathematical rudiments for parts of the developed models of this work and is to be found in our paper on topologically consistent models for efficient big geo-spatio-temporal data distribution [30].

> The use of simulation models for environmental processes, infrastructure, noise or flooding require 3D-models including tetrahedra [41, 12, 62, 69]. However, it is topology that captures the relationships between entities, be they geometric or not. The use of topology in geo-informatics is widespread, leading to a vast literature, beginning with the application of point-set topology in [20]. The idea there is to encode various relationships between regions, deemed topological. Already, P. Alexandrov proved that any binary relation gives rise to a topology on a finite set of objects [1], and this led to the notion of topological databases in [11]. The article [13] reviews present and future directions for the application of topology in the management of geo-data.

An important notion for having data suitable for simulations is that of topological consistency, as defined in [25], where also competing definitions from the literature can be found. In that article, it was found that *CityGML* (City Geography Markup Language) data is usually not topologically consistent. Definition 2.3.1 below paraphrases this notion, which is in compliance with the ISO 19107 standard. The notion from [42] comes close to the definition used here.

Definition 2.3.1. *A configuration of geometrical objects in Euclidean space is* topologically consistent*, if two objects are either disjoint or identical.*

The main advantage of this definition is that it allows any topological query to be performed on the topological model itself without having to resort to geometry with its costly computations like intersection and others. Having such a notion of topological consistency, healing as e.g. in [70] becomes conceptually more feasible. In this work, this becomes an important pre-processing step using computational geometry in order to achieve topologically consistent data. This gives rise to numerical issues with an accumulation of floating-point errors, as already seen e.g. in [6].

Pavel Alexandrov showed that any partial ordering on a finite set gives rise to a so-called T_0-topology and vice versa [1]. This very general kind of topology captures the situations where no directed circuit paths are allowed. Its flexibility can capture a lot of applications. A particular instance is the boundary relationship between geometric objects.

Given a possibly topologically inconsistent boundary representation of a 3D model, i.e. a model not satisfying Definition 2.3.1, the overlay method below produces a topologically consistent finite T_0-space which captures the topology inherent in the geometric model. This model has a minimal representation as a directed acyclic graph, called *Hasse diagram*, such that the reflexive and transitive closure of the edge relationship yields the topology in the sense of [1]. Figure 2.6 illustrates this for the example of a triangle: The top node represents the area which is bounded by its three side edges, represented by the middle row of nodes, and each of these are bounded by two corners represented by the lower row of nodes. The edges are oriented downwards and represent the boundary relationship between objects.

In [11] it was observed that this general structure can be implemented in a relational database with one table for the ‘nodes’ and another table for the direct ‘relationships’, and the reflexive and transitive closure operation can be used to make topological queries on this database. In applications, any entities considered as ‘nodes’ and any binary ‘relationship’ thus generates a topology, thus allowing topological methods to be applied in this setting, even if the entities do not carry geometric or topological information

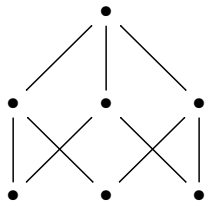
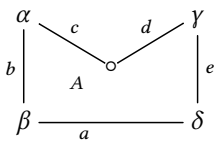


Figure 2.6: *Hasse diagram* of a triangle with its boundary relation.

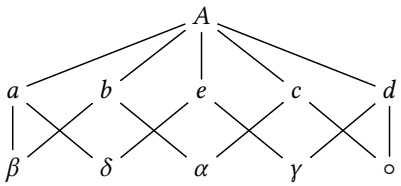
themselves. The most efficient way to store a T_0-topology is in a graph database which stores the nodes and edges of the *Hasse diagram*, as this avoids the storing of redundant relationships. In this case, topological queries like neighbourhoods or connectivity become path queries in a graph.

In the literature there are various differing definitions of the notion "topological consistency", cf. e.g. [43, 37, 59, 18]. One point of view is found in [18], in which topological consistency usually refers to the lack of topological errors, like unclosed polygons or dangling nodes. Another point of view may define it as the equality of the topological model with the topology derived from geometry in a certain way. The idea is that a model is consistent, if and only if it is properly embedded into Euclidean space. The main advantage of this definition is that in this case, costly geometric computations can be avoided in topological queries by using the topological model only. This notion of topological consistency then guarantees the correctness of topological query results.

Let P be a n-dimensional polytope. We associate to P the following finite topological space $X(P)$, which we call cell space of P: the points of $X(P)$ are the interiors of all k-faces of P for $k = 0, \ldots, n$. The topology on $X(P)$ is the one generated by the bounded-by relation $>$ on the open faces: $a > b$ (or $b < a$) if b is in the boundary of a. For example, if P is a polygon (we call it $\mathcal{P}_1$):

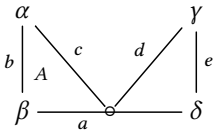


then $X(P) = X(\mathcal{P}_1)$ can be depicted as:



In the literature, the topology generated by the bounded-by relation is often called incidence graph. It is a so-called finite T_0-space or poset. The T_0 indicates that the relation $>$ yields an acyclic graph structure on the points of $X(P)$. An introduction to finite topological spaces can be found in [3].

A geometrical realization of a polytope can be obtained, e.g. by assigning coordinates to the vertices of a boundary-representation model. However, if not enough care is taken, then one can obtain something like this (we call it $\mathcal{P}_2$):



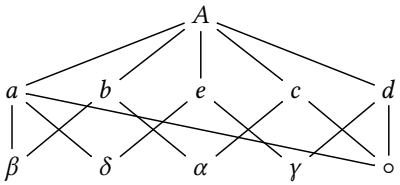
Here, there is a topological inconsistency: in the bounded-by topology, $\circ$ is in the boundary of the slanted edges only. However, in this geometric realization, $\circ$ is in the boundary also of the punctured horizontal edge. Another problem is that in this geometric realization, the interior is disconnected. This is not what we usually think of as a polytope (or here: polygon). In any case, we can construct another finite topological space $\bar{X}(P)$ which extends $X(P)$ as follows: the points are all non-empty intersubsections $a \cap b$ for $a, b \in X(P)$. The relation $\prec$ is defined as follows: Let $a, b \in X(P)$, then $a \prec b$ if $a < b$, and for $i = a \cap b \neq \emptyset$: $i \prec a$ if $i \neq a$. The space $\bar{X}(P)$ is called the overlay space of P. We say that P is topologically consistent if the overlay space $\bar{X}(P)$ coincides with the space $X(P)$. In the above inconsistent example, we have that $\circ \prec a$ in addition to $<$ in the overlay space. In any case, the overlay space $\bar{X}(P)$ is also a finite T_0-space.

This notion of topological consistency was introduced in [7] in a slightly different formulation in the context of configurations of polygons. Here, we can also consider configurations of objects of the following kind: First of all, we can define $X = X(P_1 \cup \cdots \cup P_\ell)$ and the overlay space $\bar{X} = \bar{X}(P_1 \cup \cdots \cup P_\ell)$ for polytopes $P_1, \ldots P_\ell$ in the same way as for a single polytope. Again, we call $P_1 \cup \cdots \cup P_\ell$ topologically consistent if $X = \bar{X}$. Then, a n-primitive $\mathcal{P}$ is the interior of a polytope P of dimension n minus the union H of finitely

many polytopes of dimension $\leq n$, provided that this union is topologically consistent. A primitive is defined as a n-primitive for some n. We call the closure of $P \setminus H$ the closure $\mathrm{cl}\,\mathcal{P}$ of $\mathcal{P}$, and write $X(\mathrm{cl}\,\mathcal{P})$ for the set of interiors of the faces of $P \setminus H$ including $\mathcal{P}$. Again, the overlay space $\bar{X}(\mathrm{cl}\,\mathcal{P})$ can be defined, together with topological consistency.

The final step is to build up spaces from primitives: let $C = \mathrm{cl}(\mathcal{P}_1) \cup \cdots \cup \mathrm{cl}(\mathcal{P}_\ell)$ be such a space. Then again we extend the definition of the cell space to obtain $X(C)$, and the overlay space $\bar{X}(C)$. Notice that the elements of $X(C)$ are in general not open cells in the sense of topology, but can be viewed, like in the case of cell complexes, as building blocks for a space C.

In our example of the topologically inconsistent polygon $\mathcal{P}_2$, $\bar{X}(\mathcal{P}_2)$ has the same points as $X(\mathcal{P}_2)$. The difference is in the topology: $\bar{X}(\mathcal{P}_2)$ can be depicted as



If the goal is to decide whether a geometric realization of a space is topologically consistent or not, then it is enough to find the difference between X and $\bar{X}$. However, if one wants to tell by how much the geometric realization is inconsistent, one can compute the Betti numbers of the skeleta of X and $\bar{X}$ and compare them. The Betti numbers b_i of a simplicial complex can be intuitively interpreted as the number of i-dimensional holes, where b_0 is the number of connected components, b_1 the number of loops, b_2 the number of voids etc. For a finite poset X, there also exist Betti numbers by associating to X the order complex $K(X)$, a simplicial complex whose k-simplices are the chains $a_0 < \cdots < a_k$ of length k. Then $b_i(X)$ is defined as $b_i(K(X))$.

The dimension of a poset X is the length of the longest chain in X. In [10], this is seen as a form of Krull dimension. If the dimension of X is n, then the $n-1$-skeleton X_{n-1} is obtained from X by removing all points which are at the top of a chain of length n. Iterating this process yields the skeleta X_k with $k = 0, \ldots, n$, where $X_n = X$. We now define the numbers $b_i^k(X) = b_i(X_k)$ as the Betti numbers of the skeleta of X. If $X = X(C)$ for a space as above, then we finally can define the topological defect numbers as

$$\mu_{k,i}(C) = b_i^k(X(C)) - b_i^k(\bar{X}(C))$$

In our example, we have

$$\mu_{k,0}(\mathcal{P}_2) = 0, \quad k = 0, 1, 2 \tag{2.1}$$
$$\mu_{1,1}(\mathcal{P}) = 1, \quad \mu_{2,1}(\mathcal{P}_2) = 0 \tag{2.2}$$

which can be seen as follows: $X_2(\mathcal{P})$, $X_1(\mathcal{P}_2)$, $\bar{X}_2(\mathcal{P}_2)$ and $\bar{X}_1(\mathcal{P}_2)$ are all connected, and $X_0(\mathcal{P}_2) = \bar{X}_0(\mathcal{P}_2)$. This implies (2.1). In order to see the (2.2), observe that $X_1(\mathcal{P}_2)$ is the loop-graph with vertices $a, b, c, d, e, \alpha, \beta, \gamma, \delta, \circ$, which has $b_1(X_1(\mathcal{P}_2)) = 1$; and $\bar{X}_1(\mathcal{P}_2)$ has two loops with the same vertices, and its first Betti number is two. We have used the fact that for 1-dimensional posets, the Betti numbers coincide with the Betti numbers of the underlying graphs, the reason being that the order complex of a 1-dimensional poset is a graph. This explains the first part of (2.2). In higher dimension, things are not quite so simple, but if a poset has a unique maximal element, then all higher Betti numbers vanish (as such a space is contractible) [3]. This explains the second part of (2.2).

2.3.2 Euler characteristics

The Euler characteristic can be defined by the Betti numbers as follows:

Definition 2.3.2 (Euler characteristic topological)**.** *Let X be a n-dimensional cell complex built of only finitely many cells. Let $b_i(X)$ be the i-th Betti number of X with $i = 0, \ldots, n$. The topological Euler characteristic can be defined as:*

$$\chi_{top}(X) = \sum_{i=0}^{n} (-1)^i b_i(X)$$

The Euler characteristic can also be defined by the number c_m of m-dimensional cells of X with $m = 0, \ldots, n$ as follows:

Definition 2.3.3 (Euler characteristic combinatorial)**.** *Let X be a n-dimensional cell complex built of only finitely many cells. Let c_m be the number of m-dimensional cells of X with $m = 0, \ldots, n$. The combinatorial Euler characteristic can be defined as:*

$$\chi_{com}(X) = \sum_{m=0}^{n} (-1)^m c_m$$

Both definitions are equal if the cell complex X with only finitely many cells is topologically consistent (which is true by the definition of cell complex) and the cells of the cell complex X do not have holes. Practically, problems may occur due to computational geometry based on standard computational arithmetic (e.g. double precision) when

calculating the geometrically induced topological overlay space $\bar{X}$. Some intersections may not be recognized and others may not exist. Therefore, the results happen to be topologically inconsistent and the correctness of both Euler characteristics of those geometrically calculated unions are not always true and, furthermore, they may differ.

A n-dimensional sphere is the boundary of a $n+1$-dimensional ball. A cellulisation is a decomposition of a manifold to finitely many cells. A cellulisation of a ball can be deformed by flattening each cell to a polyhedron. Euler's polyhedron formula states that any 3-dimensional polyhedron has Euler characteristic two. The generalized formula to n-dimensional spheres is [36]:

Theorem 2.3.1 (general Euler's polyhedron formula)**.** *Let X be a n-dimensional cell complex, which is a cellulisation of a n-dimensional sphere. Then it holds true that:*

$$\chi_{com}(X) = 1 + (-1)^n$$

Therefore, a segment complex in the shape of the border of a polygon (1-dimensional sphere) has Euler characteristic zero. A triangle complex which is the triangulation of a polyhedron (2-dimensional sphere) has Euler characteristic two (original Euler's polyhedron formula with $n = 2$). A tetrahedron complex in the shape of a 3-dimensional sphere has Euler characteristic zero etc.

2.3.3 Boundary Representation

BREPs (Boundary Representation) are a common way of describing the geometry of spatial objects. An example is a polygon in 2D space or a planar polygon in 3D space. They are often used to represent surfaces only by memorize the boundary elements of the surface. This approach saves memory but causes some numerical problems.

As an example, the set of planar polygons in a discrete 3D space due to computer arithmetic is relatively rare. The planarity constraint is important to define the inner points of the surface. If the planarity constraint would not be required, there would not be a definition for the inner points and the surface could be anywhere in the surrounding 3D space. The same problem arises when defining a *BREP* for a moving and morphing surface by a closed (without border) surface (hull) in 4D space. Without the definition of the inner points, the moving and morphing surface could be anywhere in the 4D space with the boundary of that hull. Point in polygon/convex hull tests are also extremely expensive.

BREPs are a common representation in computer graphics and GIS. One example are convex hull meshes to represent volumes. Those meshes consist of a set of triangles. Another example can be seen in *CityGML* (City Geography Markup Language). Complex volumes are represented by a set of planar polygons which are supposed to form a hull. The hull is the first level *BREP*. The polygons themselves are also *BREPs* to represent planar surfaces. These are the second level *BREPs*. To compute the geometrically induced topology is not a trivial task.

2.3.4 Simplicial and polytope complexes

This section is a translation of parts of the final report of the DFG Project "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) published in [30] beforehand:

Another spatial model is available in order to precisely define the inner points of *BREPs*. The spatial model uses 0,…,3-dimensional simplices to support computational geometry tasks more efficiently. For a valid spatial simplex $c = \{p_0, \ldots, p_d\}$, $p \in \mathbb{R}^3$ (spatial space) of dimension $d := 0, 1, 2, 3$ at a certain time-stamp $t \in \mathbb{R}$ (temporal space) let $\bar{p}_i = p_i - p_0$ for $0 \leq i \leq d$.

The constraint:

$$d = dim(span(\bar{p}_0, \ldots, \bar{p}_d)) \tag{2.3}$$

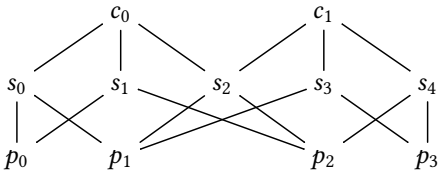
has to hold. The topological features are based on simplicial complexes with further constraints. As in most GIS geometry cores, the data types are classified by their dimension. Each 0,…,3-dimensional spatial complex is a well-defined topological object.

For a valid spatial complex C containing some simplices (or cells/polytopes, in general) the following constraints have to hold:

$$C \text{ is topologically consistent} \tag{2.4}$$
$$\text{All maximal cells are } d\text{-cells for } d \geq 0 \tag{2.5}$$
$$\text{Betti numbers } b_0 = 1 \text{ and } b_i = 0 \text{ for } i > 0 \tag{2.6}$$
$$\text{Every } (d-1)\text{-cell is boundary element of two } d\text{-cells only} \tag{2.7}$$
$$\text{All } d\text{-cells are equally oriented} \tag{2.8}$$

The notion equals to the notion used for topological consistence, as defined in Section 2.3.1. Constraint 2.5 states that all cells of a complex C are d-dimensional. It has less-dimensional sub-cells but only for topological reasons as follows. Constraint 2.6 claims that there is only one connected component and there are no holes of any dimension within a complex C. Constraint 2.7 implies a One-To-One relation of neighbouring d-dimensional cells sharing one $(d-1)$-dimensional boundary cell [14]. This constraint ensures that a complex C is a d-dimensional manifold. And constraint 2.8 ensures that every complex C is an oriented manifold.

Spatial simplices and their sets exist at certain time-stamps only (so across $\mathbb{R}^3$ only) and because of that no temporal neighbours other than themselves can be identified. Navigating through spatial complexes or even over neighbouring spatial complexes sharing boundary elements is partly done by the *G-Maps* paradigm [15]. The topological incidence graph for two neighbouring spatial triangle-simplices $c_0, c_1 \in C$ forming a valid spatial complex at some time-stamp by the new object model looks like this:



It is to mention that the segments s_0 to s_5 are not referenced by the spatial simplices of the geometry-model. Spatial simplices are defined on their points only. The edges or faces can be calculated at runtime if needed. But the cell-nodes could be instantiated to realize the topological incidence graph. How the spatial parts of those additional boundary cells are going to be integrated is a question of the applicational needs.

With the help of spatio-temporal point tubes, we are able to move and morph spatial simplices over time [14]. Spatio-temporal point tubes are interpolation functions from $\mathbb{R}$ (temporal space) and a set of points in $\mathbb{R}^4$ (spatio-temporal space) to $\mathbb{R}^3$ (spatial space). They replace all $p_i \in \mathbb{R}^3$ (spatial space) from a spatial simplex in the spatial model at time-stamp $t \in \mathbb{R}$ (temporal space) with a function f over t and a set of $p_{ij} \in \mathbb{R}^4$ (spatio-temporal space) for $0 \leq i \leq d$ and $j \in \mathbb{N}$. So each p_i becomes a function $f(t, p_{i0}, \ldots, p_{in})$ representing the i.th spatio-temporal point tube at a time-stamp t for n points $p_{ij} \in \mathbb{R}^4$.

For a valid spatial simplex and spatial complex on spatio-temporal point tubes, the above typical spatial model constraints have to hold, too. In this case, for a valid spatial simplex $c(t) = \{f(t, p_{00}, \ldots, p_{0n}), \ldots, f(t, p_{d0}, \ldots, p_{dn})\}$ of dimension $d := 0, 1, 2, 3$ in $\mathbb{R}^3$ with the explained definitions above, let $\bar{f}(t, p_{i0}, \ldots, p_{in}) = f(t, p_{i0}, \ldots, p_{in}) - f(t, p_{00}, \ldots, p_{0n})$.

The constraint:

$$d = dim(span(\bar{f}(t, p_{00}, \ldots, t, p_{0n}), \ldots, \bar{f}(t, p_{d0}, \ldots, t, p_{dn}))) \tag{2.9}$$

has to hold for each time slice at $t \in \mathbb{R}$. For a valid spatial complex C containing some cells of that kind, the above constraints 2.4, 2.5, 2.6, 2.7 and 2.8 have to hold for each time slice at $t \in \mathbb{R}$, respectively. But for each spatial complex defined like that, we are still missing true spatio-temporal polytopes to set up some spatio-temporal topology easily.

A spatio-temporal polytope complex is a collection of spatio-temporal polytope sequences for the spatial space coordinates combined with one single temporal-sequence for the temporal space coordinates. All spatio-temporal polytope sequences within a spatio-temporal polytope complex share the same temporal-sequence to reduce memory costs. A temporal-sequence is a linearly sorted interconnected collection of temporal-intervals, where else each single spatio-temporal polytope sequence describes the movement and deformation of one spatial simplex over time by temporally linear sorting interconnected spatio-temporal polytopes. Therefore, a single spatio-temporal polytope within a spatio-temporal polytope sequence could be seen as one change in movement and/or shape of one single spatial simplex by referencing a spatial pre-simplex for the first time-stamp $t_0 \in \mathbb{R}$ (temporal space) of some temporal-interval referenced by the temporal-sequence of the spatio-temporal polytope complex it is being part of and a moved/morphed spatial post-simplex at the second time-stamp $t_1 \in \mathbb{R}$ of the same temporal-interval. The next spatio-temporal polytope within the spatio-temporal polytope sequence shares the last spatial post-simplex of the last spatio-temporal polytope as spatial pre-simplex and adds a new changed spatial post-simplex to itself. Therefore, special kinds of spatio-temporal point tubes mentioned in the model defined above do exist implicitly. Anyway, all spatial pre- or post-simplices belonging to the same time-stamp of each spatio-temporal polytope sequence within a spatio-temporal polytope complex form a spatial complex as a spatial topological constraint.

For a valid spatio-temporal polytope $c = \{\hat{p}_0, \ldots, \hat{p}_{2d-1}\}$ in $\mathbb{R}^4$ with dimension $d := 1, 2, 3, 4$, points $\hat{p}_i = \begin{pmatrix} p_i \\ t_{i\%2} \end{pmatrix} \in \mathbb{R}^4$ (spatio-temporal space), $p_i \in \mathbb{R}^3$ (spatial space), two time-stamps $t_0, t_1 \in \mathbb{R}$ (temporal space) with $t_0 \neq t_1$, $0 \leq i \leq 2d-1$ and "%" as modulo operator let $\bar{p}_i = \hat{p}_i - \hat{p}_0$.

The constraint:

$$d = dim(span(\bar{p}_0, \ldots, \bar{p}_{2d-1})) \tag{2.10}$$

has to hold, where the points $\hat{p}_i$ containing the temporal coordinate t_0 belong to the spatial pre-simplex and the points containing t_1 belong to the spatial post-simplex. For a valid spatio-temporal complex C containing some cells of that kind the above constraints 2.4, 2.5, 2.6, 2.7 and 2.8 have to hold, the same way as the spatial complex constraints of the spatial model but in $\mathbb{R}^4$. Spatio-temporal polytopes of that kind and their collections/sets may exist over spatial intervals (not in case of points) but have to have a temporal expansion. They are able to have temporal neighbours at the time-stamp of their temporal boundary, and we are able to set up a spatio-temporal topological incidence graph by the new object model. As an example for a valid spatio-temporal segment-polytope complex C (see Figure 2.7) with two neighbouring spatio-temporal segment-polytopes $c_0, c_1 \in C$

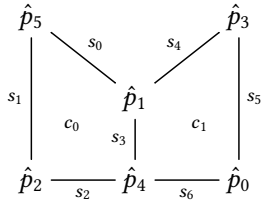
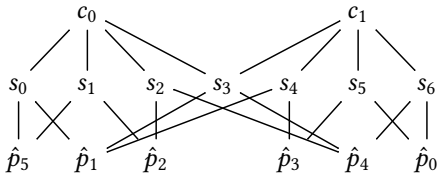


Figure 2.7: Valid spatio-temporal segment-polytope complex C for two neighbouring spatio-temporal segment-polytopes $c_0, c_1 \in C$ with $s_2, s_0 \in c_0$ and $s_6, s_4 \in c_1$ and s_2, s_6 as spatial pre-segment-simplices and s_0, s_4 as spatial post-segment-simplices and s_1, s_3, s_5 as implicit linear spatio-temporal point tubes

the spatio-temporal topological incidence graph will be:



With the help of spatio-temporal point tubes, we are able to optimize the spatio-temporal polytope complexes to a more dynamic and temporal scaleable data set and to keep the benefits of a true spatio-temporal topology. Furthermore, we are able to simplify the class-hierachy by removing the polytope sequence class used to organize for moving and morphing of a single spatial simplex. In this case, a spatio-temporal polytope complex may be a collection of polytopes only and querying a polytope sequence is a spatio-temporal topological algorithm and not a return of a specialized spatio-temporal polytope sequence instance/object within a spatio-temporal polytope complex. Anyway, the definition follows the spatial simplex model on spatio-temporal point tubes. We evaluate the spatio-temporal point tubes at special time-stamps $t \in \mathbb{R}$ (temporal space) but not only for one simplex at time-stamp t for the materialization of a simplex at the specific time-stamp but with evaluating two time-stamps $t_1, t_2 \in \mathbb{R}$ with $t_1 \neq t_2$ and use the returned spatial coordinates for the pre-, respectively post-simplex of the spatio-temporal polytope. The spatio-temporal polytope complexes in that form are an implementation of the Polthier and Rumpf model [53].

2.3.5 Relational Model and hierarchical G-Maps

The final report of the DFG Project "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) contains translated extractions of this section:

As mentioned in the section on simplicial and polytope complexes (Section 2.3.4), the simplices or polytopes are defined via their points. Higher-dimensional boundary elements may be calculated automatically. The order of the individual points also play a role in the spatio-temporal model. For spatial simplicial complexes at a given point in time t, the orientation condition 2.8 in Section 2.3.4 must be maintained for every point in time t. Therefore, the order of the implicit point tubes forms a basis for the orientation. For the higher-dimensional polytopes, this orientation forms the basis for the higher-dimensional orientation of the polytopes. In the example from Figure 2.7 the polytope c_0 must have the orientation $\hat{p}_2 \rightarrow \hat{p}_4 \rightarrow \hat{p}_1 \rightarrow \hat{p}_5$, the polytope c_1 the orientation $\hat{p}_4 \rightarrow \hat{p}_0 \rightarrow \hat{p}_3 \rightarrow \hat{p}_1$ and the segment s_6 the orientation $\hat{p}_4 \rightarrow \hat{p}_0$ if the segment s_0 has the orientation $\hat{p}_2 \rightarrow \hat{p}_4$. These paths can also be seen in the incidence graph of the example. The segment polytopes c_0 and c_1 then form an oriented 2D manifold in $\mathbb{R}^4$. The orientations of triangular polytopes (prism) in spatio-temporal triangular complexes (prism complexes) or tetrahedron polytopes (see Figure 2.8) in spatio-temporal tetrahedron complexes can also be derived.

In order to save a topology in the form of a table (*G-Map* structure) of a tetrahedron polytope, 192 entries (darts) are required, without any connection to the universe or the surrounding emptiness. There are 24 entries per spatial boundary tetrahedron (see Figure 2.10 left) at times t_0 (pre-tetrahedron) and t_1 (post-tetrahedron) and 36 entries per spatio-temporal boundary prism (see Figure 2.10 on the right), so a total of $4 \times 36 = 144$ entries plus $2 \times 24 = 48$ entries (192). With the connections to the universe or the

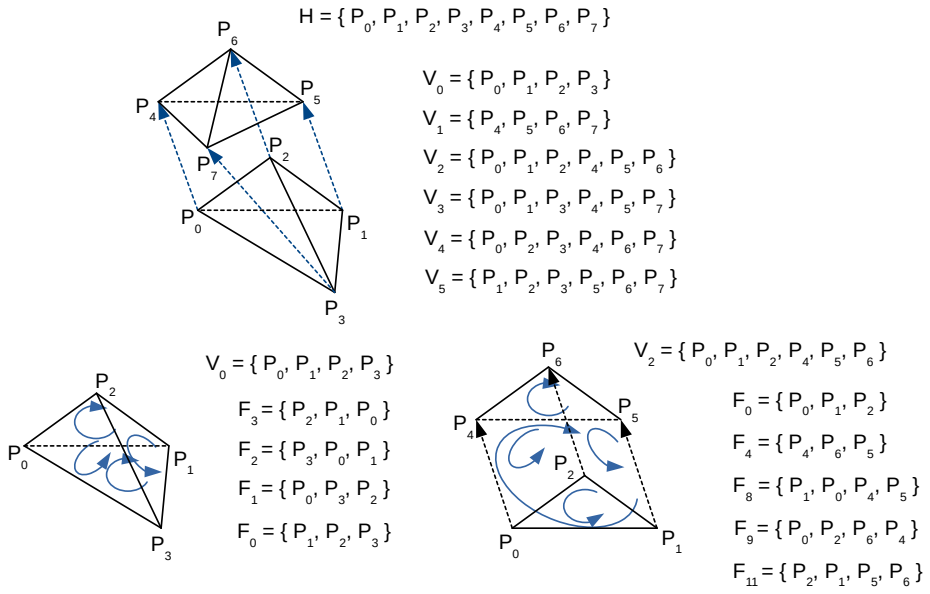


Figure 2.8: Valid tetrahedron polytope H with the six boundary volumes (top) and two boundary volumes with valid orientations (bottom). The orientation of the triangle F_0 is different in both boundary volumes.

surrounding emptiness, there are $2 \times 192 = 384$ entries for the table representing the topology of a tetrahedron polytope. The table has a relatively large number of redundant entries. It was proven that *G-Maps* tend to become verbose with increasing dimension [9]. Object-oriented or graph-based solution in the form of an incidence graph is ideal. The complete table summarizes all *G-Maps* tuples (darts) of a spatio-temporal tetrahedron polytope and shows all the different vertical paths of the incidence graph, the maximal directed path in the *Hasse diagram*, cf. Figure 2.9. It should be pointed out that a spatio-temporal triangle polytope is a subset of such a spatio-temporal tetrahedron polytope and a spatio-temporal segment polytope is a subset of the spatio-temporal triangle polytope.

Navigation via spatio-temporal *G-Maps* over a polytope complex of this type is possible by introducing involutions (see [15]). There are five different involutions ($\alpha_0 \ldots \alpha_4$) for tetrahedron polytopes, four different for triangular polytopes, three different for segment polytopes and two different for point polytopes. The new α_4-Involution navigates over the same spatial boundary elements of the tetrahedron-polytopes, so that with an α_4-involution it is possible to navigate from the tetrahedron-polytope to the neighbouring-tetrahedron-polytope. These neighbouring tetrahedron polytopes can be defined over the same time interval (navigation was then carried out using a boundary prism) or placed before or after a polytope sequence (navigation was then carried out using the boundary tetrahedron, the pre- or post-tetrahedron). The new α_0-involution for point

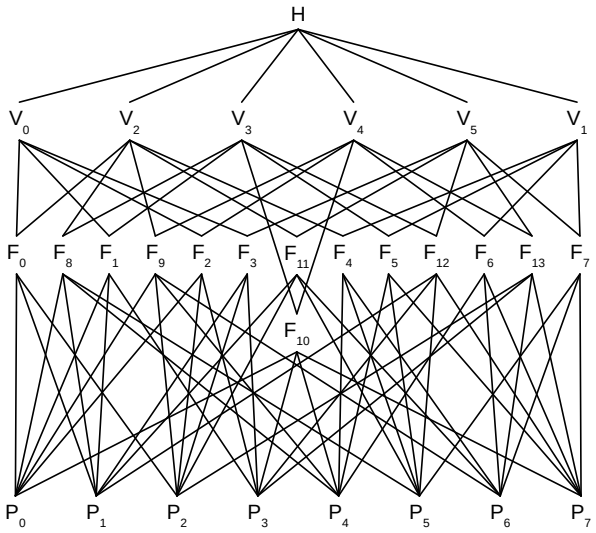


Figure 2.9: Incidence graph of the valid tetrahedron polytope H (see Figure 2.8) without boundary segment nodes

polytopes navigates from boundary point to boundary point of a point polytope, and the new α_1-involution for point polytopes navigates via a common boundary point from a point polytope to a neighbour point polytope.

$$\begin{bmatrix} P & S & F & V \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \\ 1 & 4 & 1 & 0 \\ 3 & 4 & 1 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} P & S & F & V \\ 0 & 1 & 2 & 0 \\ 2 & 1 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 3 & 2 & 2 & 0 \\ 2 & 5 & 2 & 0 \\ 3 & 5 & 2 & 0 \\ 1 & 3 & 3 & 0 \\ 2 & 3 & 3 & 0 \\ 1 & 4 & 3 & 0 \\ 3 & 4 & 3 & 0 \\ 2 & 5 & 3 & 0 \\ 3 & 5 & 3 & 0 \end{bmatrix} \quad \begin{bmatrix} P & S & F & V \\ 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 2 & 1 & 0 & 2 \\ 1 & 3 & 0 & 2 \\ 2 & 3 & 0 & 2 \\ 4 & 6 & 4 & 2 \\ 5 & 6 & 4 & 2 \\ 4 & 7 & 4 & 2 \\ 6 & 7 & 4 & 2 \\ 5 & 9 & 4 & 2 \\ 6 & 9 & 4 & 2 \\ 0 & 0 & 8 & 2 \\ 1 & 0 & 8 & 2 \\ 4 & 6 & 8 & 2 \\ 5 & 6 & 8 & 2 \\ 0 & 12 & 8 & 2 \\ 4 & 12 & 8 & 2 \\ 1 & 13 & 8 & 2 \\ 5 & 13 & 8 & 2 \end{bmatrix} \text{ and } \begin{bmatrix} P & S & F & V \\ 0 & 1 & 9 & 2 \\ 2 & 1 & 9 & 2 \\ 4 & 7 & 9 & 2 \\ 6 & 7 & 9 & 2 \\ 0 & 12 & 9 & 2 \\ 4 & 12 & 9 & 2 \\ 2 & 14 & 9 & 2 \\ 6 & 14 & 9 & 2 \\ 1 & 3 & 11 & 2 \\ 2 & 3 & 11 & 2 \\ 5 & 9 & 11 & 2 \\ 6 & 9 & 11 & 2 \\ 1 & 13 & 11 & 2 \\ 5 & 13 & 11 & 2 \\ 2 & 14 & 11 & 2 \\ 6 & 14 & 11 & 2 \end{bmatrix}$$

Figure 2.10: (left) Darts of the d-dimensional boundary elements of the pre-tetrahedron V_0 of the valid tetrahedron polytope H from Figure 2.8 (bottom, left), (right) Darts of the d-dimensional boundary elements of the spatio-temporal boundary prism V_2 of the valid tetrahedron polytope H from Figure 2.8 (bottom, right)

2.3.6 Redundancy

The final report of the DFG Project "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) contains translated extractions of this section and is published in [30] and [32] beforehand:

There are a couple of different ways of dealing with spatio-temporal changes technically. Every case has benefits for keeping track of spatio-temporal topological consistency. If a temporal change always leads to some changes in position and/or shape of some spatial simplex by definition, the spatio-temporal polytope complex needs to be split if at least one spatial simplex of the spatio-temporal polytope complex does not change over time because it contradicts with the definition. Splitting this kind of polytope complex leads to partial redundant temporal-sequences or redundant referencing of equal temporal-sequence parts. A second definition could be that no spatial changes are allowed by ongoing time, but this would lead to redundant referencing of the same simplex (or

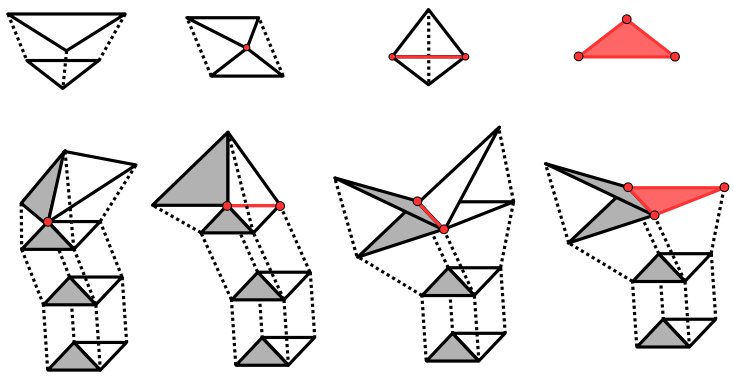


Figure 2.11: Triangle-polytope (top) spatial redundancies (red) and triangle-polytope complex (bottom) spatial redundancies (red).

parts of it) within the polytope as pre- and post-simplices (or pre- and post-parts of it, respectively). In general, redundancies are always to be expected at the spatio-temporal boundary polytopes to which at least one neighbour does not change spatially, since the common points of the boundary polytope do not spatially change with the neighbours that do not change spatially.

Figure 2.11 shows possible spatial redundancies of a triangle-polytope (top) and a triangle-polytope complex (bottom). The spatio-temporal volume of the spatially redundant triangle polytope (right, top) is the area of the triangle times the length of the associated time interval. The spatio-temporal area of an edge segment polytope is the length of the respective triangle edge segment times the length of the associated time interval, etc.

Figure 2.12 and 2.13 shows the different split methods when a triangle polytope sequence is extended by a triangle polytope. Each splitting method yields different spatio-temporal topological settings and spatio-temporal redundancies. Investigations to invent models which are spatio-temporal topologically consistent while keeping minimum redundancy and redundant referencing is part of ongoing research [30].

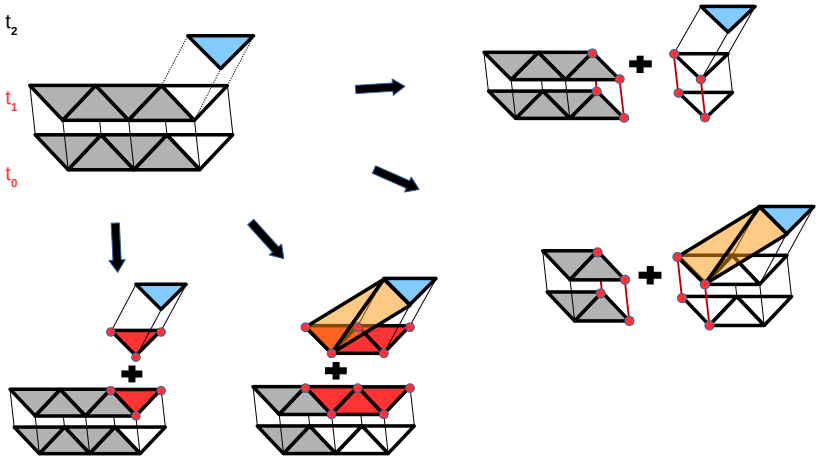


Figure 2.12: Split methods of a triangle-polytope complex when a triangle-polytope sequence is extended by a triangle-polytope.

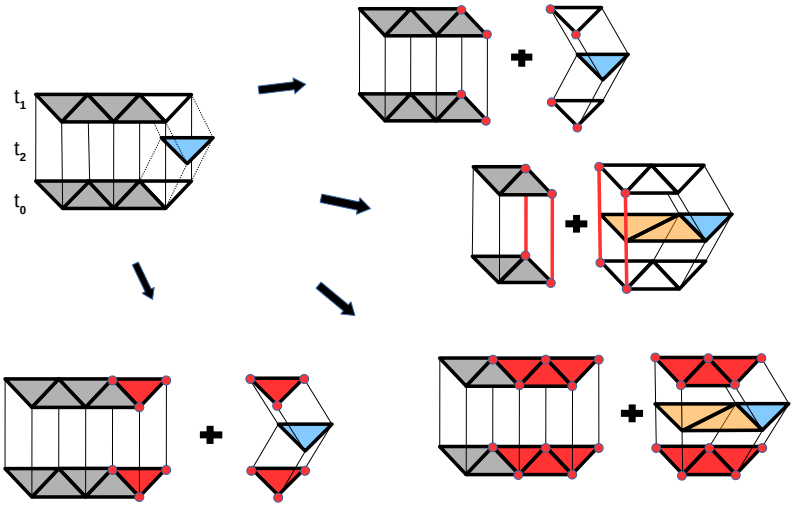


Figure 2.13: Split methods of a triangle-polytope complex when a triangle-polytope sequence is extended by a triangle-polytope.

2.3.7 Spatial Access Methods

Spatial access methods organize spatial objects (e.g. points, curves, surfaces, or solids) within a d-dimensional space and take spatial query vocabulary like “distance”, “intersects” or “hyper-volume” into account.

Quadtrees and octrees splits a hyper-cube of interest (in $2D$-space for quadtrees and $3D$-space for octrees) into sub-hyper-cubes of equal size. Each sub-hyper-cube is then subdivided into smaller sub-hyper-cubes of equal size, and so forth. The subdivision is managed by a tree. The root node represents the space of interest. Its sub-nodes represent the parts and so forth. The tree is not balanced. A hyper-cube is split if a chosen limit of contained spatial objects exceeded. All contained spatial objects of that hyper-cube are assigned to the new hyper-cubes depending on their spatial properties. The fixed size of the hyper-cubes may lead to empty hyper-cubes and long iterations through the tree while searching. Therefore, to find a point inside the tree heavily depends on how the data is spatially distributed. This is also one of the major disadvantages.

R^*-Trees organize the data in dynamic hyper-cubes of different sizes. The leaf nodes carry spatial objects of limited number and are shaped by the minimal bounding box of its content. Starting from the root-node, the hyper-cubes are shaped by the minimal bounding box of the sub-nodes' minimal bounding boxes. This approach is also called BVH (bounding volume hierarchy). The minimal bounding boxes are usually axis aligned (AABB - axis aligned bounding box). Insertion of a spatial object depends on the enlargement it may cause to the leaf nodes' minimal bounding boxes which the spatial object intersects. The leaf node gets split, if a leaf node exceeds the limited number of spatial objects per leaf node. Two spatial objects are chosen from the old leaf node together with the new spatial object (e.g. by maximal distance). The minimal bounding boxes of each spatial object will be the minimal bounding boxes for the new two leaf nodes. The remaining spatial objects are inserted to the leaf node, where the enlargement is minimal. If the parent node exceeds the limit, it has also to be repaired in the same way.

Spatial-Hash-Maps subdivides the d-dimensional space into smaller hyper-cubes by a given resolution. Each hyper-cube gets a number. This number is the hash code. A function can be defined to calculate a hash code by the spatial properties of the spatial object which suits best to the corresponding hyper-cube of the hash code. Different ways exist to define such functions. Some of them use space filling curves. However, Spatial-Hash-Maps are fast access methods and are used for coalition detection, data distribution in computer clusters and other purposes.

2.3.8 Spatial operations

This section provides a short introduction to a few important operations needed in GIS in order to calculate topologically consistent models. I do not concentrate on algorithms used in 2D space, nevertheless 2D space GIS's are still common. I will concentrate on spatial operations for simplicial complexes in 3D space as defined in Section 2.3.4, since this work concentrates on spatial and spatio-temporal data in 3D space for spatial representation and 1D space for temporal representation.

A fast basic algorithm to intersect two triangles was introduced by T. Möller [46] 1997:

Algorithm 1: triangle-to-triangle intersection by T. Möller [46]

input : Triangle A and Triangle B to intersect
output : Intersection of A and B

1 if all points of B lie on the same side of the plane of $A \rightarrow$ return null;
2 if all points of A lie on the same side of the plane of $B \rightarrow$ return null;
3 if all points of B lie on the plane of $a \rightarrow$ co-planar;
4 if co-planar $\rightarrow$ are projected onto the axis-aligned plane where the areas of the triangles are maximized and return the 2D intersection of the projected triangles;
5 if not co-planar $\rightarrow$ calculate the line between the two planes of A and B and return the intersection of the segments which result from the intersection of each triangle with the line;

Boolean operations on arbitrary polyhedral meshes are described in [40] from S. Landier in 2015. The basic idea is to find all intersections and re-triangulate all triangles at the intersections. Each new triangle can be mapped to either one or both input meshes. The triangles mapped to both meshes is the set of intersecting triangles. The triangles mapped to either one mesh is the set of differing triangles from the respectively other mesh.

A common known triangulation is named by Boris Delaunay. The Delaunay condition helps to prevent the creation of too sharp triangles. One example in 3D space is a ball-pivoting algorithm, which helps to triangulate triangle meshes from a given point cloud [5]. Delaunay tetrahedralization is also well studied. For further reading, I refer to the literature [19].

The robustness of geometric operations depends on the arithmetic used for computation. As an example, CGAL (Computational Geometry Algorithms Library - a major open source 3D processing library) provides double precision arithmetic, exact integer arithmetic using the Gnu Multi-Precision package, dynamic filtering which dynamically decides which arithmetic should be used by interval arithmetic and static filtering by evaluating with double arithmetic and compared with the worst case error bound [16].

2.3.9 Relevance for the work

The introduction of incidence graphs, topological consistency, simplicial complexes and polytope complexes, topological models and spatial access methods give insights into data management from a GIS point of view.

2.4 Related work and research

Since system scientists need to be supported by specialists of any science who perform data acquisition in their field, system scientists need to understand the tools of data acquisition. This section introduces a number of specifications, implementations, and software products related to this work.

The *OGC* (*Open Geospatial Concortium*) develops specifications in order to support the management of spatial and spatio-temporal data. Some of which became ISO standards, to be found in the ISO 191XX standards for geo-information. The specifications define global application schema based on XML. Other communities are able to use these schemata and extend them by need (e.g. CityGML).

The basic specifications are the *General-Feature-Model* and *Simple-Feature-Access*. The specifications provide concepts to model spatial and spatio-temporal objects. Features are real world objects which carry three different properties, one spatial, one temporal and one thematic property as stated in the *General-Feature-Model*. The spatial property is defined by geometric basic types which includes points, curves, polygons, and solids. Collections of those basic geometries are also supported. All geometries are defined as a *BREP* (Boundary Representation) and supposed to be topologically consistent. The global schema is defined in GML (Geography Markup Language) as XML schema.

SensorML, WaterML provide further XML schema in the context of sensors and water. *OGCs* Reference Model and SRID are specifications for coordinate systems. CityGML has become a major global schema for representation and querying of 3D city models. Whereas, GeoSciML became popular when dealing with geology. The O&O (Observations and Measurements) specifications provides XML schema concerning observations and measurements.

Service Oriented Architectures are WMS (Web Map Service), WFS (Web Feature Service), WCS (Web Coverage Service), WMTS (Web Map Tiles Service), WPS (Web Processing Service) and WCPS (Web Coverage Processing Service). The SensorThings API provides interface definitions for the IOT (Internet of Things). Furthermore, GeoSPARQL is a specification for representation and querying of geospatial linked data. 3D Tiles is a newer specification which provides rules for streaming and rendering massive 3D geospatial content such as Photogrammetry, 3D Buildings, BIM/CAD, Instanced Features, and Point Clouds. IFC (industry foundation classes) and BIM (Building Information Models) are not specifications of the *OGC* but are used in facility management and civil engineering.

JTS (Java Topology suite) is a spatial processing library written in Java for 2D operations. It is part of one of the major implementations, which is called *GeoTools*. The *GeoTools* are based on *OGC* Simple Features for SQL specification. The geometry model includes Points and MultiPoints, LineStrings and MultiLineStrings, Polygons and MultiPolygons, and heterogeneous GeometryCollections. The precision handling includes an explicit coordinate precision model and geometry precision reduction. The 3D implementations within the *GeoTools* using OpenGIS ISO Geometry interfaces are not ready yet (31.07.2021). The key features of *JTS* are [34]:

1. Geometry Operations
    a) Topological validity checking
    b) Area and Length/Perimeter
    c) Distance between geometries and WithinDistance predicate
    d) Spatial Predicates based on the Egenhofer DE-9IM model, including the named predicates: contains, within, covers, coveredBy, intersects, disjoint, crosses, overlaps, touches, equals, and the general relate operation returning the DE-9IM intersection matrix.
    e) Overlay functions including: intersection, difference, union, symmetric difference, unary union, providing fast union of geometry collections
    f) Buffer computation (also known as Minkowski sum with a circle) selection of different end-cap and join atyles.
    g) Convex hull
    h) Geometric simplification including the Douglas-Peucker algorithm and topology-preserving simplification
    i) Geometric densification
    j) Linear referencing
2. Geometric Constructions
    a) Delaunay triangulation and Conforming Delaunay triangulation
    b) Voronoi diagram generation
    c) Minimum Diameter of a geometry
    d) Minimum Enclosing Rectangle
    e) Minimum Bounding Circle

3. Metric Functions
    a) Distance between geometries, with supporting points
    b) Discrete Hausdorff distance
    c) Area and Hausdorff similarity measures
4. Spatial algorithms
    a) Robust line segment intersection
    b) Efficient line arrangement intersection and noding
    c) Snap-rounding for noding line arrangements
    d) Efficient Point-in-Polygon testing
5. Mathematical Functions
    a) Angle computation
    b) Vector arithmetic
6. Spatial structures
    a) Spatial index structures including: Quadtree, STR-tree, KD-tree, Interval R-tree, Monotone Chains
    b) Planar graphs and graph algorithms
7. Input and Output
    a) WKT (Well-Known Text) reading and writing
    b) WKB (Well-Known Binary) reading and writing
    c) GML (Geography Markup Language) Version 2 reading and writing
    d) Java Swing/AWT Shape writing
8. High-Precision Arithmetic
    a) Robust evaluation of 2×2 double-precision determinants
    b) DoubleDouble extended-precision arithmetic

GDAL (Geospatial Data Abstraction Library) is a library written in C and C++ for reading and writing raster and vector data (e.g. *GML*). It is released by the OSGeo. The library is included in popular applications like *QGIS* and *GRASS GIS*. It may be used within an ETL (Extract Transform Load) approach together with *PROJ* which is a library for conversions between cartographic projections and *GEOS* which is a port of *JTS* in C++. The features of *GEOS* are the basic features of *JTS*, which are [24]:

1. Geometries: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection
2. Predicates: Intersects, Touches, Disjoint, Crosses, Within, Contains, Overlaps, Equals, Covers
3. Operations: Union, Distance, Intersection, Symmetric Difference, Convex Hull, Envelope, Buffer, Simplify, Polygon Assembly, Valid, Area, Length, Prepared geometries (pre-spatially indexed)
4. STR spatial index
5. *OGC* Well Known Text (WKT) and Well Known Binary (WKB) encoders and decoders.

The following paragraph is taken from [23] in order to introduce *GeoMesa*:

> *GeoMesa* is an open source suite of tools that enables large-scale geospatial querying and analytics on distributed computing systems. *GeoMesa* provides spatio-temporal indexing on top of the Accumulo, HBase, Google Bigtable and Cassandra databases for massive storage of point, line, and polygon data. *GeoMesa* also provides near real time stream processing of spatio-temporal data by layering spatial semantics on top of Apache Kafka. Through GeoServer, *GeoMesa* facilitates integration with a wide range of existing mapping clients over standard *OGC* APIs and protocols such as WFS and WMS. *GeoMesa* supports Apache Spark for custom distributed geospatial analytics.

The following paragraph is taken from [61] in order to introduce SFCGAL:

> *SFCGAL* is a C++ wrapper library around CGAL with the aim of supporting ISO 19107:2013 and *OGC* Simple Features Access 1.2 for 3D operations. *SFCGAL* provides standard compliant geometry types and operations, that can be accessed from its C or C++ APIs. PostGIS uses the C API, to expose some *SFCGALs* functions in spatial databases (cf. PostGIS manual). Geometry coordinates have an exact rational number representation and can be either 2D or 3D. The features of *SFCGAL* are:
>
> 1. Geometries: Points, LineStrings, Polygons, TriangulatedSurfaces, PolyhedralSurfaces, GeometryCollections Solids
> 2. Predicates: Intersects, Touches, Disjoint, Crosses, Within, Contains, Overlaps, Equals, Covers
> 3. Operations: Intersection operations and predicates, Convex hull computation, Tessellation, Extrusion, Area and distance computation, Minkovski sums, Contour offsets, Straight skeleton generations
> 4. WKT reading and writing with exact rational number representation for coordinates

Intersection, difference and union methods support polyhedral surfaces, triangles and triangulated irregular network surfaces (TIN) [54].

Another library is *VTK*, the *Visualization Toolkit* [67]. *VTK* is used for scientific visualization. It supports different kinds of geometry types. This includes structured and unstructured grids, polydata, structured and unstructured points, rectilinear grids and field data. A detailed description of those types can be found in [68].

VTK can be seen as the geometry core of *ParaView*. The following paragraphs are taken from [49] in order to introduce *ParaView*:

> *ParaView* is an open-source, multi-platform data analysis and visualization application. *ParaView* users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using *ParaViews* batch processing capabilities.
>
> *ParaView* was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of petascale size as well as on laptops for smaller data, has become an integral tool in many national laboratories, universities and industry, and has won several awards related to high performance computation.

The following paragraphs are taken from [64] in order to introduce *ParaView Catalyst* and how to avoid data explosion:

> *ParaView Catalyst* was created as a library to achieve the integration of simulation and post-processing. It has been designed to be easy to use and introduces minimal disruption into numerical codes. It leverages standard systems such as *VTK* and *ParaView* (for post-processing) and utilizes modern scientific tools like Python for control and analysis. Overall, Catalyst has been shown to dramatically increase the effectiveness of the simulation workflow by reducing the amount of IO, thereby reducing the time to gain insight into a given problem, and more efficiently utilizing modern HPC environments with abundant FLOPS and restricted IO bandwidth.
>
> [...]
>
> A key point to keep in mind when creating Catalyst pipelines is that the choice and order of filters can make a dramatic difference in the performance of Catalyst (this is true with *ParaView* as well). Often, the source of performance degradation is when dealing with very large amounts of data. For memory-limited machines like today's supercomputers, poor decisions when creating a pipeline can cause the executable to crash due to insufficient memory. The worst case scenario is creating an unstructured grid from a topologically regular grid. This is because the filter will change from using a compact grid data structure to a more general grid data structure. We classify the filters into several categories, ordered from most memory efficient to least memory efficient and list some commonly used filters for each category:
>
> 1. Total shallow copy or output independent of input – negligible memory used in creating a filter's output. The filters in this category are:
>
> Annotate Time • Append Attributes • Extract Block • Extract Datasets • Extract Level • Glyph • Group Datasets • Histogram • Integrate Variables • Normal Glyphs • Outline • Outline Corners • Plot Over Line • Probe Location
>
> 2. Add field data – the same grid is used but an extra variable is stored. The filters in this category are:
>
> Block Scalars • Calculator • Cell Data to Point Data • Compute Derivatives • Curvature • Elevation • Generate Ids • Generate Surface Normals • Gradient • Level Scalars • Median • Mesh Quality • Octree Depth Limit • Octree Depth Scalars • Point Data to Cell Data • Process Id Scalars • Random Vectors • Resample with Dataset • Surface Flow • Surface Vectors • Transform • Warp (scalar) • Warp (vector)

3. Topology changing, dimension reduction – the output is a polygonal dataset but the output cells are one or more dimensions less than the input cell dimensions. The filters in this category are:

Cell Centers • Contour • Extract CTH Fragments • Extract CTH Parts • Extract Surface • Feature Edges • Mask Points • Outline (curvilinear) • Slice • Stream Tracer

4. Topology changing, moderate reduction – reduces the total number of cells in the dataset but outputs in either a polygonal or unstructured grid format. The filters in this category are:

Clip • Decimate • Extract Cells by Region • Extract Selection • Quadric Clustering • Threshold

5. Topology changing, no reduction – does not reduce the number of cells in the dataset while changing the topology of the dataset and outputs in either a polygonal or unstructured grid format. The filters in this category are:

Append Datasets • Append Geometry • Clean • Clean to Grid • Connectivity • D3 • Delaunay 2D/3D • Extract Edges • Linear Extrusion • Loop Subdivision • Reflect • Rotational Extrusion • Shrink • Smooth • Subdivide • Tessellate • Tetrahedralize • Triangle Strips • Triangulate

When creating a pipeline, the filters should generally be ordered in this same fashion to limit data explosion. For example, pipelines should be organized to reduce dimensionality early. Additionally, reduction is preferred over extraction (e.g. the Slice filter is preferred over the Clip filter). Extracting should only be done when reducing by an order of magnitude or more. When outputting data extracts, subsampling (e.g. the Extract Subset filter or the Decimate filter) can be used to reduce file size but caution should be used to make sure that the data reduction doesn't hide any fine features.

Scripting by wrapping *ParaView* using Python, Java etc., the use of *ParaViews* ability to run Python scripts [50] which includes programmable filters [56] or as batch processing on high performance computing clusters running in parallel using MPI (Message Passing Interface for parallel computing architectures) [55], *ParaView Catalyst* [64] and *ParaViews* web visualization features [51] make ParaView to an interesting tool in the context of this work.

One version of *ParaView* exists, which is called *ParaViewGeo*. *ParaViewGeo* extends *ParaView* with the ability to read a number of file formats commonly used within the geosciences. It has been used for visual data mining in geological, seismic and geomagnetic applications [52].

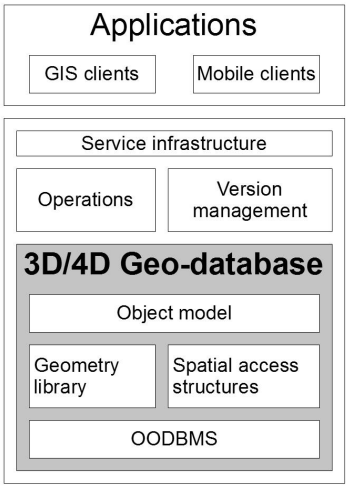


Figure 2.14: Architecture of the ancestors *DB3D*

Another interesting framework is *DB4GeO*. This framework was developed in the working group of Martin Breunig [14] as a spatio-temporal object-oriented database for research purposes. It is written in Java to be able to develop system independent network applications on top of a Java virtual machine. Figure 2.14 shows the basic architecture.

In the case of its spatial ancestors called *DB3D* the back-end is based on *ObjectStore* and *Jini Lookup-Services* as the service framework [2]. The aim of the dissertation by W. Bär [2] was the management of geoscientific 3D data in mobile database systems. The *Jini Lookup-Services* works as a sort of front end for Javas RMI (remote method invocation). Figure 2.15 shows *DB3Ds Service Oriented Architecture* using Java *Remote Method Invocation* classes on spatial *Retrieve* service as example.

In case of *DB3Ds* predecessor *DB4GeO*, *DB3Ds* service layer was rejected, and the operation layer was rudimentary integrated. *DB4GeO* uses the *Representational State Transfer* paradigm as front end and *DB4O* as back end [14]. Figure 2.16 shows *DB4GeOs Service Oriented Architecture* using *REST* on *Plane-Cut* operation as example. The implementations concerning the *REST* paradigm have been outsourced to their own development project [38] as well as the spatio-temporal framework [60] and the *G-Maps* implementations [15] before.

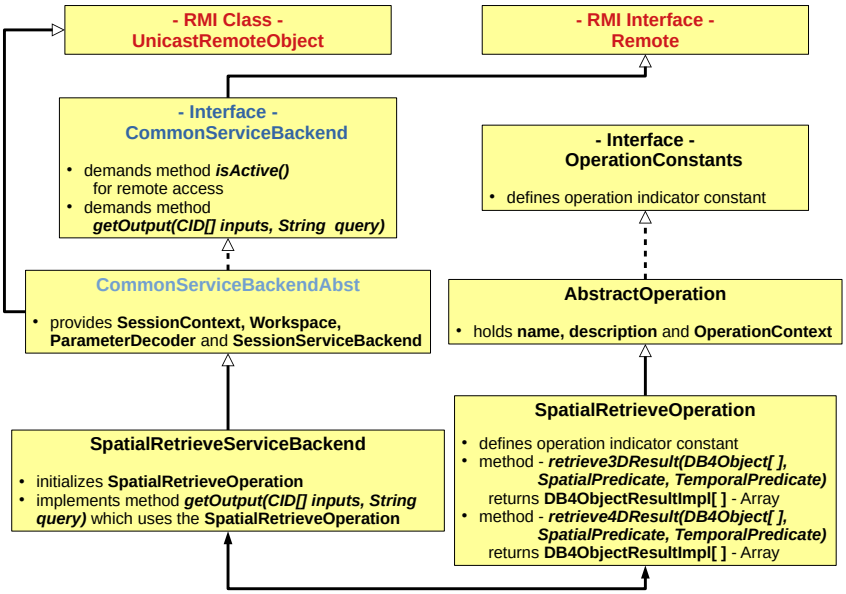


Figure 2.15: *DB3Ds Service Oriented Architecture* using Java *Remote Method Invocation* classes on spatial *Retrieve* service as example

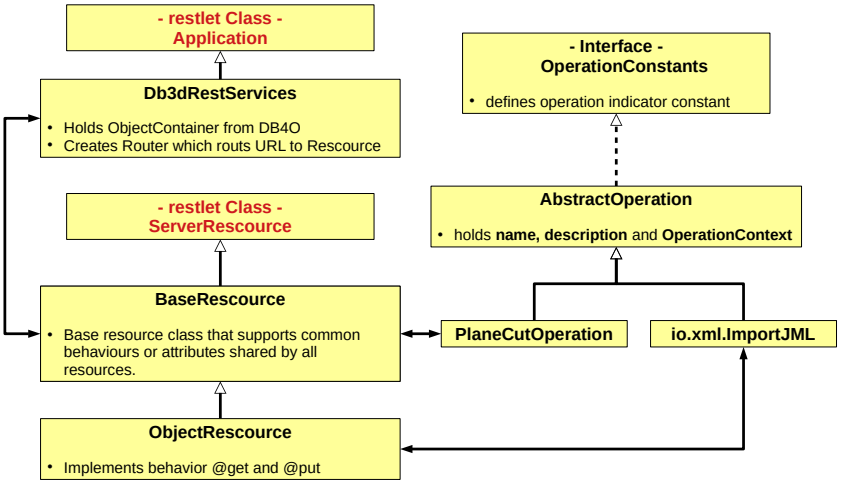


Figure 2.16: *DB4GeOs Service Oriented Architecture* using *REST* on *Plane-Cut* operation as example

3 Distributed and parallel data management for pre-processing

The last chapter provided the technical standards of three different disciplines (System sciences, computer sciences and geo-information sciences). This chapter discusses the resulting requirements and problems of distributed and parallel data management to support simulation implementations. A concept is presented at the end of the chapter.

3.1 Requirements

To model real world phenomena, system scientists filter information from a broad range of data, not only including geo-data. An abstract geo-data model which supports the creative process of system scientists is required which supports edge, fog, and cloud based solutions. If the simulation model includes spatial dependencies, the spatial model to support those simulation models needs to provide structured or unstructured geometries together with a well-defined topological model. Basic access methods need to be supported for spatial or spatio-temporal object extraction.

Furthermore, quantification of simulation parameters and validation of simulation results lead to constant communication efforts between geo-data or information sources and simulation sinks. The communication and processing needs to be as small as possible, which implies that the geo-data or information and its creation pipeline need to be analysed for bottlenecks in processing time and the amount of transferred geo-data or information. After the analysis, the outsourcing of processes is required to liquidate the bottlenecks to realize an optimal network infra-structure where each transformation process is put into the "right" place.

Therefore, standard transformation and analysis algorithms (e.g. triangulation, grid generation, topological analysis etc.) may already be integrated, but it is much more important that the user, the system scientist in this case, is able to integrate his/her own processing algorithms safely.

A plugin based approach, where each user defines its own data processing pipeline (which could be shared also) can increase collaboration. This means that the data types offered, in particular the spatial and spatio-temporal data types, must support complex abstractions (e.g. DTM, DEM, subsurface volume models, constructive solid geometry,

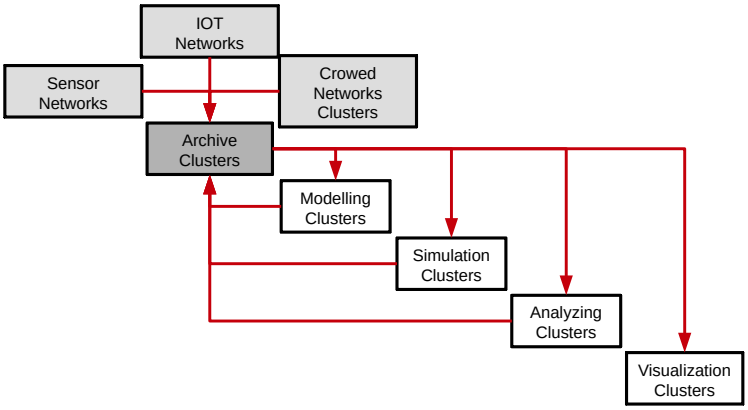


Figure 3.1: IO-Problem with centralized solutions

CityGML, finite elements, BIM etc.) and their modelling in a user-defined manner not as a requirement.

3.2 Problem description

Section 2.4 showed that processing geo-spatial data in 3D space is still not common in standard GIS or spatial databases. This section also showed that distributed and parallel processing is still not common. Even if standards, such as OGCs *Simple Feature Model* or the service oriented architectures (e.g. WMS, WPS etc.) support distributed data mining, as global standards, the downside of global standards were pointed out in Section 2.2 when they get too specialized. Special attention is required when defining such a standard, because the definition of a standard is equal to user restriction.

Currently, enormous amounts of data are archived in an archiving cluster in a first step (redundant for OLTP in several archiving clusters for high user interaction) using complex global standards which could not contain special acquisition types. For analysis, modelling or visualization, the data is then heavily filtered or reduced and transferred to a processing cluster. The extraction, transformation, and loading (ETL) is a necessary step for this, as the data volumes cannot be sent completely to a data sink at the end user or to a processing cluster. The same applies to the transfer of the processed data back to the archiving cluster, as well as to the transfer to a visualization cluster (see Figure 3.1). This problem is known as the IO problem.

The following paragraphs are taken from [64] in order to illustrate the IO problem which takes place within a single workstation or centralized high performance computing (HPC) solutions:

> This divergence between computational power and IO bandwidth has profound implications on the way future simulation is performed. In the past it was typical to break the simulation process into three pieces: pre-processing (preparing input); simulation (execution); and post-processing (analysing and visualizing results). This workflow is fairly simple and treats these three tasks independently, simplifying the development of new computational tools by relying on a loose coupling via data file exchange between each of the pieces. For example, pre-processing systems are typically used to discretize the domain, specify material properties, boundary conditions, solver parameters, etc. finally writing this information out into one or more input files to the simulation code. Similarly, the simulation process typically writes output files which are read in by the post-processing system. However limitations in IO bandwidth throw a monkey wrench into this process, as the time to read and write data on systems with relatively large computational power is becoming a severe bottleneck to the simulation workflow. Savings can be obtained even for desktop systems with a small amount of parallel processes. [...] The root problem is that due to the ever increasing computational power available on the top HPC machines, analysts are now able to run simulations with increasing fidelity. Unfortunately this increase in fidelity corresponds to an increase in the data generated by the simulation. Due to the divergence between IO and computational capabilities, the resulting data bottleneck is now negatively impacting the simulation workflow. For large problems, gone are the days when data could be routinely written to disk and/or copied to a local workstation or visualization cluster. The cost of IO is becoming prohibitive, and large-scale simulation in the era of cheap FLOPS and expensive IO requires new approaches.

The reason for this is a fundamental way of thing about data and dealing with data. Data is thought to be like a property, a piece of land, which can be sold, and the owner of that property allows what happens on the property. The owner also decides which tools are going to be used to build something. And the owner also decides what is going to be built. The added value in turn may be of use to anybody who has interest to visit that property and to enjoy, to buy, the added value being sold. The outsourcing of user defined processing steps is usually not supported by any data or service provider. If there are customers out there who know how to process the provided raw data better than the owner, it is impossible to integrate the customers' knowledge, to create needed knowledge. And this needed knowledge is the knowledge which actually needs to be made available in the first place. The communication process between customer and owner is a time-consuming process, which makes the paradigm of thinking about data as a property sluggish.

Microservice architectures are commonly used as a paradigm to optimize distributed collaboration within one organization, institution, or company. The paradigm of microser-

vice emphasizes a more heterogeneous, abstract way of linking data. The paradigm also supports edge and fog based solutions but if microservices work on the same database, the scaling of microservices is different from the scaling of a shared scalable database. A microservice is able to use a shared scaled database in a way that the provided service of the microservice is calculated on the shared scaled database in a parallelized manner by the usage of the services from the shared scaled database. This means, that only the services of a shared scaled database are efficiently usable. Scaling of microservices only works if each microservice holds its own database. This microservice needs to be part of a distributed global data stoke. This distributed global data stoke needs to provide all the data management functionalities of a centralized scalable spatial or spatio-temporal database. A distributed spatial or spatio-temporal multi-database query processing architecture to be used within a microservice ecosystem does not exist.

Most of the introduced spatial or spatio-temporal databases are centralized solutions. DB3D [2] is the only introduced database system which aimed at mobile distributed spatial database support with version management. The IO problem of restricted bandwidth and the load-balancing between mobile computing units and centralized server solutions had been pointed out as problems by W. Baer [2] in 2007. The power consumption and the lack of network coverage forced the users to work offline [2]. The solution of W. Baer was to implement a service based spatial database with version management which is a synchronization model as a revision control instance which is able to manage multiple versions of the same datasets. Version control / synchronization models are still mature issues when designing distributed and parallel data management.

It needs solution for distributed spatial or spatio-temporal access methods which are able to deal with a number of coordinates systems. The Section 2.3 showed that spatial access methods usually work on one coordinate system with one static data type for coordinates. This may cause scale problems if the spatial objects get too small. This may also cause processing problems due to bad precision. The following example illustrates the computational problems when dealing with double precision arithmetic. The example also illustrates the problem of topological inconsistency when using *BREPs* as common basic geo-data representation.

The following two paragraphs are published in [31] beforehand:

Wireframe models are one of the basic models to describe any kind of spatial entities used in city models or other geo-information models. One of the major issues is to triangulate those one-dimensional geometry types into higher-dimensional geometry types like triangle nets as surface models embedded into three-dimensional Euclidean space. Doing so will add value to the data-set. E.g. queries like “How large is the area of walls between neighbouring buildings?” can be answered thus. The same situation arises when dealing with boundary representation models (*BREP*) consisting of surfaces in order to model solids. A triangulation to true three-dimensional geometry types like tetrahedral nets provides more answerable queries than a *BREP* which only represents the boundary surface of a solid. An example is the query “How large is the overlapping volume within the city model?”. Methods dealing with the construction of solids from

wireframe models so far assume that the underlying model is topologically consistent, i.e. its building blocks form a partitioning of the space to be modelled, meaning that they do not overlap. If the model is topologically inconsistent, then such queries are not trivial to be answered using a *BREP*. The reason is that costly steps have to be taken to calculate the triangulations leading to higher-dimensional geometry types. If a city model for example consists of overlapping pieces of wall polygons, roof polygons, or even building parts are made up of overlapping planar polygon pieces, it gets much harder to retrieve those higher-dimensional geometry types they represent by being a *BREP* made of a collection of lower-dimensional *BREP*s, where I consider not only connected planar surfaces as a *BREP* for solids, as in [39], but any n-dimensional representation of the boundary of a $n+1$-dimensional model, and call this also a *BREP*.

The main problem in dealing with topologically inconsistent models is that the inconsistency can affect all dimensions, meaning that there are overlaps between objects of any dimensions in the topologically inconsistent model. This means that the increment from dimension n to $n+1$ needs to be combined with a method that produces output which is guaranteed to be topologically consistent. In other words, many intersections need to be computed. This leads to numerical issues affecting the partitioning of overlapping pieces using double-precision arithmetic. If angles get too small, then intersection points between lines or intersection lines between planes become uncertain. If algorithms are chained in pipelines to achieve more complex results, then the uncertainty accumulates, too, or the pipelines break up, producing no results at all. This major problem of computational geometry is faced when using standard double-precision arithmetic. Nevertheless, calculating a topological model is sensitive and expensive with double-precision arithmetic (e.g. due to intersections, differences, overlay etc.) and has higher memory usage than only storing topologically inconsistent *BREP*s. But it provides reusable information on how the pieces are connected, thus yielding benefits and outlays which need to be examined.

3.3 Concept

As mentioned in Section 2.1, the variety of use of geo-data or geo-information within simulation models is complex. A geo-information model is needed which allows the extraction of representations which serve as input for simulations. This data model acts as interface between sources and sinks.

It is not advantageous in terms of efficient data transfer to run geo-service on the sink, which would decrease the data transfer. Transferring a map of an area which carries all population information within one *CityGML* file and extracting the necessary information as a pre-processing step of the simulation on the simulation / sink increases the data transfer if the extraction algorithms are not provided on the source. Transferring a couple of numbers representing a population pyramid may be faster than transferring

the hole map. The source should provide an API which allows the sink to integrate the own data extraction algorithms.

It is also not advantageous in terms of efficient data transfer to run geo-services on the source, which would increase the data transfer. Those services should be executed within the sink as a pre-processing step to reduce data transfer. An example would be the creation of a high resolution raster data from vector data by scanning (e.g. point in polygon / hull tests or ray casting). The sink should provide an API which allows the source to integrate the own data.

An intuitive distribution of processing would be to agree to some basic extendable spatial or spatio-temporal API and let the scientists write their own services based on that API. Each sink or source relies on a standardized input and output data structure and does not necessarily need to process the geo-data itself nor provide the service by its own broad intelligence, which would be needed to understand what any customer wants. However, it is thinkable that other sinks rely on the same extractions which may be provided by a source through a microservice in order to scale up as needed for high user interaction. Each source would provide geo-data and services depending on its hardware properties. Complex networks of collaboration could be established where the property of any calculation unit from, for example, Raspberry Pi's to complex high performance computing cluster, along the pipeline is considered. The quantification step 9 and the validation step 12 could be easily done.

If the processing time is rather long on the source, it might be a better idea to calculate the results once, make the output persistent and provide the persistent results as a service. So, the results may be made persistent to not reprocess the input data by needs (redundant processing). This can reduce processing time. An example would be the creation of a topological consistent spatial or spatio-temporal model due to the complexity of the involved algorithms.

If services can be distributed through a common extendable API it should also be possible to transfer geo-data from a source cluster together with a service which takes the transferred geo-data as input. The execution of the service could be done on the sink and the transferred data is minimal. In this case it has to be proved if the output of the service should be provided persistently depending on the number of redundant processing.

It is not advantageous in terms of efficient processing time if processing of the data on the source takes much longer than it would on the sink and vice versa. The trade-off has to be analysed between the processing times and the transfer times in both situations. This includes the proof what happens if the output would be made persistent to reduce redundant processing. The trade-offs depend also on the business of the involved sources and sinks. A real time analysis of the trade-offs can be used for dynamic load-balancing and the pipeline can adapt to the workload.

ParaView provides nearly all needed technologies to support user-defined simulations on standard computers and high performance computing clusters, which integrates In-Situ

post-processing to reduce data transfer between simulation and archiving / visualization clusters by real-time visualization. As introduced in Section 2.4, *ParaView* provides a detailed description on how to set up a post-processing pipeline. This includes information about the input and output quality of each algorithm together with a guideline on how to put the order of the pipeline (which algorithm should come first etc.) to minimize memory usage and computation. This knowledge has been provided in order to speed up simulations which are based on *ParaViews* In-Situ post-processing abilities. It is necessary to set up efficient post-processing pipelines for data extraction, transformation, and visualization to minimize the computation efforts of the post-processing after the calculation of a simulation step. The same knowledge is important to set up distributed and interconnected pipelines for pre-processing. The problem which has not been addressed by *ParaView* is pre-processing.

Having this in mind, where savings in the post-processing can be obtained by calculating results alongside with the simulation through the In-Situ paradigm, savings can also be obtained by distributed data mining concepts in a complex pre-processing pipeline which respects the minimization of data transfer between computing units by pushing some processes to the data instead of pushing the data to the process. Figure 3.2 shows an example setup. This setup is a basic setup to illustrate a distributed environment which integrates the pipeline architecture of *ParaView* (see Section 2.4) as the commonly agreed framework for spatial or spatio-temporal data types and certain processing and visualization abilities, in this case. The example may rely on a file based synchronization software which keeps the output files of one pipeline redundant (in the same version) to the input files of another pipeline. In case of this configuration, any synchronization software will work (e.g. GIT, rsync etc.). Integrating *ParaView* into a streaming application or any other communication application could also work. So to use *ParaView* in that way actually enables certain efficient ways of distributed data-mining if the knowledge of constructing efficient pipelines is respected and some synchronization tools are integrated.

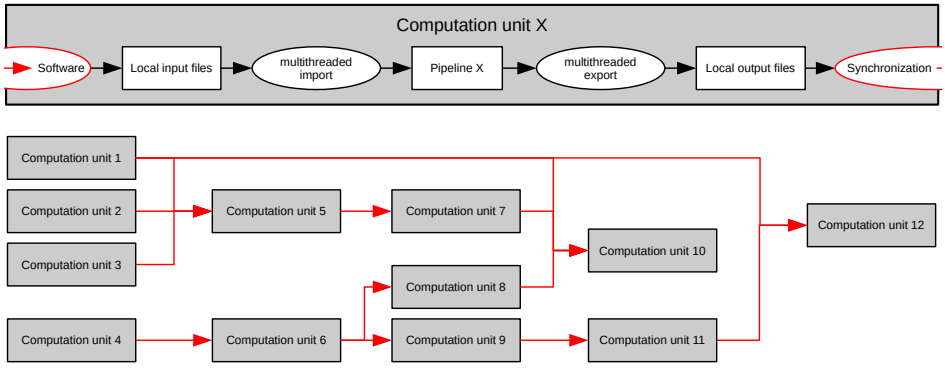


Figure 3.2: File based concept using some synchronization software together with ParaView.

A library (e.g. Python library) of data extraction and transformation pipelines could be established and spread across the distributed environment in order to reduce data transfer by analysing the pre-processing pipeline features as mentioned before. Furthermore, the simulation implementation becomes part of the distributed environment and may serve as source for different kind of applications down the pipeline. Simulation implementations which use In-Situ post-processing can also be shared through libraries or even by virtualization using "containering" (e.g. microservices) for plug and play purposes. Sharing of simulation models can enhance collaboration of multiple scientific domains. Deploying simulations leads to parallel simulation processing within a distributed environment if the boundary conditions of the simulation implementations do not exceed the IO bandwidth of each participating simulation computing unit within the distributed environment. It is questionable which type of simulation implementation supported by which type of distributed environment with its inherent IO bandwidth configuration would draw benefits from this approach. It is also questionable to contradict the In-Situ paradigm by writing each simulation step to the file system to be the source for any application down the pipeline. Even real-time streaming of simulation results has its boundaries in IO bandwidth. However, simulations become part of a distributed ETL where simulations constantly transform and provide data.

Since *ParaView* is a specialized software product for general scientific analysis and visualization, the major step is the integration of geo-information models which support spatial and spatio-temporal data mining techniques needed by system scientists. An extendable framework needs to be provided which takes care of the synchronization as shown in Figure 3.2. The framework needs to provide an abstract geo-information model, efficient access methods, need to be able to connect to different kinds of back ends and may provide basic algorithms/services. But the most important feature must be extensibility. Extensibility does not necessarily imply that a system grows to a sort of highly complex global standard. It only implies that user defined needs are able to be easily integrated, which is one of the major requirements for collaboration.

This framework can be used to define customized importers and exporters (e.g. CityGML, Kafka streams etc.) depending on the computational units which need to be synchronized. If it is possible to fuse the back end framework of a computational unit and the synchronization framework, the fused framework can be used to define the transformation pipeline and does not need to export the output nor import the input of this computational unit by using a file based approach. This is usually done through an API supported by the synchronization or back end framework of a computational unit.

An example for one of those APIs was introduced in Section 2.2.4 about the graph computing framework TinkerPop where each back end is "TinkerPop enabled" if the back end supports TinkerPop. TinkerPop becomes the main programming interface. Another example is given by GeoMesa, introduced in Section 2.4. GeoMesa provides GeoTools features on top of different back end frameworks by implementing interfaces between the GeoTools features and the different back end technologies as GeoTools-compatible data store. In this case, GeoMesa takes care of the adaption and not each

back end provider. The main user interface stays the GeoServer / the GeoTools. Another example is *ParaViews* batch processing using Python, which exposes *ParaViews* parallel computing abilities to Python. The wrapped Python classes become the main interface. I do not propose which option should be used, neither which back end should be used. Every option has pros and cons. I also do not propose wherever a streaming architecture (e.g. Kafka) should be used for real time streaming between the computational units or not. In some cases, this will be necessary (e.g. sensor or simulation networks). I also do not propose wherever microservices, REST or any other front end should be used or not. These are specialized techniques for special use cases. They should be integrated if the use case benefits from it. The use of a mediator/wrapper architecture may be considered for a multi-database query processing architecture. The steps to create the proposed framework with an appropriate API are described in the following.

The first step is to agree to a spatial or spatio-temporal model. I choose to rely on vector data in three dimensions for spatial objects and 4 dimensions for spatio-temporal objects using double precision. The topology is implemented as simplicial complexes for spatial objects and polytope complexes for spatio-temporal objects, as described in Section 2.3.4. This does not forbid to create *BREPs*. But the $(d+1)$-dimensional *BREPs* defined by d-dimensional simplicial complexes should be triangulated and stored as $(d+1)$-dimensional simplicial complexes as a matter of unambiguity (see Section 2.3.3). The *BREPs* can be retrieved by retrieving the border of the $(d+1)$-dimensional simplicial complexes. This operation is unambiguous.

The spatio-temporal model follows the Polthier and Rumpf model [53]. This allows the management of simulation results which move and morph spatial objects since a simulation step creates a spatio-temporal polytope complex with only two time steps, the spatial input data set and the processed spatial output data set with equal internal topological configuration.

Multiple types of data (structured or unstructured) are used within simulation implementations since the discretization of the domain depends on the simulation implementation in order to reduce processing time on specialized back ends. This is one of the reasons why different back ends need to be supported in some way (e.g. array databases for raster data etc.), which are optimized solutions to manage and process those special types of structured or unstructured data.

It is to say that regular and irregular topology can be managed by the proposed spatial model and that the cell type, or voxel type for voxel models, is creatable by a set of simplices. This approach is not efficient for data organized as structured grids (e.g. raster data) if the data does not include large enough hypervolumes of redundant data (e.g. an equal distributed entity saved in a high resolution raster). It is obvious that this decision is partly data driven and dependents on the use case. It can not be decided by only a common agreement to a spatial or spatio-temporal model, be it vector data or raster data. Using integer arithmetic on vector data seems to be a sort of hybrid. The problem of resolution, precision and scale remains in both types of data.

Furthermore, whenever the structure of the data or the elimination of redundancies leads to optimized representations which reduces data transfer time and processing time at the same time, the proposed spatial or spatio-temporal model should be transformed into the optimized representation and build into the pipeline. The process of optimization to find the special configuration of the pipeline in order to optimize transfer and processing time as mentioned before is part of the practical application of the proposed collaboration concept.

The proposed spatial model is a general model which is able to define precise surfaces and volumes because it is based on simplices and can be transformed in various ways to raster data or voxel models, be it by ray tracing, the inverse of a contour filter or any other method e.g. a set of simplices represent a voxel. The spatio-temporal model is also a general and precise model, since the interpolation between two time steps defines every inner point and the topology of the moving and morphing spatial object as far as the computational arithmetic allows.

For now, simplicial complexes are used for the spatial model and polytope complexes are used for the spatio-temporal model due to their versatility and simple structure in the sense of being "unstructured" grids / meshes vs. "structured" voxel models or having an irregular topology vs. a regular topology. The use of simplicial complexes as a spatial object with certain constraints is due to the precise definition of the inner points, as mentioned in Section 2.3.3 about the downsides of *BREPs* (e.g. planarity of polygons).

The final report of the DFG Project "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) and the papers [30] and [31] contain extracted, transformed and minimized paragraphs of the next five paragraphs, published beforehand:

The second step focuses on the development of topological models to provide topological queries. A suitable topological model is also the key to have control over data distribution in case of processes depending on topological constraints. This is usually the case when dealing with simulation implementations. The concept for the object model follows the *Property Graph Model* which is implemented by common graph databases but may also be translated to any other database approach, if needed, in respect to the pros and cons. The *Property Graph Model* is a basic model to structure information. A *Property Graph* consists of several nodes which are related to one another upon need. As an example, a *Property Graph* may be used to represent a group of people having relations. Each person may have different properties. They are differently shaped, they have been born on different dates, they earn different amounts of money, have different jobs or no jobs at all. Maybe the graph also includes buildings as nodes with different building properties (a shape, a construction date etc.) and each building may or may not be related to one or more persons.

The *Property Graph* itself is able to represent complex models, whose structures in turn can be analysed. One could see the group of people and the group of buildings as two layers within a standard GIS. The overlay produces another layer which colours every

building by the number of people which are connected to them. Graph algorithms need to be used in order to achieve such an overlay. In that sense, *Property Graphs* are a form of linked data. T_0-spaces can be defined by the definition of labelled relations. I focus on three different relation classes of spatial or spatio-temporal objects, which are aggregation, incidence and abstraction. Each spans two relation types which are inverse to each other: *part-of*, *border-of* and *generalization-of* and their inverses *composite-of*, *inner-of* and *specialization-of*, respectively. This object model follows the basic relations of object-oriented programming (OOP) as the abstract paradigm, which has proven to be very useful in a broad range of applications. The relation classes enable analysis concerning their spanned T_0-spaces.

Within geo-information there are formats like *CityGML* or *GML* which are XML-based, and hence semi-structured according to [22]. One basic idea of geo-information is described in OGC's *General Feature Model* and *Simple Feature Model*. As mentioned before, everything may be seen as a *Feature*, a real world object with certain properties. The model insists on three different property types, the *spatial part*, the *temporal part* and the *thematic part*. The *Simple Feature Model* defines the spatial part and which geometry types should be provided. The *General Feature Model* seems to be abstract enough to not constrain the needs of systems scientists or any other scientists who are part of the pipeline. The implementation of the object model based on OGC's *General Feature Model* and the *Property Graph Model* is provided in Section 4.2.1.

The connection between those two models and the introduction of basic relation types lead me to add node operations. Node operations build and relate nodes to a node on the basis of its *spatial part*. Each relation may be built by a node operation. Section 4.2.3 describes five different node operations. A special case is the node operation, which creates a set of disjoint $(d+1)$-dimensional simplicial complexes from a set of d-dimensional complexes, by analysing the spatial parts and the topology. The algorithm is a major need to compute watertight volumetric models from boundary representations to ensure consistent topological operations. The algorithm for transforming a set of topologically inconsistent polygons into topologically consistent volume models is described in Section 2. It is to mention in this context that the use of a graph database as back end would enable graph computing as described in Section 2.4 about TinkerPop. The node operations could be implemented using this API, which is implemented by the framework of the graph database to manage and process large graphs efficiently through the API.

A sensitivity analysis evaluates the proposed concept, which is given in Section 5.1. A couple of results generated from a LOD 2 *CityGML* input data-set of Erfurt, Thuringia (Germany) are provided in Section 5.1. Even if the GML standards require topologically consistent data, in reality they do contain overlaps which are not explicitly modelled in the data [25]. The aim here is to calculate higher-dimensional geometric models from topologically inconsistent lower-dimensional geometric models only. Therefore, the application is not restricted to city models, be they *CityGML*, IFC etc. It is also to say that any geometric model may be triangulated to higher-dimensional models in

that sense. I also do not focus on any specific application (e.g. city models, sub-surface models, land use models etc.) in order to provide a versatile spatial or spatio-temporal model to manage a broad variety of geo-data which in turn is easier to be analysed in a more general way, i.e. the way of topology and graphs. The evaluation of the proposed concept includes the evaluation of the parallelization of the algorithm.

As mentioned before, the use of a mediator/wrapper architecture may be considered for a multi-database query processing architecture using this proposed concept as global schema. Since the proposed schema is based on the *Property Graph Model* in combination with OGC's *General Feature Model*, it is also possible to implement alternative concepts (e.g. *peer-to-peer* architectures or *linked-data* concepts, see Section 2.2.3).

The third step is to implement and evaluate access methods which provide efficient spatial, spatio-temporal or topological access and can be used in a multi-database query processing architecture or centralized solution. This step is important when dealing with large graphs and graph queries. A common way of indexing graph nodes is by indexing a property. The properties in the case of spatial or spatio-temporal properties are multidimensional and topologically connected and need to be queried as such by using spatial, spatio-temporal or topological access methods. It is also important to know that the inherent structure of simplicial or polytope complexes and their interconnection through intersections (overlay) etc. form special graph structures. The knowledge about the possible structures helps to improve algorithms, proof results and speed up queries by integrating this knowledge into the development of new access methods. The implementation of a new scalable space filling curve index is given in Section 4.7.5. The evaluation of the new scalable space filling curve index is given in Section 5.2. The implementation of the new topological access method is given in Section 4.7.6. The evaluation of the new topological access method is given in Section 5.3.

4 Core framework for distributed and parallel data management

The ancestors of the newly developed core framework are *DB4GeO* and *DB3D*, which were developed by the working group of Martin Breunig [14]. Most of the ancestors sub frameworks have been refactored, revised and/or completely redesigned. The former operation layers and the service framework have not been taken into the new core framework. The technologies being used as back ends and front ends were too specialized or out of date for the desired network infrastructure. In order to provide one core framework, a new core framework unites some of *DB3Ds* spatial core sub frameworks and some of *DB4GeOs* spatio-temporal core sub frameworks and adds some extensions. I will refer to the new core framework as *DB4GeOGraphS* core framework. The following section gives a short overview. The section afterwards will discus each sub framework in greater detail.

4.1 Overview

Classes written in Java are collected in packages. Packages are able to have sub packages. A Java framework is a collection of classes which may be structured through the use of packages. Therefore, a Java framework is a package itself. Compiled versions are provided as libraries for deployment. *DB4GeOGraphS* core framework consists of 11 packages. Some of which contain sub packages.

1. ***graph*:** This package contains implementations concerning the new *Property Graph Model*, which replaces the object model of *DB3D*.
2. ***inaccuracy*:** This package contains implementations concerning numerical error handling of Javas double precision arithmetic.
3. ***spatial3d*:** This package contains an API package with all interfaces, a geometry package which contains all spatial geometries (simplices and helper geometries like vector, line, plane etc. for double arithmetic calculations) and a package for implementations concerning simplicial collections (simplicial elements, their topological collections, spatial predicate). It is the revised version of *DB3Ds* core framework.

4. ***spatial4d*:** This package contains an API package with all interfaces and a package for implementations concerning polytope collections (polytope elements, their collections, spatio-temporal predicates, interpolation). It is the revised version of *DB4GeOs* spatio-temporal framework. It uses the spatial3d package and the temporal package to define the polytopes used within the spatio-temporal model.
5. ***multidimensional*:** This package contains implementations concerning all multidimensional geometry types. It had been added to support the multidimensional scalable space filling curve index structure, which will be discussed in Section 4.7.5.
6. ***temporal*:** This package contains implementations concerning the temporal model. It is the revised version of *DB4GeOs* temporal framework.
7. ***thematics*:** This package contains implementations concerning the handling of basic node properties within the new *Property Graph Model*. It is the revised version of *DB4GeOs* thematic framework.
8. ***indices*:** This package contains implementations concerning all indexing methods. It contains the revised package of *DB3Ds* spatial index package, a revised package of *DB4GeOs* spatio-temporal index package and a new package for indexing multidimensional nodes of the new *Property Graph Model*.
9. ***operations*:** This package contains implementations concerning all implemented external operations (Import-, Export-, Triangulation-Operations)
10. ***exceptions*:** This package contains implementations concerning exception handling
11. ***tests*:** This package contains implementations for testing certain features of *DB4GeOGraphS* core framework.

The following sections will describe each package in greater detail.

4.2 Property Graph Model

The final report of the DFG Project "3D Data and model management for the geo-sciences with special consideration of topology and time" (project number: 94512227) contains translated extractions of this section and the papers [30], [31] and [32] contain extracted, transformed and minimized parts of this section, published beforehand:

4.2.1 The new object model

The object model of *DB3D* and *DB4GeO* has been replaced. The new object model is based on the *Property Graph Model* and OGCs *Simple Feature Model* and *General Feature Model* design patterns. Figure 4.1 shows the class diagram of the newly implemented *Property Graph Model*. Every node of a graph within the *Property Graph Model* carries properties. The property types have been chosen from *Simple Feature Model* and *General Feature Model* where every feature is build up by a spatial part/property, a temporal part/property and a thematic part/property. The properties are going to be closely discussed in Sections 4.3, 4.4, 4.5 and 4.6.

Three different node types have been implemented. One for non-spatial or non-spatio-temporal contents, one for spatial contents and one for spatio-temporal contents. Each of them contain the basic graph node properties: name (String), description (String), identification (*TONodeID*), temporal (Temporal) and thematic (Thematic). The *TONode* interface claims fundamental methods to implement the *Property Graph Model* and parts of the *General Feature Model* by adding methods which deal with the sets of connected *TONode* objects, the temporal property and the thematic property.

Two sub interfaces extend the *TONode* interface to implement the *Simple Feature Model* by adding methods which deal with the spatial property. It may be a bit confusing, but each extended sub interface of the *TONode* interface extends a *Spatial-3D/4D* interface, which the realizations (*TONode-3D/4D-Impl*) actually contain. It is NOT possible to add a *TONode-3D/4D* object as spatial property to a *TONode-3D/4DImpl* object. An exception will be thrown. The reason for this approach is to wrap a *TONode-3D/4D* as *Spatial-3D/4D* type to be able to use the spatial and spatio-temporal access methods for each *TONode-3D/4D* object, too. The basic spatial and spatio-temporal types are shown in Figure 4.2.

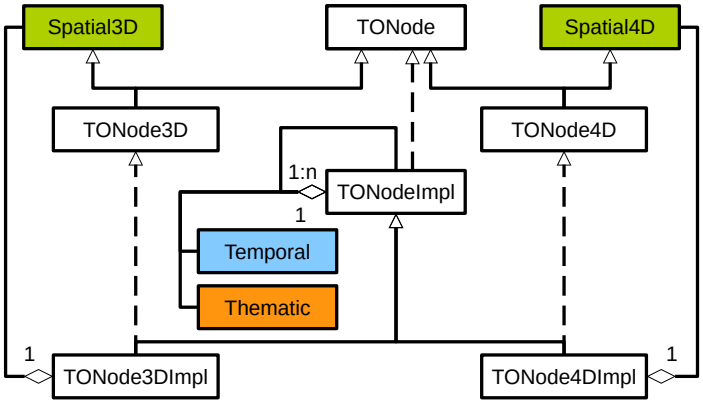


Figure 4.1: *Property Graph Model* (follows *OGCs General Feature Model*)

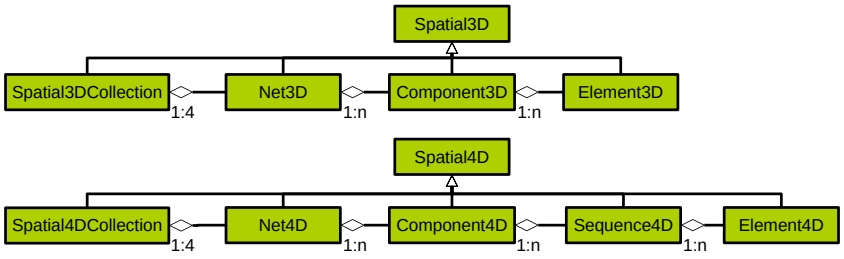


Figure 4.2: Spatial and spatio-temporal types

The *Property Graph Model* can be arranged by needs e.g. level of detail (see Figure 4.3), incidence graphs, aggregations, modelling steps by building a pipeline (e.g. from point cloud to simulation data set), *CSG* (constructive solid geometry) based on simplicial complexes or data distribution etc. A standard model has been implemented to provide the functionalities claimed by the API to manage their child-instances and their spatial-, temporal- and thematic-property and the interconnection of those parts.

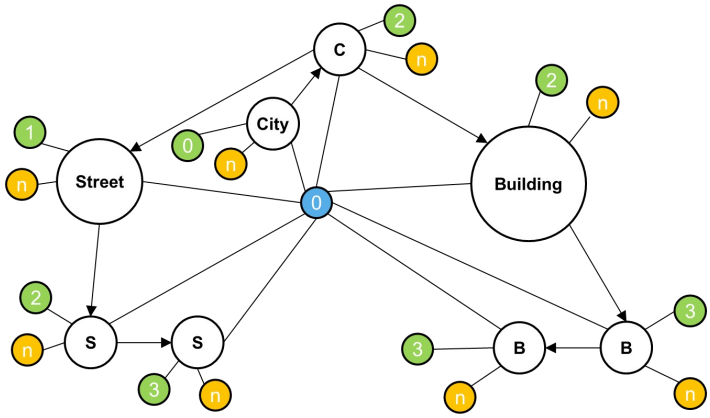


Figure 4.3: LOD-graph using new object-model
GREEN: spatial parts with dimension
BLUE: time-stamp with dimension
ORANGE: thematic parts with n dimensions
WHITE: Property Graph Nodes

This general approach shall provide the ability to adapt to distributed and parallel database concepts. It is still flexible enough to support any simulation implementation with or without spatial or spatio-temporal dependencies to not constrain the creativity of any involved scientist. Basic relation types have been distinguished in order to provide a graph schema which reflex common relations between features/geo-objects. The following subsection will discuss those relation types in greater detail.

4.2.2 Relations

The requirement analysis for 3d data and model management for geosciences, with special consideration of topology and time, has shown that a flexible and simple management of data and models with regard to geometry, topology, and the subject of matter is desirable. This has been confirmed in many discussions with the applied geosciences at KIT, but also especially in the cooperation with the French colleagues in the field of deep geothermal energy within the Soultz project [33]. This had a direct impact on the conception of the 3d and 4d data and model management.

Within the *G-Maps* implementations [15] the *one-to-one* relation between a database object with a thematic and a *Net3D* object of *DB3D* was softened by introducing intermediate levels. A *Net3D* object could be divided into several *Net3D* objects (aggregation / decomposition) which could be divided into several *Net3D* objects also and so on. The shapes of the *Net3D* objects on the intermediate levels were therefore dependent on each other and followed a hierarchy based on the level of detail as defined by [15]. Thus, it was not intended to build up hierarchies whose shapes have different geometrically functionally dependencies (e.g. a city is represented as a geometric centre up to tetrahedral nets in different level of details) or even being geometrically functionally independent of each other (e.g.: Paris is represented as the Eiffel Tower up to abstract representation of several 3d buildings of tourist interest) which is generally the case with specialization / generalization relations.

A specialization / generalization may not have similar spatial topological properties, depending on the geometrically functional dependency. Thus, it does not necessarily have to be created by the controlled introduction of separating points, separating edges or separating facets as it was the case in the former topology framework by [15] which followed the basic relation type of aggregation / decomposition only. Therefore, the former topology framework is too special to serve as a general basic model desired within *DB4GeOGraphS* core framework.

The former topology framework lacks implementations concerning a thorough thematic model for managing the thematic properties of any objects or the elements of the intermediate levels. In addition, the connection of the former topology framework with the new spatio-temporal model [38] or the previous spatio-temporal model [60] was limited to temporal snapshots. Real spatio-temporal topology in terms of *G-Maps* based on polytope complexes have not been introduced. All in all, the relations were too specialized in terms of conception and implementation.

The new object model introduces three basic relation types (*part-of*, *border-of* and *generalization-of*, where *border-of* links a border object into the direction of an interior object e.g. an edge with its bounded faces) and their inverses (i.e. *composite-of*, *inner-of* and *specialization-of*, where *inner-of* links an interior object into the direction of a border object e.g. a face with its bounding edges). The choice of using exactly those three basic relation types is inspired by the object-oriented programming paradigm which uses

abstractions and aggregations, and the incidence graph to model topological consistence. Therefore, modelling complex geo-objects where each part is topologically connected by incidence (e.g. a building as a solid with an awning as surface and an antenna modelled as curve), or aggregated to a more complex geo-object or interconnected to other geo-objects is possible. Even abstractions are possible.

Figure 4.4 and 4.5 illustrate two examples of the spatial model and spatio-temporal model. It is to mention that all relations may be bidirectional or unidirectional. Figure 4.4 and 4.5 also illustrates the α_1- and α_2-involutions known from *G-Maps*. An α_1 involution is applied by exchanging the dissimilar point ID of a segment with the dissimilar point ID of its neighbouring segment. This results in a movement along the border of C_0 or C_{12}. An α_2 involution is applied by exchanging the C_* IDs. This results in a movement from surface to surface using their common edge. The next sections will discuss each relation type in greater detail.

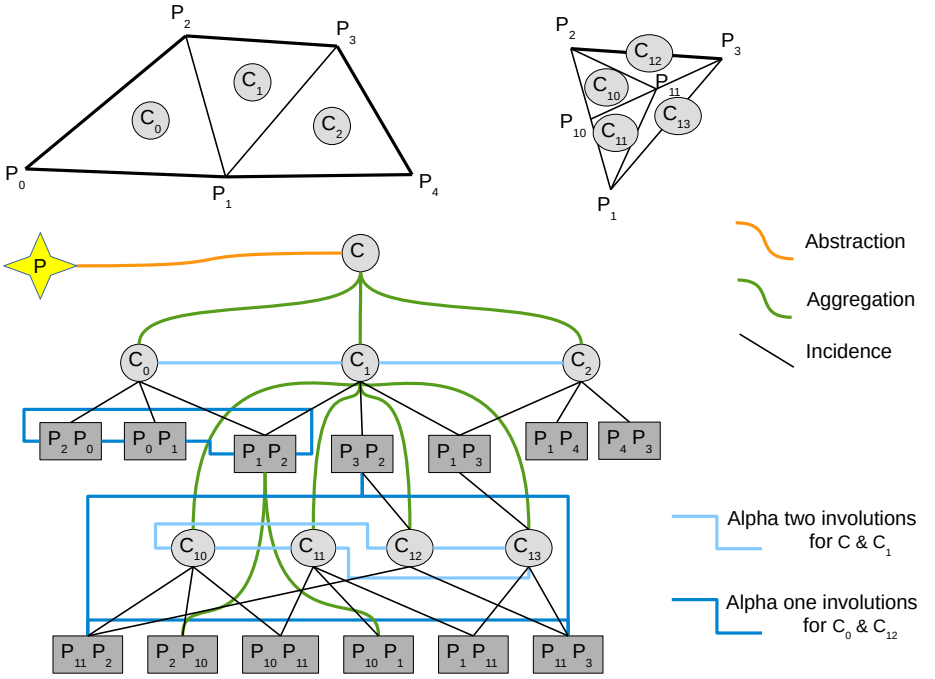


Figure 4.4: Relation types of a *Triang3DComponent* object *C* using *Property Graph Model*
GREY: Nodes of *Property Graph Model*

The relations to other nodes are organized by *Map*s as seen by the result type of the operation *getTargets()* shown in Figure 4.8. The *TARGETDIRECTION* enumeration type controls the relation types and is easily to be extended if needed. The *TONodeID* identifier type has been implemented with *Integer* identification and has to be unique for each

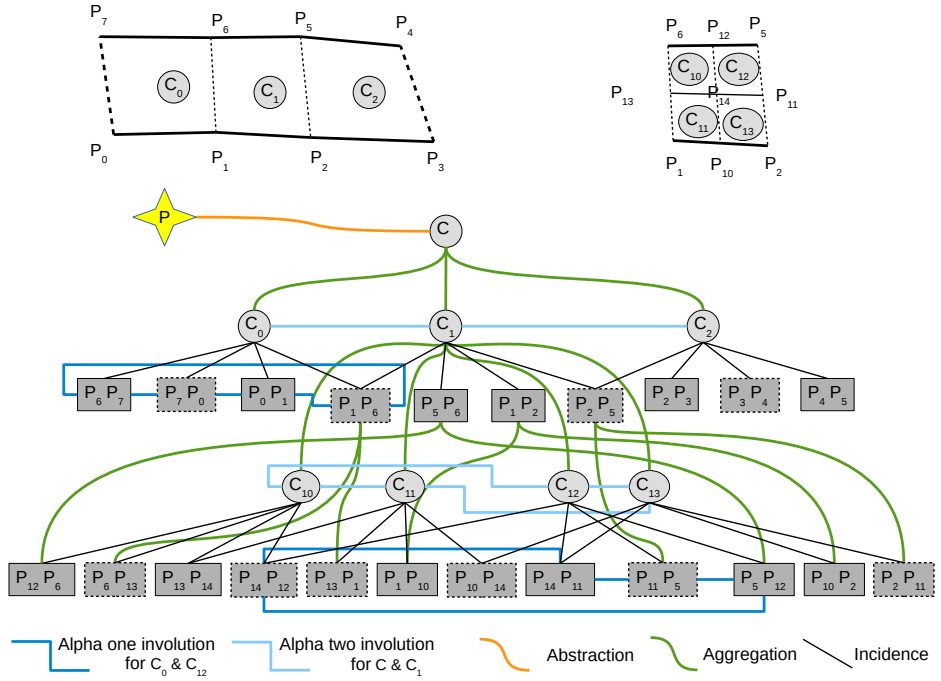


Figure 4.5: Relation types of a *Segment4DComponent* object *C* using *Property Graph Model*
GREY: Nodes of *Property Graph Model*

node in the system. Since the *TONode* interface extends the *Comparable* interface, the implementation of the *compare()* operation was based on the *TONodeID* type.

4.2.2.1 Aggregation relations → part-of and composite-of

The spatial properties described in Section 4.3 yield four different aggregation levels, and the spatio-temporal properties described in Section 4.5 yield five different aggregation levels. The different aggregation levels are also illustrated in Figure 4.2. If the spatial property of a *TONode3DImpl* object is a *Net3D* object, aggregation relations may be built to nodes which carry the contained *Component3D* objects. The *Component3D* objects in turn consist of *Element3D* objects. *TONode3DImpl* objects may be created for each *Element3D* object, which relate to their specific *Component3D* objects.

If the spatial property of a *TONode4DImpl* object is a *Net4D* object, aggregation relations may be built to nodes which carry the contained *Component4D* objects. The *Component4D* objects in turn consist of *Sequence4D* objects. The *Sequence4D* objects in turn consist of *Element4D* objects (spatio-temporal polytopes). The polytopes consist of exactly two

simplices (start and end state). For each spatio-temporal aggregation level, *TONode4D* objects may be created in the same way. Figure 4.6 illustrates this relation type.

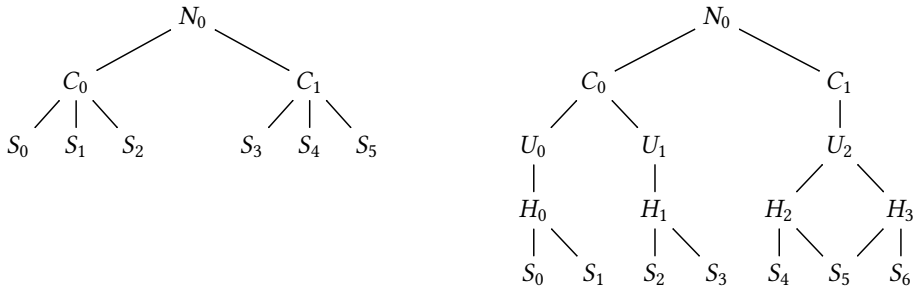


Figure 4.6: Example of aggregation relations, where the typical vertical hierarchy is given by N_i (Net) $\rightarrow$ C_j (Complex) $\rightarrow$ S_k (Simplex) using spatial properties (top) and N_i (Net) $\rightarrow$ C_j (Complex) $\rightarrow$ U_k (Spatial Sequence) $\rightarrow$ H_l (Polytope) $\rightarrow$ S_m (Pre-Simplex) or H_l (Polytope) $\rightarrow$ S_{m+1} (Post-Simplex) using spatio-temporal properties (bottom)

4.2.2.2 Incidence relations → border-of and inner-of

The spatial properties described in Section 4.3 and the spatio-temporal properties described in Section 4.5 yield different incidence levels. In case of $d = 0$, borders do not exist. A $(d-2)$-dimensional border of the $(d-1)$-dimensional border does not exist because all $(d-1)$-dimensional border properties are closed (without boundary) or do not exist.

Depending on an integrated aggregation relation, different kinds of border relations exist. The aggregation level of the border equals to the aggregation level of the spatial object, instead of the sequence and element levels. A *one-to-one* relation is built if the border of a node is directly linked to the node. But, for example in the case of a *Net3D* object, it is also possible to build the border *Net3D* object of the main node and retrieve all *Component3D* objects from the border *Net3D* object and link them to a set of new sub nodes which relate to the main node in an incidence relation. This concept has the advantage to minimize the *Property Graph* if some aggregation levels are not needed to be part of the *Property Graph*.

4.2.2.3 Abstraction relations → generalization-of and specialization-of

Functionally independent relations can be created under this kind of relation type. It is the relation type where none of the previously discussed relation types can be applied. This type has been applied to relate the results of a special node operation called *buildCores(…)* described in Section 4.2.3 to *TONode3DImpl* objects which carry a *Net3D* object as spatial property. In some cases the *Net3D* object is not retrieved from

the triangulation results which in turn may be interpreted as a special kind of viewing the *Net3D* object, a specialization, where else the *Net3D* object may be interpreted as generalization.

Figure 4.7 shows an example. It is clear to see that the border of the *Tetrahedron3DComponent* object which will be a closed (without boundary) *Triangle3DComponent* object (hull) is not equal to the *Triangle3DNet* object and that the *Triangle3DNet* object is not retrievable from the *Tetrahedron3DComponent* object. The algorithm looses input information (e.g. the shapes of the topologically inconsistent *Triangle3DNet* object) and adds information by transforming a surface type into a solid type with information about inner points and outer points which are not distinguishable by the *Triangle3DNet* object which *Triangle3DComponent* objects are not well-connected to form one (without boundary) *Triangle3DComponent* object as one hull like the border of the *Tetrahedron3DComponent* object.

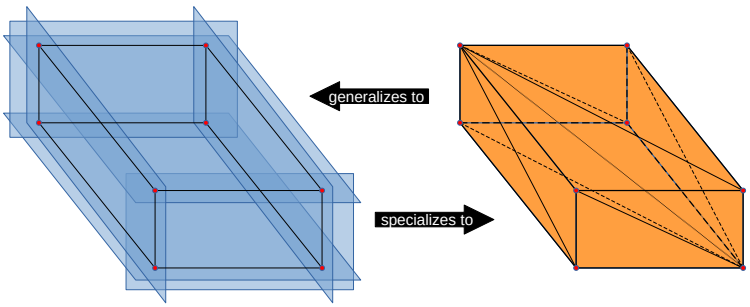


Figure 4.7: abstraction relation
BLUE: topologically inconsistent *Triangle3DNet* object made off six *Triangle3DComponent* objects
BLACK: surface intersections (left) triangle borders (right)
ORANGE: *Tetrahedron3DComponent* object as result of the tetrahedralization

4.2.3 Node operations

The *TONode*, *TONode3D* and *TONode4D* interfaces demand the implementation of basic node operations for manipulating the properties and the relations. The UML diagram is shown in Figure 4.8. The implementing classes are called *TONodeImpl*, *TONode3DImpl* and *TONode4DImpl* as the UML diagramm 4.1 reveals. The node operations of the *TONode4D* interfaces have not yet been implemented within the *TONode4DImpl*.

Nodes are able to build their own relation nodes based on their spatial or spatio-temporal types and relations. Those building operations may be outsourced to external *Builder* classes if needed. Five basic building operations have been implemented within the spatial model to support simulation implementations which are: *buildAggregation(…)*, *buildSubComponents(…)*, *buildOverlays(…)*, *buildBorders(…)* and *buildCores(…)*.

Figure 4.8: Interfaces *TONode*, *TONode3D* and *TONode4D*

Each operation takes an access method as parameter to cross-check if any of the resulting nodes is already part of the access method. If they are not part of the access method, they will be added to that access method. If they are part of the access method, only new relations will be added to the found node. Each access method provide different equality types in order to decide if nodes are equal. Those types are described in section 4.3.4. If the access method contains the global graph, the global graph will always be up-to-date.

The *buildAggregation(…)* collects the spatial properties of all *composite-of* neighbours into one *Spatial3DCollection* object. A *TONode3DImpl* node which carries the *Spa-*

tial3DCollection object replaces the node which called that operation within the *Property Graph*.

Its inverse, the *buildSubComponents(…)* operation, creates all sub nodes by analysing the spatial property. In case of a *Spatial3DCollection* object as spatial property, all non-empty *Net3D* objects are linked with new sub nodes. Each new sub node is related to the main node in an aggregation relation. Figure 4.2 illustrates all possible aggregation relations within the spatial and spatio-temporal model. The standard behavior as described may be changed by manipulating the spatial predicate within the *buildSubComponents(…)* operation, e.g. a node with a *Spatial3DCollection* object as spatial property may create new sub nodes which are linked to all contained *Element3D* objects (simplices) which removes the aggregation levels in between and minimizes the *Property Graph*.

The *buildOverlay(…)* operation calculates all strict intersections (border intersections are ignored) with a set of given nodes. Each *node-to-node* intersection is linked to a new node, and the new node is related to its parents in an aggregation relation.

The *buildBorders(…)* operation creates the border nodes of a node by calling the *getBorder()* operation of the spatial property and adds the nodes as described in Section 4.2.2.2. A spatial predicate described in Section 4.7.1 was used to control which aggregation level is retrieved from the border object and linked with the new sub nodes, as described for the *buildSubComponents(…)* operation above.

The *buildCores(…)* operation is rather complex (see Algorithm 2). Figure 4.7 illustrates an example input (left) and an example output (right). The main idea is to picture a d-dimensional simplicial complex as a d-dimensional patch, and stitch a couple of d-dimensional patches together wherever there is a $(d-1)$-dimensional intersection between them to form a patchwork. This patchwork can be transformed into an incidence graph which in turn can be analysed to find d-dimensional simplicial complexes, parts of the patchwork which form a d-dimensional ball. Those d-dimensional balls can be triangulated to $(d+1)$-dimensional simplicial complexes. The disjoint set of $(d+1)$-dimensional simplicial complexes can be found by rejecting all triangulated $(d+1)$-dimensional simplicial complexes which intersect strictly with any of the patchwork patches.

The pre-processing steps 1 to 5 build the T_0-space until a pool of threads is able to triangulate the disjoint $(d+1)$-dimensional simplicial complexes for the set of $(d-1)$-dimensional intersections. This step 6 of Algorithm 2 is described in Algorithm 3 for one thread, which tries to find disjoint $(d+1)$-dimensional simplicial complexes for each pair of patches which are connected to one specific seam. The final Step 7 calculates the difference if $(d+1)$-dimensional simplicial complexes created at step 1 did not connect to the T_0-space of the rest. Two examples are given by Figure 4.9 to illustrate the basic steps of Algorithm 2. The example shown at the top of Figure 4.9 illustrates the situation to calculate the difference with the d-dimensional simplicial complex as input (black), its triangulated $(d+1)$-dimensional simplicial complex in Step 1 (grey) and the difference in Step 7 (grey).

Algorithm 2: computing a set d-dimensional simplicial complexes into a set of $(d+1)$-dimensional simplicial complexes

input : set of d-dimensional simplicial complexes
output: disjoint set of $(d+1)$-dimensional simplicial complexes

1 collect $(d-1)$-dimensional intersections;
2 create disjoint d-dimensional patch nodes;
3 create disjoint $(d-1)$-dimensional seam nodes;
4 stitch patch nodes which form manifolds and clean up;
5 stitch seam nodes which share the same patch nodes;
6 stitch patch nodes to form *BREPs* and triangulate them to disjoint $(d+1)$-dimensional simplicial complexes;
7 create difference;

Algorithm 3: run() operation of a thread for Step 6 of Algorithm 2. Needs to run for every single seam node.

input : seam node s, spatial access method SAM^d containing all d-dimensional patch nodes, spatial access method SAM^{d+1} containing all disjoint $(d+1)$-dimensional simplicial complexes
output: spatial access method SAM^{d+1} containing all disjoint $(d+1)$-dimensional simplicial complexes

1 **for** *patch* p_0 *inner-of* s **do**
2 **for** *patch* p_1 *inner-of* s *with* $p_1 != p_0$ *or any other predecessors of* p_0 *and* p_0 *and* p_1 *are both not border-of some* c *in* SAM^{d+1} *and* p_0 *and* p_1 *are on the same plane or line (in case of CURVE3D)* **do**
3 L = new empty set of patch nodes that closed patchworks;
4 q = false;
5 **while** *not* q **do**
6 q = true;
7 M = new empty set of nodes for the patches of the patchwork;
8 R = new empty route map;
9 Dijkstra(s, p_0, p_1, L, M, R);
10 glue all patches in M to one closed (without boundary) manifold m^d;
11 m^{d+1} = triangulate m^d
12 **if** m^{d+1} intersects strict any patch in SAM^d **then** reject m^{d+1} and set q = false;
13 **if** m^{d+1} intersects strict any simplicial complex in SAM^{d+1} **then** reject m^{d+1} and set q = false;
14 **end**
15 **end**
16 **end**

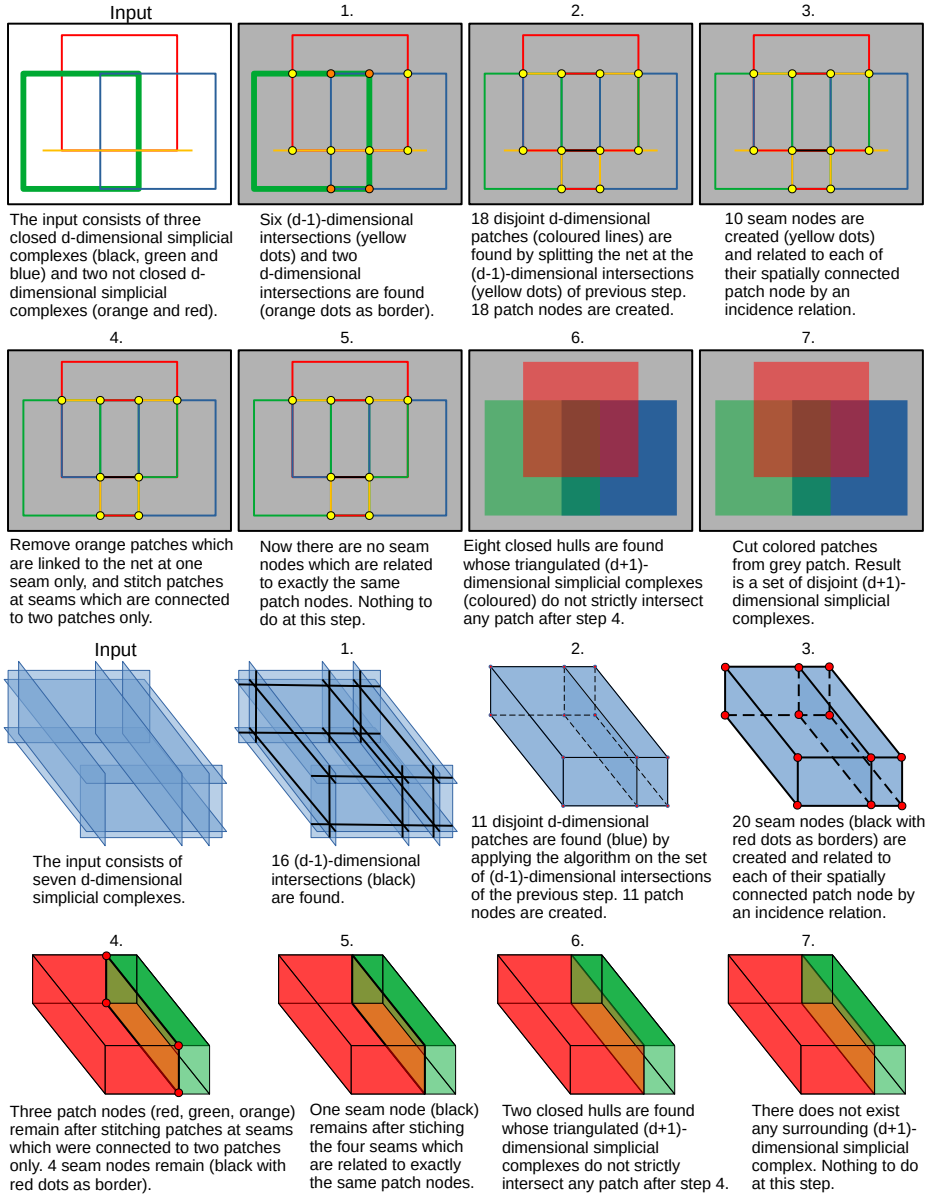


Figure 4.9: Example (top): Five 1-dimensional simplicial complexes triangulated to eight 2-dimensional disjoint simplicial complexes. Example (bottom): Six 2-dimensional simplicial complexes triangulated to one 3-dimensional simplicial complex.

The complexity of finding all $(d-1)$-dimensional simplicial complex intersections in Step 1 depends on the used spatial access method for the n input d-dimensional simplicial complexes. There are m seams which are all $(d-1)$-dimensional intersections, the borders of each d-dimensional intersection and the borders of each d-dimensional simplicial complexes. Creating k inner disjoint d-dimensional patches for Step 2 depends on the dimension d. If $d = 1$, it also depends on the spatial access method to retrieve the d-dimensional simplicial complexes which need to be split by those m seams. And in case of $d > 1$ the complexity of Step 2 is equal to the complexity of Algorithm 2 with $d' = d - 1$ and m seams as input since this algorithm creates the patches from the m seams, recursively. The creation of the spatial access method for the patches needs to be considered within the calculation of complexity. The complexity of Step 3 depends on this spatial access method for k patches to find all l seams by intersecting each of the k patches with another. The creation of the spatial access method for the m seams needs to be considered, too. Step 5 takes $\mathcal{O}(m)$ steps to glue patches to one manifold by iterating over the list of m seams. Step 6 to stitch patches to form valid B-Reps is parallelized for each seam through Algorithm 3. Figure 4.10 which illustrates an example of Algorithm 3 and the complexity of Algorithm 3 will be discussed in the following two paragraphs. The next Step 7 of Algorithm 2 depends on the number of patches w which could not be connected to the T_0-space. The complexity to calculate the difference to the list of v inner disjoint $(d+1)$-dimensional simplicial complexes resulting from Step 6 of Algorithm 2 depends on the used spatial access method for v inner disjoint $(d+1)$-dimensional simplicial complexes queried by w patches. The creation of the spatial access method for v inner disjoint $(d+1)$-dimensional simplicial complexes need also to be considered.

Figure 4.10 illustrates Algorithm 3 of Step 6 of Algorithm 2. The input of Algorithm 2 is shown in the top right (5 segment complexes) of Figure 4.10. The result of Steps 1 to 5 of Algorithm 2 is shown in 6.1 (patches are coloured, seams are yellow dots) of Figure 4.10. There is also one triangulated grey surface already found, bounded by the black segment complex, which did not intersect any of the other coloured segment complexes. Two patches (thick blue and black line) are connected through one seam (red dot) in 6.2 of Figure 4.10. The goal is to stitch a patchwork with those two patches, which forms a valid B-Rep. The idea is to follow the connected patches (dotted lines) on the green seams (see 6.3 of Figure 4.10). One patch is ignored (black, red dotted line). It was rejected in a former iteration. All yellow seams have to be checked in 6.4 of Figure 4.10. Starting with the blue seam, two options are available (patch a in orange and patch b in red). Taking the patch a first leads to an invalid loop since it maps to the same green seam of the black starting patch where the blue seam is connected to. This loop is ignored. Taking the patch b leads to a valid loop, since it maps to the green seam of the blue starting patch, where the blue seam is not mapped to. The orange square can be triangulated and checked against any one dimensional intersection, which would indicate that the found surface is not a distinct surface in 6.5 of Figure 4.10. The patch b is added to the list of patches which closed some invalid loop to be ignored when restarting at Step 6.2 of Figure 4.10.

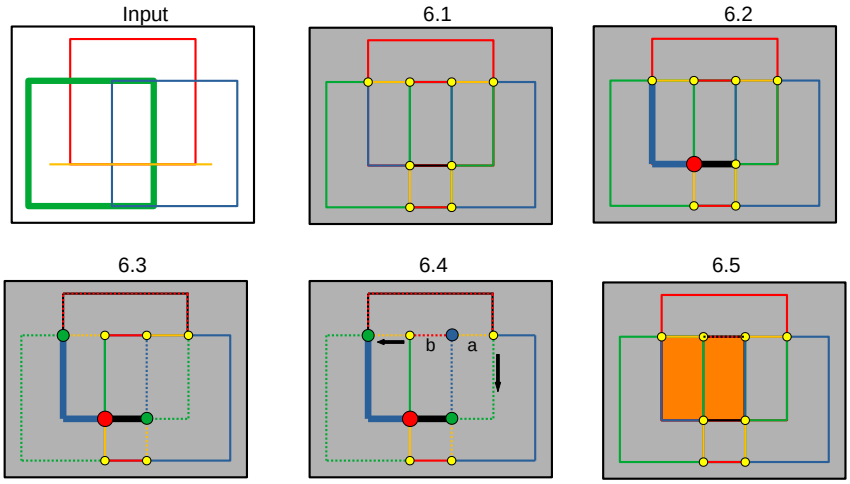


Figure 4.10: Illustration of Algorithm 3, Step 6 of Algorithm 2.

As mentioned before, Algorithm 3 describes how to find inner disjoint $(d+1)$-dimensional simplicial complexes for each pair of patches which are connected to one specific seam, which realizes Step 6 of Algorithm 2. In the 1-dimensional case, each patch is supposed to be a planar curve. Within the second FOR-loop (see Step 2 of Algorithm 3) patches p_1 will be rejected if it is not on the plane of p_0. If a plane cannot be defined by patch p_0 (patch lies on a straight line), a plane is defined by p_0 and p_1 for later checks within the Step 9. If a plane cannot be defined by patch p_0 and p_1, then every p_* which will be determined within Step 9 may help to define a plane. If a plane was defined before, every p_* will be ignored in step 9 which does not lie on that plane. If not, and the union of all patches from Step 9 are not able to define a plane, then all patches are on a straight line and a triangulation is not possible, since a straight line cannot be closed. However, the step to create a patchwork which can be triangulated to some $(d+1)$-dimensional simplicial complex is Step 9. This step will be discussed in Algorithm 4. After that step, the patchwork is glued to one closed (without boundary) d-dimensional simplicial complex and triangulated to a $(d+1)$-dimensional simplicial complex. If the patchwork could not be glued to one closed (without boundary) d-dimensional simplicial complex or the triangulation returned NULL, the algorithm does not try to find more closed (without boundary) d-dimensional simplicial complexes for the patches p_0 and p_1. If the triangulation was done successfully, the resulting $(d+1)$-dimensional simplicial complex is tested as follows. If the resulting $(d+1)$-dimensional simplicial complex strictly contains any patch or strictly intersects any previously created $(d+1)$-dimensional simplicial complex (see Step 12 and 13) the final patch which closed the corresponding d-dimensional simplicial complex within Step 9 is added to a set of patches for the patches which closed any of the previously generated patchworks. Those patches will be ignored for the next trials of Step 9.

Algorithm 4: Dijkstra(s, p_0, p_1, L, M, R) operation, Step 9 of Algorithm 3.

input : s, p_0, p_1, L, M, R
output: L,M,R

1 add entries $(s, NULL)$, (p_0, s) and (p_1, s) to R;
2 D new Queue (with first in first out);
3 P_0^b = border-of p_0;
4 add all s_0 in P_0^b with $s_0 != s$ to R as (s_0, p_0) and to D;
5 P_1^b = border-of p_1;
6 add all s_1 in P_1^b with $s_1 != s$ and s_1 is not contained by R to R as (s_1, p_1) and to D;
7 **while** *D is not empty* **do**
8 s_D = first of D;
9 p_D = value of s_D in R;
10 **for** *patch p inner-of s_D with $p != p_D$, p not bounded by s and L does not contain p* **do**
11 **if** p_0 is a CURVE3D **then** check if p_0, p_1 and p are on the same plane;
12 **if** previous check passed in case of CURVE3D and addPatch(p, s_D, R, D, M, L) returned "true" **then** break the for-loop;
13 **end**
14 **end**

Step 9 of Algorithm 3 follows the idea of a Dijkstra algorithm with edge values of one as first straight forward implementation (see Algorithm 4) to find closed loops within the T_0-space. The set of loops represents balls containing the two patches p_0 and p_1 and the connecting border seam s. Any loop may fulfil the geometrical constraint (see Step 12 and 13 of Algorithm 3). In fact, it is possible that the longest loop may fulfil the geometrical constraint only. Therefore, it is not the purpose to find the shortest loop here. It is a fact that most path-finding algorithm have a similar idea as Dijkstra's which is explained as follows. Let B be the set of borders from two patches p_0 and p_1 without the connecting border seam s. Algorithm 4 starts with this set of borders by the use of a Queue. This is a collection designed for holding elements prior to processing (with first in first out). The seam s functions as a barrier. Following the relations of the T_0-space, each border is connected to a set of patches. But each border needs to be stitched to only two patches in order to fulfil the manifold constraint. The first patch is where the Dijkstra comes from, and the second patch is where the Dijkstra goes to. As mentioned before, in case the patches are planar curves, only patches are taken into account which lie on the previously defined plane. If a plane was not defined, each of the patches may help to define a plane. However, the patches themselves are also bounded by some border seams. Each patch will be added by Algorithm 5. If this algorithm returns "true" then the second patch closed the d-dimensional simplicial complex or a path back to some border(s) in B. Algorithm 5 returns "true" only if a non-self-intersecting loop was found. The patch is added to the set of patches M which represents the resulting

patchwork. The complexity of Algorithm 5 depends on the number of interconnected borders and patches. The worst case would be that the non-self-intersecting loop with the most edges in the T_0-space forms the d-dimensional simplicial complex, which fulfils the constraints of Step 12 and 13 of Algorithm 3.

Algorithm 5 checks if one of those new border seams was already visited and is connected to a different border seam within B. If so, a loop is found. It may happen that the second patch is bounded by only one border. This patch is like a capping, and closes the border it is connected to the hole of the first patch. However, if a second patch like this or a loop was found, all the patches of this loop or the path back to the seam s are added to the result set M. If the second patch brings in some new borders, then they need to be closed in the same manner. Algorithm 5 filters the set of borders D which need to be visited in the next step of the Dijkstra if any second patch was found. All borders which contain any patches in M on their path back to the seam s are rejected. On the other hand, all new borders of the second patch need to be added to D if the second patch contains borders which need to be closed.

If a seam is connected to n patches, the complexity will be $\mathcal{O}(n^2)$ times the complexity within the two for loops (see Algorithm 3). If there are k closed (without boundary) d-dimensional simplicial complex which can be triangulated to k $(d+1)$-dimensional simplicial complexes, then it takes $\mathcal{O}(k)$ steps to find the one which fulfils the two constraints (see Step 12 and 13) in the worst case. The worst case is that the d-dimensional simplicial complex consists of the maximal number of patches relative to the other d-dimensional simplicial complexes. This one would be found at last within the Step 9. The Dijkstra itself will take $\mathcal{O}(m)$ for m patches.

Algorithm 5: addPatch(p, s_D, R, D, M, L) operation, Step 12 of Algorithm 4.

input : p, s_D, R, D, M, L
output: boolean

```
1  add entry (p, s_D) to R;
2  if s_D is connected to p_0 using R then b = border of p_0 on the route;
3  if s_D is connected to p_1 using R then b = border of p_1 on the route;
4  q = "false";
5  if p has only one connected seam (p is capping) then
6      add all r's in R (patches only) with r! = p_0 and r! = p_1 following from key s_D
         until p_0 or p_1 is reached to M;
7      add p to M and L;
8      fix Queue D;
9      q = "true";
10 else
11     for seam s border-of p with s! = s_D do
12         if R contains s then
13             if D contains s then
14                 remove s from Queue D;
15                 if s is connected to p_0 using R then b' = border of p_0 on the route;
16                 if s is connected to p_1 using R then b' = border of p_1 on the route;
17                 if b! = b' then
18                     add all r's (patches only) with r! = p_0 and r! = p_1 following from
                         key s until p_0 or p_1 is reached to M;
19                     fix Queue D;
20                     q="true";
21                 end
22             end
23         else
24             add entry (s, p) to R;
25             add s to Queue D;
26         end
27     end
28     if q then
29         add all r's in R (patches only) with r! = p_0 and r! = p_1 following from key s_D
             until p_0 or p_1 is reached to M;
30         add p to M and L;
31         fix Queue D;
32     end
33 end
34 return q;
```

4.3 Spatial properties

The spatial property implementations of the *DB4GeOGraphS* core framework can be found in the package called *spatial3d*. Figure 4.11 shows the class diagram of the refactored, revised and redesigned spatial model of DB4GeO. The basic subdivision into different class hierarchies concerning dimension and aggregation types did not change. It was reduced by removing the *Volume* Interface for closed (without boundary) triangle complexes. The functionalities are transferred into the triangle complex implementation and are callable if the triangle complex is closed (without boundary). *Volume* objects in *DB4GeOGraphS* core framework are always interpreted as true *Solid* objects made of tetrahedra, not as *BREPs*.

The major differences between the old model and the new model lies in the implementations concerning the intersection operations, lies in adding new difference operations, new equality tests and a set of new contains test (point and simplex), in retrieving border objects, adding a new helper interface called *Spatial3DCollection* to manage different dimensional types in one basic object and adding a new inaccuracy implementation described in Section 4.3.1. Each of the differences will be discussed in greater detail. An overview of the operations contained by the interface *Spatial3D* is given in Figure 4.12 (left) and an overview of the operations contained by the interface *Simplex3D* is given in Figure 4.12 (right).

As seen in Figure 4.11, different kinds of aggregation types exist, *Spatial3DCollection*, *Net3D*, *Component3D* and *Element3D* where each higher aggregation level consists of objects of the lower aggregation level. Therefore, the implementation of the interface *Spatial3DCollection*, which is called *Spatial3DCollectionImpl* manages four different *Net3D*s (*Point3DNet*, *Segment3DNet*,*Triangle3DNet* and *Tetrahedron3DNet*). This is the highest aggregation level.

The abstract realization of *Net3D*, called *Net3DAbst*, manages a set of *Component3D* objects with the help of a *R^*-Tree* implementation. It is the topological sum of this set of *Component3D* objects. The topological sum equals the disjoint union of the topological spaces. Therefore, the combinatorial Euler characteristic for *Net3DAbst* objects is implemented as the sum of each combinatorial Euler characteristic of the contained *Component3D* objects.

A *Component3D* object represents a d-dimensional simplicial complex (see Section 2.3.4). The abstract realization of *Component3D* is called *Component3DAbst*. The *Component3DAbst* abstract class realizes most of the basic operations concerning d-dimensional simplicial complexes with the help of the *Element3D* interface. This includes operations for adding or retrieving *Element3D* objects or operations like returning all triangles, segments, or points of a tetrahedron complex, for example. Realizations of the *Element3D* interface provide lists of *Element3D* objects to link neighbouring *Element3D* objects in order to provide topological access in the sense of Section 2.3.4). The extended

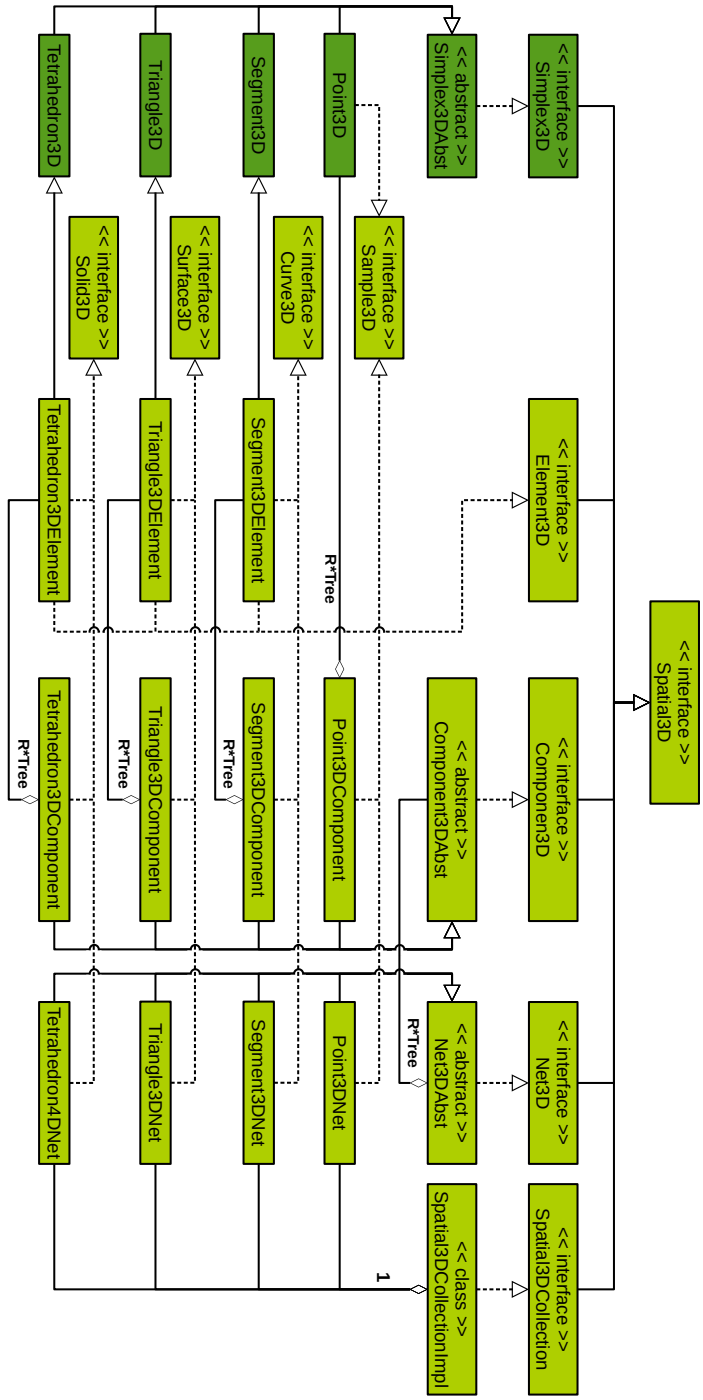


Figure 4.11: Spatial property model

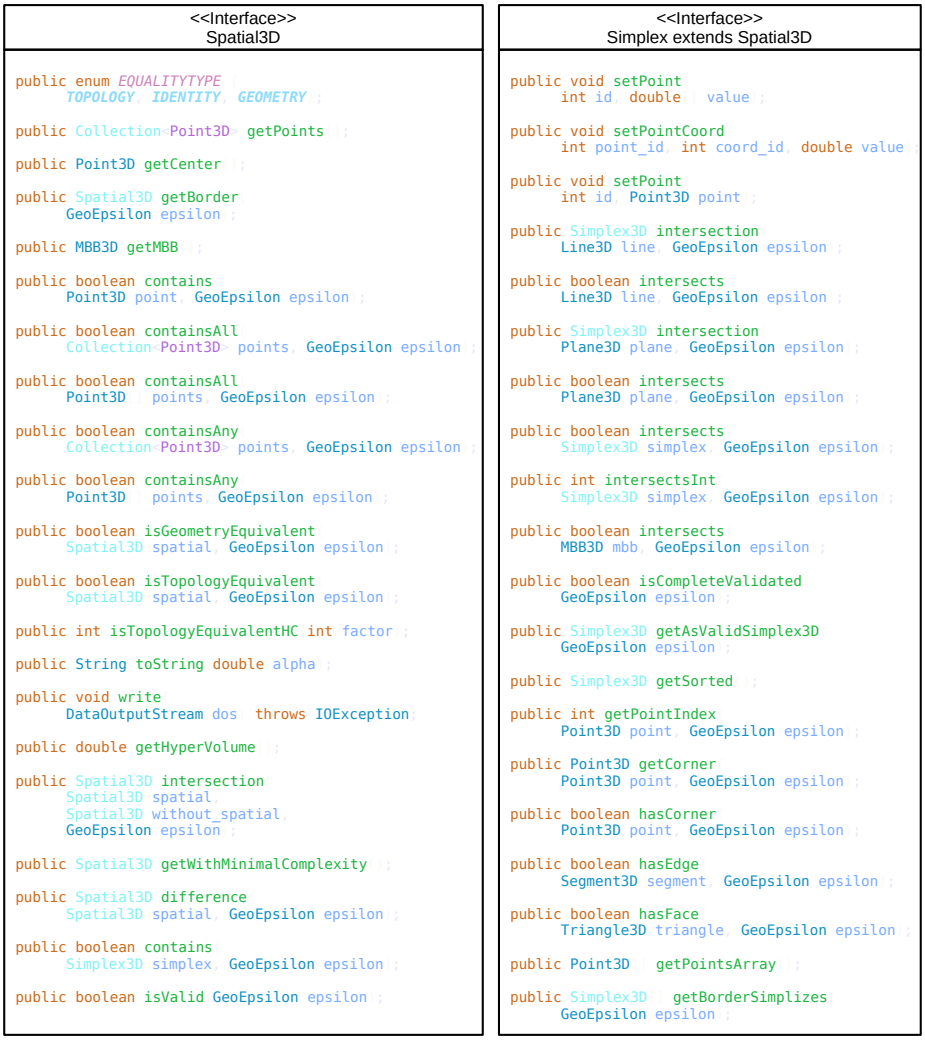


Figure 4.12: Interfaces *Spatial3D* (left) and *Simplex3D* (right)

d-dimensional realizations of *Component3DAbst* manage their *d*-dimensional *Element3D* objects with the help of a R^**-Tree* also.

DB4GeOGraphS core framework allows holes inside of triangle complexes and tetrahedron complexes. This is different to its predecessors. Therefore, the Euler characteristic of those *Net3D* objects may not be equal to the number of simplicial complexes contained by those *Net3D* objects.

All geometry calculations are done by the classes within the *geometries* package, which contains certain helper classes to establish double precision arithmetic within the 3-dimensional space. The API may be extended to spline geometries or other models, if needed. Different ways of numerical solvers (e.g. integer arithmetic or interval precision) are also integrable by implementing new geometry classes, if needed.

DB4GeOGraphS core framework uses the *VTK* library (see Section 2.4) with Java wrapping. Therefore, *VTK* as library is usable within the whole framework. The *VTK* library is used to export *XML*-files which can be imported into *ParaView* for visualization purposes. But the geometry types of *VTK* can be wrapped by using the *Spatial3D* interface or sub-interfaces (e.g. *Net3D*, *Component3D* and *Element3D*) as an extension. This is a straight forward solution to use *VTK* as geometry core of *DB4GeOGraphS* core framework which also adds the management of spline types, structured grids, image data or raster data with regular topology of standard voxel types by the object model of *DB4GeOGraphS* core framework, the *Property Graph Model*. The integration of *VTK* also adds the processing abilities of *VTK*. This enables the establishment of complex processing pipelines within the *DB4GeOGraphS* core framework.

4.3.1 Inaccuracy

The inaccuracy model was revised. The main class is called *GeoEpsilon*. This class provides all operations to compare two double values ($a < b$, $a > b$, $a == b$, $a <= b$ and $a >= b$). Five different scaling factors need to be chosen in addition to one fixed value as the basic constant α. The basic constant α has to be set to 10^{-n} with n being a desired integer number and $n > 0$. So there is one precision parameter α which scales all precision types and five different factors for each precision type:

1. **Base precision value** α
   as base value for all the different precision types
2. **Positional precision** $\epsilon = \alpha\epsilon'$
   with factor ϵ' for distance, length, area and volume comparison
3. **Positional line precision** $\varepsilon = \alpha\varepsilon'$
   with factor ε' for point or line on-line comparison
4. **Positional plane precision** $\epsilon = \alpha\epsilon'$
   with factor ϵ' for point, line or plane in plane comparison
5. **Angular precision** $\zeta = \alpha\zeta'$
   with factor ζ' for cosine comparison
6. **Skew precision** $\lambda = \alpha\lambda'$
   with factor λ' for skew comparison of two lines

The positional precision ϵ is used within the operations *isTopologyEquivalent(…)*, *isGeometryEquivalent(…)*, *isValid()*, *isRegular()* and *contains(Point3D)*. Smaller values reduce the buffers around points which leads to less point equality as well as smaller lengths of segments, areas of triangles and volumes of tetrahedrons are allowed. The positional line precision ε is used within the *Line3D* operations *distance(Line3D)*, *intersectsInt(Line3D)*, *intersection(Line3D)* and *contains(Point3D)*. Smaller values reduce the buffer, which leads to fewer identifications. The Positional plane precision ϵ is used within the *Plane3D* operations *contains(Point3D)* and *intersectsInt(Line3D or Plane3D)*. Smaller values lead to fewer identifications. The Angular precision ζ is used within the $Vector3D$ operations *isCollinear(Vector3D)*, *isOrthogonal(Vector3D)* and *isParallel(Vector3D)*. Smaller values lead to thinner angles. This in turn leads to less collinear, less parallel and less orthogonal identifications of planes and lines. The skew precision λ is used within the *Line3D* operation *isSkew(Line3D)* - operation. Smaller values lead to fewer identifications.

4.3.2 Intersection

Paper [31] contains extracted, transformed and minimized parts of this section, published beforehand:

The intersection model has been revised with focus on the simplification of the implementations. The intersection operation includes a difference operation if the parameter for the spatial object to be subtracted is set. In order to calculate the intersections, each d-dimensional simplex provides operations to calculate its intersection with a line and a plane as defined within the *Simplex3D* interface. The abstract class *Simplex3DAbst* in turn uses these operations for its own intersection operation with any type of spatial object. Inside this intersection operation, each intersection calculations with higher aggregation levels will be delegated to their intersection operations. So the intersection operation of a simplex reduces to a simplex-to-simplex intersection.

Algorithm 6 shows the basic steps taken to calculate a simplex-to-spatial intersection. Step 11 of Algorithm 6 includes a triangulation, which is described in Section 4.8.1.1.

Algorithm 6 is used within the intersection operation of the abstract realization *Component3DAbst*. Intersections with higher aggregation levels will be delegated to their intersection operations as seen within the intersection operation of *Simplex3DAbst*. Let A be the *Component3DAbst* object from where the intersection operation is called. Let B be an arbitrary *Spatial3D* object to intersect A with, and let C be the *Spatial3D* object which will be subtracted from the intersection. An intersection with a *Simplex3D* B is calculated by querying the R^**-Tree* of A. For all *Simplex3D* objects which *MBBs* intersect the *MBB* of the *Simplex3D*, B the intersection is calculated and C is subtracted from each intersection. Intersections with the same aggregation level will be calculated for each simplex of A which *MBB* intersects with the *MBB* of B using the R^**-Tree* of A and which is not contained in C. The set of intersecting simplices of B for each of those simplices of A is retrieved by querying the R^**-Tree* of B for each of those simplices of A.

Algorithm 6: simplex-to-spatial intersection

input : Simplex3D A and Spatial3D B to intersect with another and Spatial3D C to subtract from the intersection

output : Intersection of A and B without C

1 if A or B is an instance of Point3D
  → return results of contains-point test;
2 if aggregation level of B > aggregation level of A
  → return B intersection with A;
3 if MBB(A) does NOT intersect MBB(B)
  → return null;
4 if B contains-all-points of A
  → return A difference with C and vice versa;
5 create Wireframe3D W containing each point of A which is contained by B and vice versa;
6 if A is a SOLID3D
  → $B' = B$;
7 if A is a SURFACE3D
  → $B' = B$ intersection with plane of A;
8 if A is a CURVE3D
  → $B' = B$ intersection with line of A ;
9 intersect border of B' with A and add results to W;
10 intersect border of A with B' and add results to W;
11 triangulate W into a Spatial3D with maximal dimension of minimum(dimension(A), dimension(B)) and return result;

The hypervolume for each simplex of A is tested against the sum of the hypervolumes of all intersections to every simplex of B, which intersected the simplex of A. If the hypervolume is equal in accordance to the inaccuracy definitions, the whole simplex of A will be added to the result set to minimize subdivisions. A *Spatial3DCollectionImpl* (realisation of *Spatial3DCollection*) object collects all intersections and tries to build a valid spatial object as result object.

The intersection operations defined in *Net3DAbst* and *Spatial3DCollectionImpl* call the intersection operations for each contained simplicial complex and collects the intersections inside a *Spatial3DCollectionImpl* object to build a valid spatial object as well.

4.3.3 Difference

Paper [31] contains extracted, transformed and minimized parts of this section, published beforehand:

A difference operation is added to be able to calculate all boolean operations needed in set theory. Algorithm 7 shows the basic steps to calculate a simplex-to-spatial difference defined within the abstract class *Simplex3DAbst*.

Algorithm 7: simplex-to-spatial difference

input : Simplex3D A and Spatial3D B
output : A without B

1 if MBB(A) does NOT intersect MBB(B) → return A;
2 d = dimension(A);
3 I = intersection between A and B;
4 if dimension(I) != dimension(A)
  → return A;
5 if I is a NET3D
  → subtract each simplicial complex of B from A and return difference;
6 A_b = border of A;
7 I_b = border of I;
8 A'_b = A_b without I_b;
9 I'_b = I_b without A_b;
10 A'_{bb} = border of A'_b;
11 D_b = intersection of A'_b and I'_b without A'_{bb};
12 if D_b! = $NULL$ →
  triangulate simplices around simplices of A'_{bb}
  add them to result set
  add their united border to A'_b;
13 if D_b == $NULL$ and A'_{bb} == $NULL$ →
  triangulate simplices between A'_b and I'_b
  add them to result set
  add their united border to A'_b;
14 glue A'_b and I'_b and triangulate all *BREPs*;
15 add the triangulated *BREPs* to result set;
16 build a valid *Spatial3D* from result set and return it;

Algorithm 7 is used for each simplex of a *Component3DAbst* object if the *Spatial3D* object is an instance of *Component3D* or *Element3D*. If the *Spatial3D* object is an instance of *Net3D* or *Spatial3DCollection*, the difference is calculated between the *Component3DAbst* object and all the simplicial complexes of the *Net3D* or *Spatial3DCollection* object. A new *Spatial3DCollectionImpl* object collects all differences and to build a valid spatial object as

result object. The difference operations defined in *Net3DAbst* and *Spatial3DCollectionImpl* call the difference operations for each contained simplicial complex and collects the differences into a new *Spatial3DCollectionImpl* object, which is able to create a valid spatial object as well.

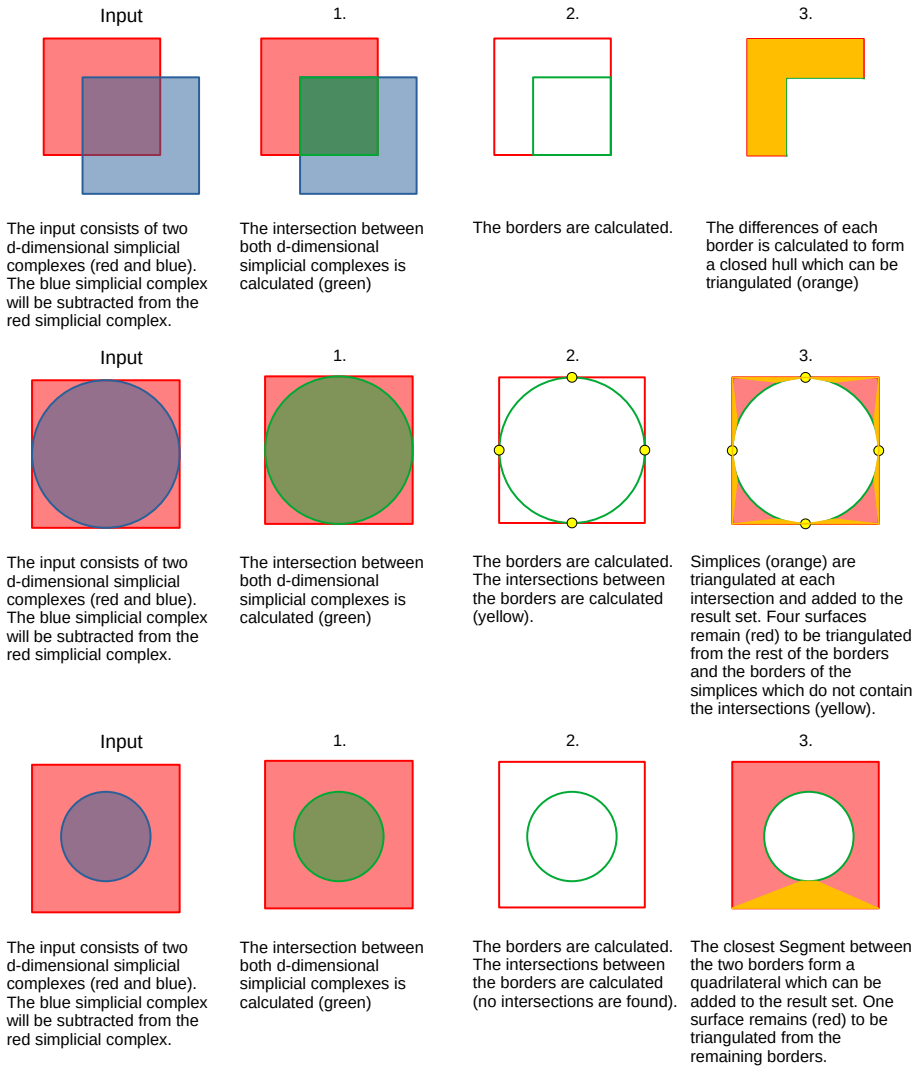


Figure 4.13: Examples: difference of two 2-dimensional simplicial complexes. The blue 2-dimensional simplicial complex is subtracted from the red 2-dimensional simplicial complex

Algorithm 7 includes a standard triangulation in Step 14 which is described in Section 4.8.1.2 and two special triangulations in Steps 12 and 13 which are described in Section 4.8.1.3. Figure 4.13 shows two examples of 2-dimensional simplicial complexes. The last step from the example at the top illustrates the standard triangulation called by Algorithm 7 in Step 14. The bottom example shows the steps needed to be taken if the border of two strictly intersecting 2-dimensional simplicial complexes touch each other. The last step within the example at the bottom includes the special triangulation from Algorithm 7 Step 12 which triangulates simplices around the touching points in order to create polygons which in turn are triangulated by the standard method and finally glued with the result set.

4.3.4 Equality

Two equality tests have been part of *DB3D* and *DB4GeO*. The *identity-equality* test is the usual way to test equality by comparing the addresses of each object within the memory. This type of equality test will not check if the object contents are equal, just like two *String* objects may be equal if they are equal in length and if they contain the same characters in the same order. *DB3D* and *DB4GeO* introduced a more complex equality test based on the inherent topology, which will be described in the next section. *DB4GeOGraphS* core framework introduces a *geometrical-equality* test, which is described after the section about *topological-equality*.

4.3.4.1 Topological-equality test

Operations concerning the *topological-equality* tests are defined in each simplex type (e.g. *Point3D*, *Segment3D*, *Triangle3D* and *Tetrahedron3D*), in the abstract classes *Component3DAbst* and *Net3DAbst* and the implementation of the interface *Spatial3DCollection* which is the class *Spatial3DCollectionImpl*.

Within each operation, an *identity-equality* test is done as the first step. After this test, an *instance-of* test makes sure that the objects are of the same instance if they are equal. The next steps differ within each *topological-equality* test operation definitions. In case of two simplices A and B, if all points of A are corner points of B and vice versa, the two simplices pass the *topological-equality* test. In case of two simplicial complexes A and B, the size of the R^**-Tree* of A needs to be equal to the size of the R^**-Tree* of B and each simplex of A needs to be found in B using the R^**-Tree* of B and the *topological-equality* test operation of simplices. In case of two *Net3D* objects A and B the size of the R^**-Tree* of A needs to be equal to the size of the R^**-Tree* of B and each simplicial complex of A needs to be found in B using the R^**-Tree* of B and the *topological-equality* test operation of the simplicial complexes.

In case of two *Spatial3DCollection* objects A and B the sizes of the R^**-Trees* of the contained *Net3D* objects of A needs to be equal to the corresponding sizes of the R^**-Trees* of the contained *Net3D* objects of B and each *Net3D* object of A needs to pass the *topological-equality* test for its corresponding *Net3D* object in B.

4.3.4.2 Geometrical-equality test

Operations concerning the *geometrical-equality* test are defined in the abstract class for all *Spatial3D* objects called *Spatial3DAbst* and in the class *Spatial3DCollectionImpl* which overrides the operation of the abstract class *Spatial3DAbst* in order to call the operations of each consisting *Net3D* object separately with respect to the dimension of the *Net3D* objects. Algorithm 8 presents all necessary step to test *geometrical-equality*.

Algorithm 8: *geometrical-equality* test

input : Spatial3D A and Spatial3D B
output : test result

1 if *identity-equality* test of A and B is passed → return true;
2 if *topological-equality* test of A and B is passed → return true;
3 if dimension(A) != dimension(A) → return false;
4 if MBB(A)!=MBB(B) → return false;
5 if A does not contain all points of B → return false;
6 if B does not contain all points of A → return false;
7 if A and B are simplices → return true;
8 A_b = border of A;
9 B_b = border of B;
10 if $A_b == NULL$ and $B_b == NULL$ → return true;
11 if $A_b == NULL$ xor $B_b == NULL$ → return false;
12 return result of *geometrical-equality* test of A_b and B_b

4.3.5 Border

Each *Spatial3D* implementation must implement a *getBorder()* operation, which returns a suitable border object. Starting from the *Element3D* aggregation level, each simplex class e.g. *Point3D*, *Segment3D*, *Triangle3D* and *Tetrahedron3D* implements this operation by building a $(d-1)$-dimensional simplicial complex from the border simplices. *Point3D* objects return null.

Each simplicial complex class e.g. *Point3DComponent*, *Segment3DComponent*, *Triangle3DComponent* and *Tetrahedron3DComponent* implements this operation by building a $(d-1)$-dimensional *Net3D* object from the $(d-1)$-dimensional border simplices. *Point3DComponent* objects return null. The $(d-1)$-dimensional border simplices are

collected by iterating over the set of d-dimensional simplices. If the d-dimensional simplex does not have a neighbour on a certain side, the corresponding $(d-1)$-dimensional simplex of the side is added to the resulting set. The resulting set is glued to a set of valid $(d-1)$-dimensional simplicial complexes, which are collected in a new *Net3D* object. If for $d > 1$ holes are inside the object, more than one $(d-1)$-dimensional simplicial complex will be part of the resulting *Net3D* object. If any borders of the holes and the outer border touch each other, the result may not be correct, since the gluing algorithm might glue parts of the outer border together with parts of the border of the hole which touches the outer border. But since d-dimensional simplicial complexes with holes that touch the outer border are defined as invalid d-dimensional simplicial complexes for $d > 1$ the behaviour of the implementation will return a reliable result containing one $(d-1)$-dimensional simplicial complex as the outer border and the other $(d-1)$-dimensional simplicial complexes as the borders of the inner holes.

Each class which is implementing the *Net3D* interface (e.g. *Point3DNet*, *Segment3DNet*, *Triangle3DNet* and *Tetrahedron3DNet*) implements this operation by collecting all $(d-1)$-dimensional simplicial border complexes of each contained d-dimensional simplicial complex into one $(d-1)$-dimensional simplicial border *Net3D*. *Point3DNet* objects return null. Intersections may happen since topological inconsistencies are allowed on the *Net3D* aggregation level since a *Net3D* object is the topological sum, the disjoint union of the topological spaces or the disjoint union of the contained simplicial complexes.

4.3.6 Overlay of simplicial complexes

The class *Spatial3DCollectionImpl* and the abstract class *Net3DAbst* implements a *getOverlay()* operation by Algorithm 9. This operation returns a *SpatialHashMap* object. The keys are the simplicial complexes of the *Spatial3DCollectionImpl* object or the *Net3DAbst* object, respectively. The values are the collections of the intersections of each key complex with the rest of the simplicial complexes. The intersections must not be equal (see Section 4.3.4) to the key complex. The return type is a *SpatialHashMap* which will be discussed in Section 4.7.2.

An example of three different triangle complexes is given in Figure 4.14. One of the major problems with this algorithm is the sensitivity on double precision. It may happen that some intersections are not equal even if they should be equal or the hash codes differ, which leads to bad results or even termination problems. This algorithm is special for these two aggregation levels and is not to be confused with the *buildOverlay(…)* operation described in Section 4.2.3 of the class *TONode3D/4DImpl* for *Property Graph* nodes.

Algorithm 9: overlay of simplicial complexes

input : Spatial3DCollectionImpl or Net3DAbst A
output : SpatialHashMap M

```
1  M = new SpatialHashMap;
2  for simplicial complex c in A do
3      collect all intersections of c with the simplicial complexes of the input which are
         not equal to c (choosing one of the equality tests of Section 4.3.4) into a set S;
4      C = M.get(c);
5      if C == NULL then
6          C = new Spatial3DCollectionImpl object;
7          add key-value pair (c, C) to M;
8      end
9      add S to C;
10     M' = C.getOverlay();
11     for key-value pair (c', C') in M' do
12         C = M.get(c');
13         if C == NULL then
14             add key-value pair (c', C') to M;
15         else
16             merge C and C'
17         end
18     end
19 end
20 return M;
```

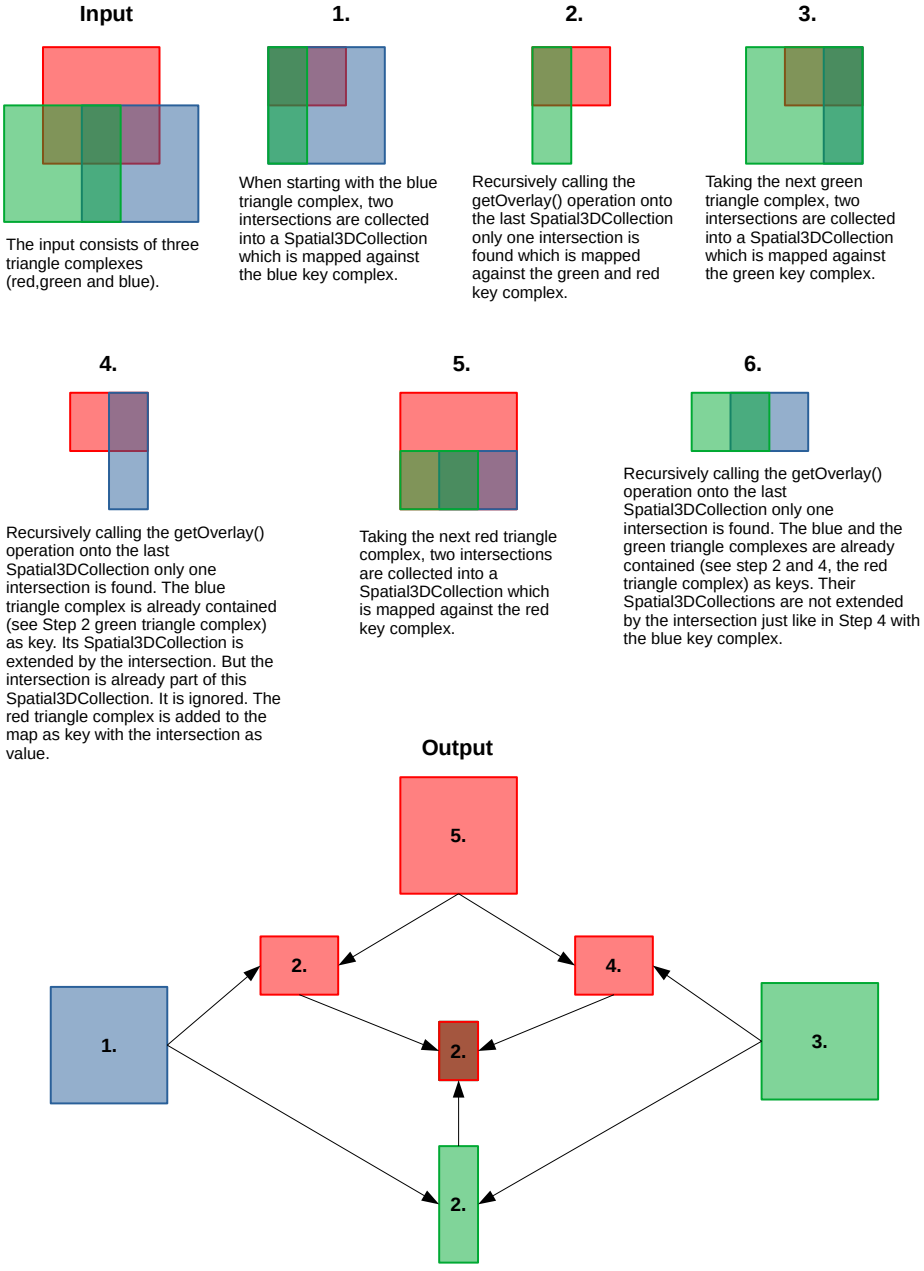


Figure 4.14: Overlay of three triangle complexes.

4.3.7 Glue simplicial complexes

The gluing of simplicial complexes which are neighbours tries to unify two simplicial complexes of equal dimension. The glue operation has been implemented in the classes *Point3DNet*, *Segment3DNet*, *Triangle3DNet* and *Tetrahedron3DNet*. In case of *Point3DNet*, all points of the *Point3DNet* object are collected into one complex. In case of the other dimensions, one simplex is picked from the *Net3D* object as entry for a new simplicial complex which will be extended step by step with all the other simplices of the *Net3D* object, if possible. If some simplices remain, the process starts again with a new entry for a new simplicial complex with the remaining simplices.

This algorithm is used in Algorithm 2 to glue manifolds into hulls or polygons, the *BREPs* or balls. It is also used within each intersection or difference operation of simplicial complexes to glue the results to valid spatial objects with their proper aggregation level.

If the border simplices do not fit together by not sharing a border edge or face, respectively, this algorithm fails to glue simplicial complexes which share some border parts. Therefore, in case of *Triangle3DNet* or *Tetrahedron3DNet*, the first step is always to make the borders of each simplicial complexes topologically consistent if they share some border parts. The algorithm is described in the next section.

4.3.8 Make topologically consistent borders of neighbouring simplicial complexes

A topologically consistent border helps to glue simplicial complexes together. If two simplicial complexes of the same dimension share parts of their borders and those parts are geometrically equal, the algorithm tries to make those parts topologically equal. The idea is to calculate each border intersection and segment-wise split the triangles which need to be split. Algorithm 10 summarizes the idea. The Algorithm is implemented within the class *Triangle3DNet*, only.

The intersection algorithm described in Section 4.3.2 for simplicial complexes will only return a reduced subdivision for the simplicial complex it was called from. The subdivision of the simplicial complex where the intersection operation was called from will be maintained. So if the intersection is called from a segment complex a with a segment complex b as argument, which is the case when intersecting the borders of two triangle complexes with borders, and the intersection is a curve (e.g. *Segment3D*, *Segment3DComponent*, *Segment3DNet* or a *Spatial3DCollection* containing a non-empty *Segment3DNet*) the subdivision of the caller a is maintained. This helps to split the triangle complex belonging to the argument b at the border triangles which intersect at their border with smaller segments of the intersection as its edge. Two different splitting may occur, which are shown in Figure 4.15 (red triangles). One simple example illustrates the steps of the Algorithm 10.

Algorithm 10: makes the borders of each simplicial complex topological consistent if they share some border parts

input : Triangle3DNet A
output : Triangle3DNet B

```
1  for triangle complex a in A do
2      b = a (deep copy);
3      if b has a border then
4          b_b = b.border();
5          for a' in A which intersects b do
6              if a'! = a and a' has a border then
7                  a'_b = a'.border();
8                  I_b = a'_b.intersection(b_b) (← I_b carries the sub-division of a'_b) ;
9                  if I_b is a CURVE then
10                     for i_b in I_b which is a segment complex do
11                         for i'_b in i_b (along segment complex) do
12                             for b' in b which intersects i'_b do
13                                 if b' contains i'_b and i'_b is not an edge of b' then
14                                     split b' into two triangles and exchange them
                                       with b' in b;
15                                 end
16                             end
17                         end
18                     end
19                 end
20             end
21         end
22     end
23     add b to B;
24 end
25 return B;
```

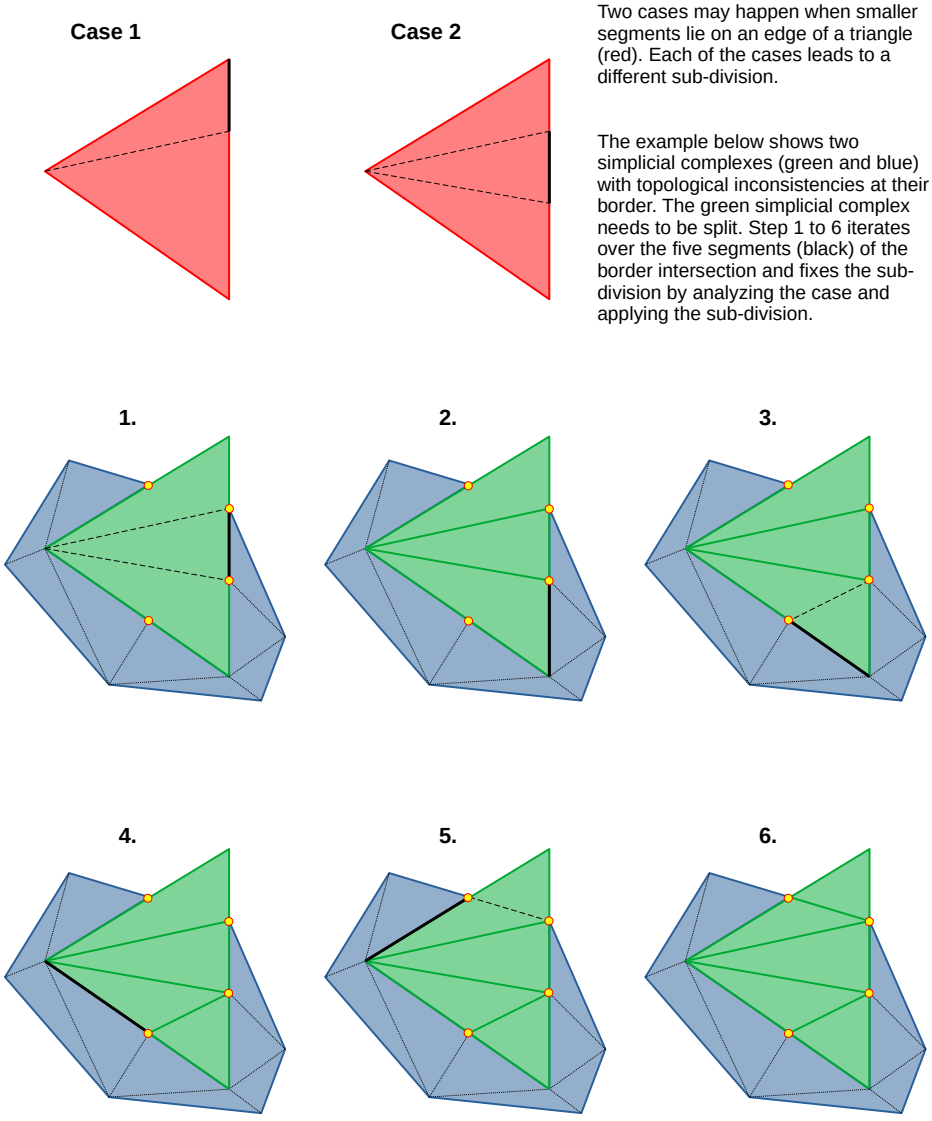


Figure 4.15: Cases for different subdivisions (top row) and making the border intersection of two triangle complexes (green and blue) topologically consistent by subdividing too large triangles.

4.3.9 Split simplicial complexes

Splitting simplicial complexes can be done in various ways. One of the easiest ways is to define a plane and cut through the simplicial complex. This was the method used of the *3D-to-2D*-service from *DB3D* to cut through spatial objects. Another way is by defining a threshold (e.g. all segments on-line l, all triangles with normal in direction range d etc.) which selects some simplices. The following implementations focused on splitting d-dimensional simplicial complexes by a $d-1$-dimensional simplicial complex. The splitting algorithm was necessary for Algorithm 2 of Section 4.8, see Example 4.9 (top) Step 2.

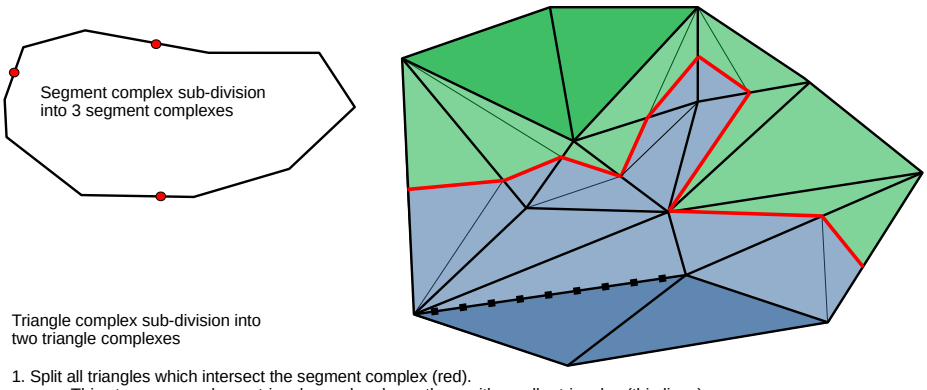
Algorithm 11: splits a *Triangle3DComponent* object into two *Triangle3DComponent* objects by a given *Segment3DComponent* object which lies on the *Triangle3DComponent* object.

input : Triangle3DComponent A, Segment3DComponent B
output : Triangle3DNet R

1 for each segment b in B and triangle a in A containing b split a along b;
2 **if** *A did not change* **then**
3 | return NULL;
4 **end**
5 collect all triangles which intersect any segment b in B in a triangle collection C_1;
6 collect all triangles which do not intersect any segment b in B in a triangle collection C_2;
7 create a Spatial3DCollection C' from C_2 and glue its Triangle3DComponent objects;
8 for each Triangle3DComponent in C' attach all triangles of C_1 whose glue spots (edges) are not in B and delete those triangles from C_1 (step by step);
9 **while** *C_1 is not empty* **do**
10 | take and remove one triangle of C_1 and insert it into a new Triangle3DComponent c_1;
11 | attach all triangles of C_1 to c_1 whose glue spots (edges) are not in B and delete those triangles from C_1 (step by step);
12 | insert c_1 to the Triangle3DNet R of C';
13 **end**
14 return Triangle3DNet R of C' ;

Algorithm 11 describes the steps to split a triangle complex. The algorithm to split one triangle by a segment, which is used at the start of Algorithm 11 relies on a triangulation which is described in Section 4.8.1.1. The split algorithm of one triangle by a segment will return two triangle complexes, one which lies on one side and another which lies on the other side of the splitting segment. The return type is a *Triangle3DNet* object, which consists of the two triangle complexes. This algorithm was also important for a special triangulation, see Section 4.8.1.3.

Figure 4.16 shows two examples. The first example illustrates a split of a segment complex by a point complex, and the second example a split of a triangle complex by a segment complex. The implementation of splitting a triangle complex is restricted to special kinds of segment complexes in order to use the previously mentioned splitting algorithm for triangles. The border of the segments needs to lie on the border of the triangle. Therefore, each segment needs to subdivide exactly one triangle or lies on a border segment of one or two triangles. One triangle may be subdivided by a number of segments in respect to the previously described constraints. Splitting tetrahedron complexes by a triangle complex has not been implemented, yet.



1. Split all triangles which intersect the segment complex (red).
   This step removes larger triangles and replaces them with smaller triangles (thin lines).
2. Collect all triangles which intersect the segment complex into a collection (light green and light blue).
3. Create complexes from the remaining triangles which do not intersect the segment complex (dark green and dark blue)
4. Iterate over the collection of Step 2 and insert all triangles to the dark blue complex
   whos glue spots (edges) with the dark blue complex are not part of the segment complex
   (glue spot example is given by doted edge) and remove them from the collection of Step 2
5. Repeat Step 4 until no triangle is able to be added to the dark blue complex
6. Repeat Step 4 and 5 for all other complexes (the dark green).
7. If the collection of Step 2 is not empty then create as many complexes from the remaining triangles as needed
   by adding only triangles to a new complex whos glue spots (edges) are not part of the segment complex.

Figure 4.16: Examples: Splitting a *Segment3DComponent* object (top left) by *Point3DComponent* object (red dots, top left) and splitting a *Triangle3DComponent* object by a *Segment3DComponent* object (top right).

4.3.10 Hypervolume

The hypervolume is calculated by the *getHyperVolume()* operation of each implementing class of the *Spatial3D* interface. Higher aggregation levels (e.g. *Spatial3DCollectionImpl*, *Net3DAbst*, *Component3DAbst*) calculate the hypervolume by the sum of the hypervolumes from the contained lower aggregation levels. Therefore, the hypervolume of a spatial collection is calculated by the sum of the hypervolumes of the four contained *Net3D* objects. The hypervolume of a *Net3D* object is given by the sum of the hypervolumes of the contained complexes, and the hypervolume of the complexes is calculated by the sum of all hypervolumes of the contained simplices. The implementation of the

getHyperVolume() operation is defined within the abstract classes of each aggregation level greater than the *Element3D* aggregation level. The hypervolumes of the simplices e.g. *Point3D*, *Segment3D*, *Triangle3D* and *Tetrahedron3D* are calculated by the typical formulas shown below.

Definition 4.3.1. *Hypervolume of a* Point3D *object:*

$$V = 0$$

Definition 4.3.2. *Hypervolume of a* Segment3D *object (using Euclidean distance):*

$$V = sqrt((x_{start} - x_{end}) + (y_{start} - y_{end}) + (z_{start} - z_{end}))$$

where x_{start}, y_{start} and z_{start} are the coordinates of the first point of the segment and x_{end}, y_{end} and z_{end} are the coordinates of the second point of the segment.

Definition 4.3.3. *Hypervolume of a* Triangle3D *object (Heron's formula):*

$$V = sqrt(s * (s - a) * (s - b) * (s - c))$$

where a,b and c are the hypervolumes of the border segments of the triangle and $s = (a + b + c)/2$ is the half perimeter.

Definition 4.3.4. *Hypervolume of a* Tetrahedron3D *object (via determinant):*

$$V = abs(det \begin{pmatrix} x_0 - x_1 & x_0 - x_2 & x_0 - x_3 \\ y_0 - y_1 & y_0 - y_2 & y_0 - y_3 \\ z_0 - z_1 & z_0 - z_2 & z_0 - z_3 \end{pmatrix} / 6.0)$$

where x_i,y_i and z_i are the coordinates for each if the four corner points p_i.

4.3.11 Contains-simplex test

The implementations of the *contains-simplex* test are defined in the abstract classes of each aggregation level. The dimension of a spatial object *A* needs to be greater than or equal to the dimension of simplex *B*, which it gets tested with. If this is not the case, "false" will be returned, because lower dimensions are not able to contain higher dimensions. If the aggregation level is greater than or equal to the *Net3D* aggregation level, the *contains-simplex* test operation is called for each contained simplicial complexes. If any of those calls does not fail, "true" will be returned, "false" otherwise. If the aggregation level is equal to the aggregation level of simplicial complexes, the intersection will be

calculated and the hypervolumes of the intersection and the simplex are compared. If the hypervolumes are equal in the sense of ϵ-equality (see Section 4.3.1 Factor 1), depending on the dimension d, "true" will be returned, "false" otherwise. If the aggregation level of a spatial object A is equal to the *Element3D* aggregation level, the result of a "contains-all-points" test of A with the points of simplex B is returned.

4.4 Temporal properties

As described in Section 2.3.4 on spatio-temporal polytopes, the temporal coordinate is separated from the spatial coordinates, since many spatial coordinates share the same temporal coordinate (complex spatial objects usually share a single time coordinate as a sort of snapshot). The 4D model of the geometry model from *DB4GeOGraphS* core framework was and is still closely linked to the temporal classes *TemporalInterval* and *TemporalIntervalSequence* as a (3D + 1D time) model. But the geometry classes and the temporal classes were separated into two different Java packages. The 4D model of the geometry model uses the classes from the temporal package. An overview of the classes are given in Figure 4.17. In addition, the temporal package was supplemented by the *TemporalStamp* and *TemporalStampSequence* classes for mapping a point in time or a series of points in time, respectively. It is now possible to link the 3D model with one of those time markers at cell level. For this reason, the old 3D model can be viewed in conjunction with the new node model as a (3D + 0D time) model described in Section 2.3.4 on spatial simplices.

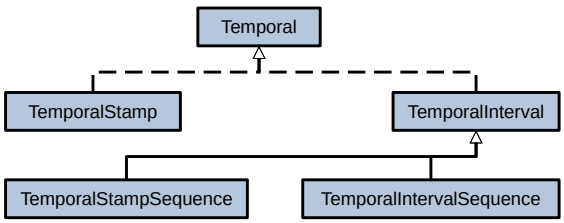


Figure 4.17: Temporal property model

4.5 Spatio-temporal properties

The spatio-temporal property implementations of the *DB4GeOGraphS* core framework can be found in the package called *spatial4d*. Figure 4.18 shows the class diagram of the refactored, revised and redesigned spatio-temporal model of DB4GeO. The basic subdivision into different class hierarchies concerning dimension and aggregation level did not change.

The major differences between the old model and the new model lies in retrieving border objects and adding a new helper interface called *Spatial4DCollection* to manage different dimensional types in one basic object. The intersection operations, difference operations, the new equality tests and the set of new contain tests are prepared to be implemented as seen in the overview of the operations contained by the interface *Spatial4D* which is given in Figure 4.19.

As seen in Figure 4.18 different kinds of spatio-temporal aggregation types exist e.g. *Spatial4DCollection*, *Net4D*, *Component4D*, *Sequence4D* and *Element4D* where each higher aggregation level consists of objects of the lower aggregation level. Therefore, the implementation of the interface *Spatial4DCollection*, which is called *Spatial4DCollectionImpl* manages four different *Net4D*s (*Point4DNet*, *Segment4DNet*,*Triangle4DNet* and *Tetrahedron4DNet*). This is the highest aggregation level.

The abstract realization of *Net4D* called *Net4DAbst* collects all *Component4D* objects with the help of a spatio-temporal access method, which will be discussed in Section 4.7.4. The abstract realization of *Component4D* is called *Component4DAbst*. The *Component4DAbst* abstract class realizes most of the basic operations concerning polytope complexes with the help of the *Sequence4D* interface. The *Sequence4D* objects are collected with the help of a spatio-temporal access method also. The abstract implementation of the *Sequence4D* interface called *Sequence4DAbst* provides a list of *Sequence4D* objects to link neighbouring *Sequence4D* objects in the sense of Section 2.3.4. It manages the polytopes (*Element4Ds*) by a spatio-temporal access method also. An *Element4D* represents a moving or morphing simplex, as described by the Polthier and Rumpf model [53]. The abstract realization of *Element4D* is called *Element4DAbst*. A detailed description can be read in [60].

4.5.1 Border

The spatio-temporal model handles border operations differently than the spatial model. Where the spatial model provides only one operation, the spatio-temporal model provides three operations to retrieve all parts of a border of a *Spatial4D* object.

The first operation is called *getStart()* and retrieves the *Spatial3D* object of the first time step. If the *Spatial4D* object is a *Net4D* or a *Spatial4DCollection*, every *Component3D* object of the first time steps of each *Component4D* is collected into a *Net3D* object and returned. In case of *Component4D*, *Sequence4D* and *Element4D*, the spatial representation of the first time step is returned. The second operation is called *getEnd()* and returns the last time step(s) just as the *getStart()* operation. The third operation, called *getBorder()*, returns the spatio-temporal border of the *Spatial4D*. This operation returns null if the *Spatial4D* object is a *Sample4D*. This operation will return an array of *Point4DSequences* or *Point4DElements* if the *Spatial4D* object is of type *Curve4D*. This operation will return an array of *Segment4DSequences* or *Segment4DElements* if the *Spatial4D* object is of type *Surface4D*. And finally, this operation will return an array of *Triangle4DSequences*

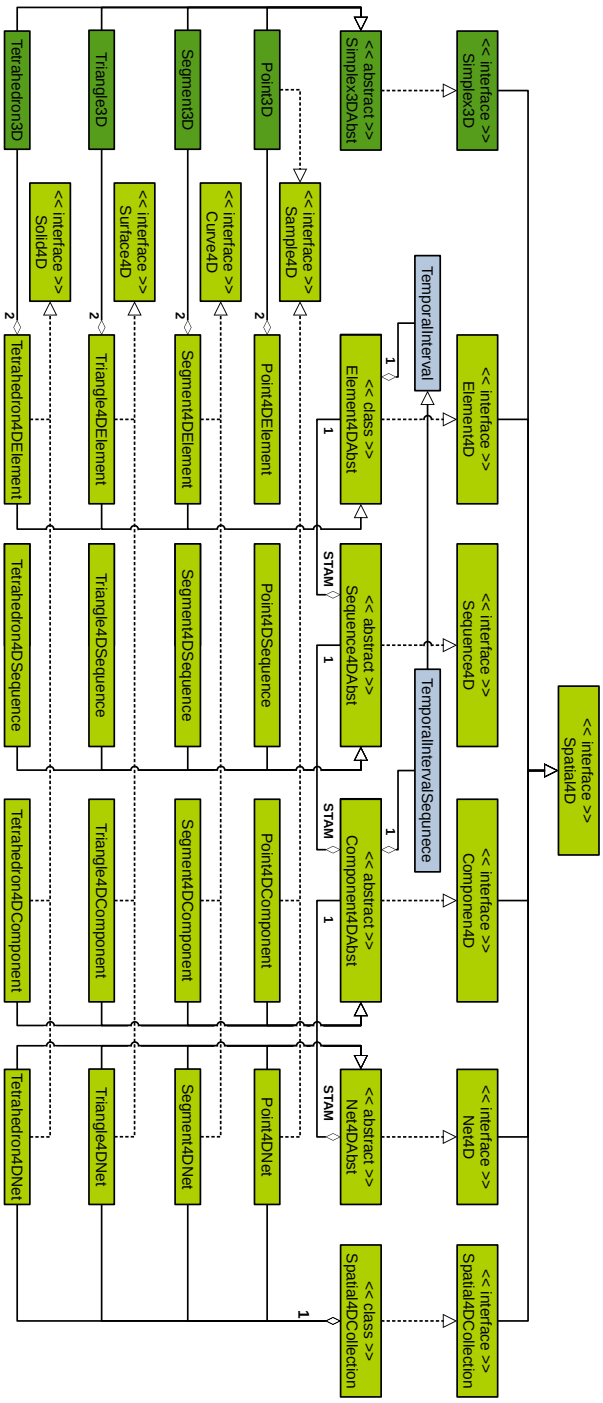


Figure 4.18: Spatio-temporal property model

Figure 4.19: Interface *Spatial4D*

or *Triangle4DElements* (prisms) if the *Spatial4D* object is of type *Solid4D*. Figure 4.20 demonstrates the spatio-temporal objects and their border.

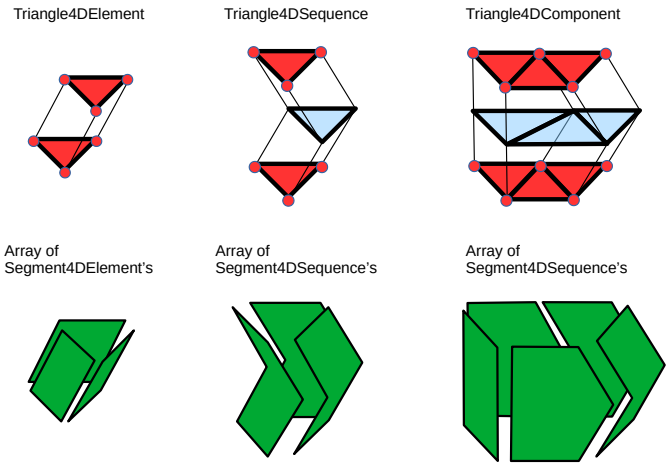


Figure 4.20: Three *Surface4D* aggregation levels and their border objects (red and green)

4.5.2 Interpolation

Interpolation of *DB4GeOGraphS* core framework did not change comparing to its predecessor, *DB4GeO*. The *Interpolation4D* interface is still part of the *spatial4d.api* package. The standard implementation which implements a linear interpolation function has been customized to work with the new spatio-temporal predicate, which is described in 4.7.1. *Interpolation4D* objects are necessary for every spatio-temporal access methods in order to be able to retrieve any spatial representation of the spatio-temporal object at any time step. The implementation of a spatio-temporal access method is discussed in Section 4.7.4. Since spatio-temporal objects of higher aggregation level then *Element4D* manage their contained objects with the help of a spatio-temporal access method and the *Interpolation4D* object is a fixed part of the spatio-temporal access method, the *Interpolation4D* object is essentially connected with the spatio-temporal shape of the spatio-temporal object.

4.6 Thematic properties

The thematic model of the predecessor *DB3D* was very closely linked to the structure of the geometry and object model. Only the *Point3D* objects, the *Element3D* objects and *Net3D* objects could be linked with thematic attributes managed by the *Object3D* and *Object4D* classes. Thematic attributes could also be assigned to a *Space3D* or *Space4D* objects, which are special classes to describe the mathematical space of the contained

objects. Each of these classes took on the management of its thematic attributes in special thematic classes.

Due to the restructuring to a dynamic *Property Graph*, these special implementations of the thematic model also had to be adapted. As shown in Section 4.2.1 Figure 4.1, the special classes are now part of the *TONodeImpl* class.

The new thematic model is closely linked to the *Property Graph* in order to offer a general thematic model. Each node manages its own record instances in a map, where the key is a *RecordDefinition* object and the value is a *Record* object. A *Record* object is created by using a *RecordDefinition* object. A *RecordDefinition* summarize data types similar to a table header in a record class. The record class is compiled at runtime, which is then available to the system. In that way, each node belongs to a number of attribute groups or record classes.

Few basic attribute types are available e.g. BOOLEAN, INT, LONG, FLOAT, DOUBLE, BYTE, STRING, VECTOR3D, VECTORND, POINT3D, POINTND, TONODEID and DATE. If more attribute types are needed, the *RecordClassBuilder* class needs to be customized as needed and the type classes have to be made available.

This model adapts dynamically to the new *Property Graph*, which promises more intuitive handling. Figure 4.21 illustrate how the thematic model may integrate into a dynamic *Property Graph* where each node belongs to a set of *RecordDefinitions* which provide the table headers for the records of each node.

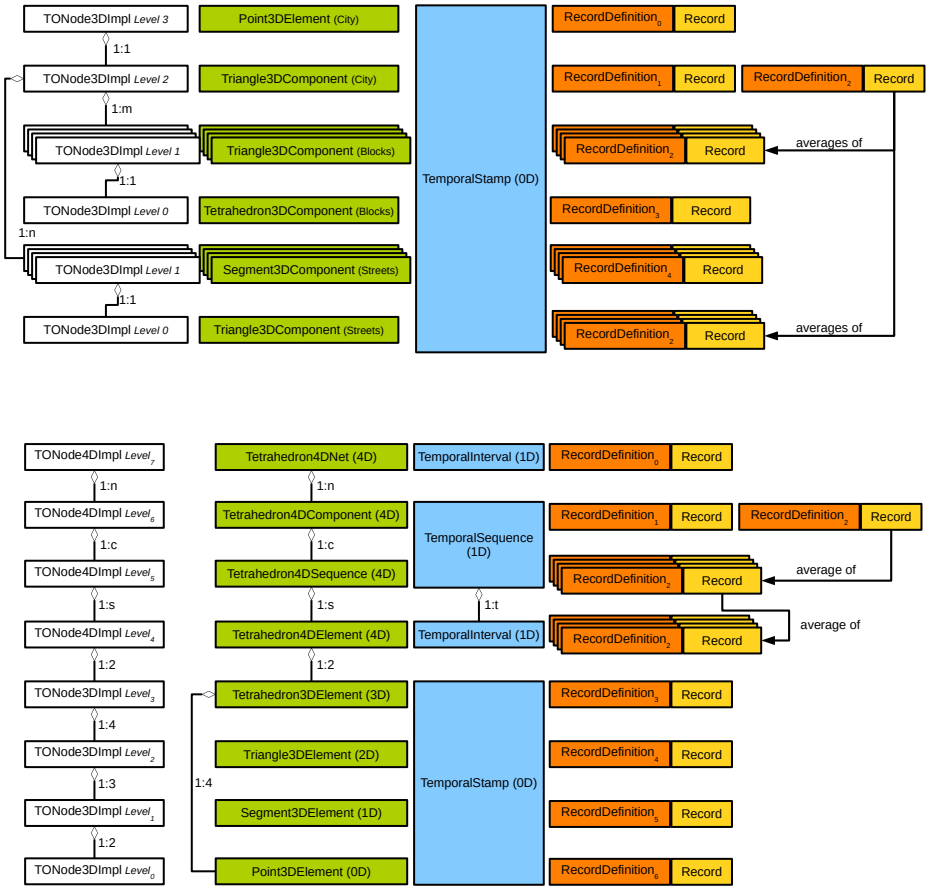


Figure 4.21: Top: LOD graph of a city with blocks, buildings, and streets represented by different dimensions and aggregation levels suitable for the level of detail.
Bottom: Typical aggregation graph of a *Tetrahedron4DNet* object. Incidence nodes for the pre- and post-objects of a tetrahedron polytope are included in this example. Each node carries thematic attributes.

4.7 Access methods

Access methods play one of the most important roles in GIS-Systems of any scale, as mentioned in Section 4.7. The following sections describe all the important implementations and the underlying mechanisms of *DB4GeOGraphS* core framework. The *indices* package is divided into five sub-packages e.g. the *api*, *spatial3d*, *spatial4d*, *temporal* and *graph*. To understand more complex access methods, it is necessary to take a closer look at spatial predicates, temporal predicates and spatio-temporal predicates at first. Then spatial implementations *Spatial-Hash-Map*, *Oct-Tree* and *R*-Tree*, the temporal implementation *Segment-Tree*, the spatio-temporal implementation as a combination of both are discussed next. They were taken from the predecessor *DB4GeO*, revised and integrated into the new framework. The following sections introduce new multidimensional access methods based on fixed and scalable space filling curves indices. And the final section on access methods gives a brief introduction on a special variant of a discrete *R*-Tree* used to manage *TONodes* based on the quality of their relations and position within the *Property Graph Model* as a first idea for a topological access method which is able to answer topological queries like: "Retrieve all spatial objects which are part of exactly two other spatial objects."

4.7.1 Predicates

Predicates help to retrieve parts of complex spatial, temporal or spatio-temporal objects. The spatial predicate has been revised, the temporal predicate has been customized to fit into the framework and the spatio-temporal has been created as a new predicate to be able to query complex spatio-temporal objects by one predicate similar to the spatial model. The predicates provide enumerations or a *PredicateCode* for the kind of query. The spatial and spatio-temporal predicate also provides enumerations for dimension and aggregation level definitions. With the help of those enumerations or *PredicateCodes* queries can be defined to retrieve whatever is possible.

In case of a temporal predicate, a temporal interval needs to be defined together with a *PredicateCode*. The *PredicateCode* can be of value: any, disjoint, before, after, overlap, adjacent, adjacent at start, adjacent at end, strict overlap, identical, symmetric overlap, contains at start, contains at end, asymmetric overlap, contains, contains at start, contains at end, inside, inside at start and inside at end. In case of a spatial predicate, a minimal bounding box needs to be defined together with an enumeration for the type of spatial query. The enumeration can be of value: any, equals, disjoint, touches, contains, covers, intersects, inside, covered by, crosses, overlaps and nearest. The spatial predicate also provides an enumeration for the dimension type (sample, curve, surface and solid) and the aggregation type (element, component and net). In case of a spatio-temporal predicate, a temporal predicate need to be provided and a minimal bounding box together with an enumeration for the type of spatial query. This enumeration has the same definitions as the spatial predicate. Predicates also provide retrieving operations which handle the

Figure 4.22: Class *Spatial4DPredicate*

query. The spatio-temporal predicate also provides an enumeration for the dimension type (sample, curve, surface and solid) and the aggregation type (element, sequence, component and net).

The spatial and spatio-temporal predicates also provide helper operations which are able to classify the aggregation level and dimension of any spatial or spatio-temporal object. Figure 4.23 gives an overview of the implemented predicates. Figure 4.22 shows the UML diagram of the new predicate. The major difference to the original *Spatial3DPredicate* is that the predicate contains a *TemporalPredicate* object.

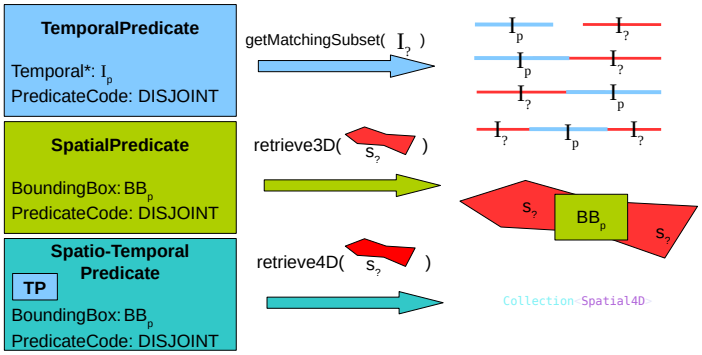


Figure 4.23: The temporal predicate (top row) retrieves the red temporal intervals (right). The spatial predicate (second row) retrieves the red *Traingle3DNet* object (right) from the red triangle complex (middle). The spatio-temporal predicate (last row) retrieves a collection of *Spatial4D* objects (right) from the red spatio-temporal triangle complex (middle).

4.7.2 Spatial access methods

Every spatial access methods (SAM) needs to implement the *SAM* interface. Figure 4.24 shows the UML diagram of the *SAM* interface (left). The *OctreeSAM* class implements the *Oct-Tree*. The implementation follows the theory described in Section 2.3.7. The *RStarSAM* class implements the *R*-Tree*. The implementation follows the theory described in Section 2.3.7. A read/write lock has been added to the *RStarSAM* in order to work in multithreaded environments. The *R*-Tree* is used in various situation. It is the main data structure for any aggregation levels higher than *Element3D*.

As described in Section 2.3.7 *Spatial-Hash-Maps* are maps whose keys are calculated by a hash function which dependends on the shape of a spatial object. The implementing class is called *SpatialHashMap* and is extended from the *java.util.Map* as part of Javas Collection API. As seen in Figure 4.12 each *Spatial3D* must implement the *isTopologyEquivalentHC()* operation. The implementation in *DB4GeOGraphS* core framework limits the functionalities to the functionalities of the *java.util.Map* interface. It does not implement the operations of the *SAM* interface. However, the typical hash code operations of any spatial object are given through Algorithms 12 and 13.

Algorithm 12: $p.isTopologyEquivalentHC(f)$ operation for calculating the hash code h of *Point3D* objects. f controls the cube size. With ">>>" as unsigned right shift operator (32 bits to the right).

input : Point3D p, Integer f
output : Integer h

$x = \text{round}(p_x * f)/f$;
$y = \text{round}(p_y * f)/f$;
$z = \text{round}(p_z * f)/f$;
$a = 31$;
$h = 1$;
t =Double.doubleToLongBits(x);
$h = a * h + (int)(t \text{ XOR } (t >>> 32))$;
t =Double.doubleToLongBits(y);
$h = a * h + (int)(t \text{ XOR } (t >>> 32))$;
t =Double.doubleToLongBits(z);
$h = a * h + (int)(t \text{ XOR } (t >>> 32))$;
return h;

Algorithm 13: $s.isTopologyEquivalentHC(f)$ for calculating the hash code h of any *Spatial3D*. f controls the cube size.

input : Spatial3D s, Integer f
output : Integer h

$a = 31$;
$h = 1$;
P = sorted points array of s;
for $p \in P$ **do**
  $h = a * h + p.isTopologyEquivalentHC(f)$;
end
return h;

4.7.3 Temporal access methods

DB4GeOGraphS core framework *indizes.api* contains a *TAM* interface for temporal access methods (TAM). A UML diagram is shown in Figure 4.24. The temporal access method implemented within *DB4GeOGraphS* core framework is based on a *Segment-Tree*. *Segment-Trees* are binary trees which are able to find overlapping intervals relatively fast. Their implementing class is called *SegmentTreeTAM*. As seen in Figure 4.24 the *TAM* interface takes *Spatial4D* objects as operation arguments and not *Temporal* objects. Temporal access methods within *DB4GeO* were tightly connected to the spatio-temporal model. It is a part of the spatio-temporal access method, which will be discussed in the next section. A detailed description can be read in [60].

4.7.4 Spatio-temporal access method

DB4GeOGraphS core framework *indizes.api* contains a *STAM* interface for spatio-temporal access methods (STAM). A UML diagram is shown in Figure 4.24. The spatio-temporal access method implemented within *DB4GeOGraphS* core framework is based on a *Segment-Tree* and a *R*-Tree* which are explained before. Their implementing class is called *RStarSegmentTreeSTAMImpl*. A detailed description can be read in [60].

4.7.5 Space-Filling Curves

This section is published within *The Computer Journal* [8] beforehand:

The motivation of this section is to provide an implementation which allows the construction of a dynamic tree-shaped index structure for geo-data stored in geo-databases by using space-filling curves. A related dynamic tree structure should be imposed directly on the nodes in the *Property Graph Model*, and should not be restricted to binary graphs, i.e. any number of children should be allowed. So, the approach will be a combination of two methods: space filling curves and tree structures. This allows to combine the advantages of both: the neighbourhood-preserving property and logarithmic-time access of data. This means the consideration of p-adic space-filling curves in which p can vary, to have another parameter which can be optimized in future work in order to obtain the best possible index structure. An advantage of taking p different from $p = 2$ can arise in the case that data is clustered into homogeneous groups; then forcing the branching number to be 2 imposes a hierarchy onto the clusters which may be arbitrary and can lead to unnecessarily high depth in larger clusters.

A space-filling curve with properties of the Hilbert curve will be used, and the Hilbert values should be implicitly contained in the tree structure in the form of paths from the root node. As we have in mind geometric data, we will consider only the continuous case. Z-order curves or other space-filling curves which do not have the nice neighbourhood-preserving properties of Hilbert curves are not the focus of this section. The data structure discussed in this section is essentially a kind of quad-tree in which the leaf nodes are ordered in a neighbourhood-preserving manner. A dimension reduction is performed in a data-driven manner by deleting all nodes which have precisely one child node. The main idea of this section is to supplement the existing quad-tree methods by such which can deal more efficiently with issues containing neighbourhood relationships, e.g. in shadow casting.

A tree has been chosen as the basic data structure for the implementation of p-adic Hilbert index on point clouds from scratch. The Hilbert value can be obtained by interpreting the movement to the first sub-node of a node as false/zero, the movement to the second sub-node as true/one, the movement to the third sub-node as two and so on (in case $p > 2$). A list of numbers (or true and false values in the 2-adic case) is built up step

Figure 4.24: Interfaces *SAM*,*TAM* and *STAM*

by step up the tree. This list of numbers serves us as the Hilbert value of the final leaf node.

The tree at the root-node covers a maximal n-dimensional hyper-cuboid. Starting from root node with n as the dimension, n steps are needed to fulfil one complete curve iteration process. In order to insert, delete or retrieve data points, the tree must be queried from root to leaf node. One step along a branch splits the hyper-cuboid of a node

into p parts equal in size by cutting $p-1$ times orthogonally to a splitting coordinate d through the node's hyper-cuboid. For each step up the tree, it has to be decided which splitting coordinate d should be taken and how to map the p hyper-cuboid parts to the p sub-nodes.

There are two cases how to map p hyper-cuboid parts to p sub-nodes. Hence, an identifier $0 \leq a < p$ for each sub-hyper-cuboid which maps the p parts of the hyper-cuboid in positive direction of the coordinate axis d and another identifier $0 \leq b < p$ for each sub-node, where lower identifiers indicate a shorter distance to the starting leaf node of the curve. If $b = a$ the curve is said to *move* along the positive direction of coordinate d and if $b = p - 1 - a$ the curve is said to *move* in opposite direction of the coordinate axis d.

How to split a hyper-cuboid within one curve iteration process depends on the path taken within the previous curve iteration process only. To connect the ends of each sub-hyper-cuboids with the start of the next neighbouring sub-hyper-cuboid correctly, it is necessary to know if the start or end should lie in the lowest or highest part of the sub-hyper-cuboid regarding each coordinate. If the path taken within the previous curve iteration process indicates that the curve *moved* along the positive direction of a coordinate, d then the sub-hyper-cuboid curve should also start in the lower part and should end in the higher part respectively that coordinate d.

There are two main cases where a curve starts and where it ends within one hyper-cuboid. If p is odd, the coordinates of the end point will always lie diagonally from the start point (cf. Figure 4.25, left). If p is even, the coordinates of the end point will always be the same as the start point except in exactly one coordinate (cf. Figure 4.25, right). This coordinate needs to be the first splitting coordinate of an iteration process. Therefore, it is important which splitting coordinate comes first in order to rotate the sub-hyper-cuboid correctly into the direction of the curve. The order of the following coordinates does not matter when building a space filling curve.

The *static* variant of the p-Adic Gray-Hilbert curve index deals with a constant number of curve iterations k. Therefore, every leaf node of the resulting tree is on the same level of the tree. The tree will contain $\ell = k \cdot n$ levels and p^ℓ leaf nodes. The hyper-cuboids of the leaf nodes are equal in size, and the tree will have under- and overfilled leaf nodes. So, if the data are clustered in a certain region, then there will be leaf nodes containing many points, while other leaf nodes contain fewer or even none of the points (see Figure 5.12 or Figure 5.13, bottom left).

The developed *scaled* variant of the p-adic Gray-Hilbert index deals with a maximal number of data points contained within each leaf node, called the *bucket capacity* s. It dynamically splits a leaf node if the maximal number is exceeded, or collapses a leaf node if no data points are left within the leaf node. This mechanism takes place on any level of the tree. Therefore, each data point is not indexed on a certain curve iteration level. The curve shown in figure 4.25 with iteration number $k = 4/3$ (notice that the fractional iteration number k used here is the tree-level number divided by the dimension number)

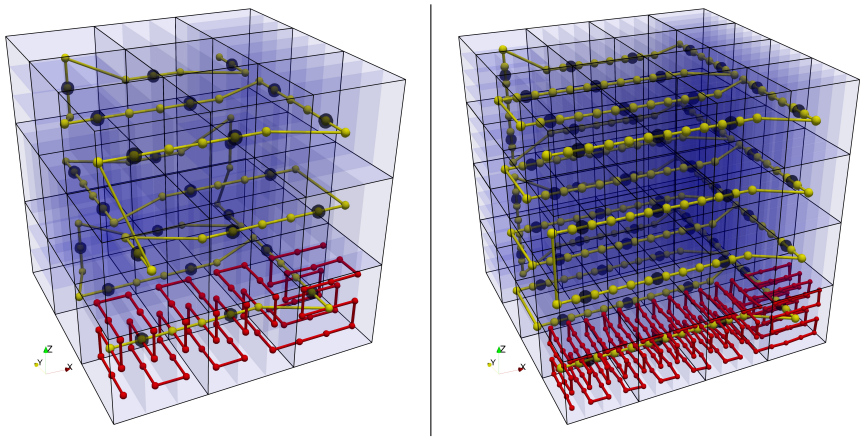


Figure 4.25: Uniformly distributed points in dimension 3 indexed by p-Adic Gray-Hilbert curves with $p = 3$ (left), $p = 4$ (right), $k = 1$ (black points and cubes), $k = 4/3$ (yellow points and curves with blue transparent sub-hyper-cuboids) and first three sub-cuboids of curves with $k = 2$ (red points and curves).

belongs to a special case where all leaf nodes of the scaled curve are on level 4 of the tree while the points are in dimension 3, and is given in the figure by yellow points and curves with blue transparent sub-hyper-cuboids. All levels and therefore each sublevel or sub-hyper-cuboid, respectively any node within a curve iteration process, can be a leaf node. In Figure 4.26 all points are normally distributed. The scaled version (bottom of each) adapts to the distribution, where else the static version (top of each) does not. In that way, the dynamic variant is data driven and provides all benefits of data driven structures in contrast to its static alternative. We call this a *scaled p-adic Gray-Hilbert tree*.

In the following pseudocode, we define for a hyper-cuboid c the quantities $c.\text{pMin}[d]$ and $c.\text{pMax}[d]$ as the values of the d-th coordinate of the lower left corner and the upper right corner of c, respectively.

Algorithm 14 FINDNODE(T,C,D,x) describes a way of searching a leaf node containing some query point. The basic idea of Algorithm 14 is to remember the path of the previous iteration process and to add those changes to the new iteration process. Lines 1 to 4 deal with starting conditions. The list of coordinate identifiers D is set up by any ordering. Line 5 is the loop within which one step from some hyper-cuboid to its sub-hyper-cuboid is taken. Therefore, the loop has to be gone through n times to fulfil one complete curve iteration process. Line 8 determines the sub-hyper-cuboid identifier a. Line 10 and 11 adjust the next sub-hyper-cuboid extrema. Lines 12 to 16 determine the next sub-node identifier b. If $b = a$ the order of the sub-nodes is along the coordinate axis d in positive direction, which means that the curve points into the positive direction of the coordinate axis d. If $b = p - 1 - a$ the order of the sub-nodes is in the opposite direction of the

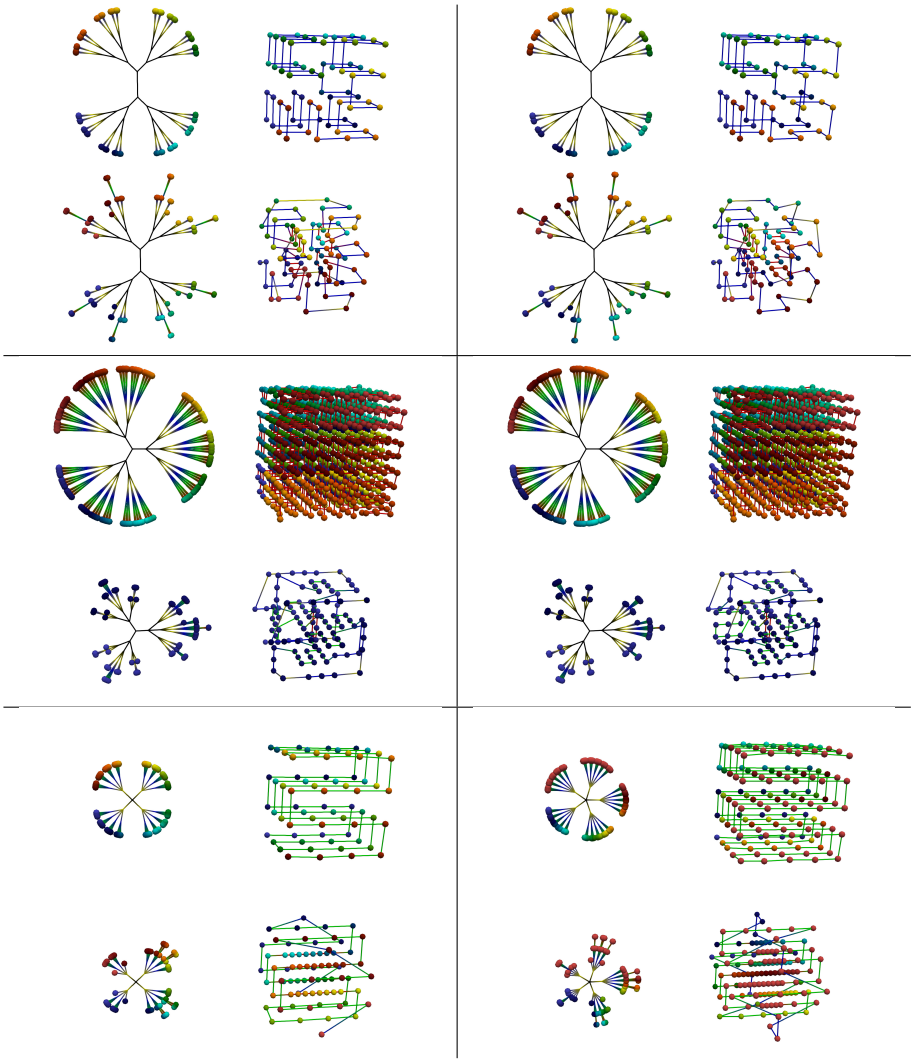


Figure 4.26: Normally distributed points in dimension 3 indexed by p-adic Gray-Hilbert trees/curves. From top left to bottom right: bubble curve with $p = 2$, $k = 2$; ring curve with $p = 2$, $k = 2$; bubble curve with $p = 3$, $k = 2$; ring curve with $p = 3$, $k = 2$; bubble curve with $p = 4$, $k = 1$; bubble curve with $p = 5$, $k = 1$. The top left part of each figure shows the tree for fixed k iterations. The bottom left part of each show the scaled tree with dynamic k iterations. The right parts show their geometric curves. The colours of the spheres indicate the IDs of the leaf nodes, which are ordered from the start to the end of the curves. The colours of the trees and the colours of the curves indicate the level of the tree.

coordinate axis d, which means that the curve points into the opposite direction of the coordinate axis d.

To decide in which direction we need to point, a helper array H has been used which saves the pointing directions of the last iteration process ($H(d) = false$ implies opposite direction of coordinate axis d, $H(d) = true$ implies in direction of coordinate axis d). If p is odd, a boolean g is also needed to integrate the information if a flip of the direction has been done within the last step. Line 17 moves the node reference to the next sub-node. If the iteration process is not finished, Lines 19 to 27 follow the rules of the Gray code to support the next step up the tree. If the sub-node identifier b is odd, then we must invert the helper variables for the next step. If the iteration process is finished, we need to run the following three routines in order to calculate the preconditions for the new iteration process: Algorithm 15 FINDNEWSTART() in Line 29, Algorithm 16 SETH() in Line 30 and Algorithm PERMUTATE() in Line 31.

Algorithm 15 FINDNEWSTART() retrieves the new start coordinate in order to rotate the new sub-curve for the next iteration process along the direction the curve is pointing within the new sub-hyper-cuboid. Most of the sub-curve directions are along or in the opposite direction of the last coordinate axis within D. Starting at the end of D and moving down the tree we check if the sub-hyper-cuboid identifier a or the sub-node identifier b is not equal to zero or $p-1$ (see Algorithm 15 Line 3) which indicates that the sub-hyper-cuboid is one of the inner sub-hyper-cuboids of the hyper-cuboid respectively to $d \in D$. If this is the case, we can return the last coordinate identifier of D to be the new start coordinate for the next iteration process. These are the cases where the sub-curve directions are along or in the opposite direction of the last coordinate axis within D. If the sub-hyper-cuboid is one of the outer sub-hyper-cuboids, we do a step down the tree. We can return the incremented coordinate identifier d either if the sub-node identifier b is even and if the last sub-node identifier was equal to $p-1$ or if the sub-node identifier b is odd and if the last sub-node identifier was equal to 0 (see Algorithm 15 Line 7). This coordinate identifier indicates a turning of the curve into a new direction along the coordinate axis d. We repeat this process until we checked every $d \in D$. If no coordinate identifier was found, then the last coordinate identifier of D is returned. Filtering the new start coordinate could also be done within the iteration process, but the pseudocode would be more complicated to read.

Algorithm 16 SETH() takes care of the curve direction in order to move from a geometrically higher sub-hyper-cuboid to a lower sub-hyper-cuboid or vice versa. If p is odd and the last sub-hyper-cuboid identifier a is also odd, we need to flip the whole new sub-hyper-cuboid by inverting g in order to point into the opposite direction as before, and we need to flip the direction of the new start coordinate by inverting H at the new start coordinate in order to stay in the same direction as before (see Line 13 to 15). If p is even, we must fix the helper array H at each coordinate instead of the new start coordinate. If the sub-hyper-cuboid lies on a part of the curve which points in the direction of a coordinate axis ($H(d) = false$) and the sub-hyper-cuboid identifier a is

Algorithm 14: FINDNODE(T,C,D,x) to find a leaf node containing some query point within its p-adic sub-hyper-cuboid.

input : A tree T covering an n-dimensional p-adic hyper-cuboid C with a list of coordinate identifiers D and some point $x \in C$.
output : The leaf node where x lies within the leaf node's sub-hyper-cuboid c.

```
1  node = root(T);
2  H = list of booleans of length n initialized with false values;
3  g = "false", c = C;
4  d = D.start;
5  while node is not a leaf do
6      p_min = c.pMin[d];
7      p_max = c.pMax[d];
8      a = (int)(p * (x[d] − p_min)/(p_max − p_min));
9      if a == p then a = a − 1;
10     c.pMin[d] = p_min + a * (p_max − p_min)/p;
11     c.pMax[d] = p_min + (a + 1) * (p_max − p_min)/p;
12     if p is even then
13         if H[d] then b = p − 1 − a; else b = a;
14     else
15         if g XOR H[d] then b = p − 1 − a; else b = a;
16     end
17     node = node.getSon(b);
18     if iteration process is not finished e.g. D.hasNext() then
19         if b is odd then
20             if p is even then
21                 H[D.next(d)] = !H[D.next(d)];
22             else
23                 g = !g;
24                 H[d] = !H[d];
25             end
26         end
27         d = D.next(d);
28     else
29         d = FINDNEWSTART();
30         g = SETH(g);
31         D = PERMUTATE(d);
32     end
33 end
34 return node;
```

Algorithm 15: FINDNEWSTART() to find the new start coordinate for the next iteration process.

output: The new starting coordinate

```
1  d = D.end;
2  while d != D.start do
3      if not moved to the first or to the last identifier b within the coordinate step taken
         at d then
4          return D.end;
5      end
6      d = D.before(d);
7      if (moved to an even identifier b within the coordinate step taken at d AND moved
         to the last identifier b within the coordinate step taken at D.next(d)) OR (moved to
         an odd identifier b within the coordinate step taken at d AND moved to the first
         identifier b within the coordinate step taken at D.next(d)) then
8          return d;
9      end
10 end
11 return D.end;
```

odd, then we must flip the direction by inverting $H(d)$ in order to point into the opposite direction (see Line 5 and 9).

Algorithm PERMUTATE() pulls up the coordinate d to the front of the list for the next curve iteration process using one of the following permutations:

$$\begin{pmatrix} n-1 & n-2 & \dots & d & d-1 & \dots & 1 & 0 \\ d & n-1 & \dots & d+1 & d-1 & \dots & 1 & 0 \end{pmatrix}$$

To this does 'pull' in the code refer. We called this the *bubble* permutation. We also used a ring of coordinate identifiers and used the following permutation:

$$\begin{pmatrix} n-1 & n-2 & \dots & n-(d+1) & \dots & 0 \\ d & d-1 & \dots & 0 & \dots & d+1 \end{pmatrix}$$

To this does 'jump' refer in the code. We called this the *ring* permutation.

It is also possible to use this algorithm to find all leaf nodes which intersect a query hyper-cuboid. The returned nodes will contain all data points from the union of their sub-hyper-cuboids which intersect the query hyper-cuboid. It is not needed to query down to each leaf if a sub-hyper-cuboid in between is covered by the query hyper-cuboid. All leaves of that sub-hyper-cuboid will be part of the result set, and it would be enough to return the whole set of leaf nodes.

Algorithm 16: SETH() sets the directions.

```
1  if p is even then
2      d = old start coordinate;
3      while d! = old end coordinate do
4          if d! = new start coordinate then
5              if (!H[d] AND moved to odd identifier a within the coordinate step taken
                 at d) OR (H[d] AND moved to even identifier a within the coordinate step
                 taken at d) then
6                  H[d] =!H[d];
7          d = D.next(d);
8      if d! = new start coordinate then
9          if (!H[d] AND moved to even identifier a within the coordinate step taken at
               d) OR (H[d] AND moved to odd identifier a within the coordinate step taken
               at d) then
10             H[d] =!H[d];
11     return "false";
12 else
13     if moved to odd identifier a within the coordinate step taken at old end coordinate
         then
14         g =!g;
15         H[new start coordinate] =!H[new start coordinate];
16     return g;
```

4.7.6 Topological Access Method

DB4GeOGraphS core framework *indizes.api* contains a *TOAM* interface for topological access methods (TOAM). A UML diagram is shown in Figure 4.27. The implementation of a topological access method, called *TOAMRStarImpl* is based on a discrete R^**-Tree* which manages multidimensional discrete points. The first coordinate of a point as key for a node is given by the sum of the shortest distances to every other node within the *Property Graph* using a Dijkstra algorithm.

Using the sum of the shortest path distances is rather complex to calculate, but it yields some interesting results. For example, if every node is connected with every other node within the *Property Graph*, the corresponding points p_i with $i < n$ and n the number of nodes, each node has the same distance which is equal to one. The first coordinate would be equal to $n - 1$ for every node. If we picture an p-adic tree as a second example, the root node would be in the centre and gets the lowest sum of distances and the leaf nodes get the highest sum of distances since the distance over some parent node to another leaf node will always add some extra distance instead of going directly to each of the

Figure 4.27: Interfaces *TOAM*,*TOAMNode* and *TOAMEntry*

leaf nodes from the parent node. So the first coordinate is a measure for how close to the centre the location of the node actually is.

As described in Section 4.2.2 three different kinds of bidirectional relation types exist which are only usable as six different unidirectional relation types within *DB4GeOGraphS* core framework. An example is given in Figures 4.28 and 4.29 when the definition of the other coordinates is to count the number of relations per relation type. Other definitions may be more useful or efficient (e.g. summing up the steps to find the first coordinate but separated by relation type).

The main argument to take this particular definition was that it is possible to retrieve geo-objects with certain local topological properties. It is also possible to take other properties of the node into account, e.g. some thematic attributes, its spatial dimension or aggregation level. The implementation may be extended for further research. A first impression is given in the evaluation Section 5.3 by using the example definitions. All coordinates are given in each picture below the *Property Graph*. The graph depicts only the relations which the Dijkstra algorithm has used when starting at the node with a value of zero.

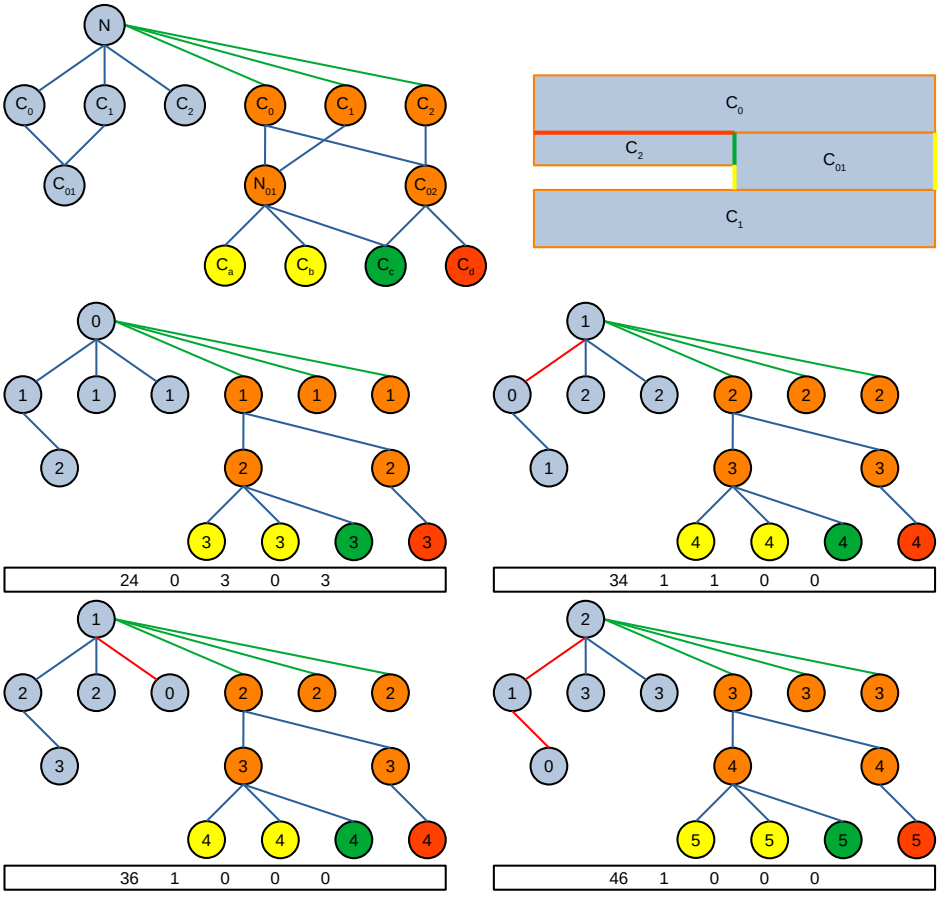


Figure 4.28: The top-left picture shows the graph of the topology from the shapes shown in the top-right picture. The relations in the *Property Graph* are part-of (red), composite-of (blue), border-of (orange) and inner-of (green). The pictures in the second and third row show the relation types which the Dijkstra algorithm had chosen when starting at the node with the value 0. The values of the other nodes are the shortest distances to the node with the value of 0.

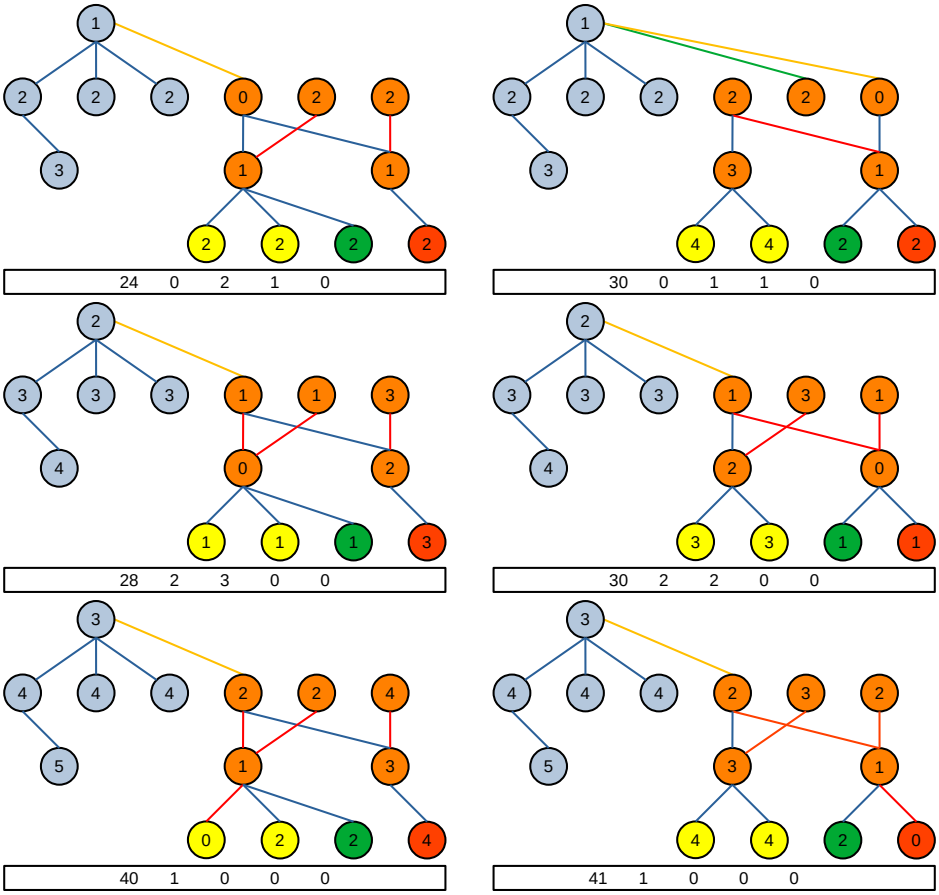


Figure 4.29: The pictures show the relation types which the Dijkstra algorithm had chosen when starting at the node with the value 0 of the example given in Figure 4.28. The values of the other nodes are the shortest distances to the node with the value of 0.

4.8 Operations

Operations are collected within the *operations* package of *DB4GeOGraphS* core framework. It is subdivided into three packages, *generators*, *io* and *filters*. The naming is inspired by the visualization toolkit *VTK* [67] from *KITWARE*. Just like in VTK, filters are seen as operations which transform objects into other objects. There is only one class within the *filters* package to solve different kinds of triangulations. The *generators* package consists of one class which generates point clouds for testing purposes. The *io* package is structured by file type. Three sub-packages exist *xml*, *vtk* and *ascii*. Each of which consists of source-classes which import objects and sink-classes which write objects. The following sections explain the triangulations and the basic importers and exporters.

4.8.1 Triangulations

The following sections describe the necessary triangulations to solve intersections and differences within *DB4GeOGraphS* core framework. The first algorithms calculate triangulations from point clouds to d-dimensional simplicial complexes. Those triangulations are needed within the intersection algorithms to triangulate wireframes. The next algorithms triangulate d-dimensional simplicial complexes to $(d+1)$-dimensional simplicial complexes. Those algorithms are necessary for Algorithm 2 in Section 4.2.3 to triangulate *Tetrahedron3DNet* objects from wireframe models. The section closes with special triangulations which are used within the difference calculations for simplicial complexes.

4.8.1.1 From point clouds to d-dimensional simplicial complexes

Three different algorithms exist which work in a relatively similar way. The first algorithm triangulates a set of points to a curve, the second to a surface and the third to a solid complex. They are Sweep-Algorithms. The first step is to sort the set of the involved points by the euclidean distance from the centre to the most distant point and build a d-dimensional complex from the first to the last point, where the first $(d+1)$ points build the first simplex and the nearest border together with the next point build the next simplex. If the nearest border together with the corresponding point lead to d-dimensional intersections with the first simplex, the next of the nearest borders of the first simplex is tried. Both simplices build a d-dimensional simplicial complex. The next point is drawn and the nearest border from the d-dimensional simplicial complex is chosen which does not lead to d-dimensional intersections to build and connect the third simplex. If a new simplex is invalid, the point is ignored. This process iterates until all points have been visited. Examples for each triangulation are given in Figure 4.30. The first example at the top may break if the points lie in one plane.

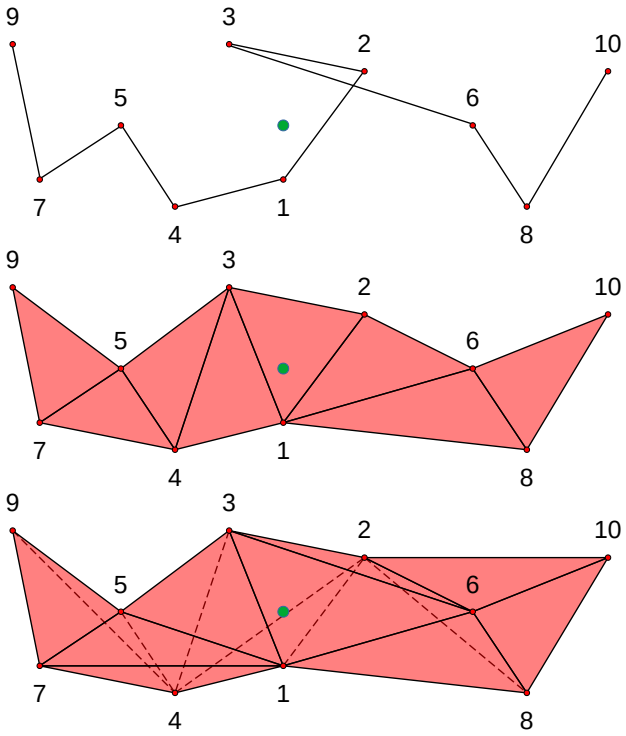


Figure 4.30: *n* points to curve (top), to surface (middle) and solid (bottom) with centre point (green)

4.8.1.2 From d-dimensional simplicial complexes to $(d+1)$-dimensional simplicial complexes

Paper [31] contains extracted, transformed and minimized parts of this section, published beforehand:

Algorithms 17 and 18 are sweep algorithms which triangulate concave polygons, represented as a *BREP*, or hulls (triangle complexes without boundary). Both algorithms were designed with two constraints. First, no additional points should be added to the point set. Second, the input is a d-dimensional simplicial complex without boundary and the output is a $(d+1)$-dimensional simplicial complex. The first constraint aims at not adding memory efforts for additional spatial information which is actually redundant since the inner points or the shape is precisely defined by a *BREP*, ignoring the fact of the numerical issues when dealing with computational arithmetic. The second constraint has its cause in geometrically induced topology. A *BREP* should bound only one "body".

Small bite by sweep: The basic idea of Algorithm 17 is to bite a triangle from a polygon, represented as a *BREP*. While the polygon itself is a collection of segments unambiguously connected through their bordering corner points, it is possible to sort those corner points and sweep over this list to generate $(d+1)$-dimensional simplices. The algorithm will fail if all sweep orders find invalid triangles (validity in accordance with the inaccuracy model, e.g. not too small, not too sharp) or valid triangles which intersect any of the remaining segments only. It is possible to add the *Delaunay* condition to reduce sharp triangles partly to the result set while sweeping over the set of corner points. The algorithm does not necessarily depend on planar input polygons, which are represented as *BREP*s. It will triangulate as long as there are no bad intersections or invalid triangles. The *Delaunay* condition reduces the number of ways a surface can be built, but since the algorithm depends on the type and order of the geometrical sort, there exists still a number of ways surfaces can be built.

Algorithm 17: Small bite by sweep

input : set of segments which form polygon represented as a *BREP*
output : set of triangles which form triangle complex

```
1  geometrically sort the corner points of the input set (e.g. ascending XYZ order);
2  for corner point in sorted corner point set do
3  |  take the two segments which share the corner point;
4  |  create a triangle by adding the missing segment;
5  |  if the triangle is not valid or intersects any of the previously generated triangles
   |    → try a different geometrical sort and go on;
6  |  remove the two segments which share the corner point from the input set;
7  |  add the newly created segment to the input set if it is not contained in the input
   |    set to close the input set;
8  |  add the triangle to the result set;
9  end
10 return the set of triangles;
```

Constraints:

1. $(d+1)$-dimensional simplices need to be valid
2. the convexity of the d-dimensional hull representation is rather simple in a way that simple geometrical sort algorithms lead to any $(d+1)$-dimensional simplex which does not intersect the rest of the d-dimensional hull representation

Big bite by sweep: The aim of Algorithm 18 was to have more control over the bite order of Algorithm 17 by using a triangulation from $(d-1)$-dimensional simplicial complexes to d-dimensional simplicial complexes also. This yields the advantage of one controlled big bite instead of randomly taken small bites. For each corner point, there exists a set of triangles forming a triangle complex (like a rice hat). The border

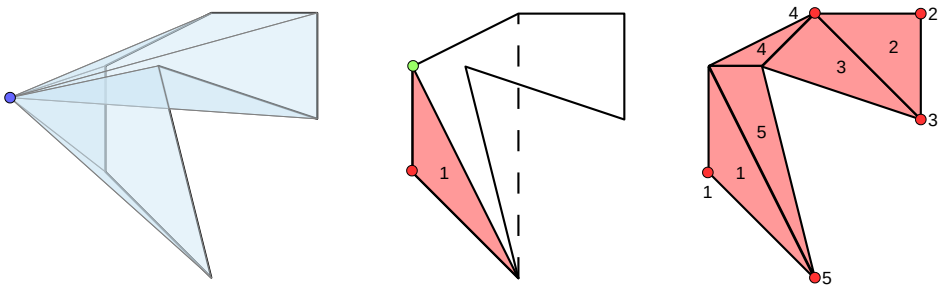


Figure 4.31: big bite by sweep, using small bite by sweep

of this triangle complex is a polygon represented as a *BREP* which can be triangulated to a second triangle complex (like a rice hat capping) by Algorithm 17. Both triangle complexes together form a hull if they do not overlap. Each triangle of the second triangle complex together with the corner point may form a tetrahedron. However, the failing issues of Algorithm 17 still remain (cf. last paragraph).

Algorithm 18: Big bite by sweep

input : set of triangles which form hull (*BREP*)
output: set of tetrahedrons which form a tetrahedron complex

```
1 geometrically sort the corner points of the input set (e.g. ascending XYZ order);
2 for corner point in sorted corner point set do
3 |   take all triangles which share the corner point (rice hat);
4 |   triangulate its border (polygon represented as a BREP) to create the second
  |     triangle complex (rice hat capping);
5 |   create the tetrahedrons from the corner point to each triangle within the second
  |     triangle complex;
6 |   try to bite each tetrahedron;
7 |   if any not solvable problems occur →try a different geometrical sort and go on;
8 end
9 return set of tetrahedrons;
```

The example in Figure 4.31 shows the steps to take one big bite from a hull to transform this *BREP* step by step into a tetrahedron complex. Sorting the points of the hull by some lexicographic order is the first step. The blue point is connected to seven triangles, forming a triangle complex (blue). The border of the triangle complex is a segment complex, which can be triangulated by Algorithm 17. This algorithm also sorts the points of the segment complex by some lexicographic order. The second point, green point, leads to an intersection (middle) and another lexicographic order is tried which finally finds all remaining triangles. The resulting triangle complex (right) together with the blue point can be triangulated to a tetrahedron complex with five tetrahedrons.

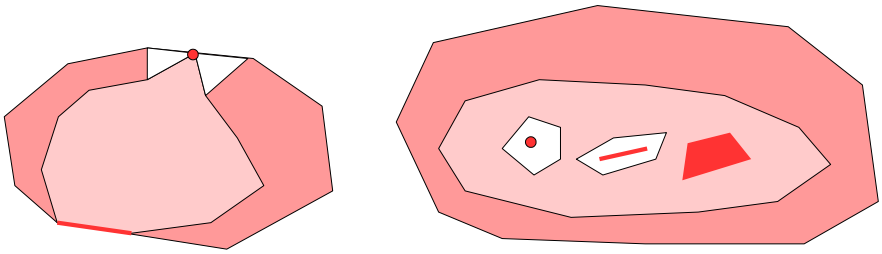


Figure 4.32: *BREP* triangulations (white); 2D case of planar polygons (left) with two border intersections (red point and line); 3D case of two hulls (right) with three intersections (red point, line, and square)

4.8.1.3 Triangulate around *BREP* intersections

The triangulations around *BREP* intersections are important to create disjoint *BREPs* which can be triangulated by the standard triangulations described in the previous section. *BREP* intersections appear when calculating the difference of two d-dimensional simplicial complexes. As described in Section 4.3.3 the first step of calculating a difference is to calculate the intersection of both d-dimensional simplicial complexes. If the intersection is also d-dimensional, then the intersection of the border of the d-dimensional simplicial complex whose difference to the other d-dimensional simplicial complex is being calculated and the border of the intersection is calculated. Those intersections may differ in dimension and need to be treated differently.

Figure 4.32 (left) shows a triangle complex in red and the inner intersection in light red. Two border intersections are also shown, a red curve and a red dot. The red curve can easily be removed from the inner and the outer *BREP* and both *BREPs* can be glued at the border of the red curve to create two disjoint *BREPs*. The red dot can not be removed from the inner and the outer *BREP* which in turn can be glued to two disjoint *BREPs*. But it is possible to create two triangles (white) and add the corresponding border parts to glue the *BREPs*.

Figure 4.32 (right) shows a tetrahedron complex in red and the inner intersection in light red. Three border intersections are also shown, a red surface, a red curve and a red dot. The red surface can easily be removed from the inner and the outer *BREP* by using the difference operation on the *BREPs* and both *BREPs* can be glued at the border of the red surface to create two disjoint *BREPs*. The red curve and the red dot can not be removed from the inner and the outer *BREP* which in turn can be glued to two disjoint *BREPs*. But it is possible to create tetrahedrons (white) and add the corresponding border parts to glue the *BREPs*.

If the inner and outer *BREP* do not intersect, a square needs to be cut out in case of triangle complexes or a prism in case of tetrahedron complexes in order to connect the inner and the outer *BREP*. Both examples are shown in Figure 4.33.

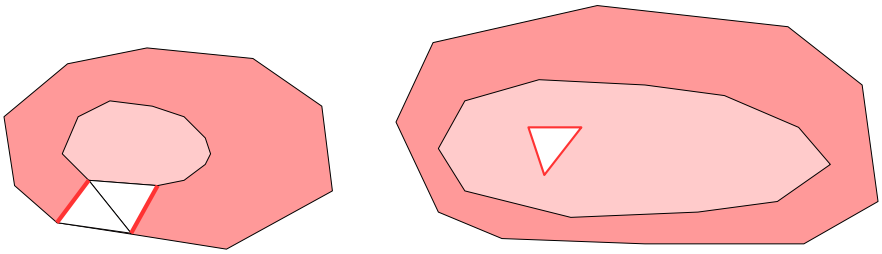


Figure 4.33: *BREP* triangulations (white); 2D case of planar polygons (left) with additional segment (red lines); 3D case of two hulls (right) with the additional triangle complex of the prism border without the two capping triangles (red triangle)

4.8.2 Input and Output

4.8.2.1 XYZ-Files Importer

XYZ-Files are used to store digital elevation data. They are organized in a sequence of point coordinates along a grid. The importer is able to manipulate the origin and to triangulate the digital elevation model into a triangle complex. The importer is also able to manipulate the sampling rate by defining a preferred segment length. This segment length must not be smaller than the cell size, otherwise it will be set to the cell size. The integer result of segment length divided by the cell size gives the jump distance. The resulting spatial object is returned as part of a *TONode3D* root node for the file. An example of the open data of Erfurt is shown in Figure 4.34.

4.8.2.2 ASC-Files Importer

ASC-Files are used to store digital elevation data. They are organized in a sequence of elevation coordinates along a grid. The header tells about the number of rows and columns, the origin, the cell size and the value for unknown data. The importer is able to manipulate the origin and to triangulate the digital elevation model into a triangle complex. The importer is also able to manipulate the sampling rate by defining a preferred segment length. This segment length must not be smaller than the cell size, otherwise it will be set to the cell size. The integer result of segment length divided by the cell size gives the jump distance. The resulting spatial object is returned as part of a *TONode3D* root node for the file. An example of the open data of Vienna is shown in Figure 4.35.

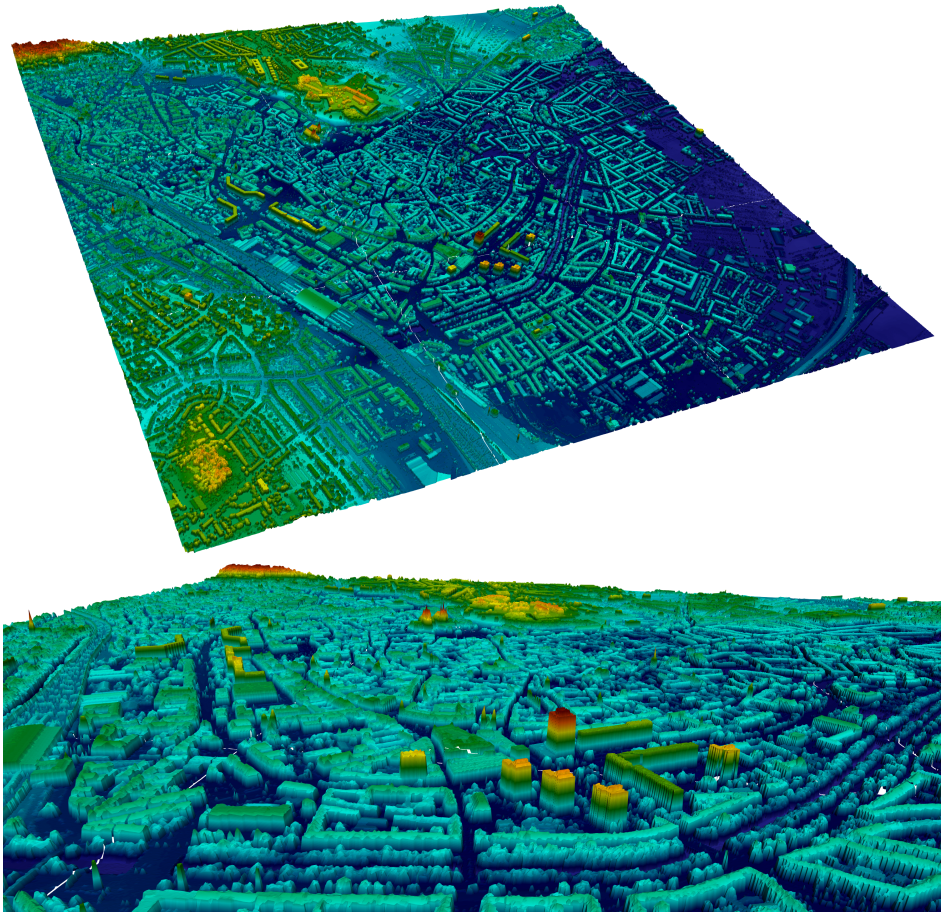


Figure 4.34: Import of nine *xyz*-files of Erfurt triangulated as triangle complexes colored by *z*-coordinate.

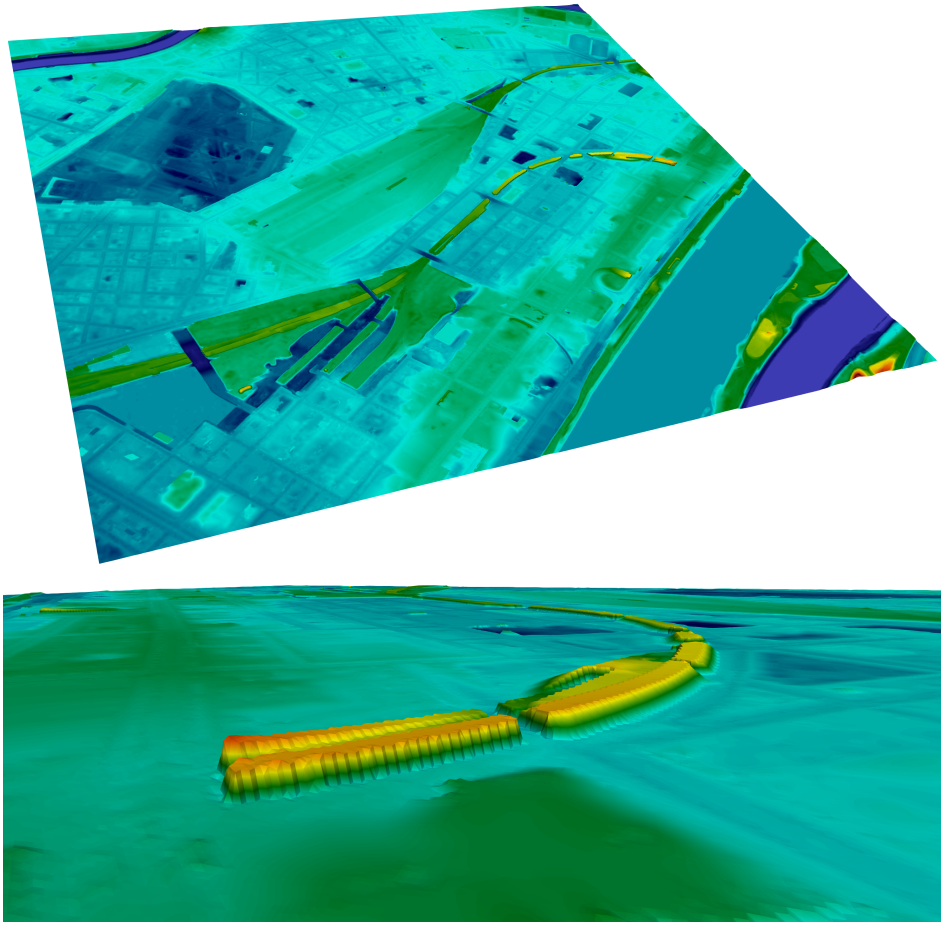


Figure 4.35: Import of *asc*-file near Vienna triangulated as triangle complex coloured by z-coordinate. Datasource: City Vienna - data.wien.gv.at

4.8.2.3 Simplex-Files Importer

Simplex-Files are *ASCII* files which carry 3-dimensional points with empty lines after one line for point sets, after two lines for segment sets, after three lines for triangle sets and after four lines for tetrahedron sets. Each simplex is collected into a *Spatial3DCollection* which tries to glue all simplizes to simplicial complexes. The resulting collection is returned as part of a *TONode3D* root node.

4.8.2.4 PointND-Files Importer

PointND-Files are *ASCII* files which carry n-dimensional points within each line. The n-dimensional points are added as thematic attributes to *TONode3D* nodes. The spatial parts of those nodes are *Point3Ds* whose coordinates can be chosen from the n-dimensional points. Each of the generated nodes are added to the root node for the file.

4.8.2.5 Forest-Files Importer

Forest-Files are *.txt files used to store field data of Barro Colorado Island [4] which can be obtained from the Smithsonian Tropical Institute. They contain records of trees. The files contain information about the tree position, tree species, subspecies, which quadrant the tree is located, the tree ID, stem ID, breast height diameter, the date on which the record of the tree has been taken and if the tree is alive or not etc. It is a spatio-temporal dataset and one of the best records of tropical rain forest development. The records are translated into a set of *TONode3Ds* with all the attributes contained by the files. An example is shown in Figure 4.36. Each of the generated nodes are added to the root node for the file.

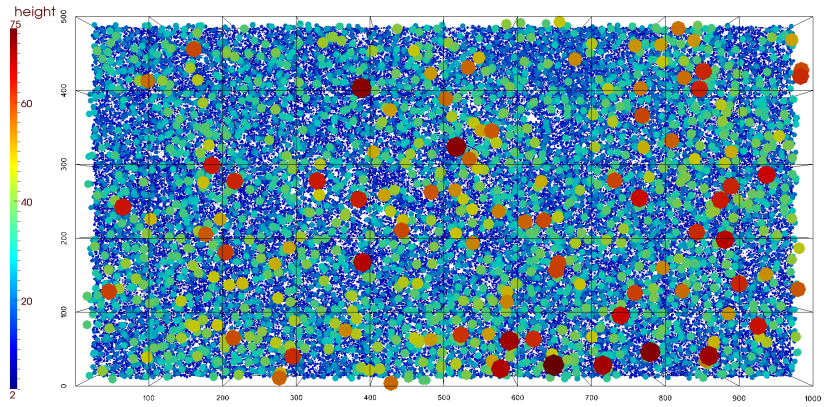


Figure 4.36: Import of a *Forest*-file from Barro Colorado Island coloured and elevated disks by height which has been calculated by the typical relation of the breast height diameter and the height.

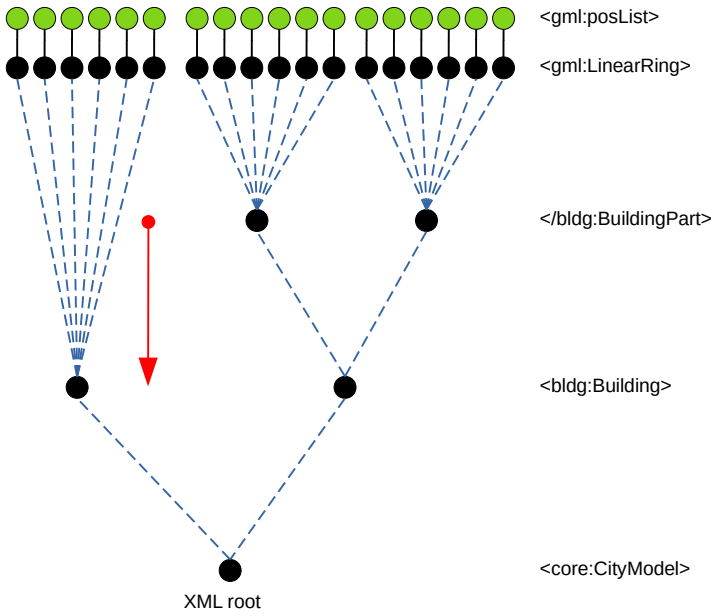


Figure 4.37: Example definitions to create aggregations by using tags of the *XML*-tree.

4.8.2.6 XML-Importer

The *XML*-importer uses an XML Java library to read in *XML*-files. The whole *XML*-tree is translated into the *Property Graph Model* of *DB4GeOGraphS* core framework. Most of the nodes are of type *TONodeImpl*. The important tags to read nodes of type *TONode3D* are the “creationDate”-tag from which the temporal property as *TemporalStamp* is created, the “name”-tag from which the node gets its name and the “posList”-tag from which the spatial property as a segment complex will be created if it does not self-intersect. If the polygon, represented as a *BREP* self-intersects, a *Segment3DNet* object will be created. A *TONode3D* node is connected to its parent *TONodeImpl* node through an abstraction relation. Figure 4.38 shows an example.

It is possible to define a set of tags which will be used to aggregate *TONode3D* nodes and add one composite *TONode3D* node which relates to the *TONode3D* nodes in aggregation relations. The composite nodes are added to the root node for the file with an abstraction relation. If one tag surrounds another tag, it will be ignored. Figure 4.37 shows an example.

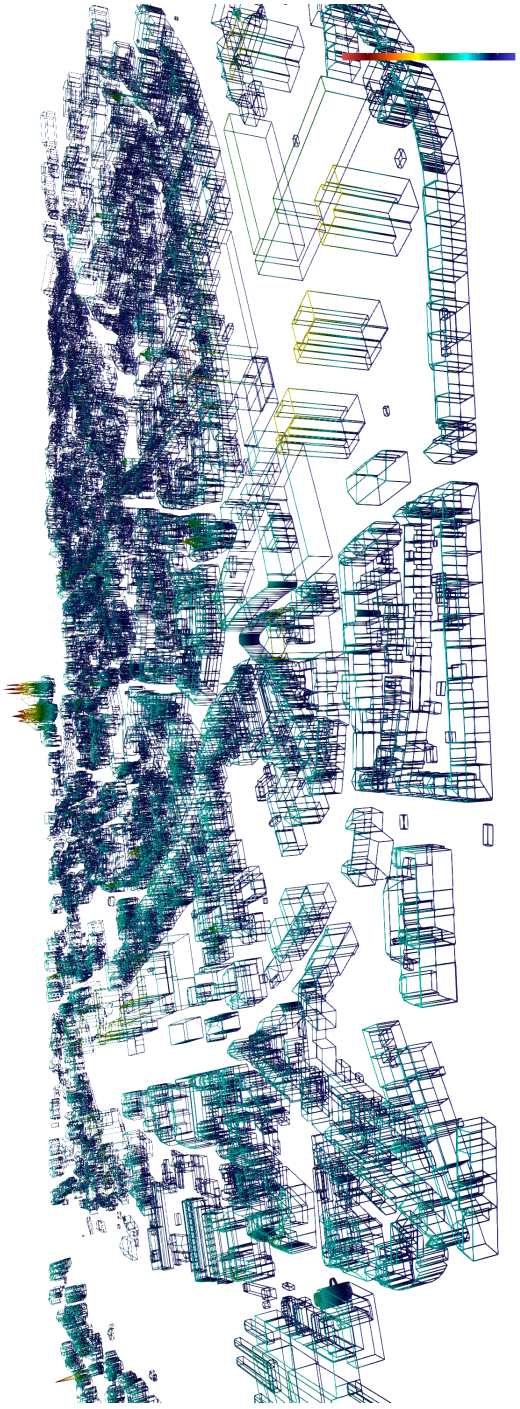


Figure 4.38: Import of a *CityGML(XML)*-file of Erfurt which contains "planar" polygons (represented as *BREPs*) as *Segment3DNet* object with the contained segment complexes coloured by z-coordinate.

4.8.2.7 XML-Exporter

The *XML*-exporter uses the *write* operation of the interfaces *TONode*, *TONodeAM* and *TOAM*. Each *write* operation writes its *XML*-representation to an output stream. In case of the interfaces *TONode*, the *write* operation also takes an argument for visited *TONodeIDs*. If some related node ID's of a node are contained in the list of visited *TONodeIDs*, the *write* operations of those nodes are not called.

4.8.2.8 VTK-Spatio-Temporal-Exporter

The visualization is as important as data management and data analysis. In this area, *VTK* [67] and *ParaView* [49] should be mentioned. In order to guarantee the connection to *VTK* and *ParaView*, *VTK* exporters were written for *DB4GeOGraphS* core framework. The exporters translate the structure of the graphs or the spatial or spatio-temporal objects into *VTK* data types and uses the writers from *VTK* library through the wrapped classes.

The spatio-temporal exporter creates *VTKs* unstructured grids from each spatial and spatio-temporal type and uses the wrapped unstructured grid writer of the *VTK* library to export the unstructured grid to the file system. Figure 4.39 or the examples of the *ASCII*-importers from Figures 4.34, 4.35, 4.36 show various objects that were successfully exported from *DB4GeOGraphS* core framework. It is possible to fill up an unstructured grid within some algorithm with some *TONode3Ds* or *TONode4Ds* objects, or to export *TONode3Ds* or *TONode4Ds* objects directly. The time series from the spatio-temporal model of *DB4GeOGraphS* core framework could be animated in *ParaView*. The parallel data format of *ParaView* can also be exported, which reduces the time of importing the files when used on a *ParaView* clusters.

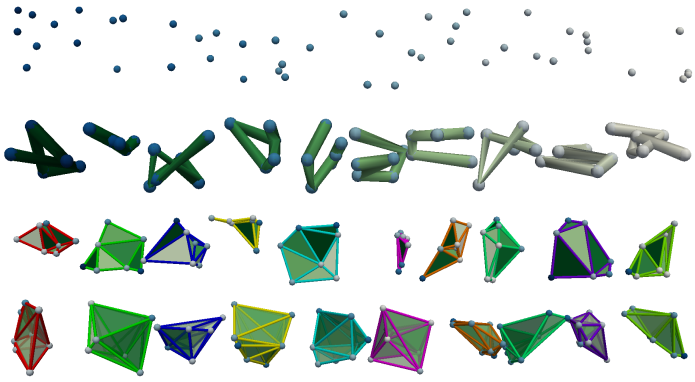


Figure 4.39: VTK-Exports in *ParaView* with thematic
From top to bottom: *Point3DNet*, *Segment3DNet*, *Triangle3DNet* and *Trahedron3DNet* object

4.8.2.9 VTK-T_0-Space-Exporter

A T_0-space can be exported in two different ways. The first method creates a tree-like structure based on radial coordinates (see Figure 4.40). Each node b which is related to a node a is positioned around the node a by the half of the radius of node a to its parent node c within half of the angle range of node b in the direction of b. The angle range is partitioned by the number of related nodes. Since there are three dimensions to use, the height is taken from the relation type. The second method lies the graph in the metric space as it would be drawn onto paper, with the root node at the top and the target nodes in a row beneath (see Figure 4.41).

4.8.2.10 VTK-TONodeAM-Exporter

The *TONodeAM*-exporter is used to export a spatial representation of the *TONodeAM* access method. Since the *TONodeAM* is based on a space filling curve, it is possible to visualize the curve, the hyper-cuboids and the tree. Figure 4.42 shows the hyper-cuboids and Figure 4.43 shows the tree of a 2-adic version within *ParaView*.

4.8.2.11 VTK-TOAM-Exporter

The *TOAM*-exporter is used to export a spatial representation of the *TOAM* access method. Since the *TOAM* is based on a *R*-Tree*, it is possible to visualize the hypecubes and the T_0-space in metric space. Figure 4.44 illustrates the *TOAM*-exporter on an example.

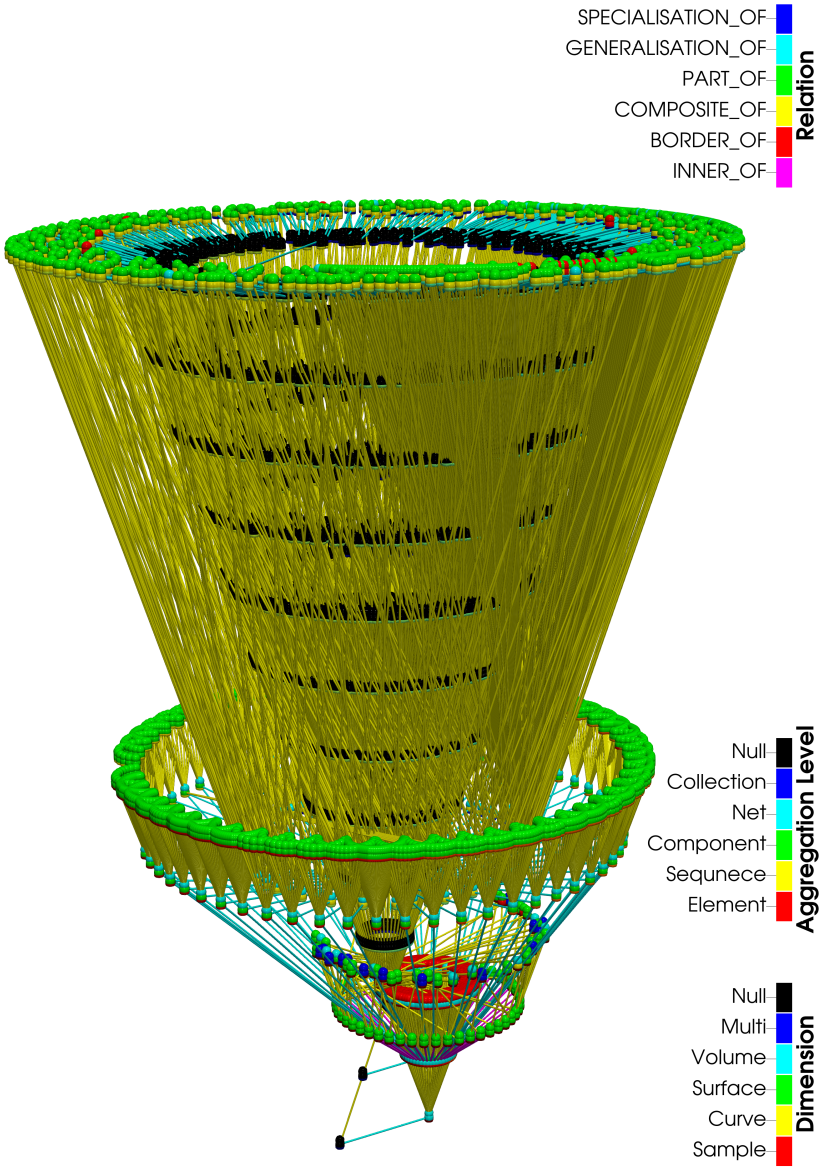


Figure 4.40: Salvador's historical city centre - *CityGML* tree and the triangulations by Algorithm 2 with spherical coordinates (see Section 5.3 for detailed explanations).

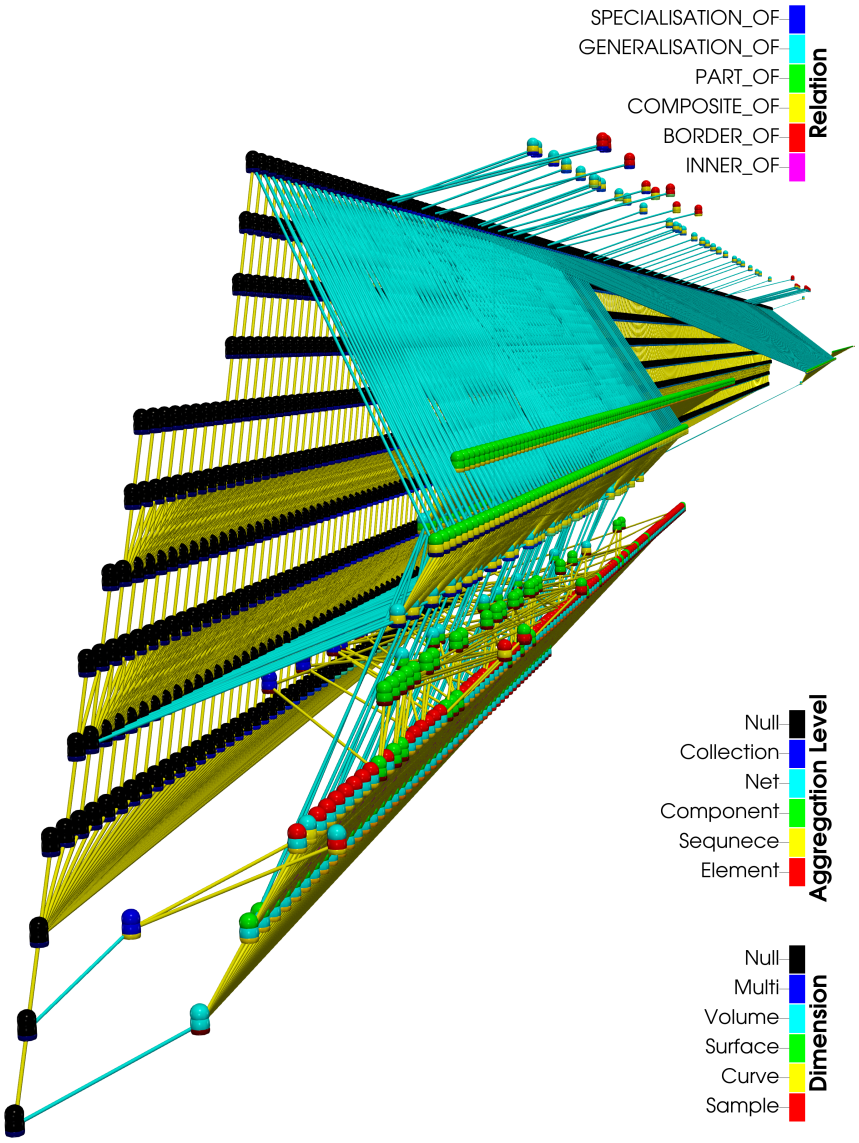


Figure 4.41: Salvador's historical city center - *CityGML* tree and the triangulations by Algorithm 2 with euclidean coordinates.

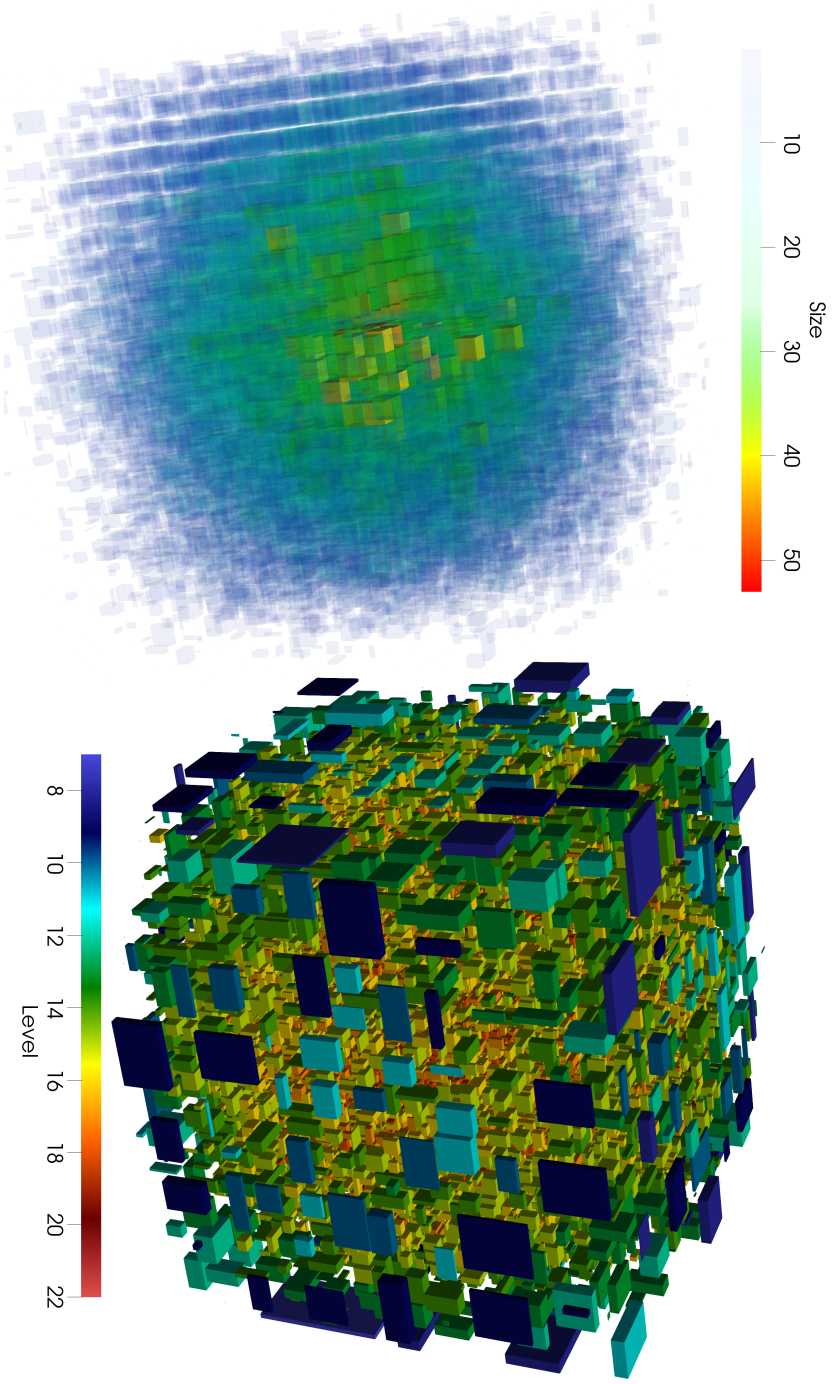


Figure 4.42: Hyper-cuboids of a fixed (left) and scalable (right) 2-adic space filling curve index managing a 3-dimensional normal distributed point cloud

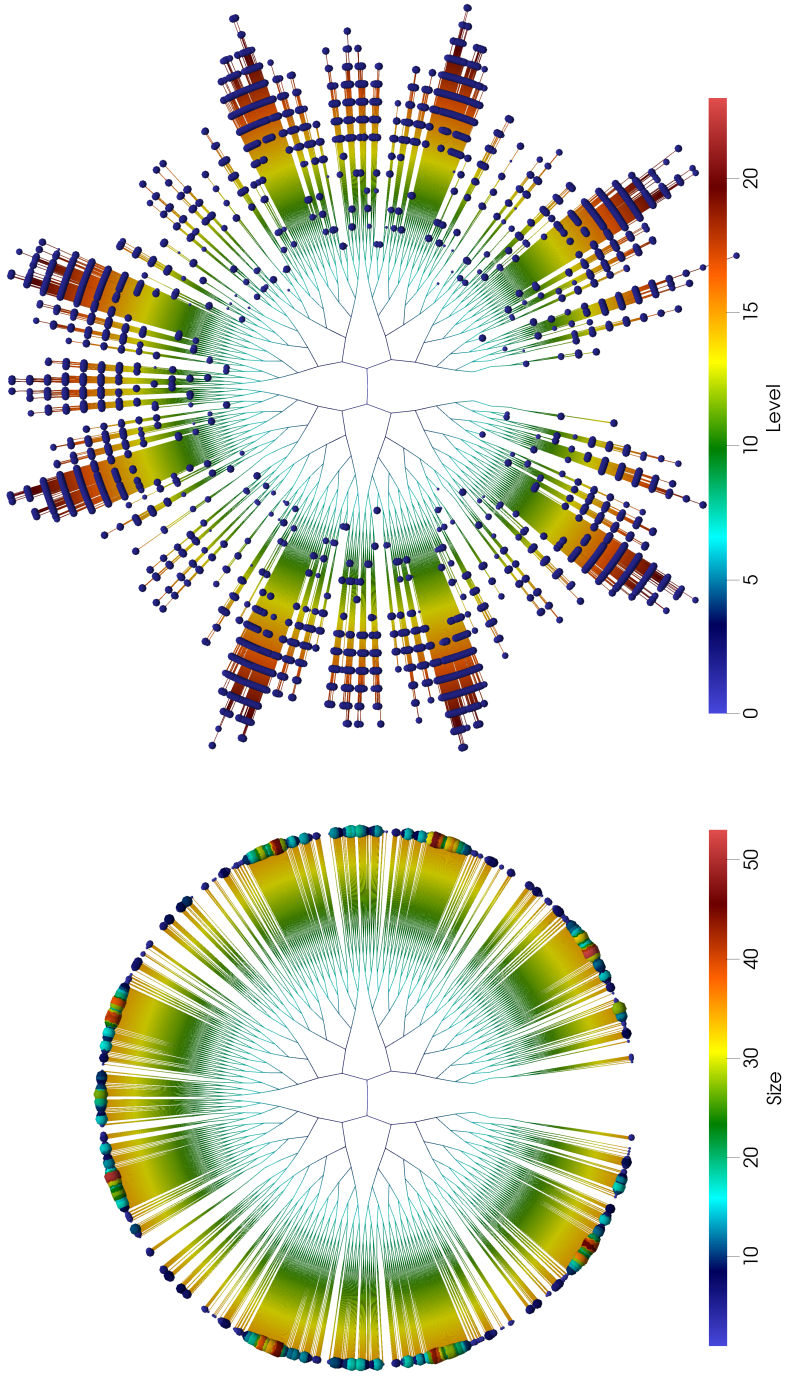


Figure 4.43: Tree of a fixed (left) and scalable (right) 2-adic space filling curve index managing a 3-dimensional normal distributed point cloud.

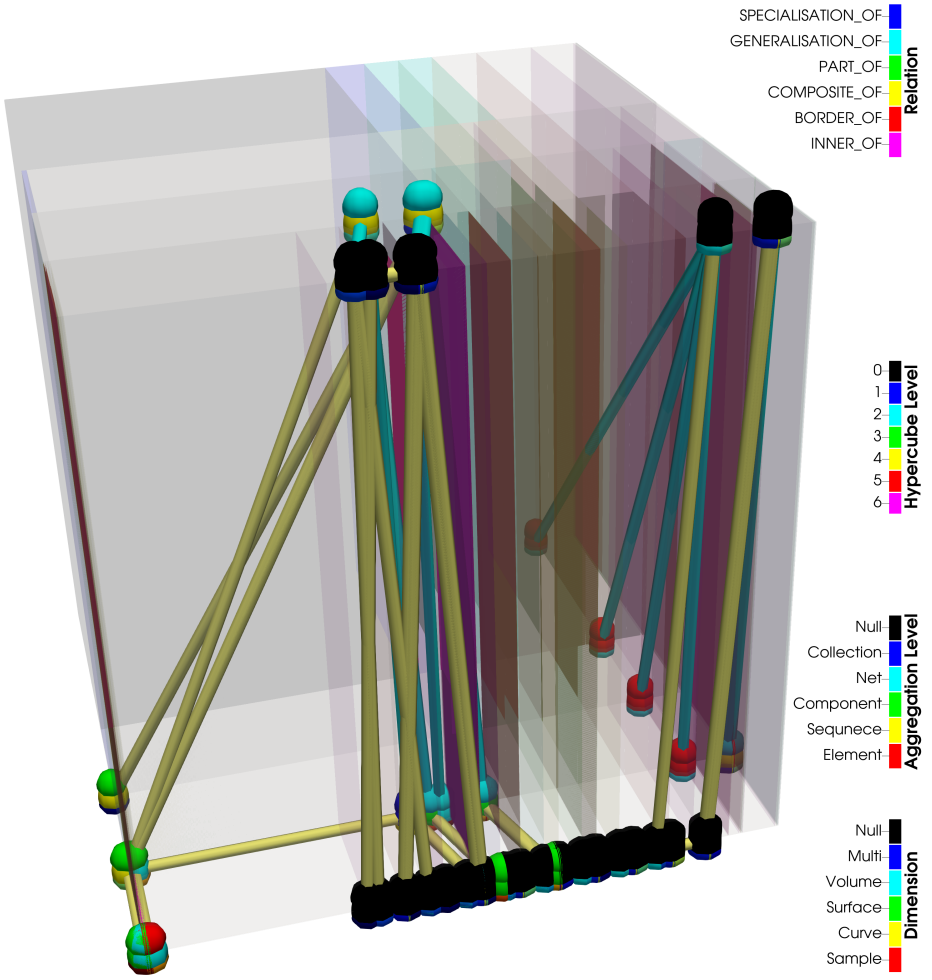


Figure 4.44: Salvador's historical city centre - Graph bend by topological access method as described in Section 4.7.6 where the x-axis represents the sum of distances of a given node, the y-axis represents the number of nodes from which a given node is a specialization and the z-axis represents the number of nodes from which a given node is a generalization.

5 Evaluation

The evaluation of the *DB4GeOGraphS* core framework is done for three different features. The geometry core and the handling of the new *Property Graph Model* is evaluated first with a sensitivity analysis on Algorithm 2 using *CityGML* field data from Erfurt, Germany. The second evaluation is testing the behaviour of the p-adic scalable space filling curve index implementation of Section 4.2.3 using a point cloud from Barro Colorado Island in Gatun Lake (central Panama) with 50 hectares of rain forest in a series of 7 censuses, beginning in 1982. The third evaluation is testing the behaviour of the topological access method of Section 4.7.6 using *CityGML* field data from Salvador, Brazil.

5.1 Creating Tetrahedral Models

5.1.1 Computing watertight Volumetric Models from Boundary Representations to ensure consistent topological operations

This section is published in [31] beforehand:

The focus was not on adding additional points to the data-set to establish "beautiful" triangles or tetrahedra which are not too thin etc. in order to operate on the point set only. We did not want to change the data. We also did not want to operate on data which has been manipulated by some *CityGML* optimizer or validator. These may be optimized in application through the modelling steps or generation steps of the input set. *CityGML* is just a showcase for a more general problem when dealing with *BREPs*. The data were produced by the *tridicon CityModeller* (Leica/Hexagon) and are manually revised with the *tridicon 3D Editor* (Leica/Hexagon). The produced *CityGML* data is stored in *3dcitydb*. The production steps were carried out by the Thuringian state office for land management and geo-information (TLBG). The data was zeroed to the coordinates $lat = 642004.72$, $lon = 5649093.259$ $z = 229.074$.

All precisions ϵ, ε, ϵ, λ and ζ were chosen to be 10^{-6} for this presentation. Figures 5.2 and 5.3 illustrate some difficult situations when dealing with planarity and double precision arithmetic. Planarity of triangle complexes is checked against the normal vector of an arbitrary triangle within the triangle complex by calculating the cosine between the reference normal vector and the other normal vectors. If the cosine is equal to one with a deviation not larger than ζ, then both normal vectors are interpreted as collinear. The top picture of Figures 5.2 and 5.3 filters non-planar surfaces of the input set for

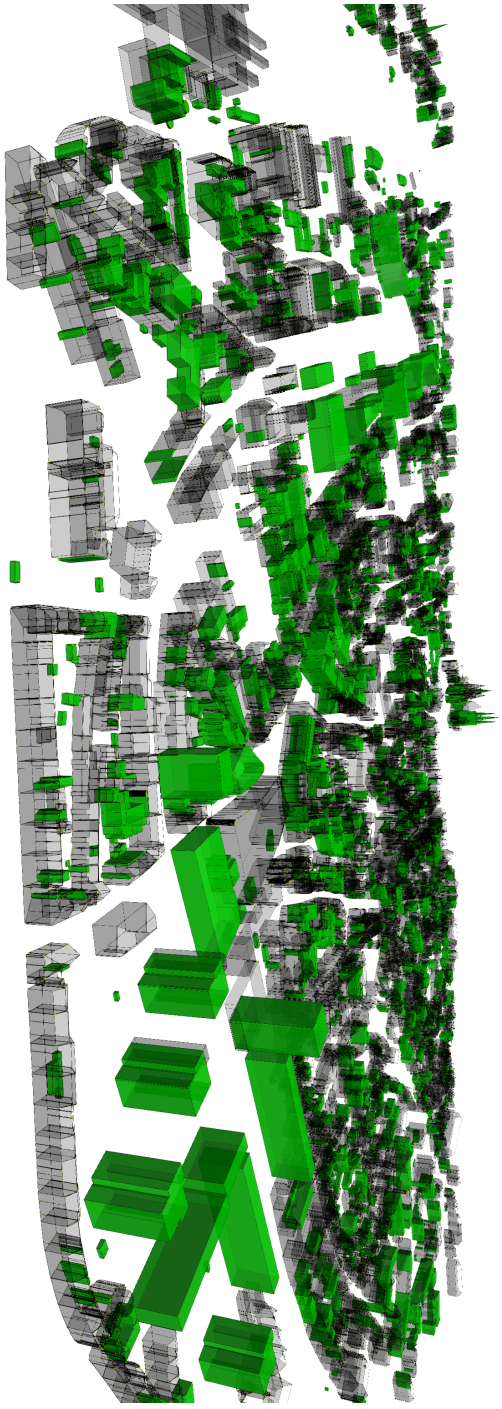


Figure 5.1: Erfurt city centre, "planar" polygons (represented as *BREP*s) of the input *CityGML* dataset (black), triangle complexes (grey), tetrahedron complexes (green)

Algorithm 2. This is one of the main reasons why Algorithm 2 fails to build solids. The second picture from the top shows triangulated polygons of the input *CityGML* dataset (orange) which are not checked for planarity. However, even if the input set contained non-planar triangulated polygons of the input *CityGML* dataset, Algorithm 2 would not walk over those patches which are not planar to the pair of patches within the *Dijkstra* in Step 6 when generating recursively the surfaces in Step 2. But the *Dijkstra* in Step 6 finds other polygons which are not part of the input set (e.g. the floor of the attic Figure 5.3 top). But if the triangulated surfaces are not part of the input set, then they will be rejected. These two situations can be handled by splitting each triangle complex into a set of planar triangle complexes as a pre-processing Step (see. Figure 5.2 third picture from the top) and ignoring the constraint that every surface of Step 2 needs to be part of the input *Triangle3DNet* object (see. Figure 5.3 third picture from the top). The roof is one tetrahedron complex and the main building is another disjoint tetrahedron complex. This example shows a valid tetrahedralization. If the building has concave structures like an open atrium, they will be closed and the results tend to be invalid. I will refer to this situation as atrium problem. However, ignoring non-planar triangle complexes lead to 40% tetrahedralized building parts. Splitting non-planar triangle complexes as pre-processing step lead to 60% tetrahedralized building parts (Figure 5.1 city centre, Erfurt). For comparison, if the precisions $\epsilon = \varepsilon = \epsilon = \lambda$ are set to 10^{-2} and angular precision ζ is set to 10^{-9}, the tetrahedralized building parts are over 80%. More results are presented later on.

Three different ways of tetrahedralizing *Triangle3DNet* objects were tried. The first method is to aggregate every triangle complex of each building part of every building into one *Triangle3DNet* object, and to tetrahedralize this *Triangle3DNet* object. The second method aggregates every triangle complex of each building part into one *Triangle3DNet* object per building and tetrahedralizes those *Triangle3DNet* objects separately. The third method tetrahedralizes each building part separately. The resulting tetrahedron complexes are aggregated into one *Tetrahedron3DNet* object for further analysis. If the stitched surfaces are not cross-checked against the input *Triangle3DNet* object, then each decomposition type has the atrium problem on its aggregation level. Nevertheless, the first method leads to disjoint tetrahedron complexes, since the *buildCores()* operation called on the node of the *Triangle3DNet* object returns distinct tetrahedralized tetrahedron complexes. The second method leads to shared solid volumes between buildings, if they overlap. The third method leads to shared solid volumes within buildings and between buildings, if the different tetrahedron complexes overlap. Figure 5.4 illustrates that situation and shows building parts of the Erfurter Dom and the Severikirche which do overlap (coloured) and need to be subtracted from another to form distinct solids. The analysis has been parallelized. Each thread calculates one *Triangle3DNet* object. Since the computation of one *Triangle3DNet* object itself was also parallelized (cf. end of paragraph on Algorithm 2 in Section 4.2.3), the computation time was reduced significantly.

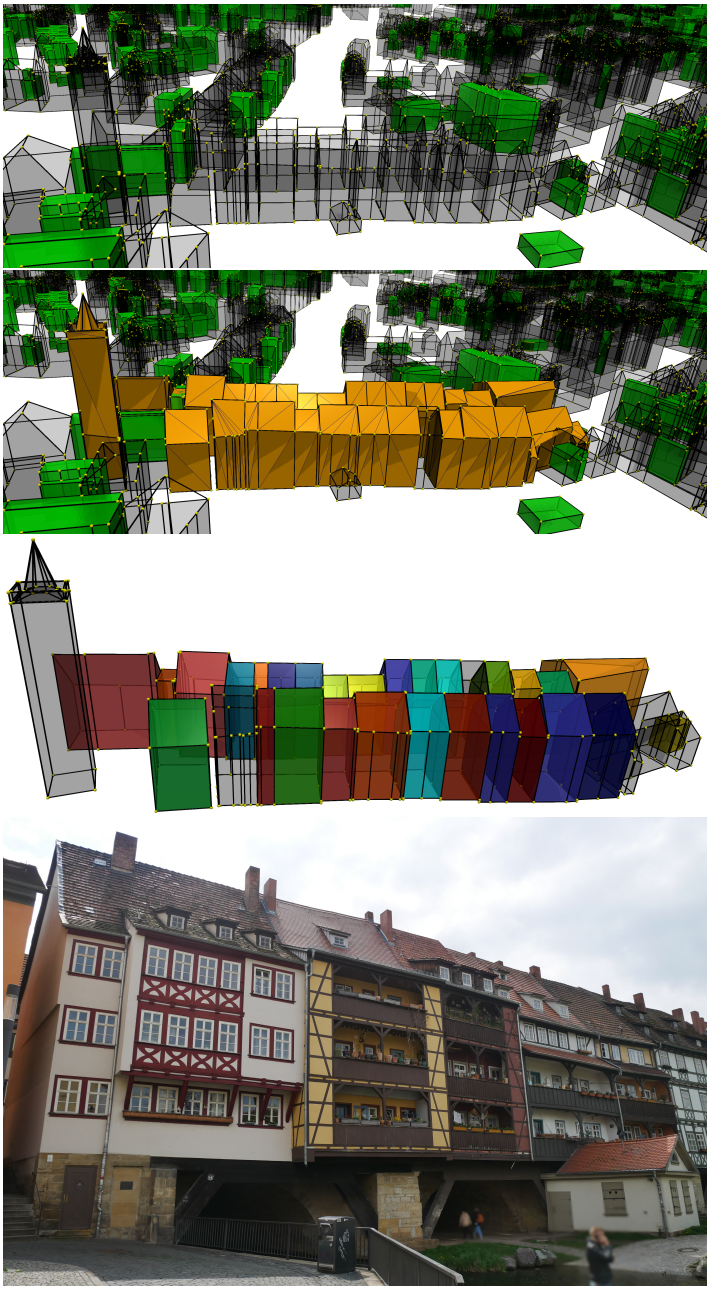


Figure 5.2: Krämerbrücke (Europe's longest bridge with fachwerk houses on both sides), "planar" polygons (represented as *BREPs*) of the input *CityGML* dataset (black), triangle complexes (grey) with ignored planar constraint while triangulation (orange), tetrahedron complexes (green) with splitting of non-planar triangle complexes as pre-processing step (coloured)

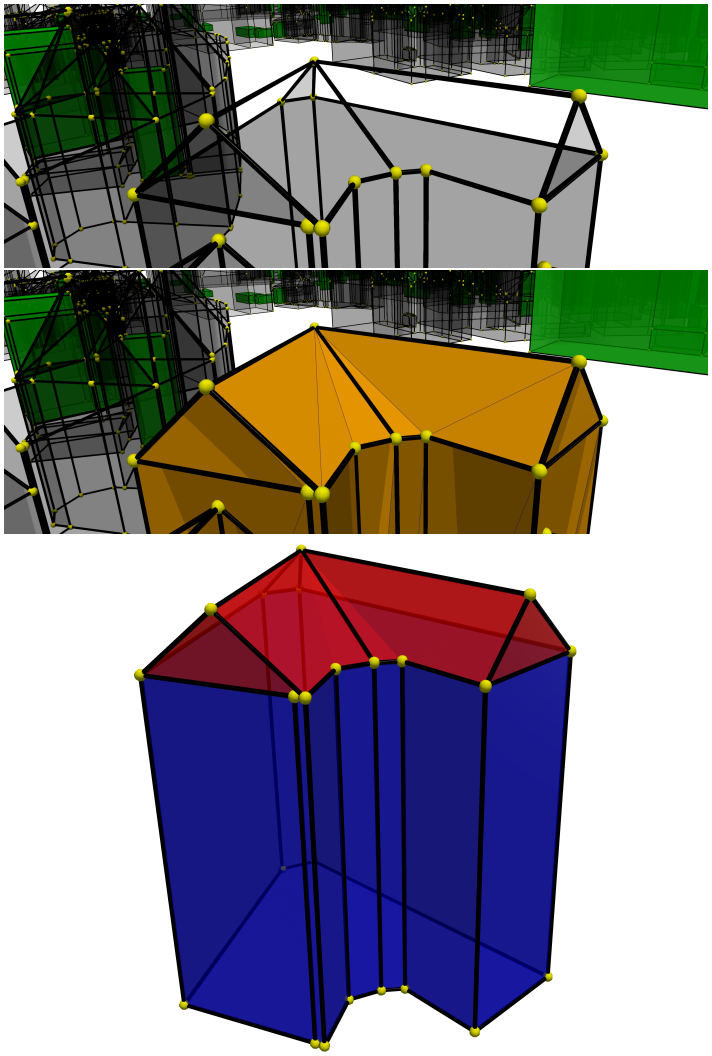


Figure 5.3: Juri-Gagarin-Ring 2, "planar" polygons (represented as *BREPs*) of the input *CityGML* dataset (black), triangle complexes (grey) with ignored planar constraint while triangulation (orange), tetrahedron complexes with ignored check of stitched surfaces against input set (red/blue)

5.1.2 A Sensitivity Analysis Using Double Precision Arithmetic

The sensitivity analysis is understood under the following two definitions provided in the lecture notes for equation-based modelling [45]:

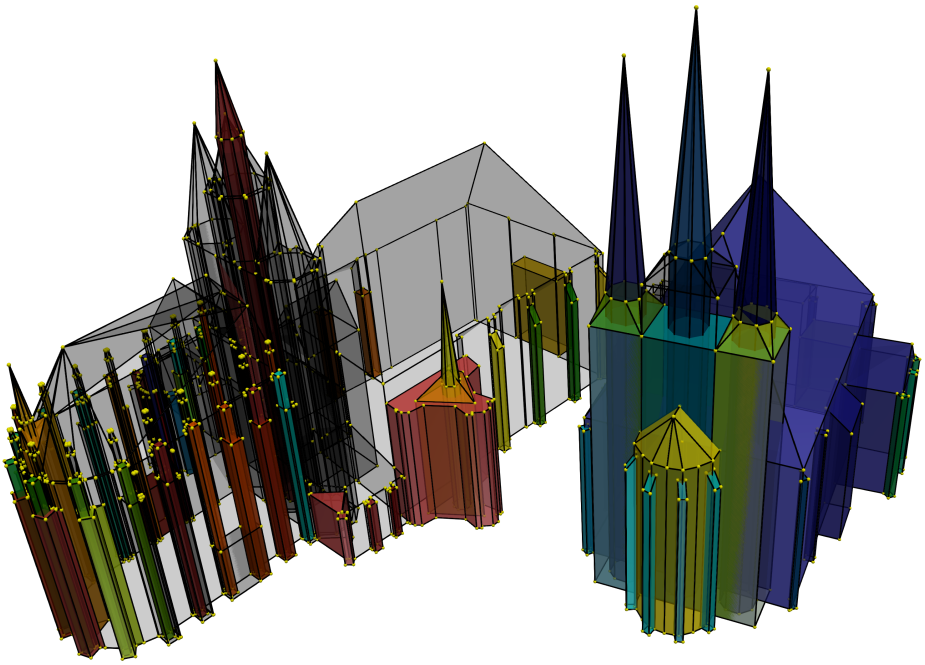


Figure 5.4: Erfurter Dom and Severikirche, "planar" polygons (represented as *BREPs*) of the input *CityGML* dataset (black), triangle complexes (grey) overlapping tetrahedron complexes (coloured)

State paths (trajectories) The representation and comparison of the dynamics of the state quantities, i.e. the state paths (trajectories) in the state space as a function of the selected parameters and external influences give indications of the general system behaviour (vibrations, breakdowns, chaos etc.) and on the effect of individual parameters.

Sensitivity The comparison of state paths as a function of variations of sensitive parameters provides information on the sensitivity of the model and the system, on uncertainties in the formulation or change of critical parameters.

Since the system does provide single states and not state paths as trajectories, I compare the states which are calculated by a parameter variation. The focus of the sensitivity analysis lies on how many decompositions (buildings/building parts or buildings as a whole) can be triangulated under a certain parameter set. For geo-information purposes, shared solid volumes are calculated and the reciprocal speed up in percentage of a certain parallelization is also presented. The test parameters have been chosen from the inaccuracy model described in Section 4.3.1. Only the factor ζ' for cosine comparison

and the base precision value α have been manipulated for the sensitivity analyses. The other factors have been set to the value of one. This implies that $\epsilon = \varepsilon = \epsilon = \lambda$.

Table 5.5 shows an overview of the data quality of some portions of the data released by the Thuringian state office for land management and geo-information (TLBG).

Building(s)	Juri-Gagarin-Ring 2	Juri-Gagarin-Ring 126c	Severi-Kirche	Krämer-Brücke	Marien-Dom
CityGML tree nodes	222	222	3385	5062	14940
buildings / building parts	1	1	31	33	86
polygones	18	18	285	373	1398
distinct polygones	18	18	283	371	1198
decimal places	3	3	3	3	3
min. point distance [m]	0.450	0.320	0.057	0.001	0.001
segments	78	96	1172	1632	5652
min. length [m]	0.450	0.320	0.073	0.031	0.002
average length [m]	8.527	21.073	11.809	7.684	7.821
max. length [m]	16.485	47.844	53.844	31.507	59.345

Building(s)	LoD2-644-5652-2-TH	LoD2-642-5644-2-TH	LoD2-644-5644-2-TH	LoD2-642-5648-2-TH
CityGML tree nodes	32630	39085	56708	1411404
buildings / building parts	278	322	486	9288
polygones	2411	2911	4119	112924
distinct polygones	2378	2837	4076	109224
decimal places	3	3	3	3
min. point distance [m]	0.001	0.001	0.001	0.001
segments	10319	12140	17280	506503
min. length [m]	0.003	0.001	0.005	0.001
average length [m]	4.495	4.669	5.502	7.014
max. length [m]	40.197	38.981	48.413	127.456

Figure 5.5: Data quality at $\epsilon = \varepsilon = \epsilon = \lambda = 10^{-3}$ and $\zeta = 10^{-9}$

The sensitivity analysis has been calculated for the last quadrant of the second table in Figure 5.5, *LoD2-642-5648-2-TH*. The algorithmic pipeline is shown in Figure 5.6. The import of the quadrant produces one node with a number of *Segment3DNet* sub-nodes. Each sub-node represents a group. The content of the group is defined by the used decomposition type. The next step is to start the tetrahedralization for each *Segment3DNets* sub-node (see. Figure 5.6 in red). Each thread triangulates the closed (without boundary) segment complexes of its allocated *Segment3DNet* sub-node. The result is one *Triangle3DNet* sub-node. The final result of each thread is created by calling the *buildCores()* operation on the previously generated *Triangle3DNet* sub-node. The result is one *Tetrahedron3DNet* node whose tetrahedron complexes are added to a global *Tetrahedron3DNet* shared by every thread. The global *Tetrahedron3DNet* is the source

for the pipeline which creates the spatial results for the sensitivity analysis. A global *Tetrahedron3DNet* node is created by calling the *buildTONode3D()* on the root node, which creates a node with the *Tetrahedron3DNet* object as a spatial property.

The first step in the pipeline to create the results is to create each sub-node for each tetrahedron complex by calling *buildSubComponents()* on the *Tetrahedron3DNet* node (see last sentence of last paragraph). All sub-nodes (see. Figure 5.6 in yellow) are collected into one *SAM* to support the next step. A thread pool calls the *buildOverlay()* operation on each sub-node to create the interior overlays in parallel. The set of the resulting tetrahedral interior overlay nodes (see. Figure 5.6 in green) are the shared solid volumes of the groups, since the *buildCores()* operation called with each group returns distinct tetrahedralized tetrahedron complexes. The next step is to calculate each border sub-node of each sub-node in order to calculate the border interior overlay of the set of border sub-nodes in parallel just as the interior overlay calculation of the sub-nodes, previously. The set of the resulting border interior overlay sub-nodes (see. Figure 5.6 in blue) are the shared border areas, curves, and samples of the border sub-nodes. The shared border areas, curves, and samples include the inner border areas, curves, and samples of all groups, since a tetrahedralization of a group may result in an aggregation of several tetrahedron complexes (see last paragraph).

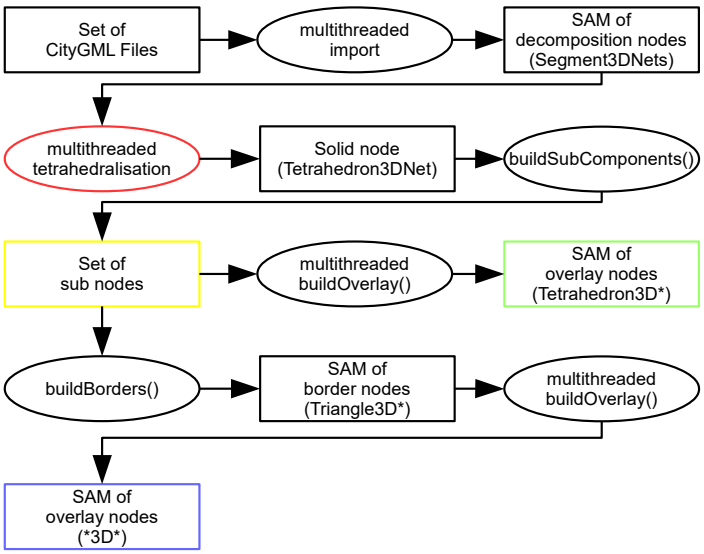


Figure 5.6: Pipeline to build one run of the sensitivity analysis

Figures 5.7 and 5.8 show the results of spatial analysis. Figures 5.9 and 5.10 show the results of the processing times analysis.

The light blue triangle is a reference point when collecting all “planar” polygons (represented as *BREPs*) of the input *CityGML* dataset into one aggregation, triangulate

the *Segment3DNet* object into a *Triangle3DNet* object and call the *buildCores()* operation of the node which is linked to the *Triangle3DNet* object. Parameters were set to $\epsilon = \varepsilon = \sigma = \lambda = 10^{-3}$ and $\zeta = 10^{-9}$. This first decomposition type (no decomposition) leads to distinct tetrahedron complexes, as mentioned before.

Figure 5.7 shows the results of the second decomposition type, which tetrahedralizes each building separately. If the building consists of multiple parts, all "planar" polygons (represented as *BREPs*) of the building are collected into one *Segment3DNet* object which is triangulated into one *Triangle3DNet* which will be tetrahedralized as one. As mentioned before, this strategy may lead to shared solid volumes between buildings only, since the *buildCores()* operation returns distinct tetrahedron complexes for each group. Therefore, if the building consists of multiple parts, they will be integrated into the whole building solid, which consists of distinct tetrahedron complexes.

Figure 5.8 shows the results for the third decomposition type, which separately tetrahedralizes each building or building part if the building consists of multiple parts. As mentioned before, this strategy may lead to shared solid volumes between buildings and/or building parts.

The top diagram of figures 5.7 and 5.8 shows the relative number of solid groups to the number of groups. Both decomposition types show nearly the same percentage results. To understand the diagram, it is important to know that a group counts into the number of tetrahedralized groups if at least one valid tetrahedron was calculated by the *buildCores()* operation of the group node. Precisions $\epsilon = \varepsilon = \epsilon = \lambda > 10^{-2}$ lead to more failures. The percentages rise with growing angular precision ζ and drops at 10^{-16}. The maximum is at 10^{-9}. The curves for each precision $\epsilon = \varepsilon = \epsilon = \lambda$ value are relatively equal in shape, but differ in their maximum. A value of $\epsilon = \varepsilon = \epsilon = \lambda = 10^{-2}$ and $\zeta = 10^{-9}$ shows the highest maximum at nearly 90% for the second decomposition type. As expected, this value is slightly higher than its counterpart in the third decomposition type. If one building part is not able to be tetrahedralized it will not count within the third decomposition type whereas a group of the second decomposition type, which contains all triangle complexes of the contained building parts, seem to have more chances to be tetrahedralized successfully. This contradicts the assumption that larger groups are more difficult to be tetrahedralized successfully. Having in mind, that a group is tetrahedralized successfully if at least on valid tetrahedron could be found, the assumption gets relativized.

The second diagram (left in second row) shows the overall volume of all tetrahedralized solids. The light blue reference (no decomposition) shows much less volume than its yellow counterpart at the same angular precision at both decomposition types. The shapes of the curves are relatively equal to the first diagram. Precisions $\epsilon = \varepsilon = \epsilon = \lambda > 10^{-2}$ lead to more failures. The second decomposition type shows generally smaller values than the third decomposition type. This may be caused by redundant volumes of overlaid building parts which are not part of the second decomposition type. It may also be caused by tetrahedralizations which have not been successfully calculated when using the second decomposition type and tetrahedralizations which have been successfully

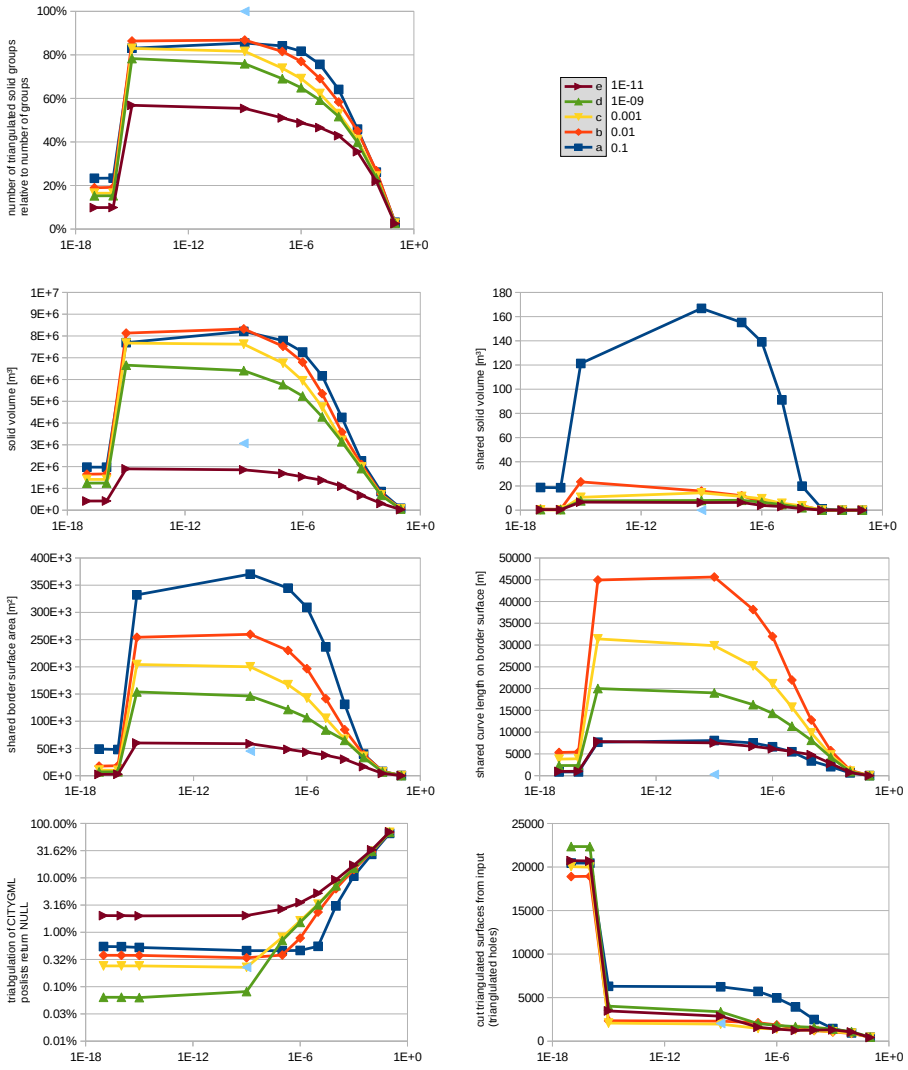


Figure 5.7: Results of city centre of Erfurt. The dataset is decomposed into several groups, where each group consists of one building together with its parts.
(x-axis: angular precision ζ, different colours for each precision $\epsilon = \varepsilon = \epsilon = \lambda$ value)

calculated when using the third decomposition type. The last two reasons are also the reasons why the shared solid volume shown in Figure 5.7 and 5.8 (right in second row) does not explain the differences. The value of the light blue counterpart affirms the assumption that larger groups lead to more failures in the tetrahedralization process, even if more groups are able to be tetrahedralized successfully.

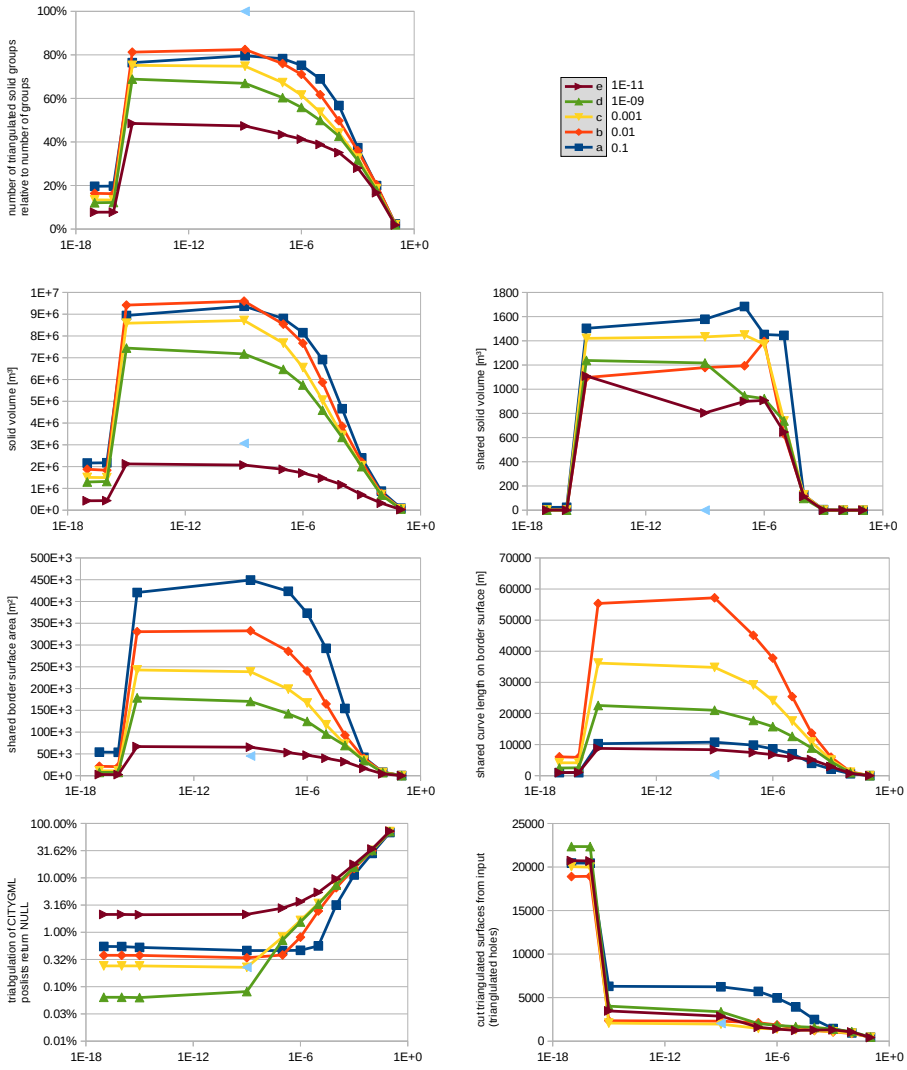


Figure 5.8: Results of city centre of Erfurt. The dataset is decomposed into several groups, where each group consists of one building or one building part if a building consists of several building parts.
(x-axis: angular precision ζ, different colours for each precision $\epsilon = \varepsilon = \epsilon = \lambda$ value)

The shared solid volume shown in Figure 5.7 (right in second row) of the second decomposition type is less than the shared solid volume of the third decomposition type shown in Figure 5.8 (right in second row). Where the second decomposition type shows 170 m^3 the third decomposition type shows values up to 1700 m^3. The third row illustrates the shared border areas and curves. Both are smaller in the second decomposition type, also.

These values are not comparable since it is unknown which solid volumes have been successfully tetrahedralized in both, either one or non decomposition type.

The last row shows how many “planar” polygons (represented as *BREPs*) of the input *CityGML* dataset were not triangulated through the pre-processing step before the *buildCores*() operation (left) and the number of rejected patches from the recursive call of the *buildCores*() are not part of the input of the *buildCores*() operation (right). As expected, the pre-processing step fails more often on larger angular precision ζ, since thinner triangles may be needed to triangulate a polygon which is represented as *BREP* whereas the number of rejected patches grows on smaller values of the angular precision ζ since the difference between a patch and the input may not be "Null" due to too high precision (less collinear, less parallel etc).

Figures 5.9 and 5.10 show the processing times of the first decomposition type using eight threads for Step 6 of Algorithm 2 (blue triangle in Figures 5.7 and 5.8) relative to the times taken by the second and respectively the third decomposition type using six threads to tetrahedralize the set of groups with only three threads for Step 6 of Algorithm 2 on an Intel® Core™ i7-7700K CPU @ 4.20GHz with eight virtual cores. The parallel processing times of Step 6 of Algorithm 2 were included in the overall processing time measurements.

It can be seen that the whole dataset is not processed six times faster than it would be expected when using six threads on the set of groups (either second or third decomposition type) against one thread for the whole dataset (first decomposition type) where the most valuable results happen. But larger groups seem to cause more tetrahedralization failures (see the solid volume diagram in combination with the shared solid volume diagram of Figures 5.9 and 5.10), which have a strong influence on the processing times, since it is unknown which part of Algorithm 2 fails and breaks its processing.

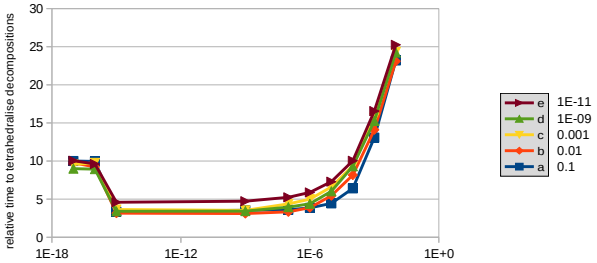


Figure 5.9: Time to tetrahedralize the whole *CityGML* dataset as one group with eight threads for the stitching of this group relative to the times with six threads for the set of groups and three threads for the stitching of each group, grouping each building with its parts into separate groups (x-axis: angular precision ζ, different colours for each precision $\epsilon = \varepsilon = \sigma = \lambda$ value).

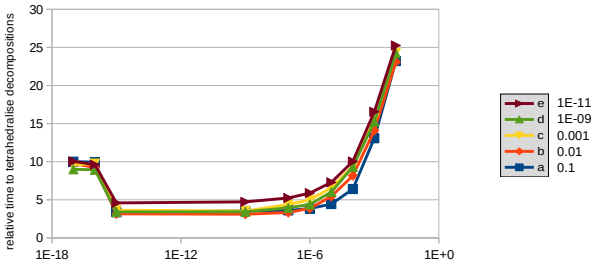


Figure 5.10: Time to tetrahedralize the whole *CityGML* dataset as one group with eight threads for the stitching of this group relative to the times with six threads for the set of groups and three threads for the stitching of each group of the city centre of Erfurt, grouping each building or building parts into separate groups (x-axis: angular precision ζ, different colours for each precision $\epsilon = \varepsilon = \sigma = \lambda$ value).

5.1.3 Distributions of combinatorial Euler characteristics of three different city model decompositions

The combinatorial Euler characteristics (see Definition 2.3.3 in Section 2.3.2) are calculated for three different decomposition types of a city model and their calculable topological inconsistencies in order to analyse the topological differences of those three decomposition types. The three different decomposition types are explained in Section 5.1.1 last paragraph. Two decomposition types are analysed in Section 5.1.2 with a sensitivity analysis of sensitive precision parameters.

As mentioned in Section 5.1.1 last paragraph: The first decomposition type tetrahedralizes all buildings in one step. The second decomposition type tetrahedralizes each building separately and if the building is a collection of building parts, all the building parts will be tetrahedralized in one step. The third decomposition type tetrahedralizes each building separately and if the building is a collection of building parts, the building parts will be tetrahedralized separately, also. Therefore, the first method leads to disjoint tetrahedron complexes which may share border objects. The second method leads to shared solid volumes between buildings, if the interiors of the tetrahedron complexes overlap. The third method leads also to shared solid volumes within buildings, if the different tetrahedron complex interiors of the building parts overlap.

I chose the values of $\epsilon = \varepsilon = \epsilon = \lambda = 10^{-2}$ and $\zeta = 10^{-9}$, which produced the highest number of successful tetrahedralizations to calculate the combinatorial Euler characteristics of tetrahedralized *Tetrahedron3DNet* object, the contained tetrahedron complexes, their interior overlays and their border interior overlays. The tetrahedralization method described in Section 5.1.1 was applied with the pre-processing step of splitting non-planar triangle complexes which were triangulated from the "planar" polygons (represented as *BREPs*) of the input *CityGML* dataset. The input *CityGML* dataset is the same as in Section 5.1.2.

The combinatorial Euler characteristics are calculated for each decomposition type presented as a distribution over the number of spatial objects which share the same combinatorial Euler characteristic. The first group of distributions are the distributions for the tetrahedralized tetrahedron complexes (see Section 5.1.2 Figure 5.6 yellow box) of all three decomposition types. The second group of distributions is the distributions of their topological sum (see Section 5.1.2 Figure 5.6 second row middle box) which has only one member, trivially. The spatial objects for the next two groups of distributions are created by the *buildOverlay(…)* operation described in Section 4.2.3 for each sub node which carries a tetrahedron complex of the tetrahedralized *Tetrahedron3DNet* object (see Section 5.1.2 Figure 5.6 pipeline yellow to green box). Where the first group of those two groups of distributions consists of the distributions for the tetrahedron complexes and the second group consists of the distributions for the *Tetrahedron3DNet* objects of the result from each *buildOverlay(…)* operation. Each tetrahedron complex of each *Tetrahedron3DNet* object also counts into the distributions of tetrahedron complexes. The spatial objects for the rest of the groups of distribution are created by the *buildBorder(…)*

and *buildOverlay(…)* operations described in Section 4.2.3 for each sub node which carries a tetrahedron complex border (see Section 5.1.2 Figure 5.6 pipeline yellow to blue box) grouped by their dimension. The groups of distributions for each dimension are also divided into two groups of distributions, by aggregation level *Component3D* and by *Net3D*, where each complex of each *Net3D* object also counts into the distributions of complexes.

As described in Section 2.3.2, the combinatorial Euler characteristic depends on the computational precision and may be wrong for the spatial database objects. The following presentation of the results is a matter of inaccuracy.

Tables 5.1 to 5.5 show the distributions. Table 5.1 shows the distributions of tetrahedron complexes of each decomposition type. Each tetrahedron complex seem not to be the border of a 4-dimensional sphere since there seem to be no tetrahedron complexes with an Euler characteristic of zero (see Section 2.3.2 Euler's polyhedron formula 2.3.1). Table 5.2 shows the distributions for the *Tetrahedron3DNet* objects of all decomposition types. Because each tetrahedron complex seem to have a combinatorial Euler characteristic of one (see Table 5.1), the combinatorial Euler characteristic could represent the number of tetrahedron complexes within the *Tetrahedron3DNet* objects.

Interior overlay of each tetrahedron complex resulting in tetrahedron complexes Table 5.3 shows the distributions of tetrahedron complexes created by the interior overlay of each tetrahedron complex. There seem to be no tetrahedron complexes found by the first decomposition type. The other decomposition types show that each tetrahedron complex, created by the interior overlay, seems not to be a border of a 4-dimensional sphere since there seem to be no tetrahedron complexes with a combinatorial Euler characteristic of zero (see Section 2.3.2 Euler's polyhedron formula 2.3.1). Table 5.4 shows the distribution of *Tetrahedron3DNet* objects of the second decomposition type, and Table 5.5 shows the distribution of *Tetrahedron3DNet* objects of the third decomposition type. Because every tetrahedron complex seems to show a combinatorial Euler characteristic of one (see Table 5.3), the combinatorial Euler characteristic could represent the number of tetrahedron complexes within the *Tetrahedron3DNet* objects. There are *Tetrahedron3DNet* objects with many tetrahedron complexes. Figure 5.11 shows one of those *Tetrahedron3DNet* objects. Some tetrahedrons resulting from the interior overlay are not valid (e.g. too sharp or too small) and get cut out of the final result. The result should be a tetrahedron complex but due to so many invalid tetrahedrons, the result is a *Tetrahedron3DNet* object with many tetrahedron complexes.

Border interior overlay of each tetrahedron complex resulting in triangle complexes Table 5.6 shows the first, second, and third decomposition type with the distributions of triangle complexes. The first decomposition type shows that there should not be any hulls (3-dimensional sphere) since there seem to be no triangle complexes with a combinatorial Euler characteristic of two (see Section 2.3.2 Euler's polyhedron formula 2.3.1).

The other two decomposition types show that most of the triangle complexes seem to have a combinatorial Euler characteristic of one. Some could shape hulls (3-dimensional sphere) which seem to show a combinatorial Euler characteristic of two (see Section 2.3.2 Euler's polyhedron formula 2.3.1). Table 5.7 shows the distribution of *Triangle3DNet* objects of the first decomposition type. Because it is known that the contained triangle complexes within each *Triangle3DNet* object seem to show a combinatorial Euler characteristic of one (see Table 5.6), the combinatorial Euler characteristics should represent the number of triangle complexes of the *Triangle3DNet* objects. Table 5.8 shows the distribution of *Triangle3DNet* objects of the second decomposition type, and Table 5.9 shows the distribution of *Triangle3DNet* objects of the third decomposition type. Because each of the contained triangle complexes within each *Triangle3DNet* object may not have a value of one (see Table 5.6), the combinatorial Euler characteristics should not represent the number of triangle complexes contained by each *Triangle3DNet* object. But *Triangle3DNet* object with higher Euler characteristics may indicate numerical errors due to double precision arithmetic.

Border interior overlay of each tetrahedron complex resulting in segment complexes Table 5.10 shows the distributions of segment complexes created by the border interior overlay of each tetrahedron complex of the first, second, and third decomposition type. The first decomposition type shows that there should not be any polygons (2-dimensional sphere) since there seem to be no segment complexes with a combinatorial Euler characteristic of zero (see Section 2.3.2 Euler's polyhedron formula 2.3.1). The other two decomposition types show nearly one quarter of segment complexes which seem to have a combinatorial Euler characteristic of zero, the other two quarters seem to have a combinatorial Euler characteristic of one. The first decomposition type consists of only one *Segment3DNet* object. Because it is known that the contained segment complexes within the *Segment3DNet* object seem to show a combinatorial Euler characteristic of one (see Table 5.10), the combinatorial Euler characteristics should represent the number of segment complexes contained by this *Segment3DNet* object. Table 5.11 shows the distribution of *Segment3DNet* objects of the second decomposition type, and Table 5.12 shows the distribution of *Segment3DNet* objects of the third decomposition type. Because there seems to be one quarter with a combinatorial Euler characteristic of zero and three quarters with a combinatorial Euler characteristic of one (see Table 5.10), the combinatorial Euler characteristics most likely underestimates the number of segment complexes within the *Segment3DNet* objects.

Border interior overlay of each tetrahedron complex resulting in point complexes Table 5.13 shows the distribution of point complexes created by the border interior overlay of each tetrahedron complex of the first decomposition type. Only one point intersection exists within the first decomposition type. Table 5.14 shows the distribution of point complexes of the second decomposition type, and Table 5.15 shows the distribution of point complexes of the third decomposition type. The border of the tetrahedron

complexes touch each other with a peak of 24, respectively 90, times. This may result from the failures in creating the overlay, as seen in Figure 5.11.

Euler	1		1		1
Count	3212		6703		7660

Table 5.1: First, second and third decomposition type (from left to right) - distributions of combinatorial Euler characteristics of tetrahedron complexes

Euler	3212		6703		7660
Count	1		1		1

Table 5.2: First, second and third decomposition type (from left to right) - distributions of combinatorial Euler characteristics of *Tetrahedron3DNet* objects

Euler	-		1		1
Count	-		518		2476

Table 5.3: First, second and third decomposition type (from left to right) - distributions of combinatorial Euler characteristics of tetrahedron complexes from interior overlay

Euler	2	3	4	5	7	8	9	10	14	16	17	18	23
Count	55	9	2	1	1	1	1	1	1	1	1	1	1

Table 5.4: Second decomposition type - distribution of combinatorial Euler characteristics of *Tetrahedron3DNet* objects from interior overlay

Euler	2	3	4	5	6	7	8	9	10	11
Count	69	17	3	7	2	6	4	2	3	2
Euler	12	14	15	16	17	18	20	23	24	27
Count	1	3	2	1	2	1	3	1	1	1
Euler	28	30	36	37	40	41	42	45	105	1088
Count	1	1	1	1	1	1	1	1	1	1

Table 5.5: Third decomposition type - distribution of combinatorial Euler characteristics of *Tetrahedron3DNet* objects from interior overlay

Euler	1		0	1	2		-1	0	1	2
Count	843		8	6040	6		1	10	7418	4

Table 5.6: First, second and third decomposition type (from left to right) - distributions of combinatorial Euler characteristics of triangle complexes from border interior overlay

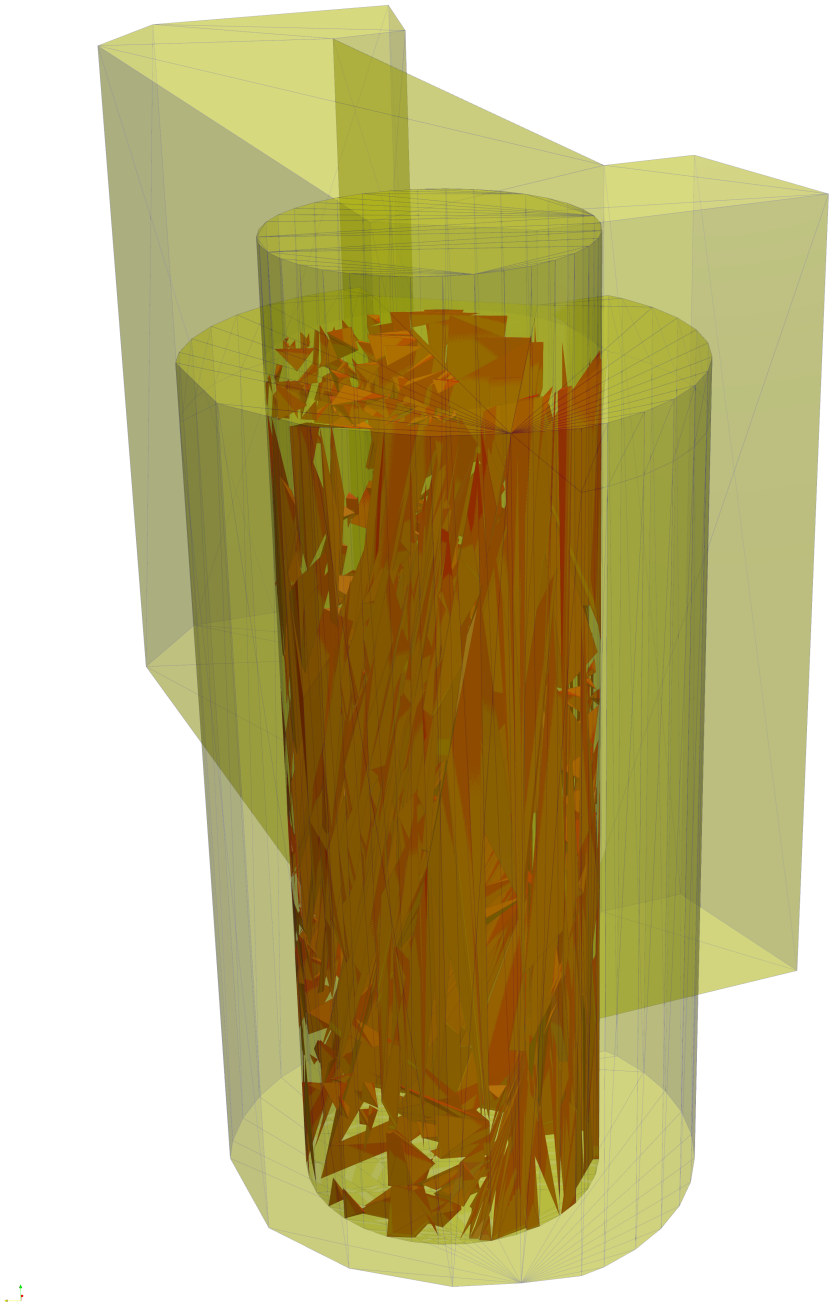


Figure 5.11: Result of an interior overlay as a *Tetrahedron3DNet* object (red) of two building parts tetrahedralized to two tetrahedron complexes (yellow) in the shape of cylinders. This is an example of a *Tetrahedron3DNet* object with many tetrahedron complexes of Table 5.5.

Euler	1	2
Count	2	1

Table 5.7: First decomposition type - distribution of combinatorial Euler characteristics of *Triangle3DNet* objects from border interior overlay

Euler	2	3	4	5	6	7	10
Count	498	96	20	13	1	1	1

Table 5.8: Second decomposition type - distribution of combinatorial Euler characteristics of *Triangle3DNet* objects from border interior overlay

Euler	2	3	4	5	6	7	9	11	13	16
Count	608	126	25	13	3	1	1	1	1	1

Table 5.9: Third decomposition type - distribution of combinatorial Euler characteristics of *Triangle3DNet* objects from border interior overlay

Euler	1		0	1		0	1
Count	49		687	1841		842	2384

Table 5.10: First, second and third decomposition type (from left to right) - distributions of combinatorial Euler characteristics of segment complexes from border interior overlay

Euler	1	2	3	4	5
Count	30	265	47	4	2

Table 5.11: Second decomposition type - distribution of combinatorial Euler characteristics of *Segment3DNet* objects from border interior overlay

Euler	0	1	2	3	4	5	6	7
Count	1	39	348	60	12	2	2	1

Table 5.12: Third decomposition type - distribution of combinatorial Euler characteristics of *Segment3DNet* objects from border interior overlay

Euler	1
Count	1

Table 5.13: First decomposition type - distribution of combinatorial Euler characteristics of point complexes from border interior overlay

Putting it together The presented analysis can help to determine which type of overlays exists when calculating interior overlays grouped by their dimension as far as the combinatorial Euler characteristic makes statements about the type of the spatial object. The presented analysis did conduct the classification by using the Euler's polyhedron

Euler	1	2	3	4	5	6	7	8	9	10	11
Count	57	42	51	18	11	9	10	7	6	1	1
Euler	12	13	14	15	16	17	18	20	21	22	24
Count	6	1	1	2	2	3	3	1	1	1	3

Table 5.14: Second decomposition type - distribution of combinatorial Euler characteristics of point complexes from border interior overlay

Euler	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Count	85	80	74	38	22	22	16	20	12	25	5	13	5	5	
Euler	15	16	17	18	19	20	21	22	23	24	25	26	33	34	90
Count	3	7	4	13	2	1	1	1	1	1	1	1	1	1	2

Table 5.15: Third decomposition type - distribution of combinatorial Euler characteristics of point complexes from border interior overlay

formula 2.3.1 described in Section 2.3.2. But there are more methods of classifications derived from the combinatorial Euler characteristics. As an example, if d-dimensional simplicial complexes seem to have an Euler characteristic of one, the visual comparison often shows d-dimension simplicial complex with a boundary, just like typical d-dimensional manifolds with boundary. Another example would be given by introducing the *genus g*, which indicates the number of tunnels 2-dimensional handle-bodies, also called g-holed tori (see [27] and [36]). However, further investigations would go beyond the scope of this work.

The presented analysis may also be used to determine failures by introducing an upper bound of possible intersections. As an example, all interior overlays of each tetrahedron complex resulting in tetrahedron complexes could always be considered as failures or bad modelling, since the overlapping of buildings is physically only possible if they share solid wall (see Table 5.3). If this situation is not thought to be bad modelling, it may help to retrieve failures by taking a closer look at the *Tetrahedron3DNet* objects resulting from the interior overlay, which show a combinatorial Euler characteristic higher than 2. Those *Tetrahedron3DNet* objects may indicate more than one shared tetrahedron complex, as shown in Figure 5.11.

5.2 p-adic scalable space filling curve index

This section is published within *The Computer Journal* [8] beforehand:

Experiments were done for some p-adic cases with projections of a data set containing information about forest trees in Barro Colorado Island (Panama) to which we will refer to as the *forest dataset*. Forest growing modelling deals with models which depend on local neighbourhood relationships (e.g. shadow casting of taller plants onto the lower forest regions). This dataset could in principle be enriched with other data from which e.g. sunlight exposure or local soil moisture could be inferred for each tree in this dataset. We experimented with attributes which do not imply an obvious topology for testing reasons.

The forest data set represents tree-specific information taken from ca. 938,000 trees in Barro Colorado Island (Panama) in a series of 7 censuses, beginning in 1982, in an area of 50 hectares [28, 17, 29]. There are 18 attributes, from which are chosen for the indexing methods 8 attributes, which seem interesting from a geo-information point of view. These are namely the following:

- x-coordinate x (60,035 different values)
- y-coordinate y (47,099 different values)
- date of measurement date (2,328 different values)
- diameter at breast height DBH (1,823 different values)
- height-of-measure hom (286 different values)
- latin name of species latin (325 different values)
- tag number of the individual stem stemtag (152 different values)
- tag number used in the field tag (423,617 different values)

The non-numerical attributes, like latin (names of the species), were transformed to integer numbers by assigning to each different string a different integer number while keeping the interval as small as possible. Every value which could not be read due to data corruption was assigned to one close NULL-value near the data range.

Thus, the data set used here consists of 2.45 million data points in a 8-dimensional space, with only 2.42 million different data points. There are 3 million missing coordinate values spread over all data points in the eight-dimensional space. There are only 1.87 million different data points with 0.37 million missing coordinate values spread over the data points in the case of the projection to the coordinates x, y, and DBH. Whereas in [63], sphere packings were used, here we will compare the performance of space-filling curve indices with respect to the influence of certain coordinates on the resulting tree structures.

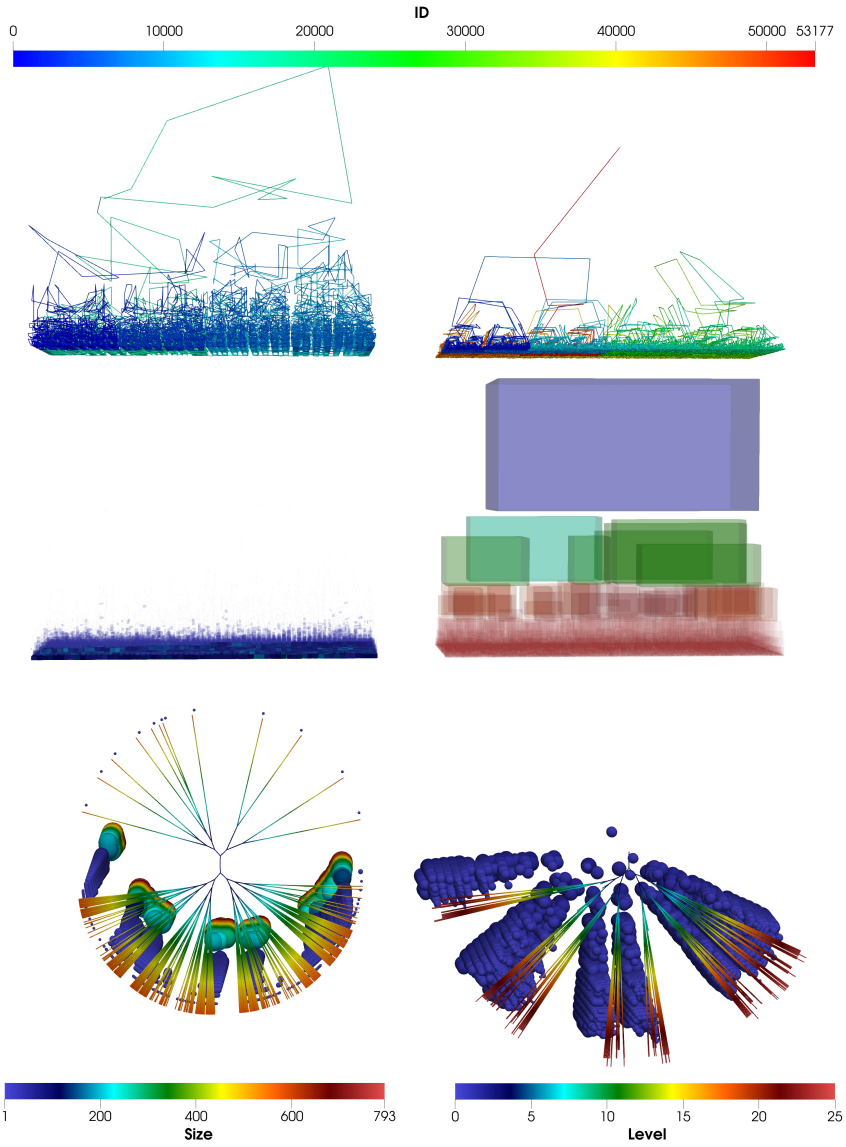


Figure 5.12: Forest dataset, attribute set x, y and DBH, $p = 2$.

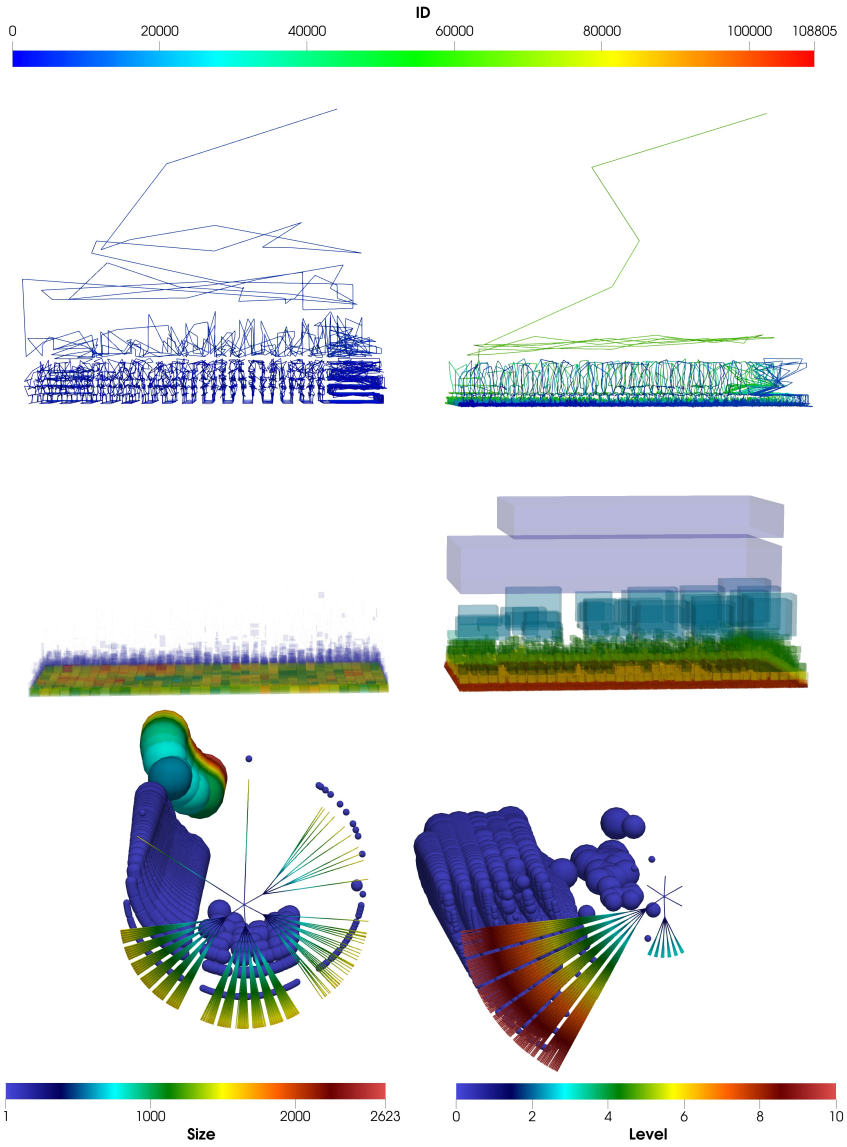


Figure 5.13: Forest dataset, attribute set x, y and DBH, $p = 5$.

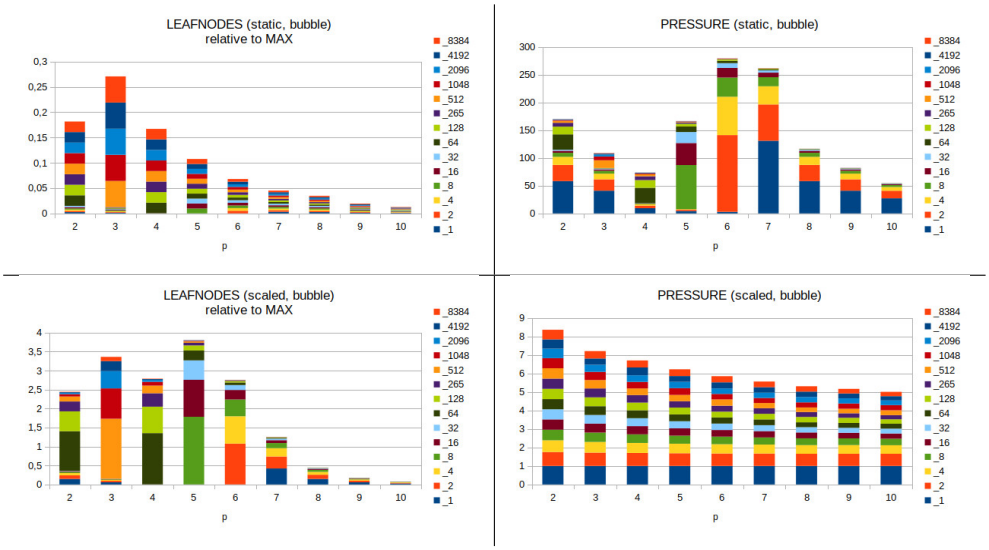


Figure 5.14: p-Adic Gray-Hilbert curves with different bucket capacity s. Results for the different the bucket capacities have been summed up for each p-value. Left: number of non-empty leaf nodes relative to the maximal number of leaf nodes; Right: number of points (constant) relative to number of non-empty leaf nodes times the bucket capacity s; Top: static curves with fixed levels; Bottom: scaled curves with dynamic levels.

Figures 5.12 and 5.13 show plots for the attribute set DBH from the forest dataset. Each left column corresponds to the static index, and each right column to the scaled index. The first row represents the ordered curve from start (blue) to end (red). The second row shows the cuboids corresponding to the leaf nodes. The third row gives the trees, in which the colouring corresponds to the size (left) or the level (right), and the size of the balls to the number of points contained in leaf nodes.

Figure 5.12: The iteration number for $p = 2$ is one higher than that for $p = 5$. The scaled tree seems more balanced than for $p = 5$. Choosing a lower value of p must be compensated for by a higher iteration number in order to obtain a similar filling degree.

Figure 5.13: As the diameter at breast height is not uniformly distributed, some leaf nodes in the static tree are heavily overfilled, and the scaled tree is heavily imbalanced. On the one hand, the static curve has fewer vertices than the scaled curve, but on the other hand, it is highly overfilled.

Figure 5.14: The number of iterations for the different static curves at the top have been calculated to take the different bucket capacities into account. Hence, for uniformly distributed points and a certain bucket capacity s, the minimal number of leaf nodes can be calculated by the number of points divided by the bucket capacity s. With the minimal number of leaf nodes, we are able to calculate the minimal number of iterations

we need to cover the minimal number of leaf nodes. With this number of minimal iterations we can calculate the level of the static tree where we put all the leaf nodes, and we can calculate the maximal number of leaf nodes a tree should have when dealing with uniformly distributed points under the constraint of not exceeding a certain bucket capacity s. It is obvious that the static version uses just a small set of leaf nodes relative to the maximal number of leaf nodes (top, left) but the *pressure*, defined as:

$$\text{pressure} = \frac{|S| \cdot s}{\#\text{ non-empty leaf nodes}}$$

is relatively high (top, right). This can be explained by the distribution of the points. The point data used for this figure is in dimension 8 (x, y, DBH, date, latin, tag, stemtag, hom). Most of these variables are not uniformly distributed. This generates a higher pressure where the density of the point cloud exceeds a certain density (number of points divided by maximal number of leaf nodes times the bucket capacity s). To reach that density in those regions, we need to create smaller sub-hyper-cuboids there. It also generates lower pressure where the density of the point cloud is lower than a certain density because we need to fuse them in order to create more pressure. This compensation can be recognized by comparing the pressures of the static versions (top, right) to the number of leaf nodes relative to the maximal number of leaf nodes of the scaled versions (bottom, left). The resulting pressure compensation (bottom, right) shows a decrease when rising value p, which indicates a better distribution of the points across the leaf nodes. This effect is stronger with higher bucket capacities because more dense point clusters get ignored if the points are able to fall into the same bucket/leaf node.

5.3 Topological access method

The topological access method is exemplary calculated for a tetrahedralized city model of a historical part of Salvador (Brazil) which had been modelled by the working group of Arivaldo Leão de Amorim. The data was produced by the Laboratory of advanced studies on City Architecture and Digital technologies (LCAD) at Federal University of Bahia (UFBA) during the research project "Establishing requirements for City Information Modelling", supported by the Brazilian federal government agency *Coordination for the Improvement of Higher Education Personnel* (CAPES) and the *German Academic Exchange Service* (DAAD). The used city model for this work is not a final version. It was used within a workshop in Salvador in the year 2019 as part of the exchange program. The key data is summarized in the following:

1. zero: 462.851043701172 376.195983886719 71.6225051879883
2. *CityGML* tree nodes: 22386
3. number of groups: 56
4. polygons: 925
5. distinct polygons: 924
6. decimal places: 15
7. min. point distance [m]: 6.216820473225959E-6
8. segments: 4446
9. min. length [m]: 3.0517578125E-5
10. average length [m]: 5.0552051851945565
11. max. length [m]: 32.06260853268529

The city model of Salvador shows more accuracy than the city model of Erfurt, with maximal 15 decimal places. The sensitivity analysis as described in Section 5.1.2 showed promising results for the second decomposition type with precision values $\epsilon = \varepsilon = \epsilon = \lambda = 10^{-2}$ and $\zeta = 10^{-14}$ even if some corner points of some polygons were ignored during import. 50 out of 56 buildings could be tetrahedralized with one tetrahedron complex per building and an overall volume of 68916 m^3 (see Figure 5.20). The distributions of the Euler characteristics emphasis the choice of the precision parameters. There are 8 tetrahedron complex created by the interior overlay (see Tables 5.16 left). Only one *Tetrahedron3DNet* object was created with 5 tetrahedron complexes (see Tables 5.16 right). This *Tetrahedron3DNet* object needs to be observed for further investigations. There are 53 triangle complexes found by the border interior overlay. One of which shows combinatorial Euler characteristic equal to zero, the others equal to one (see Tables 5.17 left). There seems to be no 2-dimensional sphere involved, since a combinatorial Euler characteristic equal to two is not contained. Seven *Triangle3DNet* objects consist of

minimum two triangle complexes, two *Triangle3DNet* objects consist of minimum three triangle complexes and only one consists of minimum four triangle complexes (see Tables 5.17 right). Those *Triangle3DNet* objects need to be observed for further investigations. The border interior overlays resulting in one and zero dimensional intersections are provided in tables 5.18 and 5.19.

Euler	1		5
Count	8		1

Table 5.16: Second decomposition type - distributions of combinatorial Euler characteristics of tetrahedron complexes (left) and *Tetrahedron3DNet* objects (right) from interior overlay

Euler	0	1		2	3	4
Count	1	52		7	2	1

Table 5.17: Second decomposition type - distributions of combinatorial Euler characteristics of triangle complexes (left) and *Triangle3DNet* objects (right) from border interior overlay

Euler	1		2
Count	21		4

Table 5.18: Second decomposition type - distributions of combinatorial Euler characteristics of segment complexes (left) and *Segment3DNet* objects (right) from border interior overlay

Euler	3	4	6	7	8	12	15	19
Count	1	1	2	2	2	2	1	1

Table 5.19: Second decomposition type - distribution of combinatorial Euler characteristics of point complexes from border interior overlay

Figure 5.15 shows the *CityGML* tree exported by VTK-T_0-Space-Exporter (radial) described in Section 4.8.2. Figure 5.16 shows the "planar" polygons (represented as *BREPs* by *CityGML*) as *Segment3DNet* object, coloured by group / building. Figure 5.17 shows the *CityGML* tree and the connected graph nodes which carry as spatial properties those "planar" polygons (represented as *BREPs*) as *Segment3DComponent* objects, the levelled out *Segment3DElement* objects or levelled out *Point3DElement* objects and their groups as *Segment3DNet* objects of the closed (without boundary) *Segment3DComponent* objects.

Figure 5.18 shows the result of the triangulation to "planar" polygons. Figure 5.19 shows those *Triangle3DNet* objects consisting of those *Triangle3DComponent* objects as wall, roof, or floor surfaces within the graph where each *Segment3DNet* object is a generalization of one *Triangle3DNet* object. The incidence relations between each polygon and its boundary is not included for clarity. Each *Triangle3DNet* object is tetrahedralized by Algorithm 2 separately.

Figure 5.20 shows the tetrahedralized results. Figure 5.21 shows the corresponding graph. The *Triangle3DNet* objects consisting of *Triangle3DComponent* objects as wall, roof, or floor surfaces are connected to their tetrahedralized *Tetrahedron3DComponent* objects. This figure also illustrates the border *Triangle3DComponent* objects of the *Tetrahedron3DComponent* objects. The aggregation relations between the wall, roof, and floor surfaces to the boundaries of the *Tetrahedron3DComponent* objects are omitted for clarity.

Figure 5.22 shows the overlays of the tetrahedron complexes as described in Section 5.1.2 Figure 5.6. Figure 5.23 shows the corresponding graph.

Figures 5.24, 5.25 and 4.44 (in Section 4.8.2 last subsection on VTK-TOAM-Exporter) show 3-dimensional projections of the 8-dimensional space spanned by the topological access method. The Section 4.7.6 on topological access methods describes seven different coordinate types. The first coordinate is the sum of distances. The definition of the other coordinates is to count the number of relations per relation type. This example adds one more coordinate type, which is the node ID. However, the discrete *R*-Tree* consists of seven hyper-cuboid levels to be able to manage all the 26317 nodes (from *CityGML*, tetrahedralization and overlay). This leads to queries of the form: Retrieve all nodes in the ID range of 0 to 26317 which have a sum of distances from 10 to 20, are a composition of 0 to 10 nodes, the border of 2 nodes and the generalization of 1 node. Figures 5.24, 5.25 and 4.44 (in Section 4.8.2 last subsection on VTK-TOAM-Exporter) show that the graph is compressed a lot in certain query ranges due to the less variety of the number of nodes per relation type of each node. The *Point3DElement* objects (red head and red body) in Figure 5.24 (right) in the lower right are “planar” polygons represented as *BREPs* by *CityGML* which had been shrunk to points while import. The most right *Point3DElement* object is a specialization of five different *CityGML* nodes.

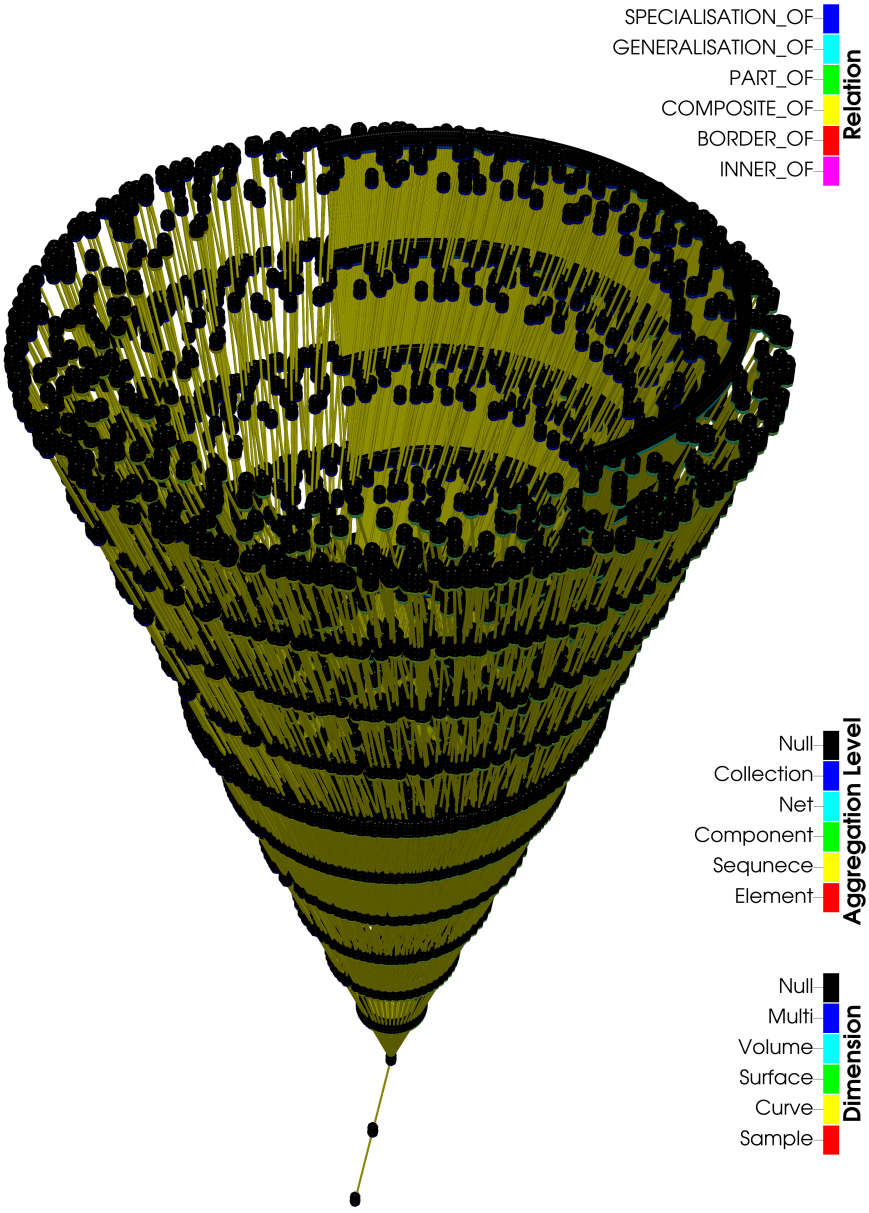


Figure 5.15: Salvador's historical city centre - *CityGML* tree (by VTK-T_0-Space-Exporter - radial)

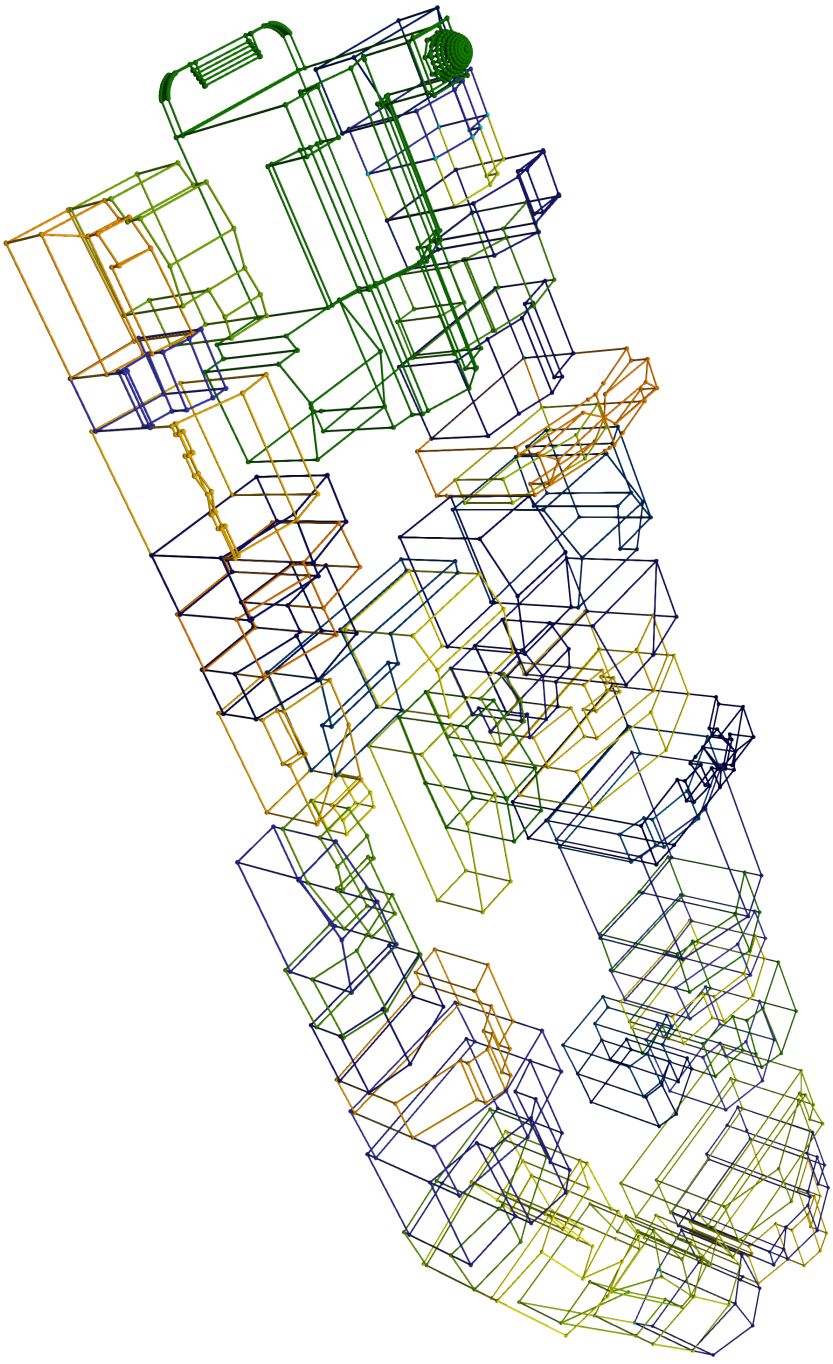


Figure 5.16: Salvador's historical city centre - "planar" polygons (represented as *BREP*s by *CityGML*) as *Segment3DNet* object coloured by group / building

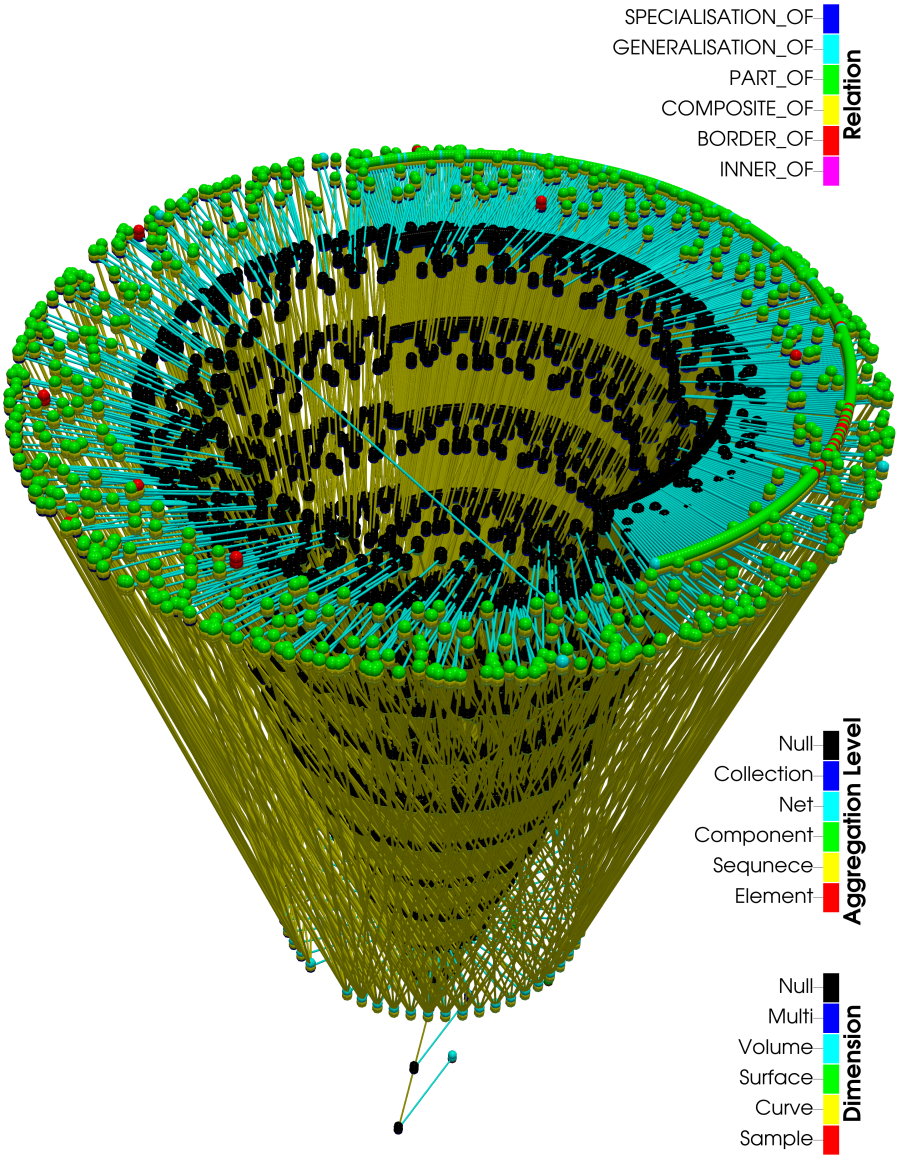


Figure 5.17: Salvador's historical city centre - *CityGML* tree with "planar" polygons (represented as *BREPs*) as *Segment3DComponent* objects (green head and yellow body), levelled out *Segment3DElement* objects (red head and yellow body) or levelled out *Point3DElement* objects (red head and red body) and their groups (second decomposition type) as *Segment3DNet* objects (cyan head and yellow body, lower outer ring) of closed (without boundary) *Segment3DComponent* objects (green head and yellow body).

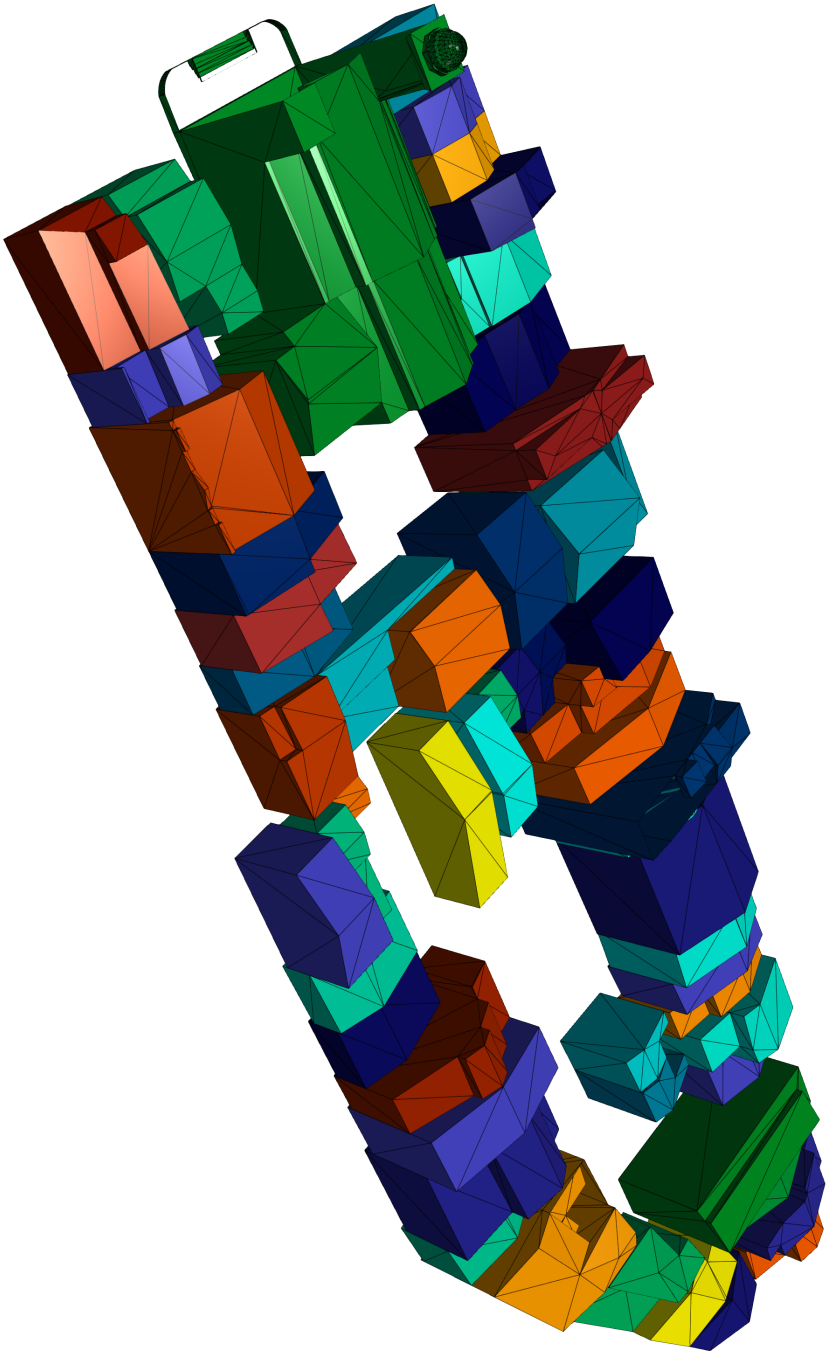


Figure 5.18: Salvador's historical city centre - "planar" polygons collected into *Triangle3DNet* objects coloured by their group ID

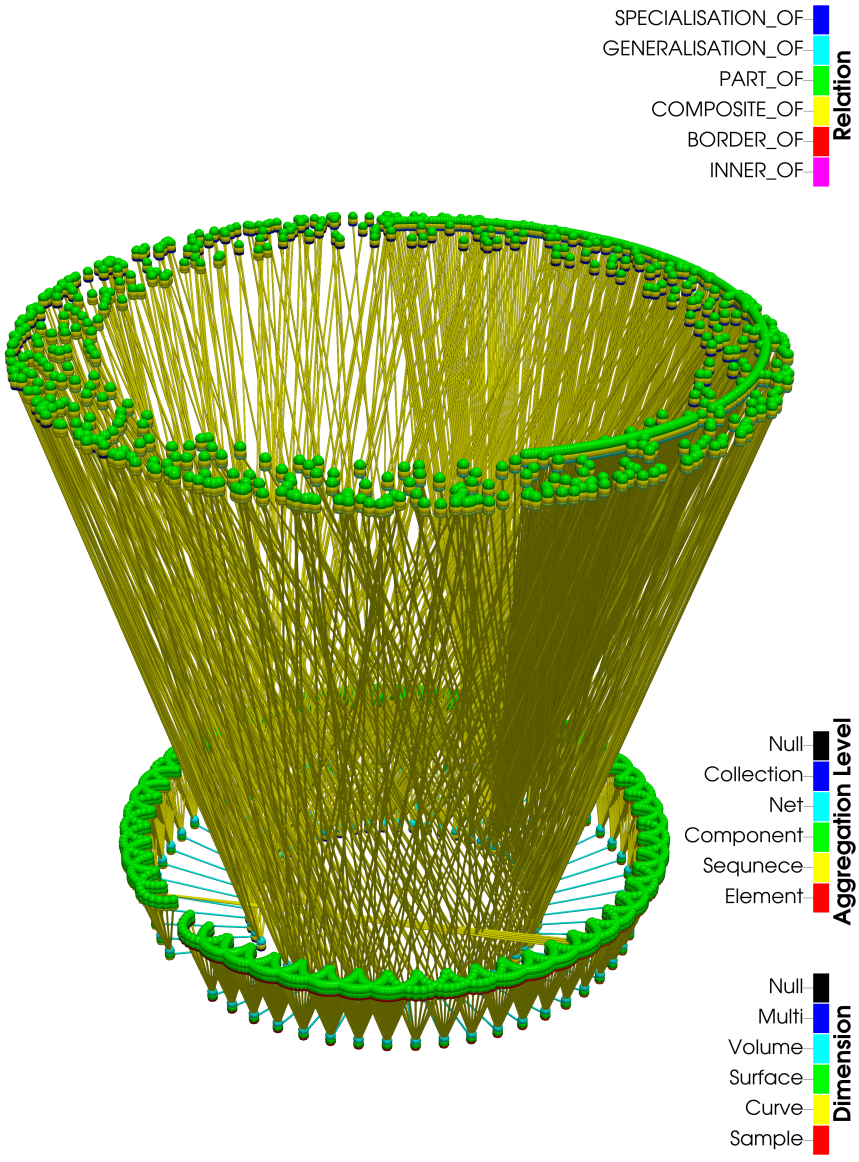


Figure 5.19: Salvador's historical city centre - Groups (second decomposition type) as *Segment3DNet* objects (cyan head and yellow body, lower outer ring) of closed (without boundary) *Segment3DComponent* objects (green head and yellow body) together with their triangulations to *Triangle3DNet* objects (cyan head and green body) consisting of *Triangle3DComponent* objects (green head and green body) as wall, roof, or floor surfaces.

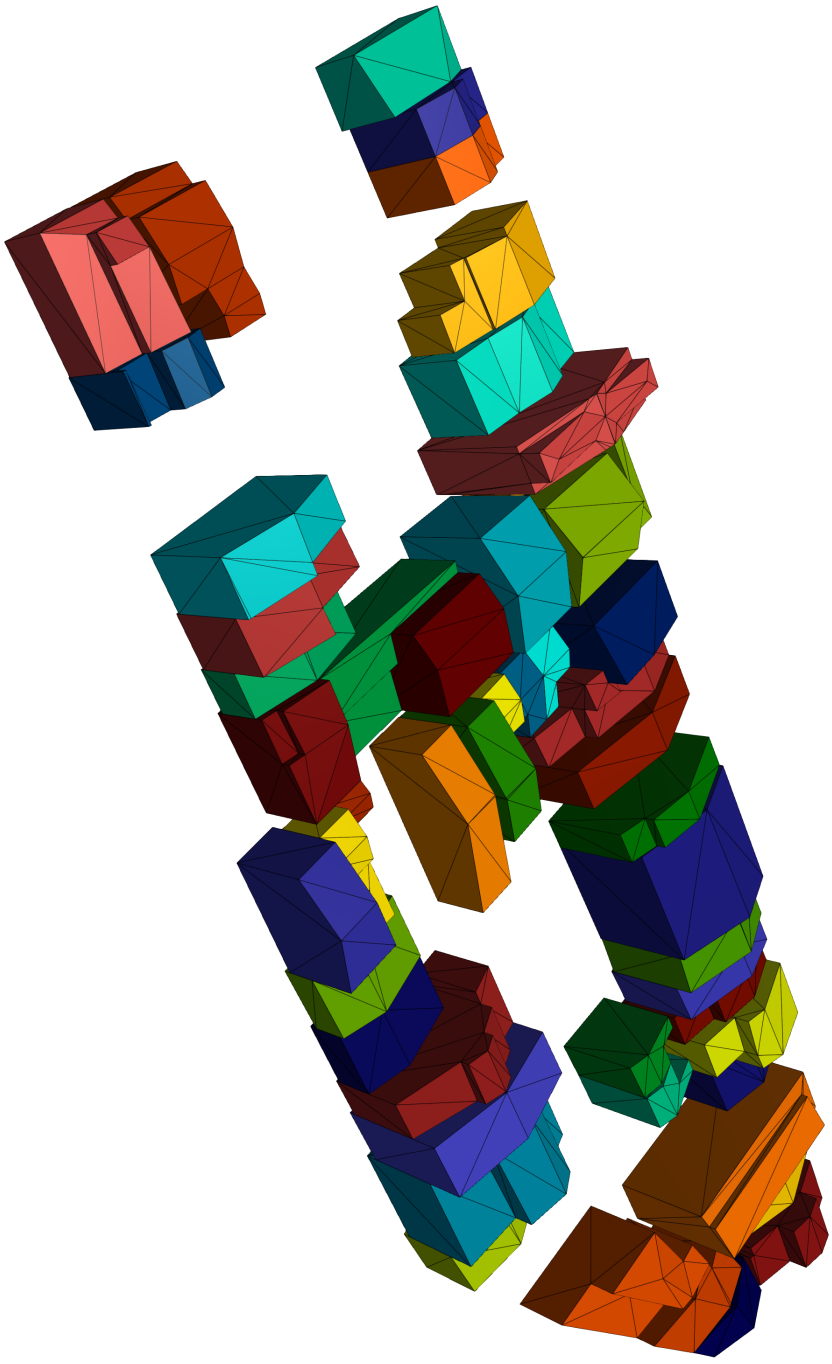


Figure 5.20: Salvador's historical city centre - tetrahedralized buildings as *Tetrahedron3DNet* coloured by tetrahedron complex ID

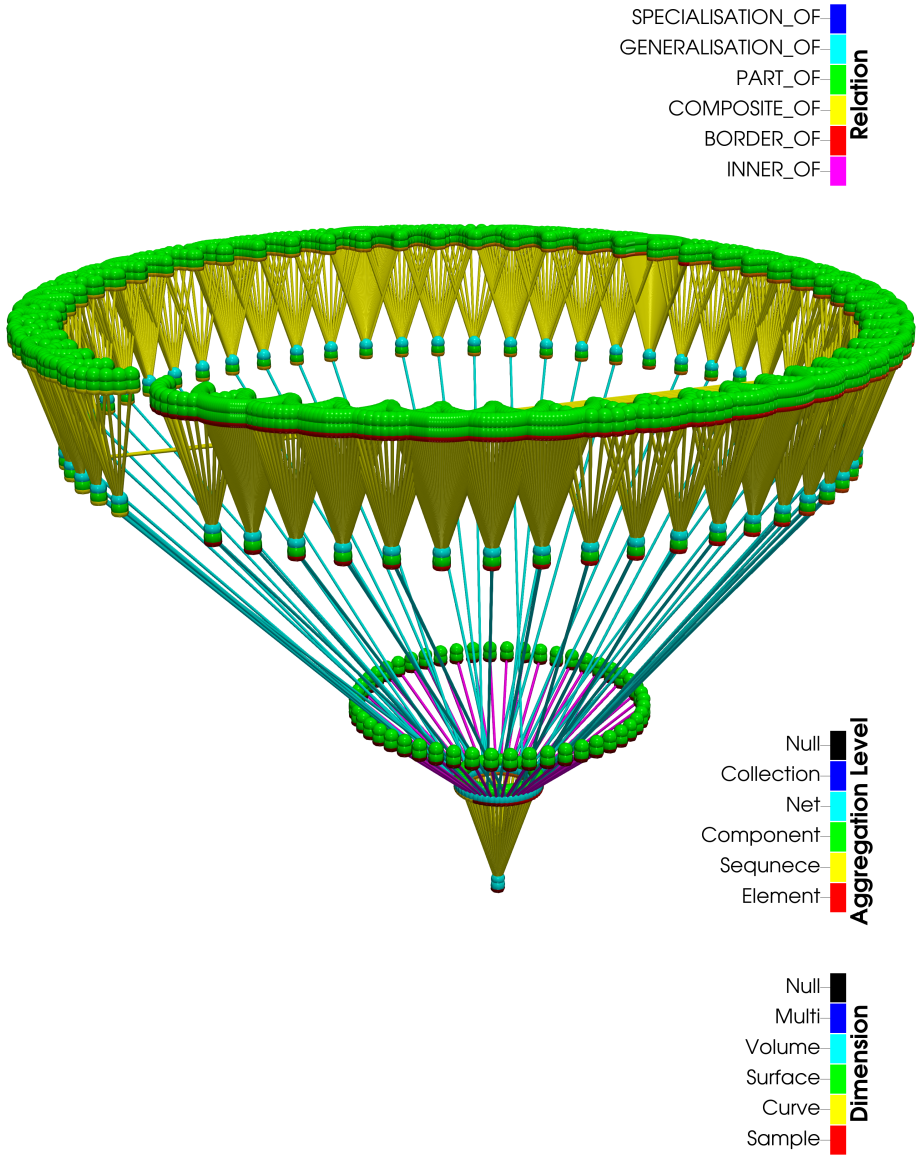


Figure 5.21: Salvador's historical city centre - *Triangle3DNet* objects (cyan head and green body) consisting of *Triangle3DComponent* objects (green head and green body) as wall, roof, or floor surfaces (top ring), their triangulations to one *Tetrahedron3DNet* object (cyan head and cyan body, bottom centre) consisting of *Tetrahedron3DComponent* objects (green head and cyan body) and their borders as closed (without boundary) *Triangle3DComponent* objects (green head and green body, lower outer ring).

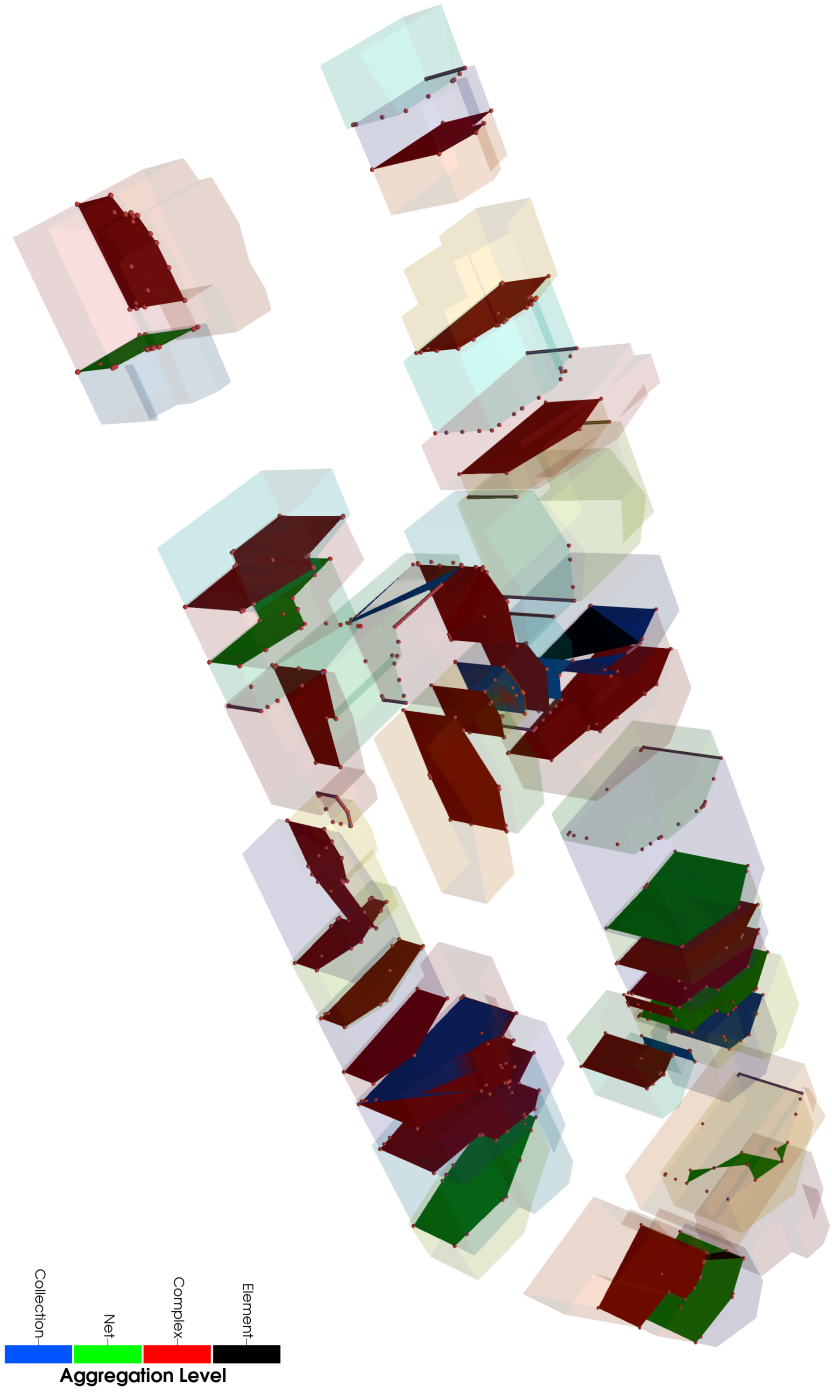


Figure 5.22: Salvador's historical city centre - overlays of tetrahedron complexes coloured by aggregation level

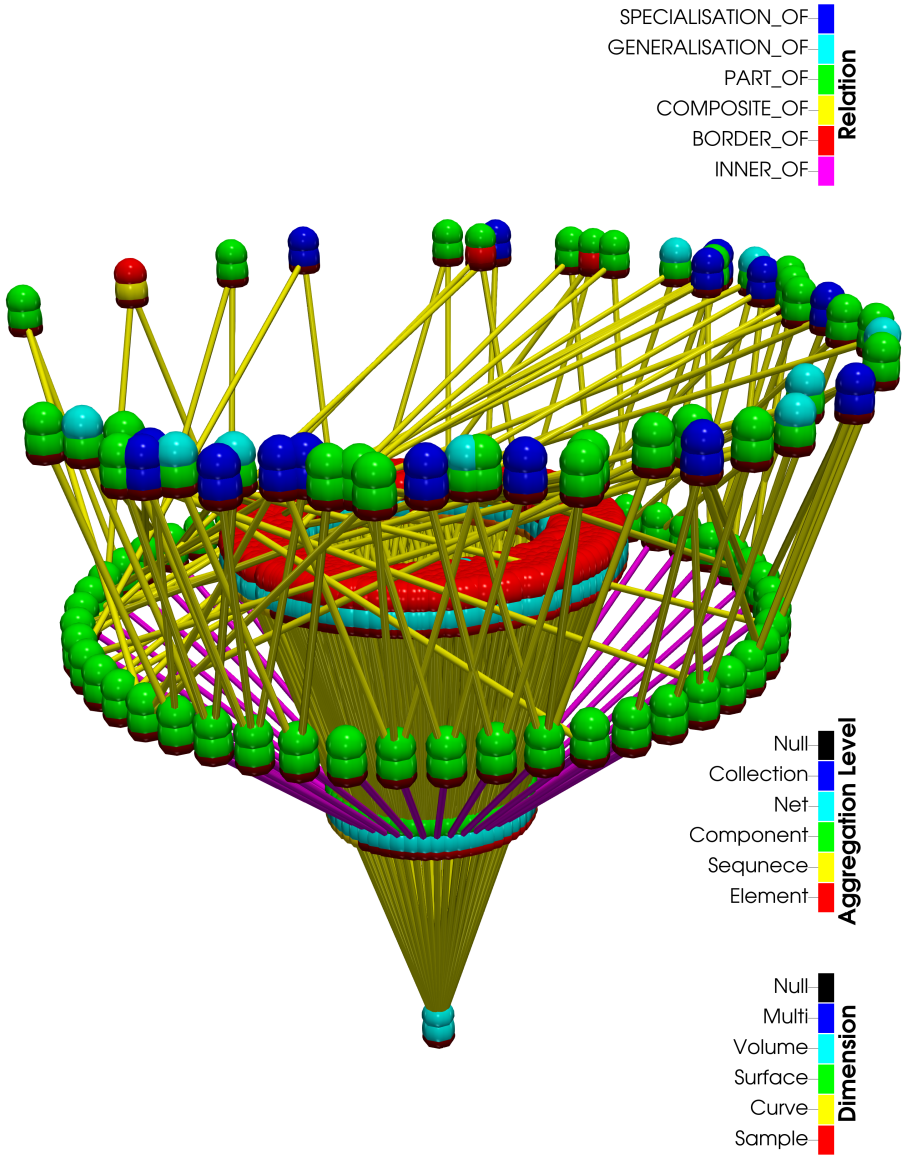


Figure 5.23: Salvador's historical city center - *Tetrahedron3DNet* object (cyan head and cyan body, bottom centre) consisting of *Tetrahedron3DComponent* objects (green head and cyan body, bottom ring) which consists of *Tetrahedron3DElement* objects (red head and cyan body, inner ring). The borders of the *Tetrahedron3DComponent* objects (green head and cyan body, bottom ring) as closed (without boundary) *Triangle3DComponent* objects (green head and green body, top outer ring) and the inner overlays (top ring).

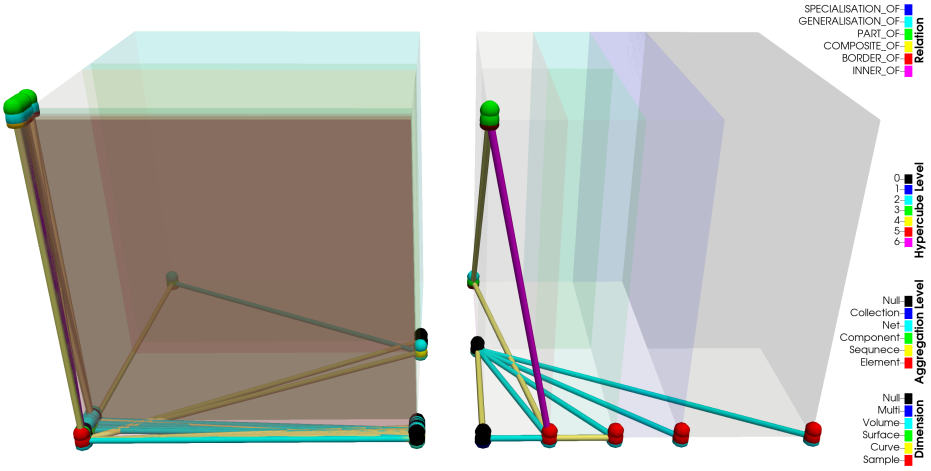


Figure 5.24: Salvador's historical city centre - Graph bend by topological access method as described in Section 4.7.6 where the x-axis represents the number of nodes from which a given node is a generalization (left) or specialization (right), the y-axis represents the number of nodes from which a given node is a composition (left) or a part (right) and the z-axis represents the number of nodes from which a given node is the inner (left) or a border (right). The colours of the hyper-cuboids overlay each other.

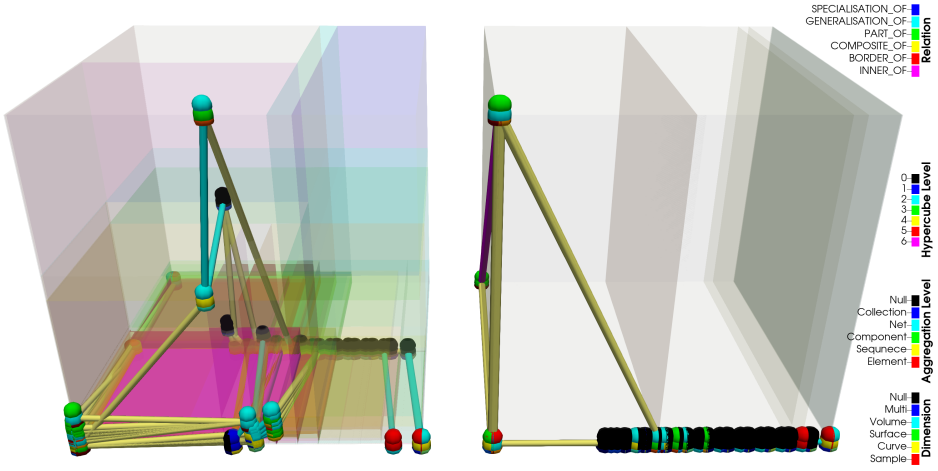


Figure 5.25: Salvador's historical city centre - Graph bend by topological access method as described in Section 4.7.6 where the x-axis represents the sum of distances of a given node (left and right), the y-axis represents the number of nodes from which a given node is a part (left) or a border (right) and the z-axis represents the number of nodes from which a given node is a composition (left) or the inner (right). The colours of the hyper-cuboids overlay each other.

6 Discussion

A scientific data model to use for interdisciplinary communication which involves spatial or spatio-temporal data is provided. The data model is ISO conform in terms of OGC's *General-Feature-Model* and *Simple-Feature-Access* specifications. This allows complex linking of spatial and spatio-temporal data, since each feature is relatable to other features. The proposed basic relation types (*part-of*, *border-of* and *generalization-of*) and their inverse (e.g. *composite-of*, *inner-of* and *specialization-of*) are able to model aggregations, geometrically induced topology and abstractions. The combination of those T_0-spaces lead to a complex graph structure. This abstract graph structure is easy to be analysed, as a cause of the limited relation types and the simple meaning behind those types.

Graph algorithms are able to use combinations of the spanned T_0-spaces in order to solve special analytics in parallel, if the graph is distributed by any graph distribution paradigm. It is also possible to define access methods for a set of graph nodes (features) which allows a spatial or spatio-temporal management, and the provided topological access method is a first step to support data management by topological properties. The topological properties are especially interesting for an ETL (extract transform load) process which supports simulation pre-processing. As such, the proposed implementation can be used for the ETL processes on data source or data sink side. The provided concept is supported by the implementation by that meaning. Any further back end or front end implementations besides the provided importers and exporters would have gone beyond the scope of this work. Besides, special back end or front end implementations would have missed the scope, since the variety needs to be supported, which the provided concept as an abstract interface layer realizes. As mentioned before, the use of a mediator/wrapper architecture may be considered for a multi-database query processing architecture using this proposed concept as global schema. Since the proposed schema is based on the *Property Graph Model* in combination with OGC's *General Feature Model*, it is possible to implement alternative concepts e.g. *peer-to-peer* architectures or *linked-data* concepts.

In this context, the choice of spatial and spatio-temporal model was done under the constraint of being a general but precise spatial and spatio-temporal model, which is not an optimized representation for all purposes. As mentioned in Section 3.3, if the structure of the data or the elimination of redundancies leads to optimized representations which saves memory or reduces data transfer without loosing information the proposed spatial or spatio-temporal model should be transformed into the optimized representation and the processing of the data should adapt to the optimized representations. It is still a

matter of research to find optimized representations for all the different purposes, be it processing optimization, memory, and IO reduction or something in between. Not only the application purposes imply the use of optimized representations for special back ends. The time to transform and transfer data needs also to be taken into account, which may lead to less IO costs, less transformation cost on the cost of processing of not optimized representations on back ends which are optimized for the not optimized representations. This could include not to transfer data at all and to consider the In-Situ paradigm by processing the data where it is and only transfer the results. The all in all efficiency might be better than the use of highly optimized high performance computation clusters for special purposes since the IO bandwidth throws a monkey wrench into the overall process. This is also still a matter of research and the optimization highly depends on the use case and the data.

An evaluation of the implementations concerning the handling of the data management model together with the double precision arithmetic have been done. Another evaluation has been done concerning the implementation of a data-driven p-adic scalable space filling curve index. And last, but not least, an evaluation was presented concerning the first implementation of a topological access method.

The following paragraph is published in [31] beforehand:

Calculating topology with double precision arithmetic remains a difficult task. The work provided straight-forward methods to calculate higher dimensional geo-objects from *BREPs*. The algorithm-pipeline produces volumes, if the objects are simple enough, but illustrates the problems which may occur. The major control parameters are the precisions $\epsilon = \varepsilon = \epsilon = \lambda$ and angular precision ζ. If $\epsilon = \varepsilon = \epsilon = \lambda$ is set large, then segments from the input set will be smoothed out, but the planarity check of curve patches within the Dijkstra step 6 of Algorithm 2 will find more valid patches to stitch together to form polygons. If $\epsilon = \varepsilon = \epsilon = \lambda$ is small, then points will not be recognized on planes (e.g. planarity of wire-frame patches), intersections between walls and roofs may also not be recognized any more, which leads to topological inconsistency. If ζ is large, then thin triangles get rejected easily and non-planar surfaces may be triangulated wrongly. If ζ is small, then triangulations get rejected, because they are found to be "non-planar", and hulls are not to be found because the wire-frame patches do not form planar polygons. These problems are not new and are highly sensitive to the quality of input data.

The evaluation included an analysis of distributions of combinatorial Euler characteristics of three different city model decompositions and their overlay. Some distributions of Euler characteristics showed unexpected results. Those results helped to identify computational failures. Besides those useful results, the presented conceptual model which combines the philosophies of graph databases with the OGC's General Feature Model helped to produce complex topological structures and added value to the data-set. The algorithm-pipeline and the pipeline to produce the results may easily be extended and optimized. Parts of the algorithm-pipeline may be exchanged (e.g. the triangulation algorithms, planarity checks etc.) or reconfigured with different precision parameters to

optimize processing efficiencies (better result or better run time through the scalability of graph databases).

As mentioned in Section 3.2, the double precision arithmetic only support processing of spatial or spatio-temporal data of a certain scale in terms of precision. For the evaluation in Section 5.1 a translation of the input data to an arbitrary point within the data set was done while importing. A scaling and translating of the input data set to a unit cube may have improved the robustness. The translation was performed to improve the robustness by reducing the number of bits used for integer digits, since the coordinate system of the input data is a traditional geo-spatial coordinate system used for cartography which usually is not optimized for computational geometry using double precision arithmetic. A scaling was not evaluated since a scaling would change the angles and the angular precision may have caused problems. But the translation of the input data was a first step towards the optimization of computational processing of spatial geo-data based on a traditional geo-spatial coordinate system.

The following two paragraphs are published in [8] beforehand:

The scaled p-adic Gray-Hilbert index has efficient access and insert algorithms and can be shown to be more storage efficient than the optimal static version of the p-adic Gray-Hilbert index tree, if the dimension is sufficiently high or, equivalently, the bucket capacity is chosen sufficiently low. Optimality is defined here as giving rise to the smallest finite p-regular tree containing the data in its leaf nodes in such a way that, on average, no leaf node is overfilled. As p increases, the same set of data becomes separable at a lower hierarchic level, which can be an advantage for methods which query the depth of trees. The index structure corresponds to a sequence of equi-sized subdivisions of hyper-cuboids with locally varying depth. This takes care of the fact that non-uniform data lead to an imbalanced static Hilbert curve index tree.

A data structure is used for encoding data, and properties of the data influence the properties of the encoded data. In this way, efficiency and other properties of data structures are influenced by the properties of the underlying data. In this way, a measure for encoded data which reflects properties of the data themselves becomes possible. The derived *p-adic local sparsity measure* thus contains information about the local distribution of the data set. [...] Potential applications for the index structures constructed here should be seen in the context of massive data in any finite dimension. Further work should deal with a fully data-driven index structure based on space-filling curves. This includes the investigation of dynamic sub-hyper-cuboid sizes which is common to *kd*-trees, data-driven permutations and automated adjustment of coordinate projections. This also includes the relation to possible generalizations of certain 2-adic curves like the Morton or Butz curves to the p-adic case.

As mentioned before, spatial or spatio-temporal access methods use the coordinate system of the data. Since it is impossible to manage spatial or spatio-temporal data of any scale efficiently by one access method and the use of basic data types like double in case of double precision, scalable data types or the support of multiple coordinate

systems would be needed which are part of the internal logic of the access method. At least the support of local and global coordinate systems (e.g. game engines) my also increase data precision. An implementation would have blown the scope of this work.

The topological access method presented in Section 4.7.6 and evaluated in Section 5.3 describes a special version of a topological access method. The graph gets compressed due to less variety of the number of nodes per relation type. But further research, the definition of alternative coordinates, would have blown the scope of this work.

6.1 Summary

Chapter 1 gave a short introduction to motivate this work, described the goals of this work followed by a short outline.

Chapter 2, gave rudiments, split into three parts. The first part introduced the workflow of system scientists, who develop simulation models concerning any kind of real world phenomena based on general system theory. General system theory and the typical workflow of applied system scientists were introduced in order to filter the needs. The second part gave an overview of basic concepts of parallel and distributed data management provided by computer sciences. The third part introduced spatial and spatio-temporal concepts used within the geo-information sciences and the technical standards within the geo-information technologies. The combination of those different subjects led to the following chapter.

Chapter 3 discussed requirements and problems at first. A collaboration concept was provided based on a proposed data management concept. This includes the integration of spatial and spatio-temporal data structures and relation types, as well as access methods which may be applied to a multi-database query processing architecture. The collaboration concept for distributed data mining is based on an interconnected pipeline architecture discussed as a theoretical example by using ParaView's pipeline architecture in conjunction with some synchronization software which interconnects pipeline outputs as sources with pipeline inputs as sinks. Since this theoretical model uses a special cluster software (ParaView) in conjunction with some synchronization software or ParaView's Python Batch processing features, this example only illustrates a special implementation.

Chapter 4 introduced all implementations which had been undertaken in this context. An implementation is presented for the spatial and spatio-temporal geometry core, which is a fusion of two different cores. The first core is a revised and extended version of the spatial geometry core of *DB3D*, and the second core is a revised and extended version of the spatio-temporal geometry core of *DB4GeO* [14]. An implementation of a thematic model and temporal model is introduced. Those implementations were connected to a new complex spatial and spatio-temporal object management model based on the *Property Graph Model* to build the base for distributed and parallel data management.

Access methods are also implemented in order to evaluate different kinds of access management.

Chapter 5 was split up into three parts. The first part evaluates the spatial geometry core and the used double precision arithmetic. It shows an example of using the core in a multithreaded environment which also illustrates the handling when writing algorithms for data import (e.g. *CityGML*), transformation (e.g. tetrahedralization of wireframe models) or extraction (e.g. pipeline to write results in Section 5.1.2) based on the proposed object model. The next two parts evaluate an p-adic scalable space filling curve index and a topological access method which had been developed in the context of this work.

The final chapter 6 concludes this work with a discussion of the concepts and the evaluations. Problems are discussed which arose throughout the implementations, which still need to be taken into consideration. It could be shown that using spatial access methods speed up graph queries with spatial content. It could be shown that parallelization based on property graphs also speeds up the application. It could be shown that a p-adic scalable space filling curve index has a balanced partitioning when the data is not equally distributed. It could also be shown that the first implementation of a topological access method based on a discrete multidimensional R^**-Tree* with the exemplary discretization of topological properties does not lead to well distributed data which can be handled efficiently by the discrete R^**-Tree*.

6.2 Outlook

The papers [30] and [31] contain extracted, transformed and minimized paragraphs of the next two paragraphs, published beforehand:

In my future research, I want to continue on the examination of the introduced topology model, focusing on dynamic objects. I want to work on volume generation out of a set of intersecting non-planar surfaces and the optimization of the algorithm-pipeline to produce scientifically reliable results which may be visualized nicely also, or used within simulations or other scientific analysis as example application. The optimization of the algorithm-pipeline to create topologically consistent datasets includes efficient calculations of intersections and Betti numbers, which will help to topologically analyse spatial or spatio-temporal data sets. This may also include the implementation of different computational arithmetic.

Future research needs to concentrate on spatial, spatio-temporal and topological access methods which are able to handle geo-objects of any scale within a multi-database query processing architecture. This includes the research on the integration of dynamic coordinate systems as part of the spatial or spatio-temporal access methods and the research on topological access methods. Alternative definitions of the “topological” coordinates for the implemented topological access method or even combinations with spatial properties like the dimension or the aggregation level is thinkable.

Bibliography

[1] P.S. Alexandrov. “Diskrete Räume”. In: *Matematicheskii Sbornik (N.S.)* 2 (1937), pp. 501–518.

[2] W. Bär. “Verwaltung geowissenschaftlicher 3D Daten in mobilen Datenbanksystemen”. PhD thesis. Universität Osnabrück, 2007.

[3] J. Barmak. *Algebraic Topology of Finite Topological Spaces and Applications*. LNM 2032. Springer, 2011.

[4] *BCI*. 2021. URL: http://ctfs.si.edu/webatlas/datasets/bci/.

[5] F. Bernardini et al. “The ball-pivoting algorithm for surface reconstruction”. In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), pp. 349–359. DOI: 10.1109/2945.817351.

[6] F. Biljecki, H. Ledoux, and J. Stoter. “Error propagation in the computation of volumes in 3D city models with the Monte Carlo method”. In: *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.* II-2 (2014), pp. 31–39.

[7] P.E. Bradley. “Supporting Data Analytics for Smart Cities: An Overview of Data Models and Topology”. In: *Statistical Learning and Data Sciences SLDS 2015*. Ed. by A. Gammerman, V. Vovk, and H. Papadopoulos. LNCS 9047. Springer, 2015, pp. 406–413. ISBN: 978-3-319-17091-6. DOI: 10.1007/978-3-319-17091-6_35. URL: http://dx.doi.org/10.1007/978-3-319-17091-6_35.

[8] P.E. Bradley and M.W. Jahn. “On the Behaviour of p-Adic Scaled Space Filling Curve Indices for High-Dimensional Data”. In: *The Computer Journal* 65.2 (July 2020), pp. 310–330. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxaa036. eprint: https://academic.oup.com/comjnl/article-pdf/65/2/310/42537740/bxaa036.pdf. URL: https://doi.org/10.1093/comjnl/bxaa036.

[9] P.E. Bradley and N. Paul. “Comparing G-maps with other topological data structures”. In: *Geoinformatica* 18 (2014), pp. 595–620.

[10] P.E. Bradley and N. Paul. *Dimension of Alexandrov Topologies*. arXiv:1305.1815v1 [math.GN]. 2013.

[11] P.E. Bradley and N. Paul. “Using the Relational Model to Capture Topological Information of Spaces”. In: *The Computer Journal* 53.1 (2010), pp. 69–89.

[12] M. Breunig et al. “Collaborative Multi-Scale 3D City and Infrastructure Modeling and Simulation”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W4 (2017), pp. 341–352.

[13] M. Breunig et al. “Geospatial Data Management Research: Progress and Future Directions”. In: *ISPRS Int. J. Geo-Inf.* 9.2 (2020), p. 95.

[14] M. Breunig et al. “The Story of DB4GeO - A Service-Based Geo-Database Architecture to Support Multi-Dimensional Data Analysis and Visualization”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 117 (Jan. 2016), pp. 187–205. DOI: 10.1016/j.isprsjprs.2015.12.006.

[15] E. Butwilowski. “3D data and model management for the geosciences with particular emphasis on topology and time”. PhD thesis. Karlsruher Institut für Technologie, 2015.

[16] *Computational Geometry Algorithms Library (CGAL)*. 2021. URL: https://www.cgal.org/.

[17] R. Condit. *Tropical Forest Census Plots*. Berlin, Germany, and Georgetown, Texas: Springer-Verlag and R. G. Landes Company, 1998.

[18] J. Dušan and B. Branislav. “Elements of spatial data quality as information technology support for sustainable development planning”. In: *Spatium* 11 (2004), pp. 88–83.

[19] H. Edelsbrunner et al. *Geometry and Topology for Mesh Generation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2001. ISBN: 9780521793094. URL: https://books.google.de/books?id=v6BybEYVGqYC.

[20] M. Egenhofer and R. Franzosa. “Point-Set Topological Spatial Relations”. In: *International Journal of Geographical Information Systems* 5.2 (1991), pp. 161–174.

[21] Amy Elser, ed. *Reliable Distributed Systems*. Springer-Verlag New York, 2005.

[22] T. Erl, W. Khattak, and P. Buhler, eds. *Big Data Fundamentals. Concepts, Drivers & Techniques*. The Prentice Hall Service Technology Series. Prentice Hall, 2015.

[23] *GeoMesa*. 2021. URL: https://www.geomesa.org/.

[24] *Geometry Engine Open Source (GEOS)*. 2021. URL: https://trac.osgeo.org/geos.

[25] A. Giovanella, P.E. Bradley, and S. Wursthorn. “Evaluation of Topological Consistency in CityGML”. In: *ISPRS Int. J Geo-Inf.* 8.6 (2019), p. 278.

[26] M.F. Goodchild. “GIS in the Era of Big Data”. In: *Cybergeo : European Journal of Geography [online]* Les 20 ans de Cybergeo (2016).

[27] F. Hirzebruch and M. Kreck. “On the concept of genus in topology and complex analysis”. In: *Notices of the American Mathematical Society, v.56, 713-719 (2009)* 56 (June 2009).

[28] S.P. Hubbell, R. Condit, and R.B. Foster. *Barro Colorado Forest Census Plot Data*. URL http://ctfs.si.edu/webatlas/datasets/bci. 2010.

[29] S.P. Hubbell et al. “Light gap disturbances, recruitment limitation, and tree diversity in a neotropical forest”. In: *Science* 283 (1999), pp. 554–557.

[30] M. W. Jahn et al. “Topologically consistent models for efficient big geo-spatio-temporal data distribution”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W5 (Oct. 2017), pp. 65–72. DOI: 10.5194/isprs-annals-IV-4-W5-65-2017.

[31] M.W. Jahn and P.E. Bradley. “Computing watertight volumetric models from boundary representations to ensure consistent topological operations”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* VIII-4/W2-2021 (2021), pp. 21–28. DOI: 10.5194/isprs-annals-VIII-4-W2-2021-21-2021. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/VIII-4-W2-2021/21/2021/.

[32] M.W. Jahn, P.V. Kuper, and M. Breunig. *Efficient spatio-temporal modelling to enable topological analysis*. Accepted at ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences.

[33] M.W. Jahn et al. “Temporal and Spatial Database Support for Geothermal Subsurface Applications”. In: *Advances in 3D Geoinformation*. Springer, Oct. 2017, pp. 337–356. ISBN: 978-3-319-25689-4. DOI: 10.1007/978-3-319-25691-7_19.

[34] *Java Topology Suite (JTS) features*. 2021. URL: http://www.tsusiatsoftware.net/jts/jtsfeatures.html.

[35] *Kafka Documentation*. 2020. URL: https://kafka.apache.org/documentation.

[36] S. Kalyanswamy. “Euler Characteristic”. In: 2009.

[37] H. Kang and K. Li. “Assessing topological consistency for collapse operation in generalization of spatial databases”. In: *Perspectives in Conceptual Modeling*. Ed. by J. Akoka. Vol. 3770. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005.

[38] P.V. Kuper. “Spatio-Temporal Data Handling for Generic Mobile Geoinformation Systems”. PhD thesis. Karlsruher Institut für Technologie, 2016.

[39] D.E. LaCourse, ed. *Handbook of Solid Modeling*. McGraw-Hill, 1995.

[40] S. Landier. “Boolean Operations on Arbitrary Polyhedral Meshes”. In: *Procedia Engineering* 124 (2015). 24th International Meshing Roundtable, pp. 200–212. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2015.10.133. URL: https://www.sciencedirect.com/science/article/pii/S1877705815032348.

[41] H. Ledoux. “The Kinetic 3D Voronoi Diagram: A Tool for Simulating Environmental Processes”. In: *Advances in 3D Geoinformation Systems*. Ed. by P. van Oosterom et al. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg: Springer, 2008.

[42] H. Ledoux and M. Meijers. “Topologically consistent 3D city models obtained by extrusion”. In: *International Journal of Geographical Information Science* 25.4 (2011), pp. 557–574.

[43] S. Li. “On topological consistency and realization”. In: *Constraints* 11.1 (2006), p. 3151.

[44] S. Li et al. “Geospatial big data handling theory and methods: A review and research challenges”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 115 (2016), pp. 119–133.

[45] H. Malchow. *Skripte zu den Vorlesungen Gleichungsbasierte Modelle I and II*. 2012.

[46] T. Möller. “A Fast Triangle-Triangle Intersection Test”. In: (1997). URL: https://web.stanford.edu/class/cs277/resources/papers/Moller1997b.pdf.

[47] P.v. Oosterom, B. Maessen, and W. Quak. “Generic query tool for spatio-temporal data”. In: *Int. J. Geographical Information Science* 16.8 (2002), pp. 713–748.

[48] M.T. Özsu and P. Valduriez, eds. *Principles of Distributed Database Systems*. Springer-Verlag New York, 2011.

[49] *ParaView*. 2021. URL: https://www.paraview.org/.

[50] “ParaView and Python”. In: (2021). URL: https://www.paraview.org/Wiki/ParaView_and_Python.

[51] “ParaView Web”. In: (2021). URL: https://www.paraview.org/web/.

[52] *ParaViewGeo*. 2021. URL: http://paraviewgeo.objectivity.ca/.

[53] K. Polthier and M. Rumpf. “A concept for time-dependent processes”. In: *In Visualization in Scientific Computing*. Springer Verlag, 1994, pp. 137–153.

[54] *PostGIS 3D Functions*. 2021. URL: https://postgis.net/docs/manual-3.1/PostGIS_Special_Functions_Index.html#PostGIS_3D_Functions.

[55] “PvPython and PvBatch”. In: (2021). URL: https://www.paraview.org/Wiki/PvPython_and_PvBatch.

[56] “Python calculator and programmable filter”. In: (2021). URL: https://www.paraview.org/Wiki/Python_calculator_and_programmable_filter.

[57] E. Rahm, G. Saake, and K. Sattler, eds. *Verteiltes und Paralleles Datenmanagement*. Springer, 2015.

[58] M. Rivi et al. *In-situ Visualization: State-of-the-art and Some Use Cases*. Partnership for Advanced Computing in Europe White Papers. 2012.

[59] M.A. Rodríguez et al. “Measuring consistency with respect to topological dependency constraints”. In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2010, pp. 182–191.

[60] C. Rolfs. “Konzeption und Implementierung eines Datenmodells zur Verwaltung von zeitabhängigen 3D-Modellen in geowissenschaftlichen Anwendungen”. Diplom Thesis. Hochschule Vechta, 2005.

[61] *Simple Feature Computational Geometry Algorithms Library (SFCGAL)*. 2021. URL: http://www.sfcgal.org/.

[62] J. Stoter, H. de Kluijver, and V. Kurakula. “3D noise mapping in urban areas”. In: *International Journal of Geographical Information Science* 22.8 (2008), pp. 907–924.

[63] F. Taubert et al. “The structure of tropical forests and sphere packings”. In: *Proceedings of the National Academy of Sciences of the U.S.A.* 112.49 (2015), pp. 15125–15129.

[64] “The Catalyst User’s Guide v2.0”. In: (2021). URL: https://www.paraview.org/documentation/.

[65] *TinkerPop Documentation*. 2020. URL: https://tinkerpop.apache.org/docs/current/.

[66] A. Vaisman and E. Zimanyi. “What is Spatio-Temporal Data Warehousing?” In: *International Conference of Data Warehousing and Knowledge Discovery* (2009), pp. 9–23.

[67] *Visualisation Toolkit (VTK)*. 2021. URL: https://vtk.org/.

[68] *Visualisation Toolkit (VTK) File Formats*. 2021. URL: https://kitware.github.io/vtk-examples/site/VTKFileFormats/.

[69] K. Zhang et al. “A 3D visualization system for hurricane storm-surge flooding”. In: *IEEE Computer Graphics and Applications* 26.1 (2006), pp. 18–25.

[70] J. Zhao, J. Stoter, and H. Ledoux. “A Framework for the Automatic Geometric Repair of CityGML Models”. In: *Cartography from Pole to Pole*. Ed. by M. Buchroithner, N. Prechtel, and D. Burghardt. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg: Springer, 2014.