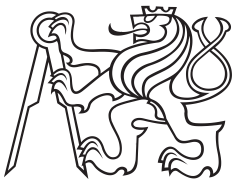
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of computer graphics and interaction

Urban data visualization

Bc. Vojtěch Tomas

Supervisor: Ing. David Sedláček, Ph.D.
Field of study: Open Informatics
Subfield: Computer Graphics
January 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tomas** Jméno: **Vojtěch** Osobní číslo: **457805**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vizualizace urbanistických dat

Název diplomové práce anglicky:

Urban data visualization

Pokyny pro vypracování:

Navrhněte prototyp modulárního systému pro zpracování a vizualizaci urbanistických dat. Identifikujte formáty vstupních dat, proveďte rešerši existujícího software umožňující vizualizaci urbanistických dat a popište jeho limitace.
Při návrhu systému zohledněte zejména charakter vstupních dat, parametrizovatelnost vizualizace, srozumitelnost a použitelnost systému, prezentační formáty, paměťovou a výpočetní efektivitu, rozšiřitelnost systému. Předpokládejte dva způsoby projekce/vizualizace dat, první na standardní projekční plochu (např. monitor, plátno), druhou bude projekce na fyzický prostorový model části města shora (ortografiká projekce). Pro realizaci projekce na model města navrhněte modul, který připraví podporu pro tvorbu takového modelu, např. postup sestavení modelu ze standardizovaných dílů/kostiček, nebo data vhodná pro 3D tisk modelu.
Implementujte prototyp systému a otestujte jej vytvořením modelu vybrané části Prahy a vizualizací dvou různých množin dat: po dohodě s vedoucím vyberte testovací množiny dat a demonstrujte tvorbu fyzických modelů (dle dostupnosti prostředků pro tvorbu fyz. modelů).

Seznam doporučené literatury:

1] Bits and Bricks: Tangible Interactive Matrix for Real-Time Computation and 3D Projection Mapping. I Winder, K Larson. Proceedings of the 2017 Future Technologies Conference (FTC), 2017
2] Tang, L.; Chen, C.; et al. Building Information Modeling and Building Performance Optimization. In Encyclopedia of Sustainable Technologies, edited by M. A. Abraham, Oxford: Elsevier, 2017.
3] Ledoux, H.; Ohori, K. A.; et al. CityJSON: A compact and easy-to-use encoding of the CityGML data model. Open Geospatial Data, Software and Standards, volume 4, no. 1, 2019

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. David Sedláček, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **10.03.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **19.02.2023**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I want to thank Ing. David Sedláček, Ph.D. for his advice and support. Thank you to doc. RNDr. Tomáš Hudeček, Ph.D., Ing. arch. Annamária Bohuniczky, and Mgr. Jiří Čtyroký, Ph.D. for their invaluable insight and for letting me in on the inner workings of city planning.

Thank you to Josef Kortan for his consistent support, guidance, and always finding a way to show me a different perspective. To Lucie Klabanová for proofreading this thesis. To my dear colleagues, Jan Petýrek, Anna Moudrá, and Giang Chau Nguyenová, for accompanying me on the year-long journey. I would also like to thank my friends and family for supporting me throughout my studies.

Thank you.

Declaration

I declare that this thesis represents my work and that I have listed all the literature used in the bibliography.

Prague, 3 January 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 3. ledna 2022

Abstract

Urban data visualization plays a vital role in sustainable city evolution. Visual media enable efficient communication which is the cornerstone of any development. This thesis presents a design and implementation of a modular visualization system of urban data. The initial research focuses on the role of data platforms in urban planning and analyses the integration of visualization with diverse data sources, including simulation models. Further, the thesis explores possible geometry representations and ways to process large geospatial datasets. A simple and extensible styling language is proposed. The system implementation is available as two python packages for data processing and visualization. On the inside, the geometry processing is implemented in C++ using spatial and temporal acceleration data structures. The visualization component is implemented as an accessible web application enabling data viewing and export. The validation of the implemented solution includes the evaluation of performance tests and an iterative qualitative user study.

Keywords: urban data, visualization systems, extensive datasets, dynamic data, open data, web application

Supervisor: Ing. David Sedláček, Ph.D.

Abstrakt

Vizualizace urbanistických dat zásadním způsobem přispívá k udržitelnému rozvoji měst. Vizuální média umožňují efektivní komunikaci, která je základním kamenem vývoje. V této práci je představen návrh a implementace modulárního vizualizačního systému urbanistických dat. Počáteční rešerše se zaměřuje na roli datových platforem v městském plánování a analyzuje možnosti integrace vizualizace s různými datovými zdroji, včetně simulačních modelů. Zpracovaná rešerše se dále zabývá možnostmi reprezentace geometrie a zpracování rozsáhlých datových sad. Práce obsahuje návrh a implementaci jednoduchého a rozšiřitelného stylovacího jazyka. Systém je implementován formou dvou Python balíčků umožňujících zpracování a vizualizaci dat. Interní zpracování geometrie je implementováno v C++ a využívá prostorových a časových akceleračních datových struktur. Vizualizační komponenta je implementována jako přístupná webová aplikace umožňující prohlížení a export dat. Validace implementovaného řešení zahrnuje vyhodnocení výkonnostních testů a iterativní kvalitativní uživatelskou studii.

Klíčová slova: urbanistická data, vizualizační systémy, rozsáhlé datové sady, dynamická data, otevřená data, webová aplikace

Překlad názvu: Vizualizace urbanistických dat

Contents

Figures

Tables

Chapter 1

Introduction

A city is often compared to an ever-evolving organism. As humans do with most known complex systems, we try to observe them, describe them, and predict their behavior. We also try to shape the city structures in response to our needs and current events. The city is a complex system, and to plan and control its evolution, we need to understand it as a whole.

An inherent loop arises as decision-makers influence citizens' daily lives; citizens adjust and influence plans by feeding new data back into the system. The effectiveness of this loop can be increased by enabling cooperation throughout the participating groups, which opens up space for data visualization as a communication medium. Visual media enable efficient communication and therefore are essential for achieving sustainable city evolution.

The base for the visualization includes more than just data. Predictive models, the know-how of the participants, and their diverse views are all equally significant inputs. The main topic of this thesis is the specification and implementation of tools that could help find common ground and provide a unified view of the city's state.

The primary goal of this thesis is to design and implement a visualization system that supports both virtual and physical media. The developed system should be easily accessible, extensible, and performant. Existing visualization tools often prioritize some of these qualities over the rest; however, an ideal visualization tool should balance them all.

Thesis Structure

Chapter 2 focuses on the role of data platforms in urban planning; diverse forms of data visualization are examined. Chapter 3 examines urban data visualization principles and methods enabling the processing of extensive datasets. Chapter 4 explores the existing geospatial data formats and information systems. The design of the visualization tool is presented in chapter 5. Finally, the chapter 6 presents the implementation details, the evaluation of performance tests, and a small user study.

Chapter 2

From Data to Models

This chapter maps the current state of urban data sources and approaches to visualization using various types of media. As data alone is not always a sufficient source of information for decision-making, alternative approaches using simulations and models are discussed. The implications of data and model integration are examined in the context of user interface and software architecture design.

2.1 Urban Informatics

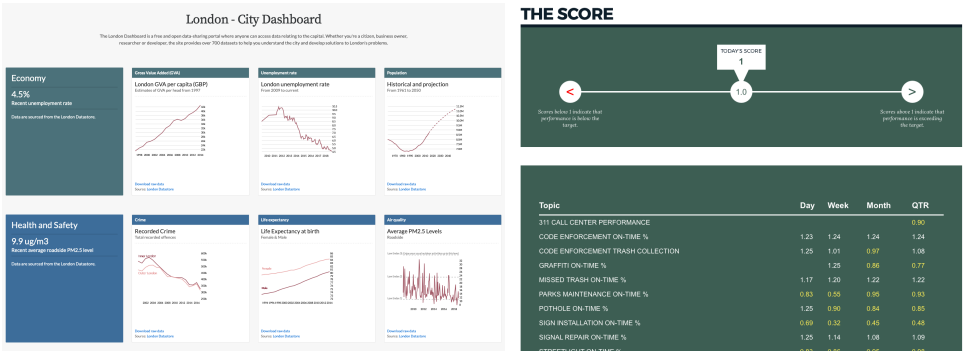
In 2011, Forth et al. [1] described Urban Informatics as a separate field of study. It focuses on three key aspects - place, technology, and people in the context of urban environments. The urban environment is described as a *"complex techno-social network; the city only meaningfully exists when a sustained stream of people occupies it"* [1]. The authors outline and study four dominant trends: the emergence of Ubiquitous Computing, the accessibility of real-time information, informed sustainability and planning based on citizen participation.

One of Urban Informatics' objectives is to help develop communication channels between local authorities and citizens. Moreover, thanks to the gathered information, the citizens can, directly and indirectly, influence the development of the urban area. The communication channels, as understood by Urban Informatics, are omnidirectional.

2.2 Data Sources

Robinson et al. [2] presented a classification of the data sources into two categories — Open Data and Ubicomp sensing. The authors further demonstrate how the gathered collective knowledge could serve as a basis for optimizing daily activities, such as trip planning.

Open Data. The most prominent source of data are governments and local institutions. The data is commonly released in several open formats, which are further described in the section 4.1. This data often includes maps,



(a) : London City Dashboard [7] **(b) :** Boston CityScore [8]

Figure 2.1: Examples of City Dashboards

building layouts, terrain models, public transport data, etc. The previously mentioned data are mainly static; some cities also offer real-time weather information or public transport vehicle locations.

Ubicomp. Ubiquitous Computing was first presented by Weiser in 1991 [3]. The concept of ubicomp relies on small embedded computers, which communicate together and allow for seamless interaction between users and technology. As these devices are used by users or exposed to the environment, they can gather the information that will later serve as input for predictive models and as a basis for decision-making. Further research in this area lead to the development of fields such as Participatory Sensing [4], which enables gathering data from devices of individual users, or Urban Computing [5], which studies the impact of the ubiquitous information on the city level.

2.3 City Dashboards

The gathered data is usually made public in an open format or processed and subsequently presented in a visual form on a data platform. While this approach is reasonably straightforward and data platforms can help answer simple questions, further analysis shows that these platforms alone are insufficient for improved communication between stakeholders.

2.3.1 City Dashboard Classification

Barns [6] presents a four-class classification of dashboard-like platforms: Data Repositories, Data Showcases, City Scores, and Data Marketplaces.

Data Repositories. Data Repositories or Open Data Portals usually function as a public site where data is freely available in machine-readable formats. The data is usually authored by a public institution or city government. Example of such platforms can be NYC Open Data Portal [9], Helsinki Open Data Portal [10], Vienna Open Data Portal [11], and the Chicago Data Portal [12].

Data Showcases. Data Showcases, also generally known as Dashboards, offer a public interface to the data. The data is usually presented in a visual format, such as a map, a timeline, or a graph. The underlying data might not be publicly available. Examples of such platforms include the City of London Dashboard operated by Datopian [7], see figure 2.1a.

City Score. City Score platforms integrate several data sources to provide a performance metric against a target set by, e.g., city government. Examples of such platforms include The Greater Sydney Dashboard [13] and the Boston CityScore [8], see figure 2.1b.

Data Marketplace. Data Marketplace is a broader term, which encompasses the combination of all previously mentioned platform types. Generally, public and private sector representatives can operate the site together to provide access to a broader range of performance metrics or machine-readable data. A great example of such a platform was the City Data Exchange (CDE) project run by the Municipality of Copenhagen, the Capital Region of Denmark, and Hitachi in years 2013–2018 [14].

2.3.2 City Dashboard Analysis

All previously listed examples of data dashboards function on similar principles. Most of them inform the viewer about the current city-state using tables and graphs and appear as a static environment offering limited interactivity. The presented information has been simplified to offer a quick overview. If a deeper understanding of the situation is necessary, the user needs to download the source data and attempt to visualize it using a different tool. A problem arises if the data sources are not publicly available.

The City Data Exchange Report [14] lists several observed barriers to effective data exchange, including lack of use cases and the reluctance of the participants to share data on an open data platform. Goldsmith and Crawford argued [15] that the adoption of data platforms and data-driven governance would *"open up the machinery of government to its people, letting them collaborate to create solutions coproduced by public servants and their constituents"* [15]. Despite the ongoing efforts to implement smart city solutions by the public sector alone, it is possible to observe signs of a lack of support for the proposed approach.

Robinson writes: *"Commercial agendas for smart cities are just as likely to reduce our life expectancy and social engagement by making it easier to order high-fat, high-sugar takeaway food on our smartphones to be delivered to our couches by drones whilst we immerse ourselves in multiplayer virtual reality games."* [16] In the article [16], he proposes that one of the main reasons why the smart city transformation stalled so far is the lack of cohesion between the public and private sectors.

Most existing smart city solutions are funded as research-and-development pilot projects, and very few reach sustainable states. The commercial sector provides most of the significant investments into the smart city agenda,

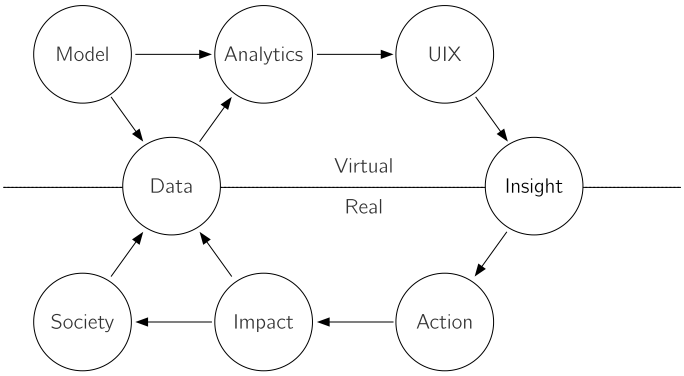


Figure 2.2: Sociotechnical system model [17]

focusing on expanding their business, hence the previous quote. The lack of incentive for the private sector to participate explains why virtually no smart city projects funded by private companies are developed in cooperation with public institutions in the public interest of city development.

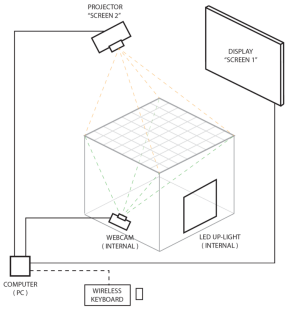
Summary. An alignment of questions posed by the public and private sector representatives is needed to make the dashboards applicable as an effective information source for both groups. City Dashboards might function well as a general source of information; however, the general-purpose dashboards lack the ability to answer targeted questions that require more profound insight into the input data. It is certainly possible to extend the dashboard to provide a more detailed view of the data, but this extension has to be provided on-demand and requires the action of the dashboard developer/supervisor.

2.4 Modeling the Urban System

Urban data alone is not always sufficient input for city planning. Winder [17] describes the paradigm of generating insight from data illustrated in figure 2.2. It emphasizes the role of the user interface, which is essential in getting familiar with the actual data and analytics. The generated insight influences further planning and development. The impact of these actions has the potential to generate new data that feeds back into the loop. The step from data to analytics can be rather intensive and involves creating analytical models presented as interactive simulations. The practical application of this approach is well illustrated in the article [18], presenting a physical environment using tangible media that acts as a control panel of the urban development modeling tool (see figure 2.3).

The MIT City Science Research Group has taken a similar approach and developed the CityScope platform. The framework integrates computational

(a) : Participants using Tactile Matrix [19]



(b) : SDK Schema [20]

Figure 2.3: Tactile Matrix framework

models and data warehouses with digital and physical user interfaces. [21] The available CityScope online documentation [22] is quite limited. It describes the overall structure of the project (see figures 2.4 and 2.5); however, it does not go into detail, and further investigation of the available source codes is necessary to replicate or extend the current state of the platform. The backend of the platform (CityIO) acts as an integration layer for simulation or computational modules (GAMA, SUMO, Brix) and user interface (CityScoPy, RoboScope). This setup is particularly effective from a software architecture perspective and allows for virtually any modeling software integration. [22]

Urban Modeling Tools. According to [23], *"the modeling process can be seen as an iterative process, in which specific knowledge is injected, and series of issues has to be discussed."* Although this description is provided in reference to the OpenMOLE modeling toolbox, it applies to other modeling tools such as GAMA [24], modeling tool used by CityScope, and MATSim [25], multi-agent transport simulation tool also utilized by urban planners [26]. The agent-based modeling approach is well suited for urban planning, as it allows to model the dynamics of the urban environment, including the interaction between agents.

A different approach to urban modeling utilizes cellular automata. Santé et al. [27] present an exhaustive analysis of the currently known cellular automata models for the simulation of real-world urban processes. *"The main strength of CA-based models is their ability to integrate the modeling of the spatial and temporal dimensions of urban processes. Yet, the main reason for the widespread acceptance of these models is their simplicity."* [27] The simplicity of CA can also act as an obstacle, and the basic cellular automata rules and principles need to be extended by relaxations. The use of cellular automata together with agent models is also possible. [28] Santé et al. also state that based on the known qualities, the cellular automata models *"will not be used to exactly predict a phenomenon, but to interactively simulate different scenarios by modifying the parameters of the model"* [27].

Integration of Modeling Tools and Visualization. A basic understanding of the utilized data structures is required to generate a visualization based on

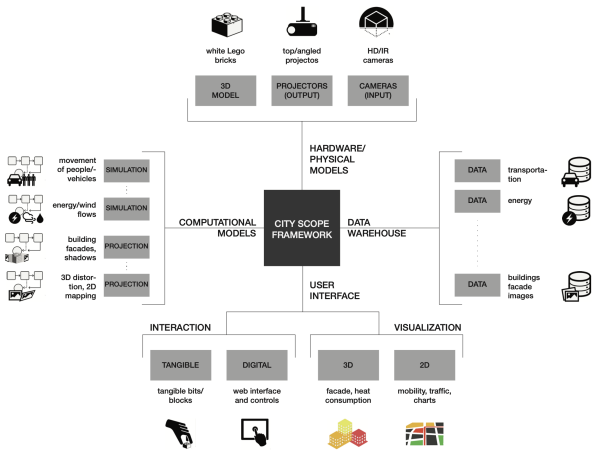


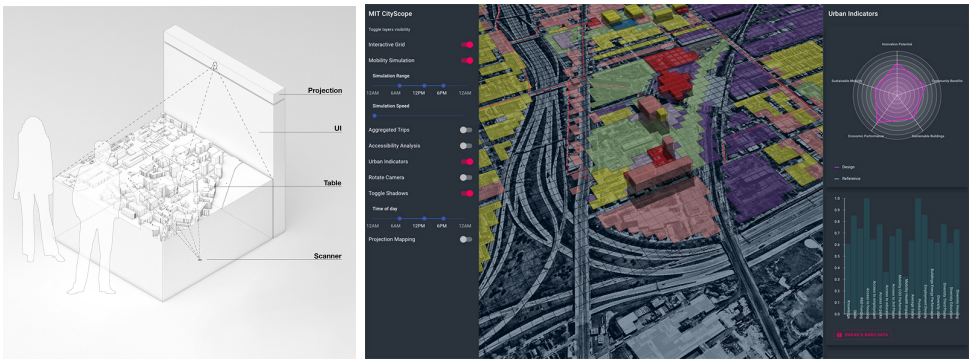
Figure 2.4: Integration Schema of CityScope Framework [21]

the inputs and outputs of the modeling tools. Both introduced agent-based tools — GAMA and MATSim — produce output files in XML format. The structure of these files is well documented, and the files can be parsed using a general XML parsing tool; however, the contents do not adhere to any generally used spatial data format. As for cellular automata, the underlying grid structure makes them quite simple to export and integrate into different applications. Therefore, the visualization platform must be easily extensible to account for the output files' specific structure.

2.5 Visualization Media

The discussion of modeling tools in the previous section mentions different visualization approaches. To overcome the inherent incompatibility of 2D digital media, 3D physical models, and dynamic simulations, Underkoffler and Ishii conducted several experiments [29, 30] and introduced a framework called Urp — a system for urban planning. The system integrates the beforementioned components and utilizes a concept of *I/O Bulb*. This setup enabled the projection of information onto objects' surfaces while the objects acted both as the projection plane and as a controller. This proof-of-concept was later extended into the Luminous Table Project [31], which extended the data integration and simulation capabilities and explored the system's potential in the context of collaborative urban planning.

Winder [18] suggested that by constraining the objects into a matrix, *"we enable scanning in a way that is cheap in terms of both computation and hardware while facilitating projection-mapping,"* which allows for smoother interaction. Based on a similar idea, Larson and Hadhrawi presented CityScope as *"the proposed framework combines physical and digital mediums: it projects precise sets of data on tangible, physical constructs made out of LEGOs"* [21].



(a) : Tangible Interface [22] **(b) :** Web Interface [22]

Figure 2.5: CityScope Framework

Later studies [32, 33] demonstrated the potential of collaborative interfaces and how consensus can be achieved by collaborative real-time plan optimization using the physical interface. Although these research papers go more into detail about the used predictive models, from the perspective of this thesis, the most interesting is the successful validation of the physical user interfaces.

The only shortcoming of the physical media and projection mapping is that it requires a permanent site for the physical model and an elaborate setup. The usage of matrix-like grids can lower the complexity of the setup; however, it is still not as easily shared with multiple participants as the dashboards and purely web-based visualization tools.

2.6 Citizen Participation

An interactive system with physical components is also a suitable tool for exhibitions and galleries to let visitors engage in city planning. In 2019, the OFICINA studio created an interactive exhibition [34] for CAMP and IPR Prague that aimed to present Strategic Plan for Prague [35]. The visitors' task was to regulate city life (e.g., tourism, transport) and set the course of future development (e.g., investments, housing). The simulation was controlled by the placement of physical blocks (see figure 2.6a) and analog pulls; the feedback was provided on a wide-screen visualization (see figure 2.6b). Another exhibition at CAMP in 2019 used a combination of the architectural model and projection mapping [36], see figure 2.7. The exhibition was not interactive; however, the projection served as a visual aid for the visitors standing around the physical model marking the transformed area.

(a) : Color blocks **(b) :** The exhibition environment

Figure 2.6: Pře(d)stav si Prahu by OFICINA in CAMP [34]

(a) : Highlighted area **(b) :** Physical model

Figure 2.7: Rohanský ostrov: nový Karlín? in CAMP [37]

2.7 Summary

Previous attempts to establish data-sharing platforms were unsuccessful due to the lack of cooperation between the private and public sectors. These platforms usually act as dashboard-like data marketplaces. While these platforms can work well as general information sources and are easily sharable, their extensibility and interaction capabilities are limited. Depending on the used software, integration with modeling tools might be difficult. The questions the visualization needs to answer are often obtained on the go, which requires the use of flexible and interactive solutions. The alternative is to use a more complex system primarily designed to integrate various forms of visualization, modeling tools, and data. While these tools are designed to be used in a collaborative manner, and studies have shown their potential in urban planning, the on-site presence of the users is required, which rules out the possibility of remote cooperation.

Chapter 3

Visualization Principles

This chapter discusses the general principles of spatial data visualization. The traditional visualization workflows following the visualization pipeline are not always sufficient; the model utilized by visual analytics is presented. Spatial data representations are discussed in the context of data manipulation and integration. Appropriate mapping of data attributes and visual elements is considered. Finally, different concepts enabling large data quantities are presented.

3.1 Visualization Pipeline

Classical visualization pipeline as presented by Telea et al. [38] consists of four steps: Importing, Filtering, Mapping, and Rendering, see figure 3.1. Several extensions of this pipeline exist [39], including extensions for out-of-core data processing [40, 41] and streaming prioritization depending on the current focus area [42] combined with level-of-detail-based accelerations [43]. The pipeline execution can be accelerated using task, pipeline, and data parallelism [44].

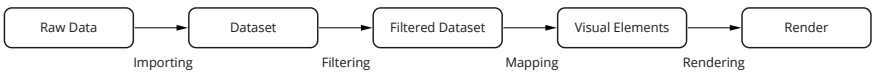


Figure 3.1: Visualization pipeline

Importing. Importing data requires choosing a suitable representation of the input data. The transformation might require representation conversions; however, the import step should be non-destructive and preserve all of the input information.

Filtering. The main goal of data filtering is to extract desired information from the input dataset. Filtered datasets are easier to both process and perceive. The input and output of the filtering function are datasets with identical domains.

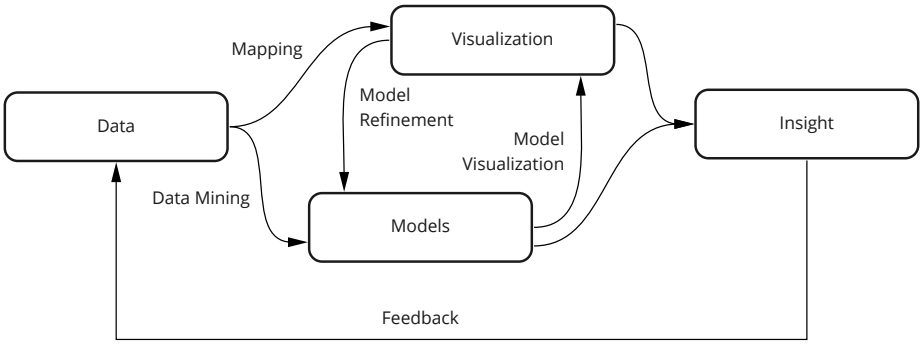


Figure 3.2: Visual Analytics

Mapping. During the mapping step, the values are encoded using visual elements. The reason why mapping and rendering steps are separated is the increased modularity of the system, which gives the user the possibility to tweak the final rendering properties.

Rendering. The final rendering step takes the visual representation of the dataset plus viewing parameters and creates the desired output — most commonly an image. The user is given the possibility to change the viewing parameters, e.g., the visible range, the size of the image, etc.

3.2 Visual Analytics

The current rate at which cities generate new data exceeds the limits of automatically analyzable and visualizable quantities. Moreover, the initial pipeline design does not support the interaction with the analytical models introduced in section 2.4 of the previous chapter. Thomas and Cook [45] suggest a new approach to the visualization process and offer a natural extension of the original pipeline by integrating data, models, visualization, and user interaction into a singular loop, see figure 3.2. When applied to the decision-making and geospatial domain, it becomes apparent that visual analytics is yet another example of a sociotechnical system introduced in the previous chapter.

According to Andrienko et al. [46], the application of visual analytics to geospatial data brought security implications and revealed a lack of evaluation standards. Analyzing specific geospatial data can conflict with privacy protection; examples of such data are one's home or workplace location, daily activities, or trips. *"Researchers (...) are typically concerned with the possible threats to privacy arising from computational data processing and from the integration of two or more datasets. They do not study the privacy issues arising from the involvement of human analysts empowered with interactive visual tools."* [46] Similarly, visualization and data analysis evaluation standards only apply to each domain individually.

Prior research in the area of geovisual analytics for decision making has been conducted by Andrienko et al. [47] It explored the area of geovisual analytics to support physical space analysis, facilitate cooperation of *"multiple actors with diverse roles, expertise, capabilities, and interests, and to integrate innovative computational technologies into the established human practices of decision-making."* [47] The objectives are consistent with those of visual analytics. Andrienko et al. also presented several practical examples of when this approach can be applied:

- site selection (placement of housing, offices) that takes into account neighborhood building relationships, traffic patterns, socio-demographic data, availability of services, future growth, etc.,
- or time-critical scenarios such as evacuation, which requires the decision to be made in a limited amount of time.

Nowadays, the first of these problems is usually solved using geospatial software and multi-criteria decision analysis when a reduced processing strategy is used, and most of the relevant information is not considered. These two situations are just examples; finding the optimal classification of spatial decision problems remains an open issue. [47] Further, Andrienko et al. propose the geovisual analytics need to build on the following principles:

Collaboration inclusion of stakeholders in the decision-making process with the focus on the groups who are affected by the decision in both the positive and negative directions, and across different institutions, cities, and between countries or cultures

Communication visual interface utilization for the information transfer

Flexibility adaptability of the system to the needs of participants

3.3 Geospatial Data Representation

This section aims to describe basic principles of commonly used spatial data representations; actual file formats are later described in the section 4.1. Following subsections introduce structures for geometry and attribute representation and algorithms for dataset integration. Integrating datasets, also known as overlay mapping, involves finding intersections and mapping attributes from the original dataset pair. Memory efficiency and extensibility of the structures and algorithms are considered. This section draws from the publication [48].

3.3.1 Křovák's Projection

Most of the data representations operate on a plane, which is inherently incompatible with the round shape of the Earth. The main focus is on preserving shapes, angles, and sizes while projecting the data onto a flat

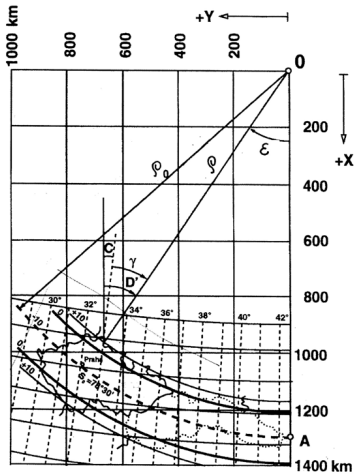


Figure 3.3: Křovák's projection — planar situation [49]

surface. A range of map projections is available, deforming some of these properties in different areas. As the main focus of this thesis is the application of urban visualization in Prague, Czech Republic, Křovák's projection needs to be mentioned.

Křovák's projection utilizes a conformal oblique equidistant conic projection; the planar situation is presented in figure 3.3. Meters are used as a unit of measurement. While the actual mathematical formulation of the projection can be found in the literature, see [49], from the application perspective, its properties are more interesting:

conformal preserves angles,
oblique does not utilize perspective,
equidistant preserves lengths.

The projection can be found under the identification EPSG:5514 (utilizing Greenwich meridian); several variants differ in the used projection parameters. There are variants utilizing Ferro meridian (EPSG:5221), modified axes (EPSG:5513, EPSG:2065), or slightly modified projection (EPSG:5515, EPSG:5524, and other).

3.3.2 Vector Data

The fundamental elements utilized by vector representation are points, lines, and polygons. These structures can be linked to additional attributes, and the basic model can be extended to capture topological relationships; the structure is illustrated in figure 3.4. The table of attributes can theoretically contain arbitrary values, and attributes can be assigned to any of the three fundamental element types. Moreover, this structure can be extended to support various grouping of the primary elements (e.g., multiple polygons can be grouped into a multipolygon) or extend the topological relations of the elements (e.g., polygons with holes).

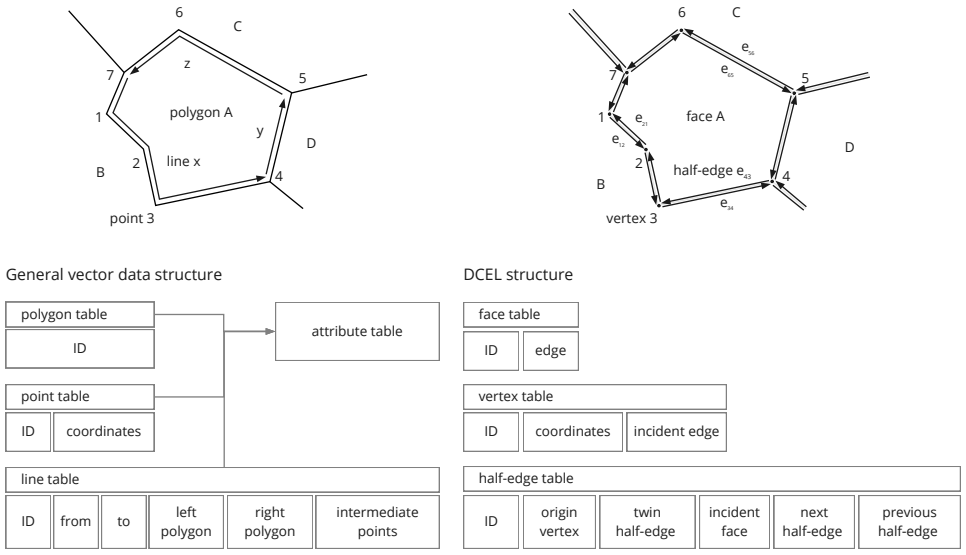


Figure 3.4: Possible Vector Data Representation

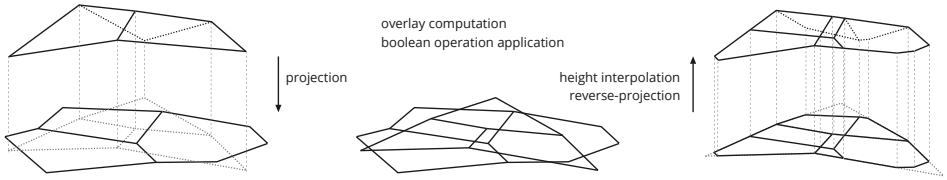


Figure 3.5: Cross-dimensional Overlay Mapping

The vector representation is memory efficient since it stores only the bare geometrical and topological information. The attribute data can be separated from the geometry or extended if necessary. When applied to two-dimensional data, the dataset can be viewed as a planar subdivision or two-dimensional line segment arrangement and represented using DCEL (Doubly Connected Edge List) data structure. [50]

Planar Overlay Mapping. In practice, planar vector datasets can be integrated using the DCEL overlay algorithm and boolean operations. The `MapOverlay` algorithm [50] is well suited for this task as it preserves the relations between the overlay DCEL and the input data.

Cross-dimensional Planar Overlay Mapping. A possible approach to two- and three-dimensional dataset integration involves projecting the three-dimensional dataset into two dimensions and then using the `MapOverlay` algorithm. The height information must be preserved for individual vertices to facilitate the reverse projection and height interpolation in the intersection points. The procedure is illustrated in figure 3.5. Degenerate polygons can appear after the initial projection if the three-dimensional dataset contained

task/property	polygonal vector data	triangulated vector data
specification ambiguities	needs specification of allowed polygon extensions (using holes, multipolygons)	none
loading inputs	requires transformation of vector inputs with unsupported extensions	triangulate input data
memory requirements	lower	higher
overlay computation	using `MapOverlay` [50]	using `MapOverlay` [50] or triangle-primitive intersections and queries accelerated by hierarchical data structures
rendering	needs triangulation	as is

Table 3.1: Comparison of polygonal and triangular vector data workflows

vertical polygons; however, these polygons must remain in the projected DCEL as overlapping groups of half-edges. The `MapOverlay` algorithm excludes self-intersections of overlapping half-edges that belong to the same input DCEL. If the plane sweep algorithm[1] yields a new point for each overlapping half-edge when an intersection is found, the output of the `MapOverlay` can be reverse-projected into three dimensions.

Graphics-friendly Approach. It is safe to assume that the data will be visualized using a graphics library for rendering in an interactive environment. Current graphics libraries can render a limited number of primitive types, including points, lines, triangles, and array-like extensions of these primitives only. Therefore, general polygons need to be triangulated prior to rendering. This situation opens up the possibility of performing the overlay mapping after the triangulation step. The benefits, as well as drawbacks of the two outlined approaches, are described in the table 3.1. The overlay mapping of triangular data can be computed using the approach outlined in pseudocode 1. It uses the same technique as collision detection algorithms for triangular mesh, traversing two hierarchical structures in parallel. Some notes on this algorithm:

- The projection of the input *dataset*3 is not stored; the tree structure for the projected data can be built without actually storing the projected coordinates of the dataset.
- BVH (Boundary Volume Hierarchy) is a suitable hierarchical structure for dealing with the mesh collisions. Here, the problem is reduced to a single plane; therefore, R-trees utilizing rectangles as bounding volumes and top-down greedy split construction strategy [51] are a valid choice.
- The projection and reverse projection, as illustrated in figure 3.5, is performed in the intersection routine.
- Degenerate cases appear after the projection of vertical triangles. A possible solution is to avoid the projection and clip the three-dimensional triangles by supporting planes of the intersected planar triangles extruded to infinity in the third dimension.

[1] The plane sweep algorithm for line-segment intersection is described in [50].

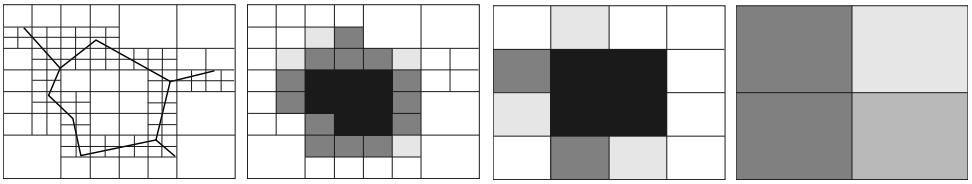


Figure 3.6: Quad-tree-based Raster Representation

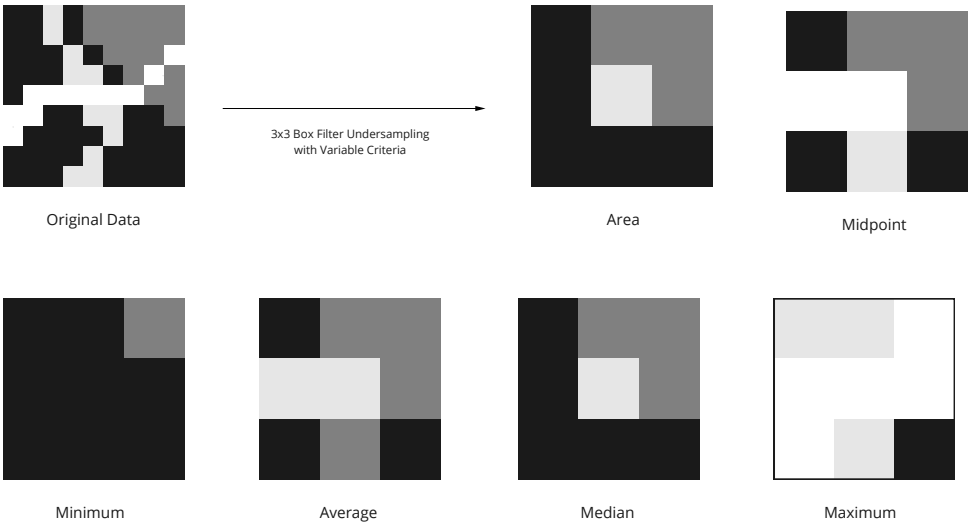


Figure 3.7: Raster Data Resampling Strategy Examples

3.3.3 Raster Data

Raster data is organized in a grid-like structure, usually utilizing rectangular cells containing singular values. The data contained in the grid can be effectively compressed utilizing chain, run-length, or block codes. Another raster representation with built-in compression utilizes quad-tree-based pyramids, illustrated in figure 3.6.

One of the main shortcomings of raster data is its precision. While vector data can, in theory, be represented with arbitrary precision, the raster resolution will always limit the precision of the stored data. Additionally, the amount of stored data grows exponentially as the resolution increases. Another issue arises during raster undersampling. There are several strategies of data resampling illustrated in figure 3.7.

As the raster usually contains a single value per cell, multi-attribute data require the layering of several rasters to store all available values. Contrary to vector data mapping, integration of multiple raster datasets is algorithmically simple — it comes down to reprojecting the rasters to match resolution and coordinate systems.

Input: two-dimensional triangulated *dataset*2, three-dimensional triangulated *dataset*3
Output: *overlay*, three-dimensional dataset

```
1  Function intersection(trianglesA, trianglesB) is
2      triangles2, triangles3 ← classify(trianglesA, trianglesB)
3      overlay ← []
4      plane2 ← get_plane(any of triangles2)
5      foreach tri2 ∈ triangles2 do
6          foreach tri3 ∈ triangles3 do
7              tri23 ← project(tri3, plane2)
8              plane3 ← get_plane(tri3)
9              itris2 ← triangulate(intersect(tri2, tri23))
10             foreach itri2 ∈ itris2 do
11                 //optionaly label the projected itri2 with tri2 and tri3
12                 overlay ← project(itri2, plane3)
13     return overlay
14
15 Function compute_overlay(nodeA, nodeB) is
16     if disjoint(nodeA.bbox, nodeB.bbox) then
17         return ∅
18     if is_leaf(nodeA) & is_leaf(nodeB) then
19         return intersection(nodeA.triangles, nodeB.triangles)
20     if is_leaf(nodeA) & ¬is_leaf(nodeB) then
21         return ∪child∈nodeB compute_overlay(nodeA, child)
22     if ¬is_leaf(nodeA) & is_leaf(nodeB) then
23         return ∪child∈nodeA compute_overlay(nodeB, child)
24
25     nodeSml, nodeLrg ← sort_by_bbox_size(nodeA, nodeB)
26     return ∪child∈nodeLrg compute_overlay(child, nodeSml)
27
28 tree2 ← build_rtree(dataset2)
29 tree3 ← build_projected_rtree(dataset3)
30 return compute_overlay(tree2.root, tree3.root)
```

Algorithm 1: Two-to-Three-Dimensional Overlay Mapping

Visual attribute	Quantitative	Ordinal	Selective	Associative
Position	✓	✓	✓	✓
Size	✓	✓	✓	
Brightness		✓	✓	
Texture		✓	✓	✓
Hue			✓	✓
Orientation			✓	✓
Shape			✓	✓

Table 3.2: Organization levels of visual attributes [38]

3.4 Visual Elements and Attributes

This section describes the mapping of the data to the visual elements. According to Munzer [52], it is necessary to consider the properties of the visual elements in terms of the following qualities:

Accuracy how the perceived value correlates with the actual data,

Discriminability allows discriminating between value categories,

Separability how individual visual channels interfere,

Popout how quickly can the encoding be spotted.

Additionally, it is possible to characterize the visual channels in terms of organizational structures they can convey [38], see table 3.2:

Associative does not require multiple instances to convey the value,

Selective allows discriminating between value categories,

Ordinal allows comparing values,

Quantitative allows computing the amount of difference between values.

Following the expressiveness principle, matching channel and data characteristics is desirable. [52] The nature of spatial data limits the use of position or orientation to the expected use cases. Table 3.3 illustrates appropriate matching of data representations and visual elements. The table also illustrates which visual attributes can convey nominal and ordered data attributes well.

Color is the common attribute among all of the available visual elements. Telea [38] suggests a set of rules to ensure optimal color mapping; however, the invertibility rule generally applies to the entire mapping process. The principle suggests that optimal mapping should allow deriving the original values from their visual representation. The visualization should be accompanied by a legend to ensure this principle is followed.

representation		visual elements	visual attributes encoding	
			nominal values	ordered values
	point	glyphs	color, shape	color, size
vector	line	lines, curves	color, type, thickness	color, thickness
	area	polygons	color, texture	color, texture
raster		pixels, grid cells	color	color

Table 3.3: Visual Elements and Visual Attributes Mapping

3.5 Data Processing

The raw computational power is often insufficient to deal with large quantities of data. The increasing amounts of gathered data require a special memory and performance management approach. Similar problems arise when the amounts of data are not overwhelming, but the availability of computational resources is limited.

Out-of-core memory. Suppose it is impossible to fit the entire dataset into memory. This situation arises when processing sizeable topographical areas, dense datasets, or rich metadata sets. As presented in [40], the idea is to process the data segment by segment. The data needs to be stored in a structure that will allow bulk-loading to optimize the performance. The goal is to minimize the number of reads from the external memory. According to [41], this approach can be used only by separable, result invariant, and mappable algorithms:

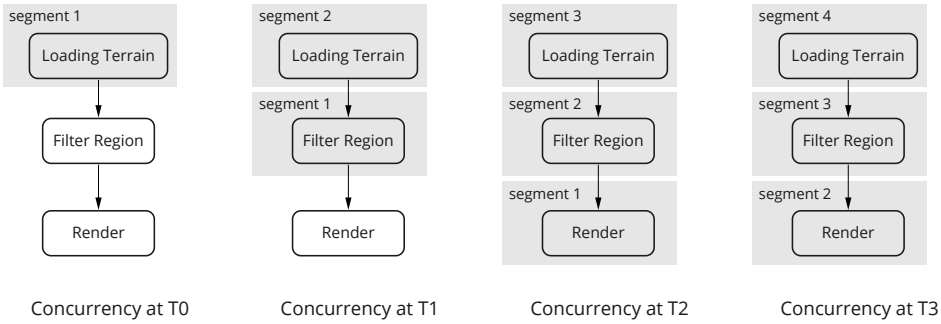
Separable the algorithm requires only a segment of the data at a time,

Result invariant the order of the data segments does not matter,

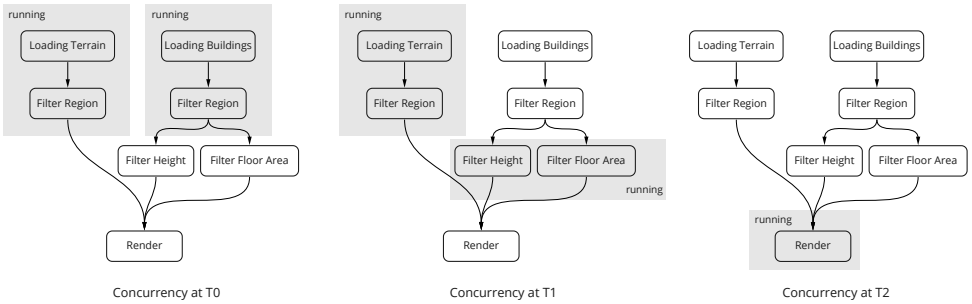
Mappable required inputs are identifiable for each output.

Parallelization. As mentioned in section 3.1, there are several approaches to parallelization of the visualization pipeline, see [44]. Data parallelism runs several pipeline instances, effectively processing multiple data segments concurrently, see figure 3.8a. Task parallelism focuses on finding which parts of the visualization pipeline can run in parallel, see figure 3.8b. Pipeline parallelism builds on the out-of-core memory principle and runs several pipeline stages in sync with the data loading procedure, see figure 3.8c.

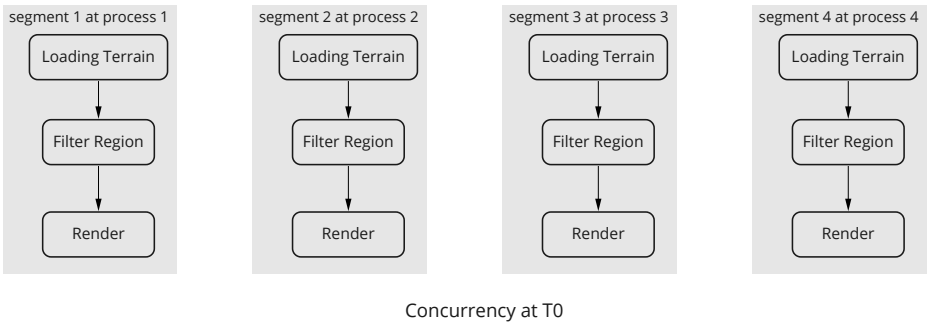
Prioritization. Often only a portion of the datasets is in focus. This situation presents an opportunity to prioritize processing of the visible areas. Furthermore, visibility is not the only available criterium. Ahrens and Desai [42] suggest a set of rules to ensure optimal prioritization. In summary, the prioritized areas should possess at least one of the following characteristics: being close to the observer or least likely to get culled, containing values in the range of interest, or containing highly variable data.



(a) : Data Parallelism Example



(b) : Task Parallelism Example



(c) : Pipeline Parallelism Example

Figure 3.8: Parallelism Examples, inspired by [39]

3.6 Summary

The classical visualization pipeline can be utilized for geographical data visualization; however, it might be necessary to employ additional data processing techniques — out-of-core memory, prioritization, or parallelization — to deal with the large amounts of data. The principles of visual analytics present a way of integrating visualization with analytical models, creating a collaborative and interactive simulation tool. The shortcomings of this approach include a lack of validation standards and a potential for creating privacy issues. Finally, vector and raster data representation can be utilized for spatial data representation; two approaches for vector data processing were suggested.

Chapter 4

Geospatial Information Systems

This chapter presents an overview of existing geospatial formats and tools. Further analysis of the formats focuses on the utilized data model and representation capabilities. The selection of the analyzed formats is based on their occurrence at Geoportál hl. m. Prahy [53]. The final section of this chapter contains a brief survey of the available geospatial information systems, visualization tools, and frameworks.

4.1 Formats

Previous chapter outlined the basic concepts of geospatial data representation and manipulation. This section takes this topic one step further and introduces some of the most common geospatial data formats. There are several factors worth considering for each format — the purely technical aspects (internal data structure, data compression), adoption (loader/writer availability and support), and the utilized data model (semantic structure of the data).

The table 4.1 provides a summary of the utilized formats available at Geoportál hl. m. Prahy [53][1]. The data is divided into 38 categories; some contain more than one dataset. The format occurrence is weighted by the number of datasets. Shapefile, CityGML, GeoJSON, and DXF are the most utilized file formats, and the vast majority of the publicly available data utilizes vector representation. Without exception, all vector data is available in the Shapefile format. The following section describes these file formats in more detail.

4.1.1 Esri Shapefile

Esri Shapefile [54] is a vector format with a database-like structure illustrated in figure 4.2. The actual data is divided into several files and stored in a binary format, see table 4.2. The files do not contain any topological information; it is necessary to recreate it manually from the geometry description. Attributes are tied directly to the geometry. The data model allows linking multiple geometries to the same attribute. The format structure suggests it would

[1] Evaluated at 4. 1. 2021, updated at 27. 12. 2021

Topic	Dataset #	Raster representation			Vector Representation							
		JPG	TFW	TIFF	SHP			GeoJSON	GML	DGN	DWG	DXF
					2D	multipatch	polygonZ					
Absolute Building Height	1	-	✓	✓	-	-	-	-	-	-	-	-
Relative Building Height	1	-	✓	✓	-	-	-	-	-	-	-	-
Terrain and Buildings Absolute Height	1	-	✓	✓	-	-	-	-	-	-	-	-
Building 3D Model	1	-	-	-		✓	✓	-	-	✓	✓	-
Digital Surface Model	1	-	✓	✓	-	-	-	-	-	-	-	-
Digital Terrain Model	1	-	-	-	-	-	✓	-	-	✓	✓	-
Bridges 3D	1	-	-	-		✓	✓	-	-	✓	✓	-
Police Stations Utilities and Regions	2	-	-	-	✓	-	-	✓	✓	-	-	✓
Networks	10	-	-	-	✓	-	-	-	✓	-	-	✓
Bike Roads - Signs	1	-	-	-	✓	-	-	✓	✓	-	-	✓
Bike Roads - Tracks	1	-	-	-	✓	-	-	✓	✓	-	-	✓
Road Maintanance	2	-	-	-	✓	-	-	✓	✓	-	-	-
Parking Meters	1	-	-	-	✓	-	-	✓	-	-	-	-
Parking Lots	1	-	-	-	✓	-	-	✓	-	-	-	-
Reserved Parking Lots	5	-	-	-	✓	-	-	✓	-	-	-	-
Pedestrian Paths	1	-	-	-	✓	-	-	-	-	-	-	✓
Public Transport	6	-	-	-	✓	-	-	✓	✓	-	-	-
Roads	1	-	-	-	✓	-	-	✓	✓	-	-	-
Geology	2	-	-	-	✓	-	-	✓	✓	-	-	✓
Noise maps	4	-	-	-	✓	-	-	✓	✓	-	-	✓
Noise Reduction Utilities	3	-	-	-	✓	-	-	✓	✓	-	-	✓
Gargabe Collection	5	-	-	-	✓	-	-	✓	✓	-	-	✓
Geographical Maps - Layout	7	-	-	-	✓	-	-	✓	✓	-	-	✓
Geographical Maps - Vector	3	-	-	-	✓	-	-	-	-	-	-	-
Geographical Maps - Raster	4	-	✓	✓	-	-	-	-	-	-	-	-
Height Contours	3	-	-	-	✓	-	-	-	✓	-	-	-
Civic Amenities	2	-	-	-	✓	-	-	✓	✓	-	-	✓
Nature Reserves, Parks, and Wildlife	4	-	-	-	✓	-	-	✓	✓	-	-	✓
Orthophoto	3	✓	-	-	-	-	-	-	-	-	-	-
Environment	9	-	-	-	✓	-	-	✓	✓	-	-	✓
Zoning Plan - Land Use	8	-	-	-	✓	-	-	-	-	-	-	-
Zoning Plan - Changes	4	-	-	-	✓	-	-	✓	✓	-	-	-
Price Map	1	-	-	-	✓	-	-	-	✓	-	-	-
Water Management Infrastructure	9	-	-	-	✓	-	-	✓	✓	-	-	✓
Floor Count	1	-	-	-	✓	-	-	-	-	-	-	-
Current state of land use	1	-	-	-	✓	-	-	-	✓	-	-	✓
Closures (transport and other)	2	-	-	-	✓	-	-	✓	✓	-	-	✓
Lookout points	2	-	-	-	✓	-	-	✓	✓	-	-	✓
$\sum_{partial} dataset\ \#$	115	3	8	8	101	2	3	73	81	3	3	65
$\sum_{total}$	115	3	8	8		104		73	81	3	3	65

Table 4.1: Format Occurance at Geoportál hl. m. Prahy [53]

be possible to store different geometry types in a single file; however, the specification explicitly states that the geometry type must be the same for all geometries.

According to the specification [54], there are 14 different geometry types supported, including no geometry (*Null shape*), two-dimensional geometries (*Point, Polyline, Polygon, MultiPoint*), three-dimensional geometries (*PointZ, PolylineZ, PolygonZ, MultiPointZ*), two-dimensional geometries with a measurement attribute (*PointM, PolylineM, PolygonM, MultiPointM*), and a type capable of representing a three-dimensional compound geometry with a measurement attribute (*MultiPatch*). The attributes stored in the DBF file are structured according to the *dBase IV* specification using a table-like schema. There are several limitations to what can be stored in the DBF file, notably:

- the maximum key length is ten characters, and the range of available characters is also limited,
- the maximum value length is 4000 characters,
- all numbers (including floating-point numbers) are stored as a string,
- and the total allowed number of attributes is 255.

The broad support of the format mitigates these disadvantages; namely, the GDAL [55] library provides reliable drivers for loading and saving Shapefiles.

File	Description	Required
SHP	geometry description	✓
DBF	attributes (table-like)	✓
SHX	index file linking the geometry with the attributes	✓
PRJ	projection information	
XML	geospatial metadata	

Table 4.2: Shapefile File Components

4.1.2 CityGML and CityJSON

GML (Geography Markup Language) is an XML-based language capable of describing the geometry, attributes, and styling rules of geographical data. CityGML [56] format utilizes GML application schema. The basic structure of the model is outlined in the figure 4.3.

CityGML introduces the concept of a semantic model isolated from the actual geometry, see figure 4.1. Individual objects correspond to classes and can be represented by a set of geometries with a different level of detail. CityGML utilizes a class-hierarchy model. The CityGML specification is under active development; the newest version, 3.0, supports static geometrical representations and object hierarchies, point clouds, time-dependent data, and versioning. CityGML standard is designed to be modular, meaning *"the implementations are not required to support the complete CityGML model in order to be conformant to the standard"* [56]. The core model can also be extended; the standard supports domain-specific extensions through ADEs (Application Domain Extensions). There is a web-friendly mutation of CityGML — CityJSON [57]. Both CityGML and CityJSON are open formats; there are converters and importers available, see [58, 59].

Both formats — CityGML and CityJSON — have a similarly extensive structure. The purpose of these formats is to capture the entire city, including the geometry, semantics, and relations, which makes the formats rather complex. This complexity makes it difficult to parse the formats without a dedicated tool. CityJSON tries to overcome this problem using a JSON-based syntax and a limited set of features; however, the overall structure and complexity remain the same. The citygml4j [59] is a java-based toolkit offering an API to access the CityGML data; a python-based toolkit cjio [61] is available for the CityJSON format.

4.1.3 GeoJSON

GeoJSON is a JSON-based format utilizing a feature-based approach — all data which belongs to a single object (*Feature*) is contained in a single JSON object. This ties the attributes (*Properties*) directly to the geometry, while the geometry is stored in place (contrary to CityJSON, which uses index-based geometry representation). The utilized data model is illustrated in figure 4.4. Features and geometries can be grouped into collections, providing

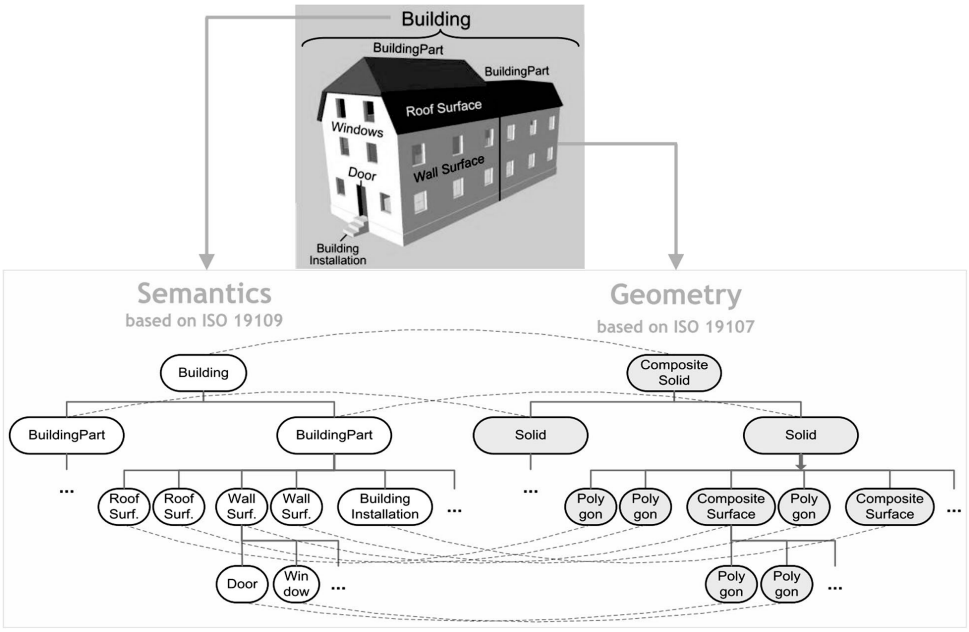


Figure 4.1: Coherence of semantics and geometry in CityGML [60]

elementary support for hierarchy representation. The format offers a very flexible specification of the attributes — an attribute can be any JSON object.

Similar to the Shapefile format, GeoJSON does not contain any topological information. According to the specification, there are nine different object types supported: 6 geometries (*Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon*), one object class (*Feature*), and two group classes (*GeometryCollection and FeatureCollection*). The GeoJSON specification explicitly forbids adding new object types; however, it allows extending the existing ones. All geometry types can store both 2D and 3D geometries. Since the format is text-based and utilizes a direct geometry representation, the file size increases quickly with the number and size of geometries. Due to the simplicity of the format, a vast majority of geospatial tools support it, including tools such as GDAL [55] or GeoPandas.

4.1.4 Other Formats

Several other formats need to be mentioned. A family of CAD formats (DGN, DWG, DXF) appeared in the table 4.1, and while these formats ranked fourth, they are highly utilized among architects and designers. In the early 2000s, the BIM standard emerged, which set the course towards a more collaborative and continuous workflow. While the BIM standard has been in development for many years, it is becoming increasingly popular. The new standard disrupts the traditional 2D workflow by integrating various data emerging throughout the life-cycle of a building.

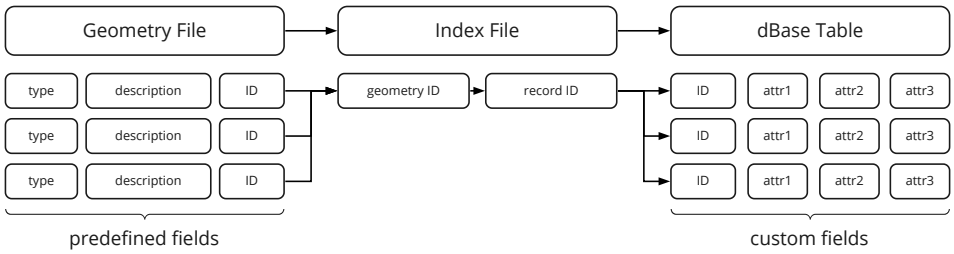


Figure 4.2: Shapefile Data Model

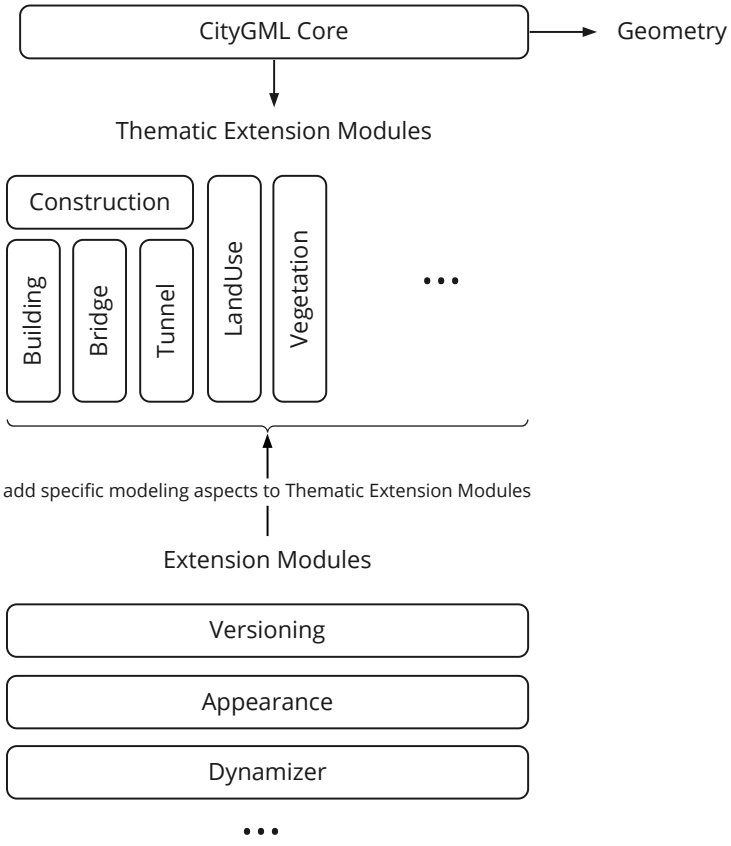


Figure 4.3: Simplified CityGML Module Overview

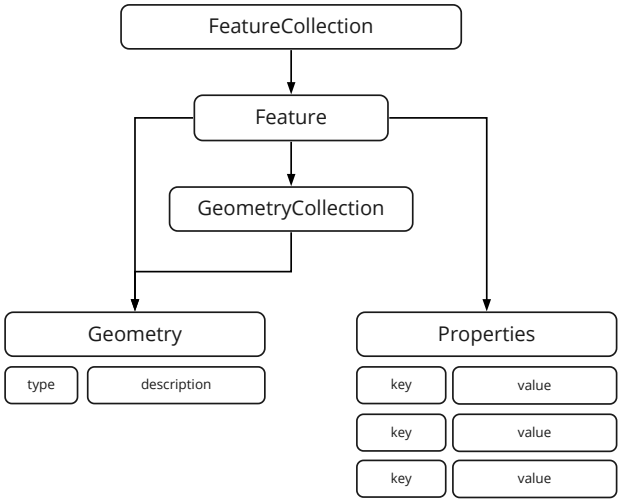


Figure 4.4: GeoJSON Data Model

DGN, DWG, and DXF. DGN and DWG are proprietary vector formats utilized by CAD editors, such as AutoCAD. The developer tools for importing and converting DWG and DGN are provided by OpenDesignAlliance (ODA) [62] and Autodesk [63]. GDAL [55] provides limited DWG and DGN loading capabilities. Unlike DGN and DWG, DXF is an open version of these formats. [64] While the DXF format has a limited set of features, open-source tools are available [65] for importing and exporting DXF files.

BIM and IFC. Building Information Model (BIM) is a standard for designing, developing, and maintaining urban projects. According to [66], the standard is designed with sharing and continual project management in mind. The complete project documentation for each development phase can be assembled using BIM, which should, in theory, speed up the designing and construction periods and reduce overall costs. The specification utilizes three-dimensional models instead of classical 2D drawings. Industry Foundation Classes (IFC) format is a platform-neutral and open file format utilized for BIM data sharing and exchange. While BIM is a future-proof standard, the format's adoption is still relatively slow, and no open data is currently available in the IFC format.

4.2 Tools

Geospatial data processing is a pretty competitive field, and many professional tools are available. The following sections present some of the most popular tools and frameworks. Some of the presented tools offer a wide range of cross-platform services, including data integration, planning, and data publishing.

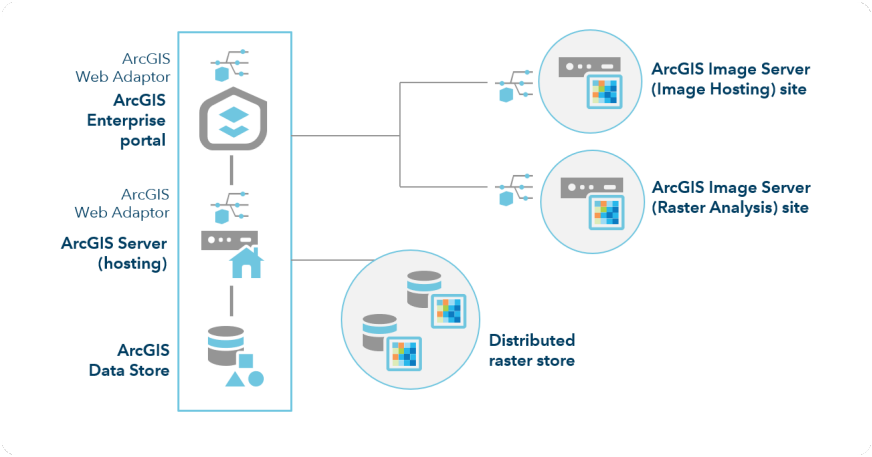


Figure 4.5: ArcGIS Enterprise deployment pattern [68]

4.2.1 Applications

This section presents a selected set of available geospatial visualization applications. These tools are either targeted at experts in the field of geospatial systems (ArcGIS, QGIS, Cesium) or provide a more straightforward way to visualize data in the spatial context (Kepler.gl, Movement). Both proprietary and open-source solutions are available. The complexity of these tools can be overwhelming and can act as a barrier to their adoption by a broader range of users.

ArcGIS. Esri offers a complete software suite for geospatial visualization and analysis under the name of ArcGIS [67]. The offered products include a full desktop environment for data analysis, server components for web map service management, and web frontend components. Complete developer documentation is available for the open segments of the ArcGIS software ecosystem. Esri offers several categories of products, including individual applications, tools for building custom map applications, and SDKs for developers to integrate Esri products into their own. A setup exposing ArcGIS services to external users is illustrated in figure 4.5. Selected products also support user scripting. As it is a commercial software with full-time support, it is a widely popular solution often utilized by local authorities.

QGIS. QGIS [69] is an open-source alternative to ArcGIS. It supports a wide range of formats (Esri Shapefiles, ArcInfo files, MapInfo files, .csv, OpenStreetMap data, PostGIS data, etc.). A QGIS plugin, Qgis2threejs, provides a way to view 3D content in the browser, but it appears that the only supported input format is .hgt - DEM (Digital elevation model) format. From version 3.0, there is built-in support for 3D content; supported formats are mostly Esri Shapefiles and raster formats (e.g., GeoTiff). It is possible to export static images and short animations created from time-series data.

Vis.gl Apps. Kepler.gl [70] is an open-source visualization application developed by vis.gl [71]. The tool was created and popularized in collaboration with Uber; scalability and good performance are the strong features of the application. The tool is based on Deck.gl, a WebGL-based rendering framework optimized for big data visualizations. The application uses a layer-based approach — a layer is an isolated entity with its base geometry (lines, arcs, points, hex or rectangular grid, heatmap, etc.) and style. The application uses OpenStreetMap [72] and Mapbox [73] as data sources for 2D maps and 3D building models. A component for displaying three-dimensional terrain is missing. It is possible to use Kepler.gl as a Python library for visualization inside Jupyter Notebooks. The supported formats include CSV, GeoJSON, and proprietary JSON-based application format. Similarly to kepler.gl, Movement is also developed by vis.gl in collaboration with Uber [74], with the primary focus to visualize transport-related data. This project stands on the border of visualization and data analysis. Unlike the previous examples, this application was developed to present a fixed set of curated datasets. The goal is to help cities (they operate only in a limited number of cities) to reduce congestion, emissions and improve road safety.

Cesium. Cesium [75] is a platform for building geospatial applications. It provides a way to push 3D content to both web (CesiumJS) and Unreal Engine. A part of the services is a platform called Ion, which automatically tiles and optimizes the content for both web and the game engine and offers a library of assets such as images, terrain, and building models. The platform is targeted towards commercial projects and offers pre-paid plans with higher bandwidth.

4.2.2 Frameworks

In terms of visualization frameworks, there are several existing solutions. Visualization frameworks usually provide a set of features (data management, rendering, etc.) used throughout the visualization pipeline; however, the final deployment is up to the framework user.

Mapbox. Mapbox offers a range of products for geospatial data visualization on the web and in mobile apps. Besides classical static maps, Mapbox offers several modules for movement data visualization and vehicle navigation. Mapbox also offers a Studio allowing developers to customize the data visualizations similarly to Kepler.gl. All Mapbox services are offered for a fee; the final cost is determined by the number of requests and monthly users.

3DCityDB. 3DCityDB [76] is a framework oriented towards effective storing, analysis, and export of urban data. Figure 4.6 presents an application structure where the framework is utilized as a geometry and metadata source along with other services. The framework utilizes the CityGML standard as the base schema for the data representation. The framework supports export into several formats, such as Collada, glTF, and KML.

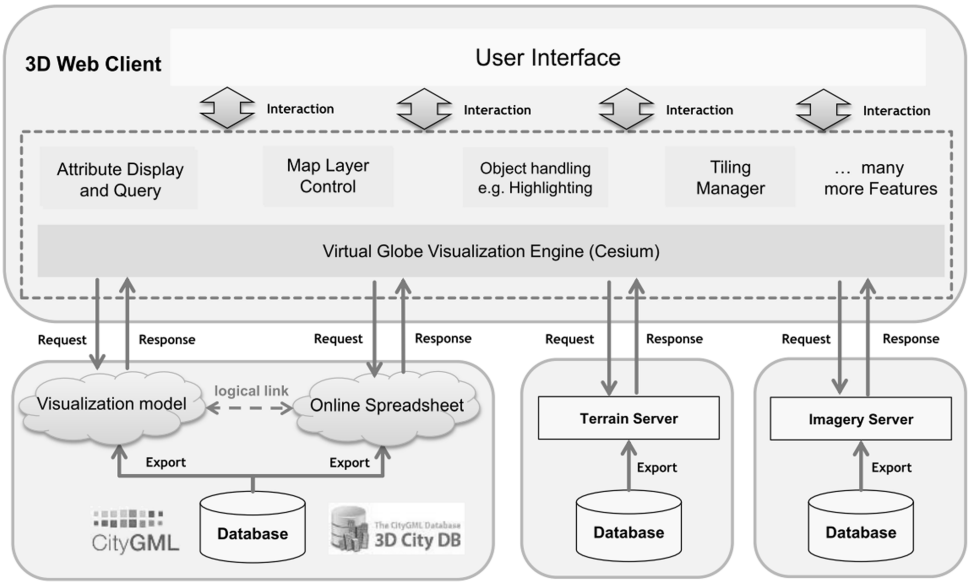


Figure 4.6: Workflow of using 3DCityDB web client coupled with Cloud-based online spreadsheets [76]

Vis.gl Frameworks. Vis.gl develops several web-based frameworks for visualization, see [71]. Deck.gl is a WebGL-based rendering framework for visual exploratory data analysis of large geospatial datasets, compatible with React and Mapbox. A more general-purpose visualization framework is Luma.gl, also a WebGL-based toolkit. To provide a way for plotting various data in 2D, vis.gl developed the React-vis library.

LuciadRIA. Hexagon offers a portfolio of geospatial visualization tools under the name of Luciad [77]. The visualization tools are mainly targeted at the web environment; the visualization toolkit LuciadRia utilizes state-of-art web technologies, including WebAssembly and WebWorkers. The toolkit is proprietary, and the documentation is targeted mainly towards end developers using the framework to assemble their applications. The framework handles various formats, including BIM and point cloud data.

4.3 Summary

The commonly utilized geospatial formats include Shapefile, GeoJSON, CityGML, and DXF. The concept behind CityGML — decoupling geometry and metadata — is beneficial because it offers a flexible approach to data versioning and integration. Standards such as BIM offer a way to represent the complete project documentation in a comprehensible and sharable way; although, the adoption of this standard is not yet observable in the open data field.

A range of geospatial information systems and visualization frameworks is available, the summary of their capabilities and limitations is presented in

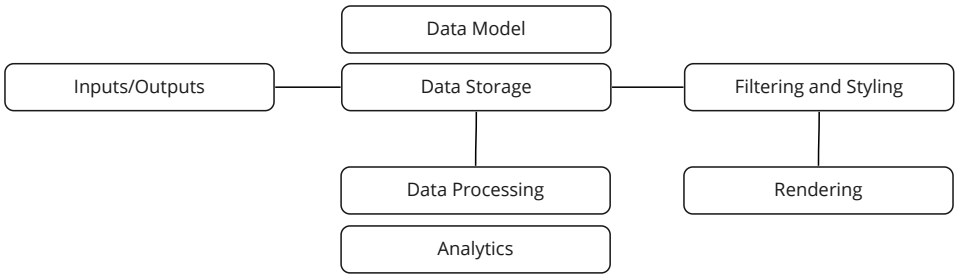


Figure 4.7: General schema of a geospatial data processing and visualization system

table 4.3. Some of these tools offer services beyond data visualization, e.g., modeling tools and data publishing. Most of these tools support visualization on the web, which makes the content easily accessible and collaboration with other users simpler. Typical limitations include high system complexity or limited visualization pipeline implementation.

Based on the presented examples, it is possible to devise a general architecture of a geospatial visualization tool. The central component manages the data representation, while additional tools service the required inputs and outputs. Some of the more extensive toolkits (ArcGIS, QGIS) offer analytical tools and ways for the user to create custom data processing scripts. An independent module manages the visualization styles (the mapping step of the visualization pipeline). The rendering is handled separately. The general schema is illustrated in figure 4.7.

	modularity and customizability	accessibility	platform requirements	limitations	notes
ArcGIS	addons, application and service ecosystem, public API for scripting and styling, stylesheets, styling UI	desktop, web	Windows OS, cross-platform viewer	toolkit complexity, proprietary code	supports content editing, includes modeling tools, includes data publishing services
QGIS	plugins open-source code, public API for scripting, styling UI	desktop, web (limited)	cross-platform	complex UI, limited web support, publishing data requires additional service (QGIS Cloud)	supports content editing
Kepler.gl	open-source code public API for styling styling UI	web	cross-platform	fixed map sources (Mapbox, OpenStreetMaps), 2.5D only	-
Movement	-	web	cross-platform	not available for developers	works as a datastore/marketplace with visual frontend
Cesium	open-source code, public API for styling, styling UI	web	cross-platform	focused on graphics only, limited metadata	-
Mapbox	public API for styling, styling UI	web, API for mobile	cross-platform	proprietary services, 2.5D only	developer-friendly architecture
3DCityDB	open-source code, Cesium Virtual Globe for visualization	web	Oracle or PostgreSQL, cross-platform viewer	uses Cesium for visualization	-
Deck.gl	open-source code, public API for styling, React-friendly API	web	cross-platform	JS library only	-
LuciadRia	public API for styling, compatible with any tech stack	web	cross-platform	proprietary code	supports online content editing

Table 4.3: Summary of Application and Framework Capabilities and Limitations

Chapter 5

Design

This chapter presents the design of the visualization system. The goal is to develop a modular system for urban data processing and visualization. The thesis assignment lists factors that need to be considered; each factor influences several aspects of the system. The factors to consider include:

- selection of supported input data formats
- visualization customizability
- accessibility and usability
- presentation formats and media
- memory and computation efficiency
- system extensibility

Several system outputs are expected:

- virtual model presentation
- visual outputs for video mapping
- outputs for physical model construction

The first section explains the high-level system structure to make the design process more comprehensible. The remaining sections of this chapter introduce the individual system components and the utilized concepts.

5.1 System Structure

The high-level structure presented in figure 5.1 is based on the research in chapter 3 and examples in chapters 2 and 4. The design follows the general system schema presented in figure 4.7 at the end of the previous chapter. The system is divided into three main components — a user interface, a server component, and a processing toolkit. The following paragraphs offer a brief description of the individual components; more details are presented in the following sections.

Processing Toolkit. The processing toolkit serves as the baseline component of the system. The goal is to create a standalone package that can be utilized both by the system and the user in scripts or from a command line. The toolkit's purpose is to handle data imports and allow data filtering and

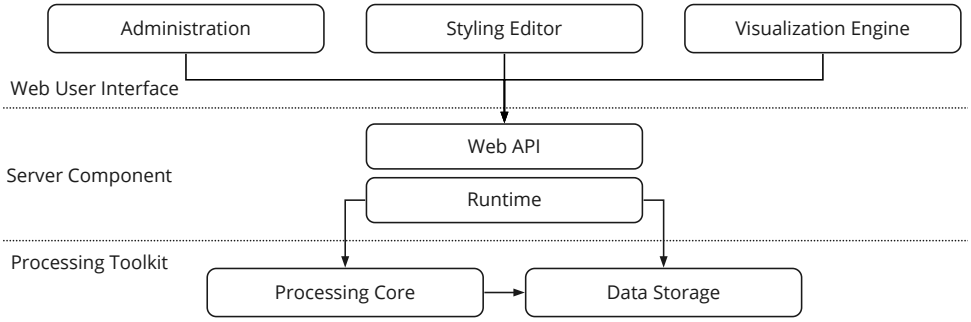


Figure 5.1: System Structure

modification. The structure of the utilized data model, the extensibility of the toolkit architecture, and the performance of the utilized algorithms and data structures are key focus areas of the design.

Web User Interface. The user interface is designed as a web app to make the visualization and data administration accessible and easily sharable. The visualization engine is designed to handle both static and dynamic data types. Styles defined in the style editor control the visualization appearance based on the present object metadata.

Server Component. The server component integrates the user interface and the processing toolkit. This component exposes a public web API and schedules data processing in the background. Several important technical details were omitted in the diagram 5.1 for clarity; the server requires supervising software to handle the interprocess communication, exceptions, etc. A description of the deployed system is provided in the next chapter.

5.2 Processing Toolkit

The processing toolkit handles geometry and metadata transformations. The data is represented using a general data model, separating metadata and geometry. The data storage utilizes an out-of-core memory approach and enables large dataset processing. The toolkit utilizes several additional spatial and temporal data structures to allow for fast data manipulation.

5.2.1 Data Model

The design of the internal data model is the primary issue. Previous research revealed that most available datasets are currently available in the Shapefile format. This observation indicates that the vector representation should be supported directly; however, the designed data model must be format-independent to remain extendable. The proposed data model, presented in figure 5.2, is designed according to the following principles:

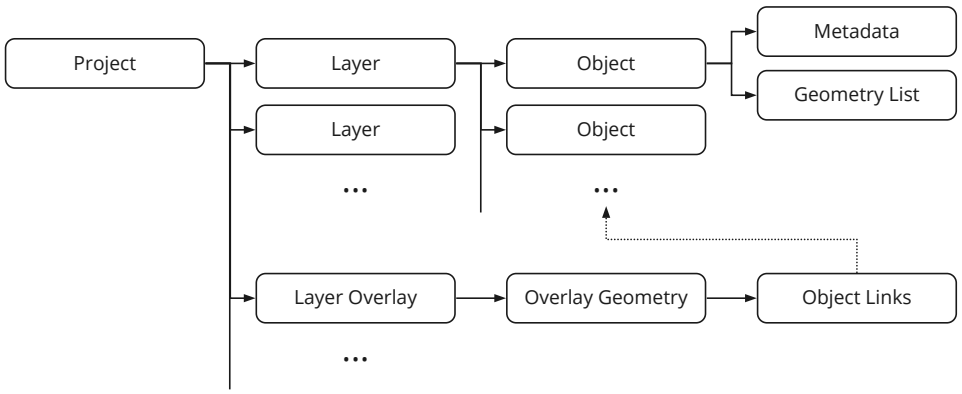


Figure 5.2: Proposed Data Model

- Format-independent data model allows the integration of various types of inputs.
- To increase cross-program compatibility, it is desirable to make the data model similar to those utilized by existing GIS and graphics software — *Projects* include *Layers* of *Objects*.
- As demonstrated by the CityGML standard, *Geometry* and *Metadata* separation offers more flexibility. Each object can be represented by a collection of geometries.
- When a *Layer Overlay* is computed, it is possible to store only the overlay geometry without duplicating the object metadata. Instead, the geometry contains links to the original objects, allowing the original metadata retrieval.

5.2.2 Data Storage

This section deals with the data storage. Utilizing a database for data storage has several advantages (e.g., scalability, performance, easy integration with web services); it also adds a system dependency and requires each user to install the database software first if they wish to use the toolkit locally. The proposed alternative, illustrated in figure 5.3, utilizes a local file system. This approach limits the ability to query data based on relations effectively; on the other hand, it removes the database dependency and can be optimized for selected operations. Some notes on the proposed approach:

- The directory hierarchy in figure 5.3 follows the design of the data model in figure 5.2 up to the layer level.
- The object geometry and metadata components are stored in separate *Data Storage Directories*, which speeds up the data retrieval when only one component is required.

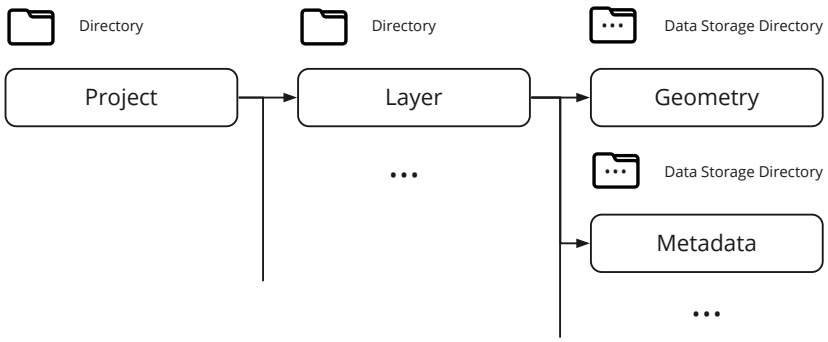


Figure 5.3: An outline of the directory structure of the local data storage; *Data Storage Directories* act as special data containers although the structure consists of regular files and directories

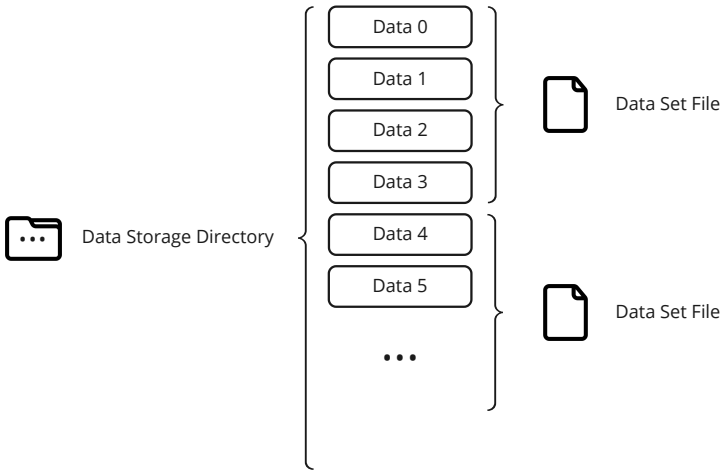


Figure 5.4: Individual data items in Data Storage Directories are packed into chunks and stored in separate Data Set Files

Data Storage Directory, illustrated in figure 5.4, is a type of directory containing data in an array-like file structure. The data management remains transparent to the user, and the storage appears as an ordinary array. The design is based on the following assumption:

- the total data size exceeds the available memory,
- the data is almost always read and updated in a sequential order,
- random access is required for the least frequent operations,
- and the items are nearly never deleted.

Figures 5.5 and 5.6 illustrate the initialization and sequential access of the storage items. An important parameter is the granularity of the data sets — one object per file is inefficient and possibly hits the limits of the file system (e.g., inode counts). Storing all objects in a single file might not be possible due to insufficient memory resources. The figure 5.7 shows the data storage design applied to geometry and metadata.

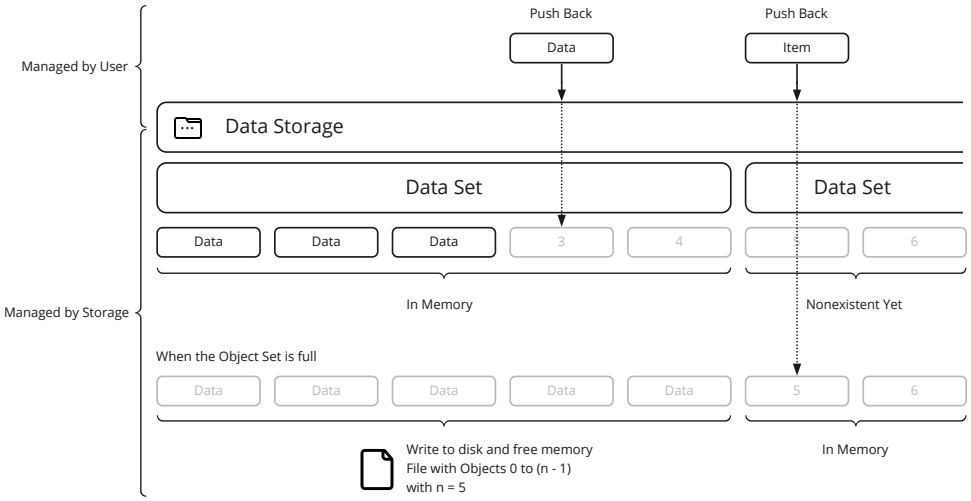


Figure 5.5: Adding data to Data Storage

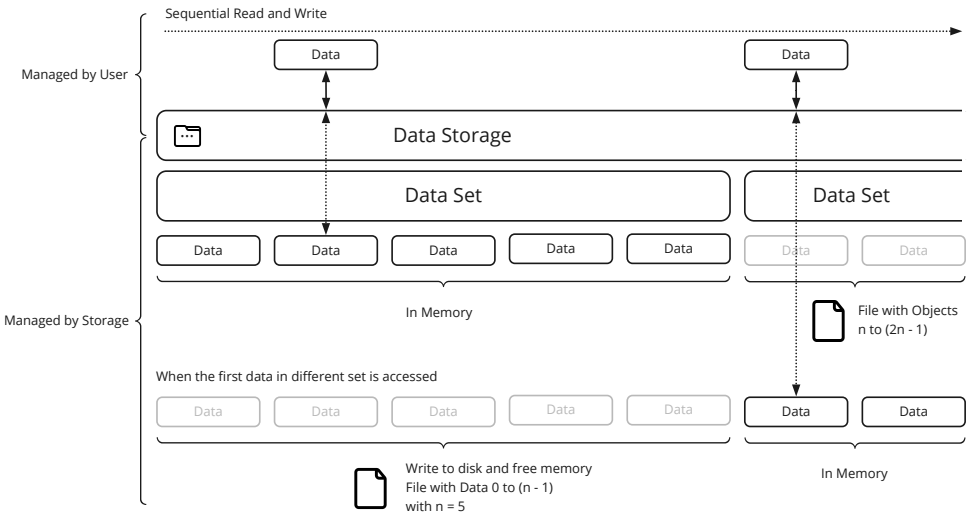


Figure 5.6: Sequential update of data in Data Storage

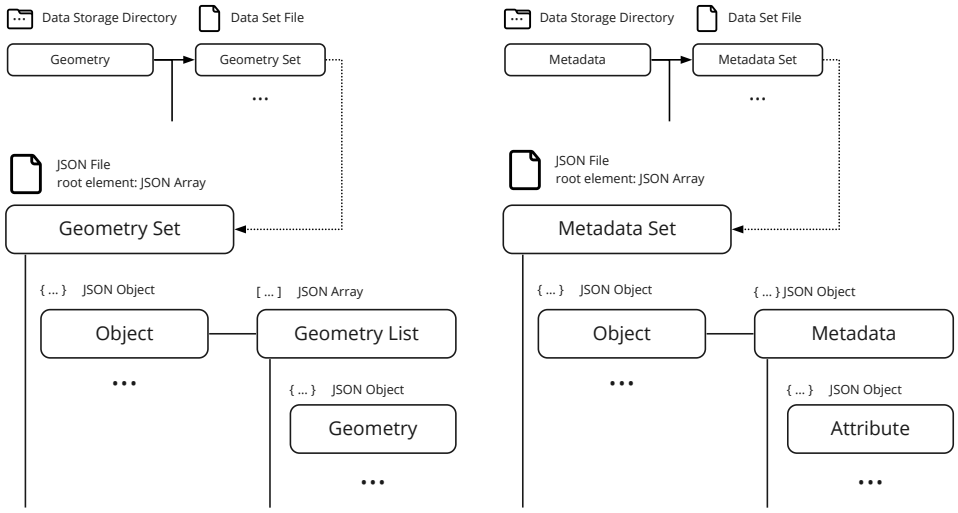


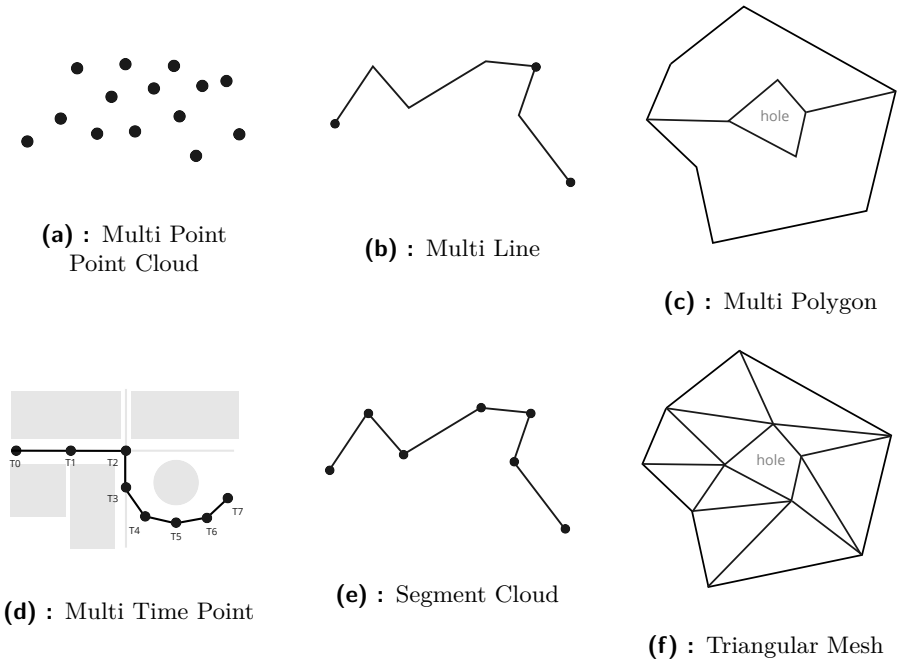
Figure 5.7: Example of Data Storage Contents

5.2.3 Geometry Representation

Similar geometric types are utilized among the formats described in chapter 4. Some types are not suitable for rendering, as described in the section 3.3.2. The preservation of the original data is desirable, as it enables precise reconstruction and re-processing of the input data. Figure 5.8 illustrates the proposed set of geometric types.

Some of the geometry types in figure 5.8 are visibly related. Each type can be directly rendered (*Multi Point*, *Segment Cloud*, *Triangular Mesh*, *Multi Time Point*) or transformed into a renderable alternative (*Multi Line*, *Multi Polygon*). *Multi Line* is a series of points bounded by beginning and ending point, while *Segment Cloud* contains a list of unconnected segments. *Multi Polygon* is a collection of arbitrary polygons (possibly with holes, as supported by Shapefile, GeoJSON, and CityGML) and can be triangulated into *Triangular Mesh*. *Multi Time Point* is an ordered sequence of points with timestamps.

The out-of-core memory approach requires the geometry representation to be persistable. Utilizing the aforementioned vector formats is not possible due to the lack of support for all expected geometry types. CityGML and GeoJSON store geometry coordinates as string values; however, a loss of precision due to string representation is undesirable. The glTF format developed by the Khronos Group suggests an alternative approach — the buffers of the geometries can be stored directly as base64-encoded strings in a JSON-like format. The base64-encoding provides a lossless geometry representation, and it is also suitable for streaming into the web visualization engine.



(a) : Multi Point Point Cloud

(b) : Multi Line

(c) : Multi Polygon

(d) : Multi Time Point

(e) : Segment Cloud

(f) : Triangular Mesh

Figure 5.8: Supported Geometry Types

5.2.4 Regular Grid

None of the concepts so far directly addressed the spatial layout of the data. The array-like approach utilized by data storage allows pre-sorting the objects based on their position; however, utilizing a spatial structure would improve the performance in the following situations:

- processing simple geometry queries based on position or range,
- and map overlay computation.

The first expected query type can be accelerated using a regular grid — it is a relatively simple yet sufficient structure. The second use case — map overlay — is described in a separate section; however, the general idea is first to query the overlaying grid tiles and then proceed with the overlay computation.

It is necessary to clip the geometry to the tile boundaries to avoid triangles or lines spanning multiple tiles. The naive grid construction algorithm could iterate over all objects, clip their geometry and assign the geometry slices to the appropriate tiles. The algorithm would effectively load the entire dataset into memory.

A more memory-efficient approach is presented in figure 5.9. While the outline of the algorithm remains the same, the intermediate geometry slices are first stored in a tile cache utilizing an out-of-core data storage. The cache

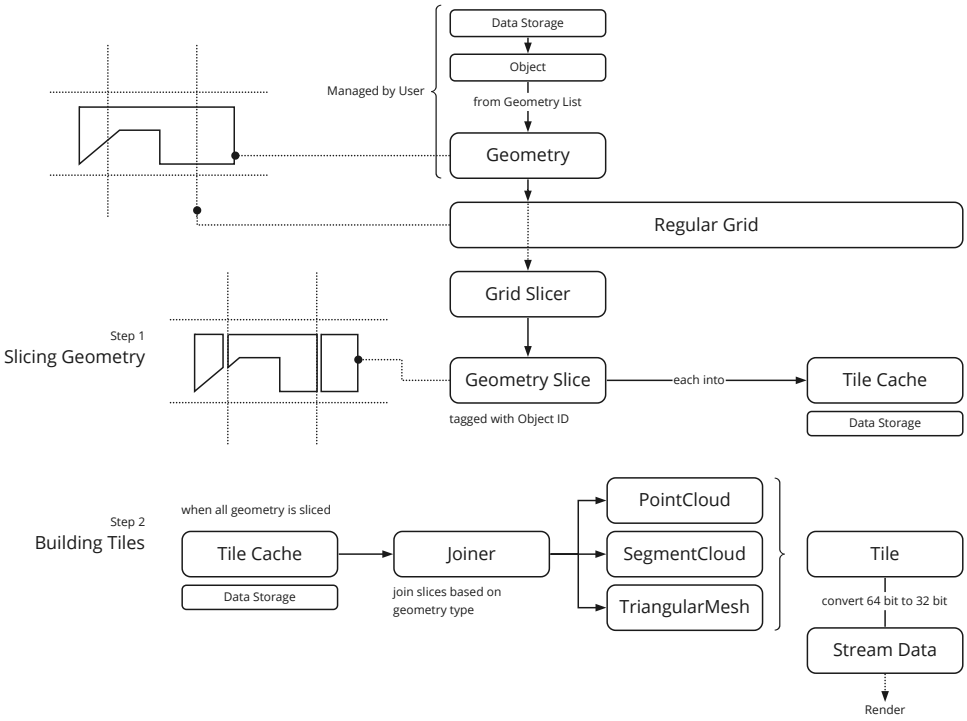


Figure 5.9: Grid Construction

allows virtually keeping all tiles in memory, while the actual data is mainly stored on a disk. The final step — building tiles from cache — requires joining geometry slices of the same type for each tile. The output tile contains up to three renderable geometry instances — one of each renderable type. Thus, the rendering engine can render the tile contents directly without further transformations.

It is possible to tag the geometry slices with the original object identifiers before storing them in the cache. It consequently allows the reconstruction of the original objects. The identifiers also allow picking and highlighting selected objects in the visualization.

5.2.5 Timeline

The queries dealing with spatio-temporal data face similar issues as the spatial queries. The expected spatio-temporal query involves locating positions of objects, e.g., agents or vehicles, at a given time. The query can be constrained by a spatial range.

As the regular grid consists of same-sized tiles, the proposed timeline acceleration structure consists of equally-sized intervals. The intervals store the location of each moving object in each second. A linear motion of the object between the specified time points is assumed. The timeline construction algorithm, illustrated in figure 5.10, is similar to the grid construction; it also utilizes identical memory optimization methods.

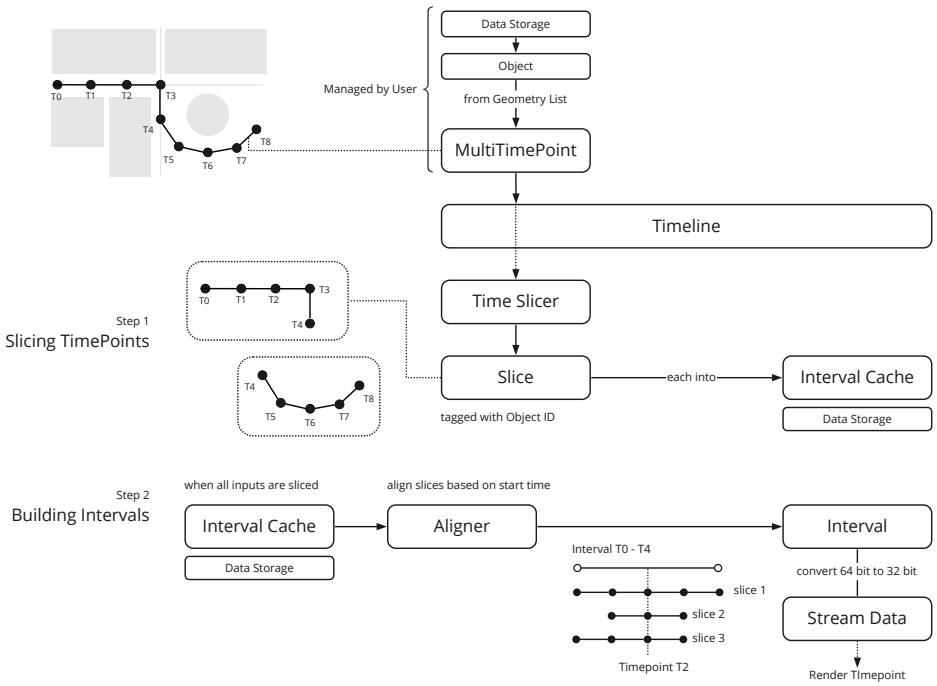


Figure 5.10: Timeline Construction

The supported temporal geometry types include only Multi Time Points. The structure of intervals is optimized for data streaming into the visualization engine. As the visualization client requests a time window, the precomputed object positions can be quickly retrieved and directly passed to the rendering engine. The interval length is configurable. The timeline structure can be combined with the regular grid, creating a spatio-temporal regular grid.

5.2.6 Overlay Mapping

This section deals with the overlay mapping in the context of previous design decisions. The overlay mapping focuses on preserving the relations between the input and output geometries since it allows filtering, styling, and modifying the overlay geometry based on the metadata of the original objects. Various approaches to overlay computation have been discussed in section 3.3.2. The approach utilizing triangulated data is preferred for the following reasons:

- the regular grid structure containing renderable geometry types can further accelerate the overlay mapping,
- and preserving relations between the triangular overlay and original geometry is simpler.

The second argument is not as obvious. The section 3.3.2 introduced `MapOverlay` algorithm that computes the overlay and links each output polygon to the input overlaying it. Since the links do not distinguish between

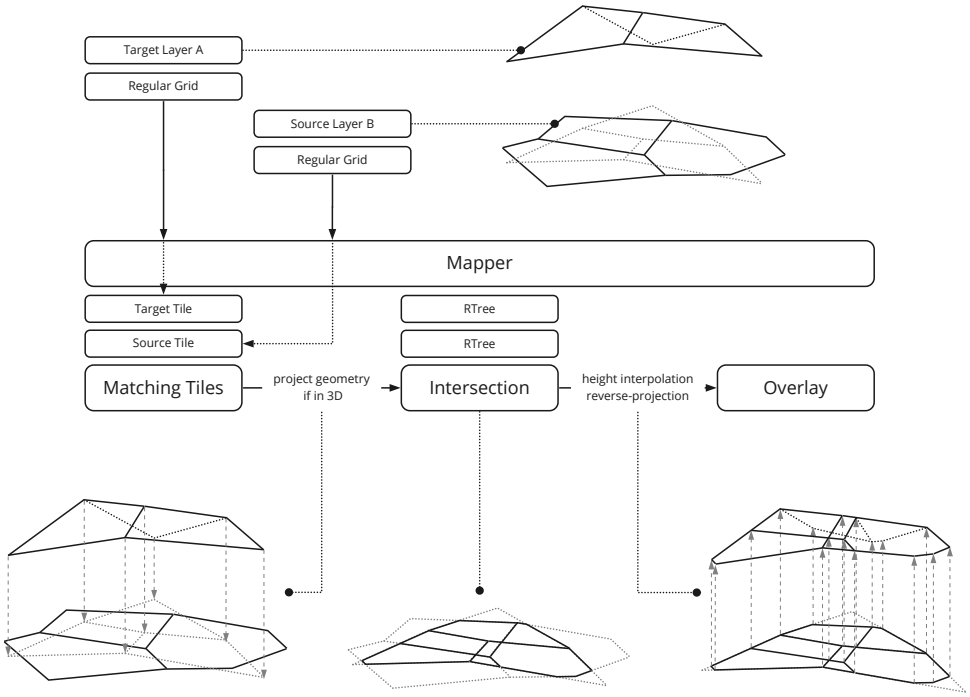


Figure 5.11: Layer Overlay Mapping

the polygons of the input, the `MapOverlay` needs to be extended to preserve more detailed relations, and additional effort is required.

The pseudocode 1 in section 3.3.2 suggested how each overlay triangle can be linked to the original geometry. Since the geometry in the regular grid is tagged with the object identifiers, all necessary information is available. The section 3.3.2 also suggested how to deal with degenerate cases. The pseudocode 1 can be further generalized to support mapping of different geometry types. The overlay mapping illustrated in figure 5.11 is applicable to four types of mappings:

- Multi Time Point to Triangular Mesh (Point-Triangle intersection),
- Point Cloud to Triangular Mesh (Point-Triangle intersection),
- Segment Cloud to Triangular Mesh (Segment-Triangle intersection),
- and Triangular Mesh to Triangular Mesh (Triangle-Triangle intersection).

If the input geometries are stored in matching regular grids, the output overlay geometry can be directly stored in a regular grid with identical parameters.

5.2.7 Styling

This section deals with the customizability of the visualization. The key idea is to change the geometry appearance based on the object metadata. As presented in section 3.4, the common visual attribute of all geometries is color. The color can convey both nominal and ordered values. For demonstrational purposes, this thesis limits the stylable properties to color. Each object

```
layer_rule_list     ::=  layer_rules* | layer_rules* legend
layer_rules         ::=  @layer ( STRING ) { rule* }
rule                ::=  layer_color   |  mapping   |  meta_rule
layer_color         ::=  @color :  COLOR ;
mapping             ::=  source   |  target
source              ::=  @source { meta_rule* }
target              ::=  @target { meta_rule* }
meta_rule           ::=  @meta ( name_link ) { (style ;)* }
name_link           ::=  STRING ( .  STRING )*
style               ::=  key_style | range_style
key_style           ::=  key :  COLOR
range_style         ::=  range :  COLOR+
legend              ::=  @legend { (legend_style ;)* }
legend_style        ::=  key_style | legend_range_style
legend_range_style  ::=  STRING : range :  COLOR+
key                 ::=  STRING | @default
range               ::=  [ SIGNED_NUMBER SIGNED_NUMBER ]
```

Grammar 5.1: The terminals are highlighted in black; nonterminal symbols are light gray. The notation utilizes an asterisk as a denotation of any number of occurrences; plus sign as one or more occurrences; round brackets in gray denote a group of symbols.

represents a virtual structure, and based on the structure properties; it is assigned a color. A prime example of a similar styling system is Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) — HTML describes the structure, while CSS describes the appearance. The proposed styling mechanism is based on identical principles.

Grammar 5.1 describes the designed styling language. The language supports both static colors as well as the definition of colormaps. Additionally, it is possible to define a legend displayed along with the visualization. The language uses the following keywords:

`@layer` scope of styles applied to a selected layer
`@meta` scope defining metadata attribute styling rules
`@color` default for a selected layer
`@default` default for a defined metadata attribute (no other rules match)
`@source` and `@target` scope of overlay source or target layer metadata
`@legend` scope of legend entries

The styling language grammar is easily extensible. Upon submission, a context-free grammar parser generates a styling table based on the predefined grammar and provided user styles. As illustrated in figure 5.12, the color of each object is evaluated and stored in a separate object-color map. When the visualization engine requests the style to be applied, the visible geometry is traversed, and the colors are assigned based on the geometry object identifiers and the object-color map.

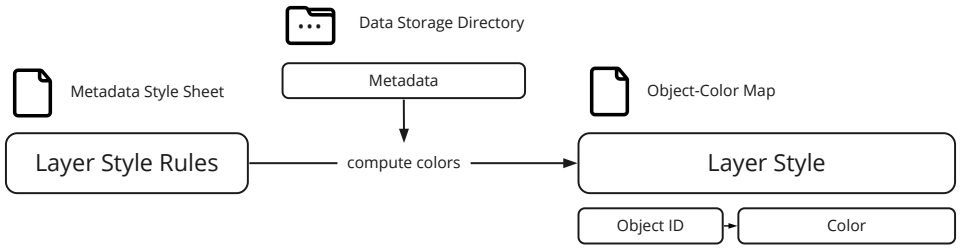


Figure 5.12: Style Schema

Examples. The first example defines styling rules for the layer *Usage*. All objects with the metadata attribute *type* equal to *roads* will render gray, *water* features will be blue, and *trees* will turn green. The rest of the objects in the layer *Usage* (objects without a defined *type*) will turn red.

```
@layer (Usage) {
      @color: #FF0000;
      @meta(type) {
          @default: #FFFFFF;
          "roads": #999999;
          "water": #0000FF;
          "tree": #00FF00;
      }
}
```

The second example illustrates a simple colormap. All objects in the layer *buildings* with the attribute *tenants* will be colored according to the linear colormap. The colormap ranges from 0 (red) to 100 (blue), and the color is extrapolated for the values beyond those limits.

```
@layer (buildings) {
    @meta(tenants) {
      [0 100]: #FF0000 #00FF00 #0000FF;
    }
}
```

The final example illustrates the styling of the layer overlay. The layer overlay *3D Usage* utilizes the metadata of the source layer. The geometry overlayed over the source layer objects with a metadata attribute *type* is colored using the same color schema as in the first example.

```
@layer ("3D Usage") {
      @source {
          @meta(type) {
              "roads": #999999;
              "water": #0000FF;
              "tree": #00FF00;
          }
    }
}
```

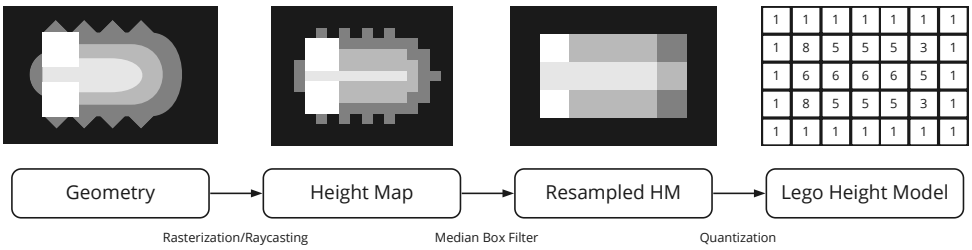


Figure 5.13: Transforming Geometry into LEGO

5.2.8 Outputs

The final required output involved the generation of physical models. Two approaches were considered — 3D printing and LEGO models.

3D printing. The system has to slice the geometry and provide geometry files in a printer-compatible format (e.g., STL, OBJ). Since the regular grid construction already requires the slicing functionality, it can be reused. The geometry file for a selected area can be created on-demand.

LEGO models. Generating the LEGO models is a more complex task. Several approaches were considered; however, the scale of the test models limited the available detail. Instead, the proposed design utilizes a heightmap, illustrated in figure 5.13. First, the heightmap is rendered with a resolution larger than the brick resolution of the output. The heightmap is then filtered using a box filter. The filtered heightmap is quantized to match the scale of the bricks to the original geometry. The choice of a suitable box filter is discussed in the following chapter.

5.2.9 Summary

The processing toolkit utilizes a data model organizing *Objects* into *Layers* and *Projects*. The semantic and geometrical object representation is separated. The data is stored directly in the file system; a complete overview of the directory structure is provided in figure 5.13. The toolkit uses out-of-core memory to enable processing of extensive datasets. This principle is reused at several points in the toolkit. A range of geometry types is supported, including dynamic point data.

The toolkit utilizes several structures to enable efficient data streaming and accelerate spatial or temporal range queries. Datasets can be integrated using the introduced overlay mapping algorithm to preserve the relations between overlay geometry and original objects. The appearance of the geometry can be customized based on the object metadata using a custom styling language. Finally, the toolkit can generate resources for 3D printing and LEGO model assembly.

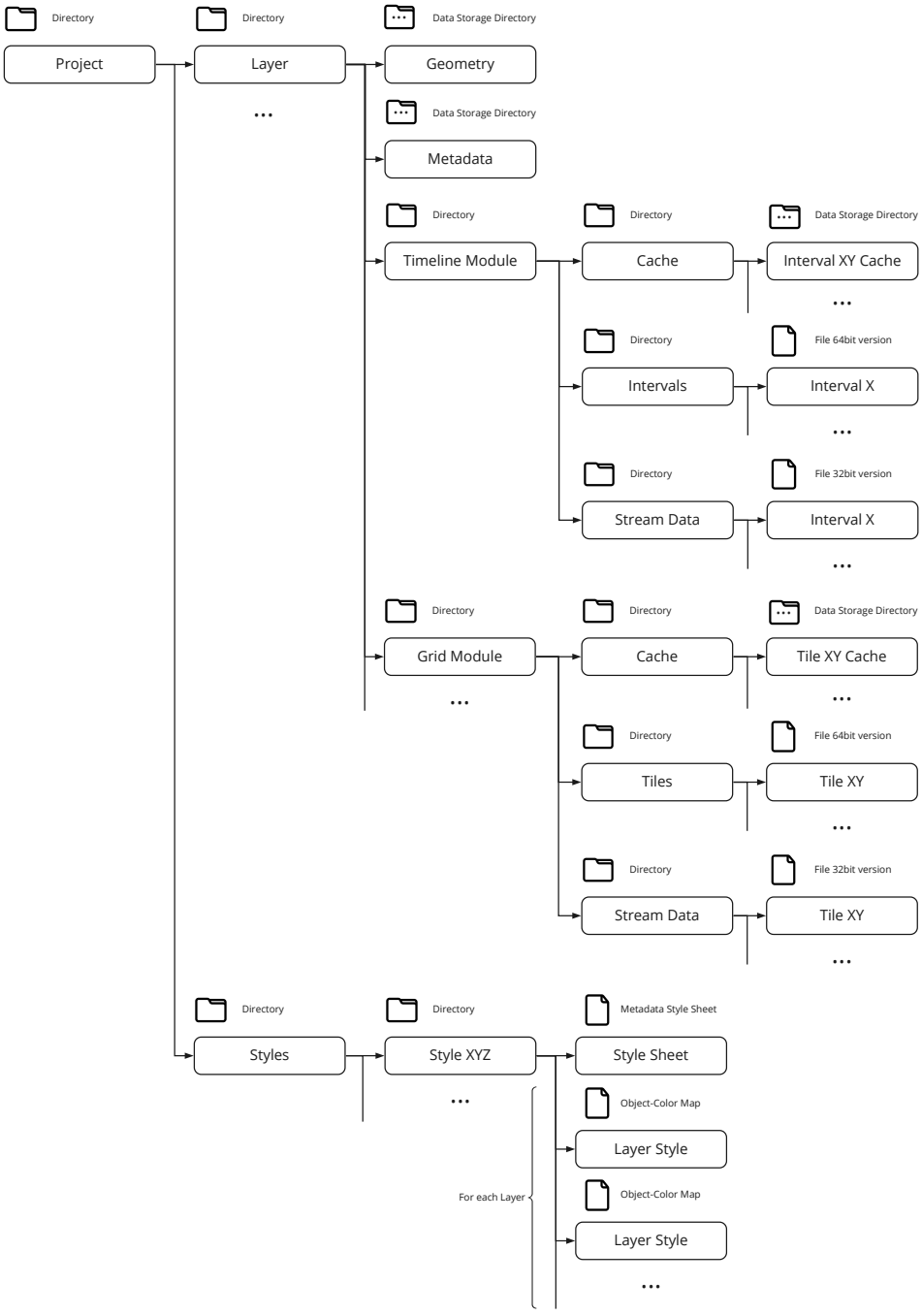


Figure 5.14: Complete Schema of the File System Structure

5.3 Visualization Engine

The visualization engine is a system component responsible for the visual outputs. The proposed software architecture of the engine is illustrated in figure 5.15. The engine design needs to adjust to the web technology limitations — capped frame rate, limited memory, and limited performance. The user interface was designed using an iterative approach; the proposed design was consulted with the participants of a user study. The results of the study are presented in section 6.3.

5.3.1 Data Processing Principles

The architecture of the engine takes into account the data processing approaches suggested in section 3.5 and the limitations imposed by the web technologies. The architecture design is based on the following principles:

- The engine architecture mirrors the layout of the processing toolkit components, as well as the data model structure.
- The engine needs to prioritize processing the tiles close to the viewer to those further away (the prioritization principle).
- The loaded data needs to require as little proce ssing as possible. Since the data representation utilized by the processing toolkit stores geometry in base64 buffers, the only processing required prior to rendering is decoding the buffers.
- The individual geometries are independent, which allows the engine to process them in parallel (data parallelism approach).
- All metadata is stored on the server only and presented on-demand.

5.3.2 Engine Components

The presented architecture of the visualization engine contains three primary submodules — Project, Decoder Worker Pool, and Renderer component.

Project. The Project component manages the visualization data. It is responsible for resource management, loading geometry, and applying styles. Each project Layer contains a Grid and Timeline class. The Grid manages the static geometry, requests tile geometry on viewpoint change, and supplies the decoded data to the Renderer. Timeline similarly manages the dynamic data. The data is dynamically loaded based on the viewer's current position. Two approaches are possible:

- Download the data, store it as long it is visible, free the memory resources when it leaves the visibility radius. This approach limits the memory footprint for the cost of higher data transfer.

- Download the data once and cache it in local memory. When requested again, the data is served from the local memory. This approach limits the data transfer and increases the memory footprint.

Since it is not clear which approach is preferable, the engine should allow the user to select the preferred approach.

Decoder Worker Pool. Decoder Worker Pool transforms the base64 encoded buffers into data arrays that can be passed to a graphic library directly for rendering. Each geometry consists of a list of buffers (vertices, normals, object ids, etc.). Since the buffers can be decoded independently, it is possible to distribute the decoding among several threads. One of the main limitations of the web environment is the default single-threaded code execution. Decoding the data in the main thread would throttle the performance and block the user interface updates, ultimately freezing the system. An alternative approach is proposed utilizing Web Workers. Web Workers are now widely supported, providing a way to run several threads in the browser. Moreover, the Web Worker implementation allows the copy-less transfer of the data array buffers, reducing the memory footprint. The figure 5.16 illustrates how the decoding jobs are submitted to a job queue. When all buffers belonging to a job are decoded, the Decoder returns the decoded geometry to the tile or interval.

Renderer. The Renderer component encapsulates the graphics library calls, handles camera controls and picking. The technical options for facilitating a web visualization are quite limited — all of the existing visualization libraries use WebGL. The existing visualization libraries offer very similar APIs. Still, it is advisable to use a graphic library and avoid raw WebGL — it helps encapsulate the rendering layer and keeps the architecture cleaner.

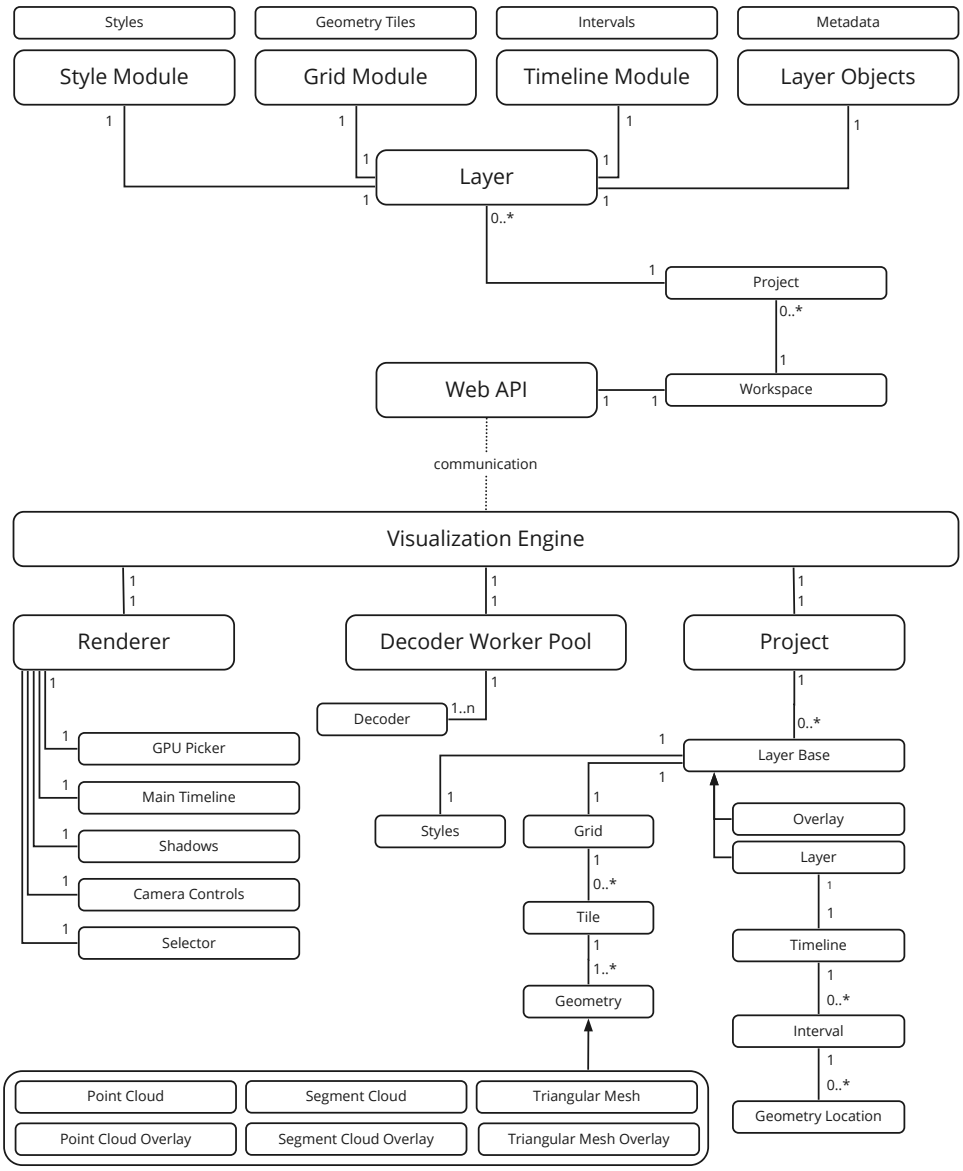


Figure 5.15: Visualization Engine Architecture

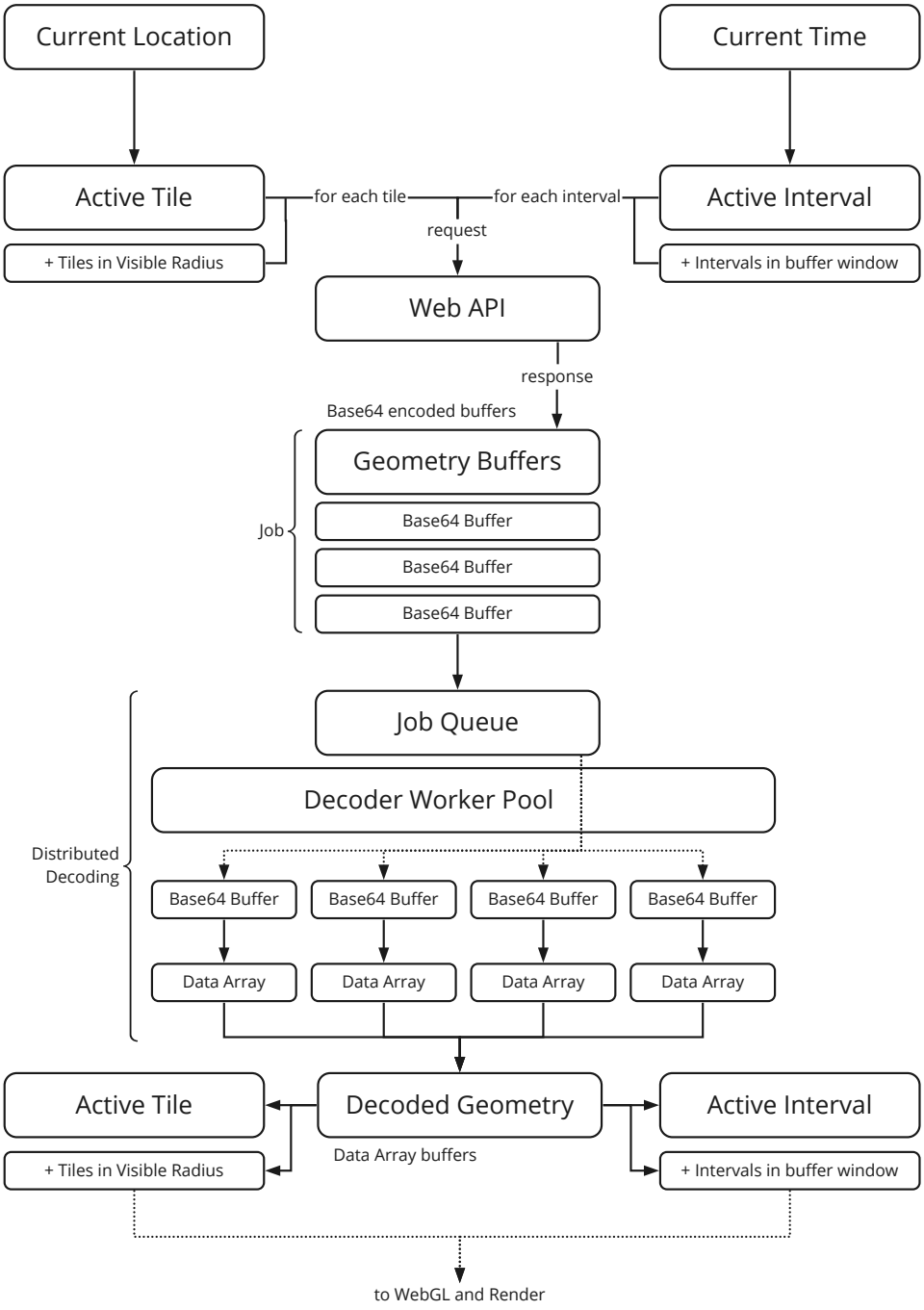


Figure 5.16: Parallelization of the geometry decoding

Chapter 6

Implementation and Results

This chapter presents the implementation of the visualization system and testing results. The system's performance is evaluated in a series of tests focused on memory usage and frame rate. A qualitative user study was conducted during development to help uncover the interface design and the performance issues. The final design was validated with the users.

6.1 Implementation

The system was implemented according to the design presented in chapter 5. The developed processing toolkit is available at PyPI as a python package `metacity`. The server component, together with the user interface, is available as python package `metacity-workspace`. The implementation is written in a combination of languages — geometry processing is implemented in C++ behind a python interface. The visualization engine and the user interface are coded in Typescript.

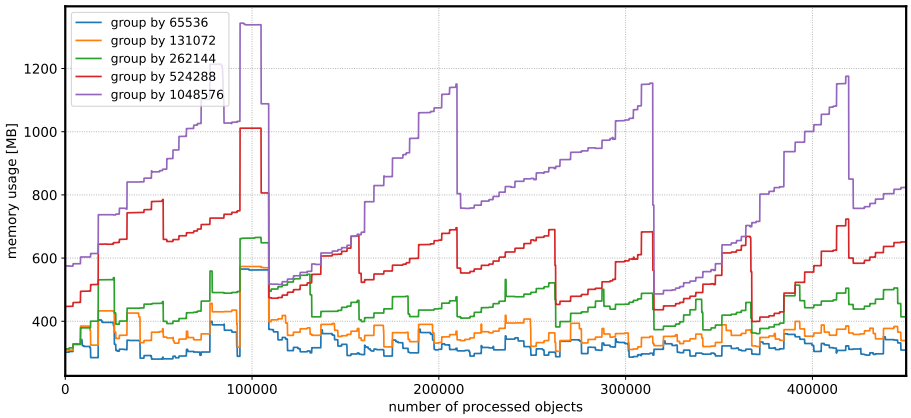
Shapefile and GeoJSON input formats are supported. The dynamic data was obtained from the MATSim simulation software; a custom SIM format was utilized for data transfer. The processing toolkit contains a custom implementation of a GeoJSON parser; the Shapefiles are loaded using the GDAL library. The deployed application runs behind an Nginx server and uses the Nginx Unit for FastAPI apps. Gzip compression is utilized to save bandwidth. The compression works well with the base64 encoded buffers.

6.2 Profiling and Validation

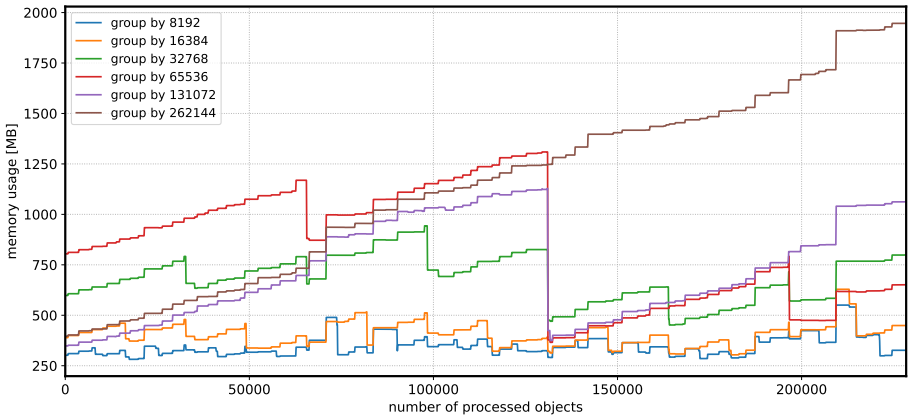
The implementation was tested and validated using a series of datasets described in table 6.1. The static datasets are publicly available at Geoportál hl. m. Prahy; the dynamic datasets were provided by Giang Chau Nguyenová[1] and Anna Moudrá[2] as an output of their synthetic population research.

The tests were conducted at a computer running Linux Kubuntu 18.04 with Intel Core i7 7700 @ 3.6GHz, 8GB of DDR4 RAM (2133MHz), NVIDIA

[1] gcngia@gmail.com
[2] anna.moudra@gmail.com



(a) : Loading the Terrain dataset consisting of 4502183 objects



(b) : Loading the Buildings dataset consisting of 228472 objects

Figure 6.1: Profiling memory consumption during datasets loading; different sizes of data storage sets are tested

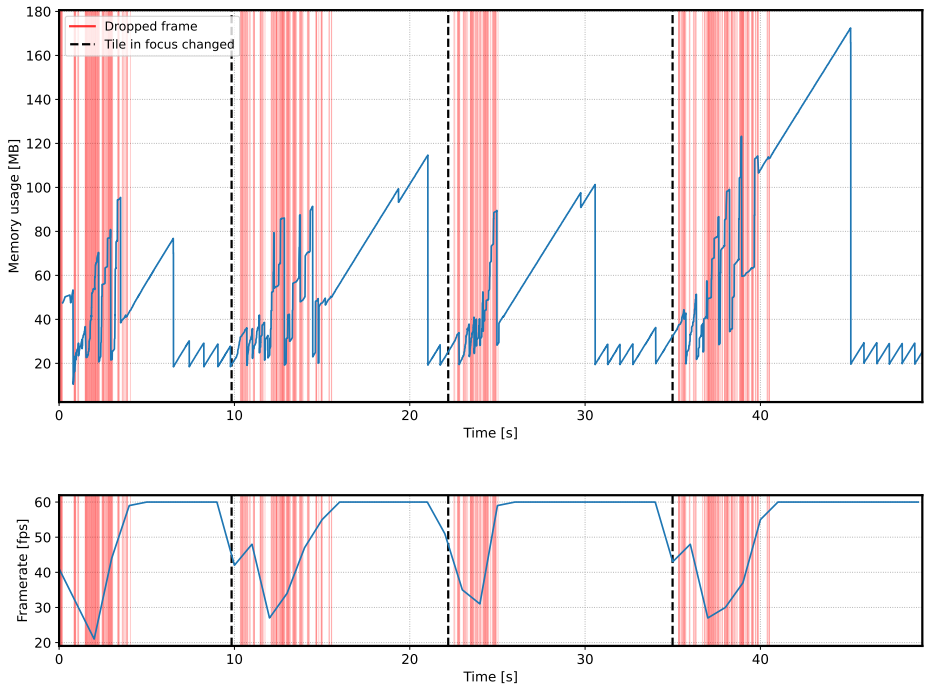
GeForce GTX 1060 6GT with compute capablity 6.1 and Cuda driver 390.116. The following tests were conducted:

Test 1 Data importing, grid and timeline construction
Test 2 Static geometry streaming and visualization
Test 3 Dynamic geometry streaming and visualization
Test 4 Box filter function for LEGO model generation

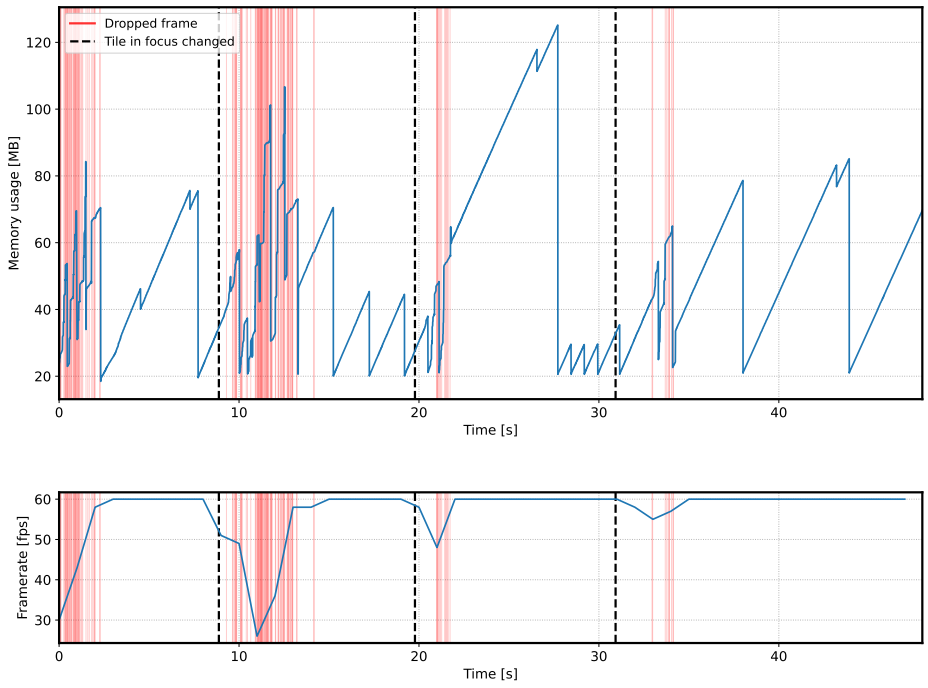
Test 1. The data was uploaded into the system while the used memory and disk space were monitored. The figure 6.1 illustrates the memory consumption for different sizes of data storage sets. The utilization of the out-of-core memory enabled predictable memory consumption. As the set sizes are all powers of two, the memory consumption graph resembles a series of sawtooth

static dataset general information					grid statistics					
layer	primitive type	object count	geometry size [MB]	metadata size [MB]	primitive count per tile					primitive count
					min	max	avg	median	stdev	
Buildings	triangle	228472	1403.228	38.964	21	625119	78490.00	58834.5	79470.00	41599698
Terrain	triangle	4502183	1587.296	837.864	2796	183204	21426.38	18840	13434.58	14055708
Roads	line segment	47522	16.228	15.520	3	4575	1149.42	972	899.65	634482
Bridges	triangle	672	10.900	0.016	9	79089	1025.73	219	5099.23	258486
Population	triangle	88572	44.832	15.284	6	33189	6737.69	4797	6685.99	3166713
Land Use	triangle	551077	597.124	124.400	6	256092	73393.40	54072	64919.63	42347994
Amenities	point	34043	2.500	3.564	3	6015	232.63	63	544.74	102126
Roads 3D	line segment	-	-	-	12	14745	4204.19	3702	3128.43	2308101
Land Use 3D	triangle	-	-	-	0	1338774	408118.73	335502	321196.00	233852034
dynamic dataset general information					timeline statistics					
layer	primitive type	object count	geometry size [MB]	metadata size [MB]	primitive count per interval					primitive count
					min	max	avg	median	stdev	
Cars	time point	21196	744.404	2.228	0	5115	899.84	519	1058.19	71213697
Buses	time point	189234	565.688	26.484	0	1872	684.63	546	510.87	52086903
Subway	time point	33506	116.752	5.028	0	285	150.97	144	69.93	10815135
Tram	time point	194448	706.308	27.456	0	1230	705.48	852	356.49	65525040

Table 6.1: Data storage, regular grid and timeline statistics for test datasets; the geometry is tiled into 1×1 km tiles or 60-second intervals. The Road 3D and Land Use 3D layers are actually layer overlays, which do not contain any objects, only a geometry of the source layers stored in the grid.



(a) : Performance with tile geometry caching off



(b) : Performance with tile geometry caching on

Figure 6.2: Profiling visualization performance; the user loaded Building, Terrain, Bridges, and Roads layers and changed the camera position twice before returning the to the initial location.

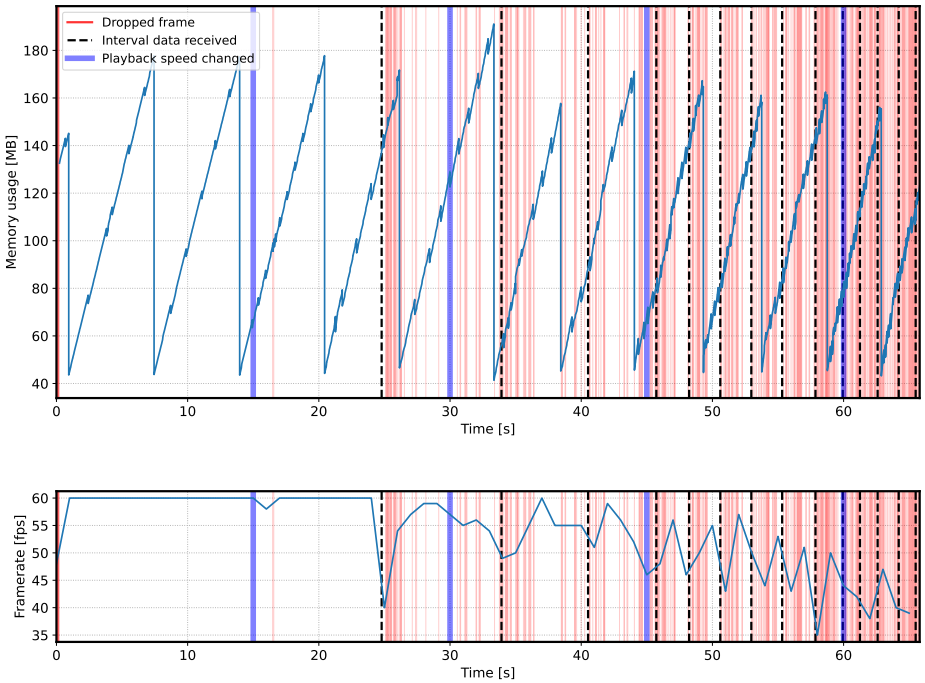
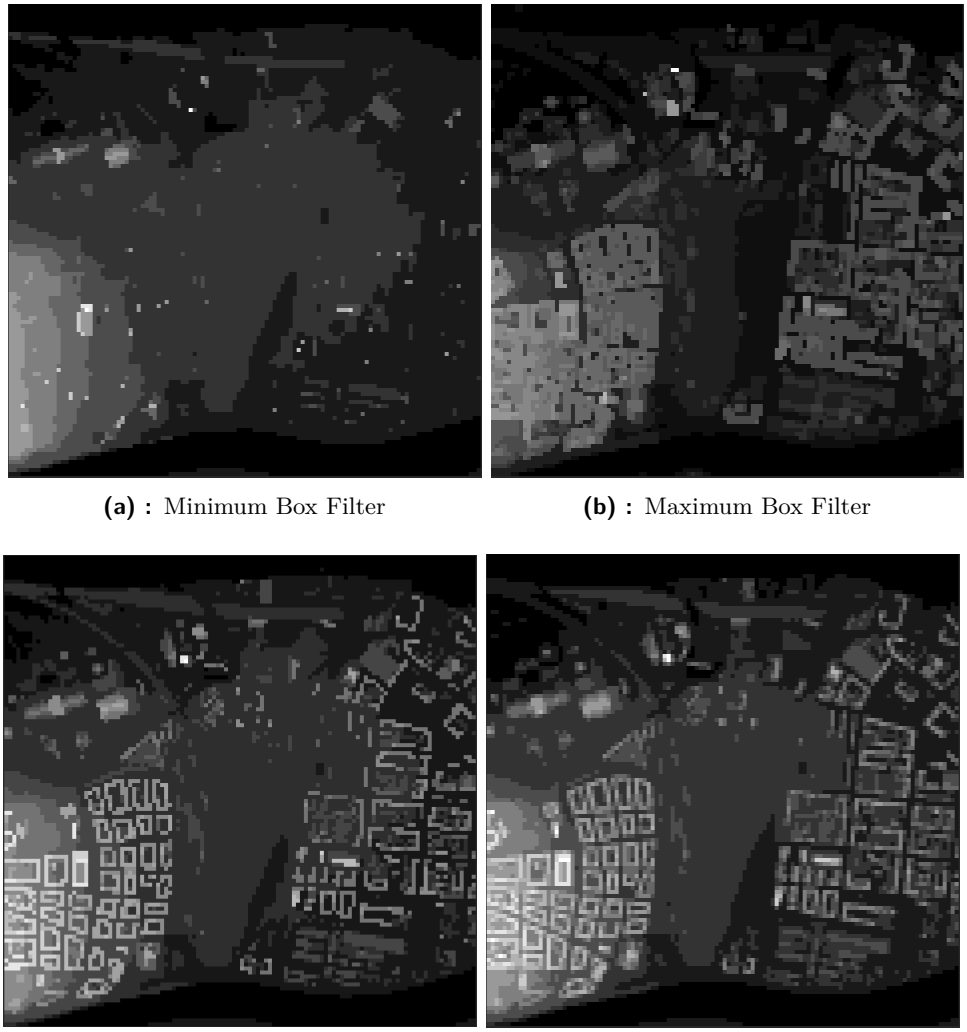


Figure 6.3: Profiling visualization performance; the user loaded all dynamic data layers and changed the playback speed from 1× to 5× at 15 seconds, to 10× at 30 seconds, to 30× at 45 seconds, and finally to 60× at 60 seconds.

wave patterns aligned at the peaks. The disk space requirements are presented in the table 6.1. The static data was tiled into 1×1 km tiles; the temporal data was split into 60-second intervals. The data presented in table 6.1 also include two overlays — Roads 3D were created by mapping Roads onto Terrain, and the Land Use 3D overlay is a combination of the Terrain and Land Use datasets.

Test 2. Building, Terrain, Bridges, and Roads layers were rendered using the visualization engine. The user changed the viewing position three times, which required dynamically loading the visible geometry. At the last position adjustment, the user returned to the starting area. The utilized memory and frame rate were monitored. The test was performed both with and without geometry caching; the figure 6.2 presents the recorded values. The implementation behaves as expected — the frame drops appear less with the caching turned on. The engine discards the data once it is passed to GPU, which explains the memory drops.

Test 3. All dynamic layers from table 6.1 were rendered at once. The user changed the playback speed several times. The utilized memory and frame rate were monitored. Figure 6.3 presents the recorded values. As the playback speed increases, frequent frame drops are observed; however, the frame rate



(a) : Minimum Box Filter

(b) : Maximum Box Filter

(c) : Median Box Filter

(d) : Average Box Filter

Figure 6.4: LEGO Transformation Outputs

stays above 30 fps. The memory consumption patterns are similar to those observed in the previous test case.

Test 4. The implementation of the LEGO model generator, as presented in section 5.2.8, was tested using the minimum, maximum, median, and average box filters. Figure 6.4 illustrates the outputs of the proposed algorithm. The median filter seems to yield the best results.

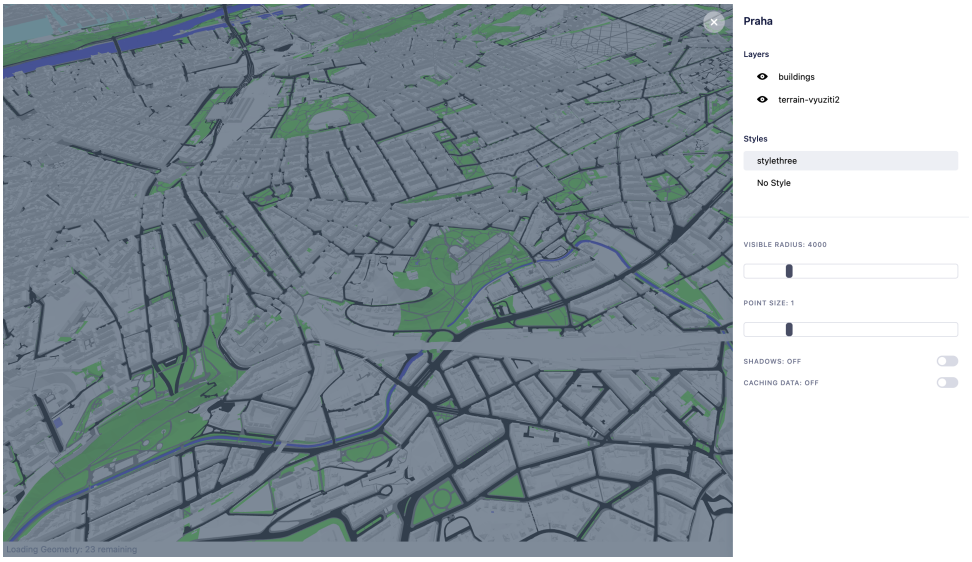


Figure 6.5: The original design of the interface controls, the color overlay, and the side panel proved to be problematic

6.3 User Testing

A qualitative user study with three participants was conducted during the development. Based on the study results, adjustments were made to the system. At the testing site, the user interface was briefly presented to the participants, who were later asked to perform the following tasks:

1. Locate a particular building on the map
2. Zoom closer to the building
3. Select the building
4. Interpret the building metadata

All participants were familiar with geospatial information systems (ArcGIS and QGIS). After the testing, the participants were asked the following list of questions:

- *What observations do you have about the interface?*
- *What tasks would you use the visualization for?*
- *Has 3D visualization any advantages over 2D for you?*

Participant 1. The first participant struggled with the controls. The difficulties were caused by the lack of a reset button, low framerate, and softened camera movement. The double click to select the building was not intuitive for the participant. The participant appreciated the possibility of seeing the city in both 2D and 3D. The 3D environment provides a richer context. The

participant suggested adding keyboard controls. Another issue was the partial overlay of the visualization when the controls were opened, as presented in figure 6.5; the participant expected the visualization to be visible without the partial overlay. They also suggested that a floating toolbox is preferable to the side menu.

Participant 2. The second participant approached the navigation by remaining in the top view position and located the selected building easily. The participant described the controls as more intuitive for game players. The double click to select was also not intuitive for the participant. The 3D model helped the participant navigate the city, and they suggested restricting the zoom at a certain distance from the buildings.

Participant 3. The last participant was pleasantly surprised by the controls — the similarity the game controls was reiterated. The participant came up with the idea of model styling on their own, and their suggestions were in line with the capabilities of the implemented styling system.

Adjustments. Modifications based on the results of the user study were proposed. The final design was validated with the users. The following changes were made to the interface:

- a single left click replaced double click to select,
- a view resent button with a compass was added,
- the original menu design was replaced by a toolbox in the top-left corner,
- the camera movement smoothing was removed completely,
- the visualization was heavily optimized by the decoding parallelization.

6.4 Limitations

New problems were identified during the development. The system does not optimize the geometry level of detail. The optimization can be implemented as an additional module of the processing toolkit in the future.

Styles support only color modification; however, the styling system can be extended by adding new rules to the styling grammar. The implementation of the visualization engine also needs to be extended to support the new visual attributes. The extended support could enable the customizability of the line and point geometry appearance.

There is a minor issue related to naming — the layers can be renamed while being processed; however, the processing job fails, as it cannot find the layer under the original name. Nevertheless, this issue can be easily fixed.

As the system relies on out-of-core memory, disk space availability might become a problem with the increasing number of layers and projects. This limitation cannot be easily overcome — it is the main tradeoff for keeping the memory footprint low.

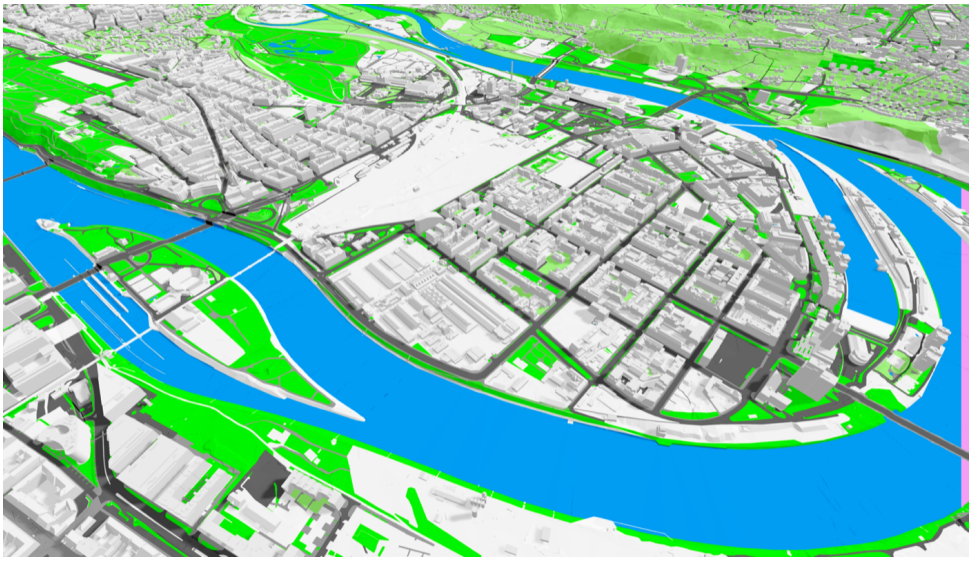


Figure 6.6: Render of the area Prague-Holešovice using datasets Buildings and Land Use 3D; the visualization is styled using the Style 1 from appendix C

6.5 Results

The figures 6.6 to 6.13 showcase selected examples of the visualization outputs. As demonstrated in figure 6.9, it is possible to select a region and export the geometry as an OBJ file or as a LEGO model. The export page of the LEGO model offers an interactive presentation (figure 6.10). It is possible to divide the model into tiles and build it in a bottom-up fashion (figure 6.11). The feasibility of this approach is demonstrated in the figure 6.12; a slice of Náměstí Míru was assembled. The projection mapping of the visual outputs was tested using the assembled LEGO model (figure 6.12) and a scale model located at the Center for Architecture and Metropolitan Planning in Prague (see figure 6.13). Examples of styles are presented in appendix C; additional outputs are presented in the appendix D.

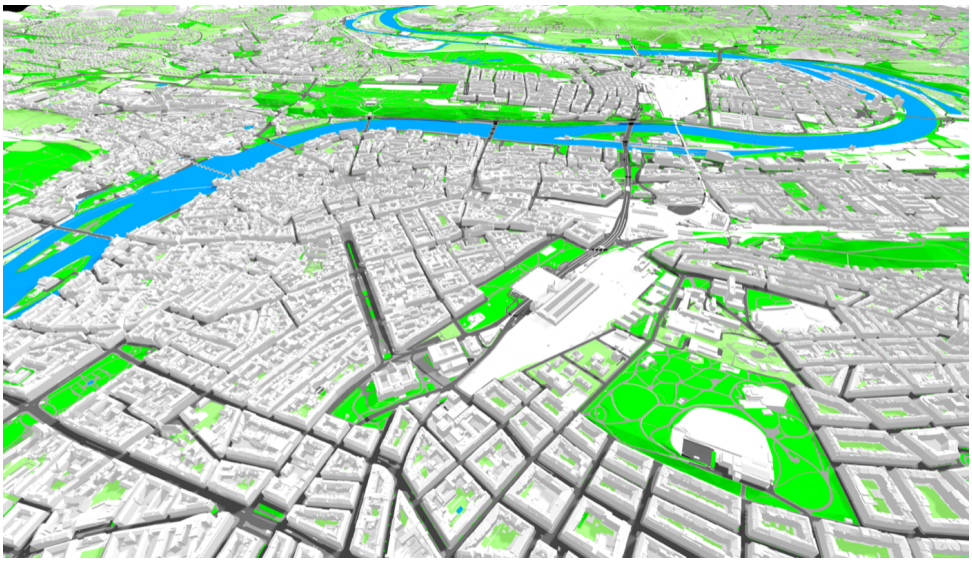


Figure 6.7: Render of the Prague city center containing datasets Buildings and Land Use 3D, styled using the Style 1 from appendix C

Figure 6.8: Visualization of traffic simulation, the color indicates the speed at which the agents are moving — read agents are waiting at junctions and stops while green agents are moving at full speed

Figure 6.9: The visualization interface allows selecting areas for OBJ or LEGO model export.

Figure 6.10: The presentation of the exported LEGO model, the example is the export of area presented in figure 6.9

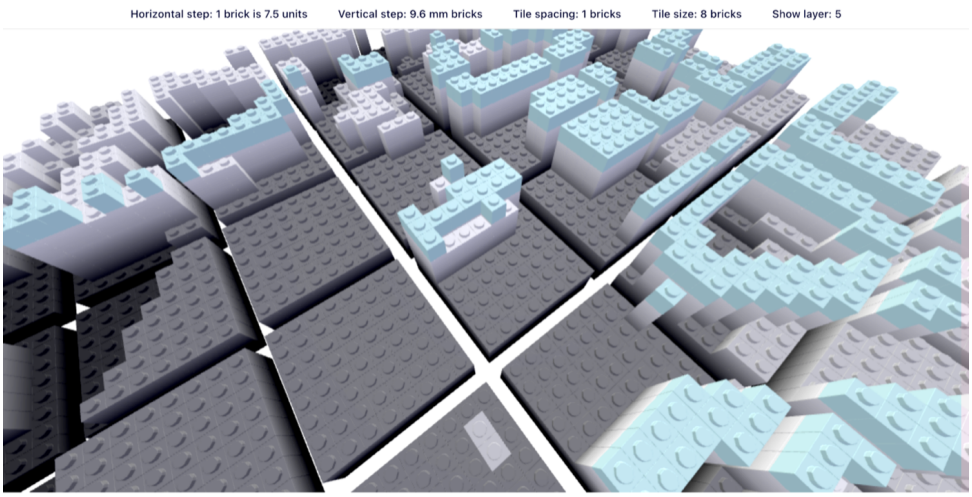


Figure 6.11: Assembly of the LEGO Model presented in the figure 6.10

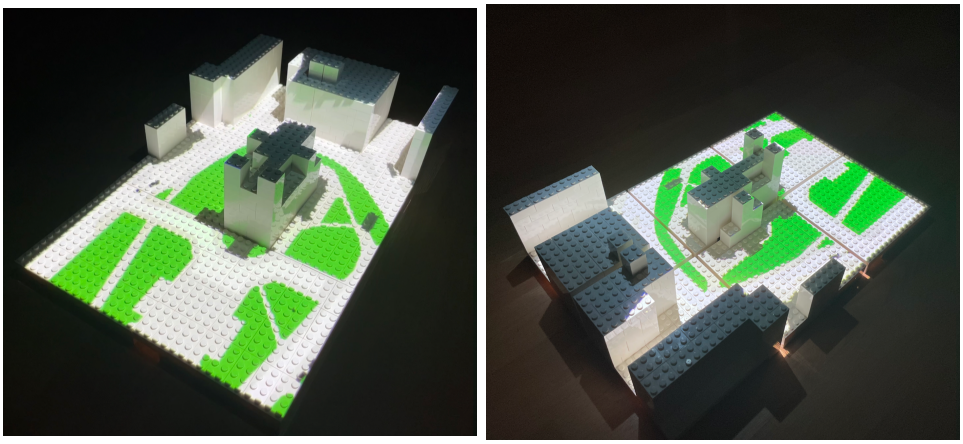


Figure 6.12: Assembled LEGO models with projection mapping; the warping of the projection was set up in TouchDesigner

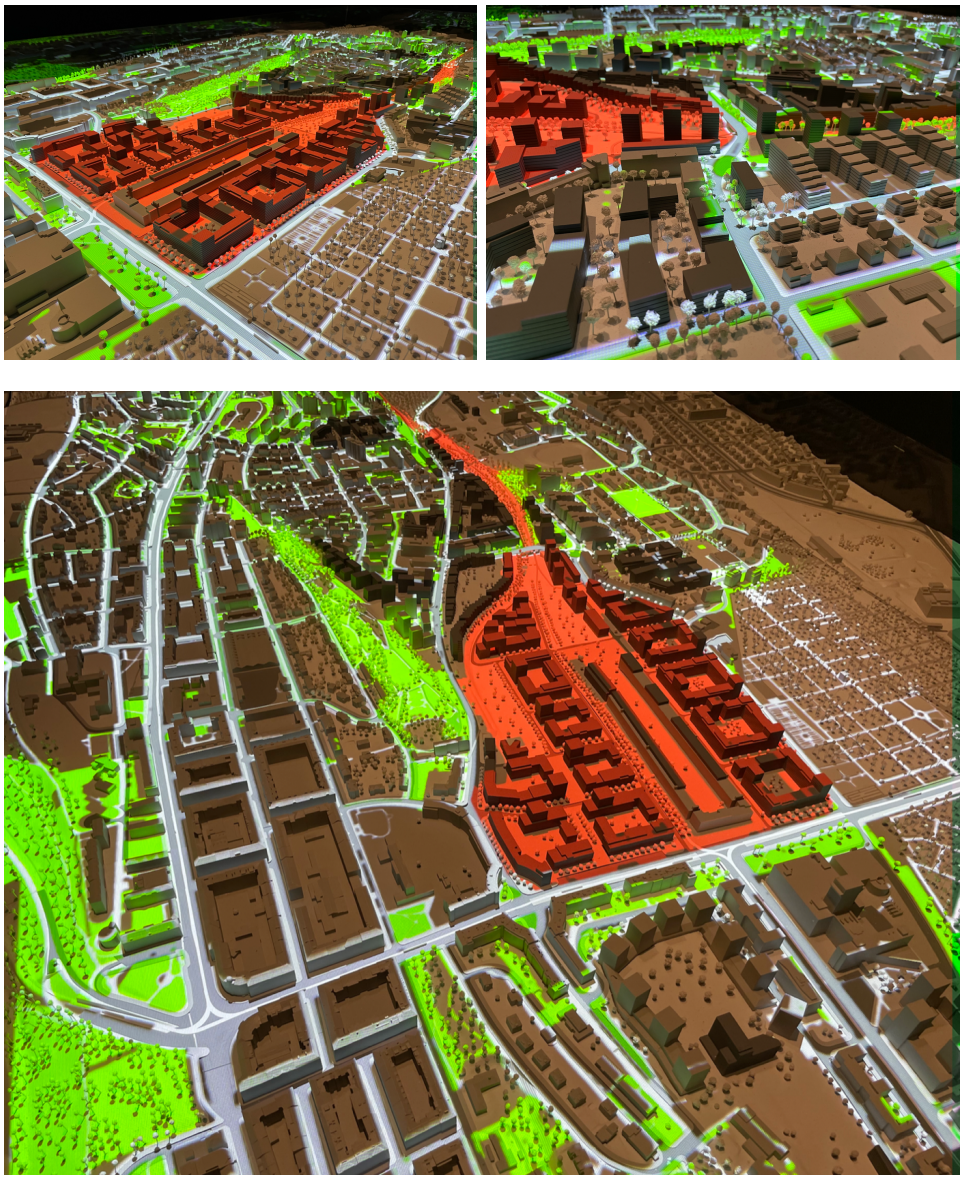


Figure 6.13: The visualization output is live projected onto a scale model of the Žižkov Freight Railway Station area, courtesy of CAMP Prague. The red zone highlights the current location of the railway station, soon to be redeveloped into a new residential area.

Chapter 7

Conclusion

The initial research was focused on the role of urban data platforms in the context of urban planning. The analysis revealed that these systems become truly meaningful only when all stakeholders are willing to participate. The systems designed around simulation models encourage higher levels of interaction. The model-based approach, also known as visual analytics, can answer questions that arise during problem exploration. Unfortunately, the validation of the model outputs can be problematic, and there is a potential risk of creating privacy issues. Further, the analysis explored possible geometry representations and ways to process large geospatial datasets. The existing geospatial formats and information systems were examined in the context of available open data.

Based on the initial research, a design of a visualization system was proposed. It focused on enabling efficient data processing and accessible visualization on the web. The internal data model was designed to facilitate a wide range of input formats. The separation of semantic information and geometry offers more flexibility. The processing of large datasets is enabled by using out-of-core memory and spatial and temporal data structures. A simple and extensible styling language was proposed to allow customizing the geometry appearance based on the object's metadata.

The system was implemented as two python packages. The first package provides the necessary tools for data processing, while the second manages the visualization. They can be deployed together as a standalone visualization system; moreover, the modular design of the packages allows them to be separately reused as libraries in a more extensive system. Behind the python interface, all computations are implemented in C++, providing an ideal combination of interface accessibility and computational efficiency. The system can be used to share and explore the data online. Users can request OBJ files or generate LEGO models of any available urban area.

The system's performance was tested using a set of static and dynamic datasets. Concurrently, a user study was conducted to validate the design of the user interface. All issues discovered in the user study were resolved.

The future objectives include extending the styling system. The open design of the current styling language offers many opportunities for experimentation. Geometry optimization poses an equally significant challenge. The complexity

of the geometry increases with the application of transformations such as overlay mapping. It is desirable to adjust the geometry level of detail and further improve the performance of the system. The most challenging objective is to integrate the developed system with a simulation modeling tool. As the initial research suggests, pursuing this goal could significantly contribute to sustainable city development.

Appendix A

Bibliography

[1] Foth, M.; Choi, J. H.-j.; et al. Urban informatics. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, 2011, pp. 1–8.

[2] Robinson, R.; Rittenbruch, M.; et al. Street computing: Towards an integrated open data application programming interface (API) for cities. *Journal of Urban Technology*, volume 19, no. 2, 2012: pp. 1–23.

[3] Weiser, M. The Computer for the 21 st Century. *Scientific american*, volume 265, no. 3, 1991: pp. 94–105.

[4] Burke, J. A.; Estrin, D.; et al. Participatory sensing. 2006.

[5] Zheng, Y.; Liu, Y.; et al. Urban computing with taxicabs. In *Proceedings of the 13th international conference on Ubiquitous computing*, 2011, pp. 89–98.

[6] Barns, S. Smart cities and urban data platforms: Designing interfaces for smart governance. *City, culture and society*, volume 12, 2018: pp. 5–12.

[7] Datopian. London — City Dashboard [online]. [cit. 17. 12. 2021]. Available from: `https://london.datahub.io`

[8] City of Boston. CityScore [online]. [cit. 17. 12. 2021]. Available from: `https://www.boston.gov/innovation-and-technology/cityscore`

[9] City of New York. NYC open data [online]. [cit. 17. 12. 2021]. Available from: `https://opendata.cityofnewyork.us/`

[10] City of Helsinki Executive Office. Open data service [online]. [cit. 17. 12. 2021]. Available from: `https://hri.fi/en_gb/`

[11] Cooperation Open Government Data Österreich. Offene Daten Österreich [online]. [cit. 17. 12. 2021]. Available from: `https://data.wien.gv.at`

[12] City of Chicago. Chicago Data Portal [online]. [cit. 17. 12. 2021]. Available from: `https://data.cityofchicago.org`

[13] Greater Sydney Commission. The Greater Sydney Dashboard [online]. [cit. 17. 12. 2021]. Available from: https://www.greater.sydney/dashboard

[14] Municipality of Copenhagen and Capital Region of Denmark. City Data Exchange - Lessons learned from a private/public data collaboration [online]. March 2018. Available from: https://cphsolutionslab.dk/en/projekter/data-platforms/city-data-exchange

[15] Goldsmith, S.; Crawford, S. *The responsive city: engaging communities through data-smart governance*. Jossey-Bass, ISBN 978-1-118-91121-1 978-1-118-91093-1.

[16] Robinson, R. Why Smart Cities still aren't working for us after 20 years. And how we can fix them. February 2016. Available from: https://theurbantechnologist.com/2016/02/01/

[17] Ira Winder, MIT Media Lab, Big Data Analytics Tokyo 2017. In: YouTube [online]. 12. 5. 2017 [cit. 10. 9. 2020]. Available from: https://youtu.be/zVmc5_KK5JU

[18] Winder, J. I.; Ma, C. Tangible Interactive Matrix for Real-time Computation and 3D Projection Mapping. *Future Technologies Conference (FTC)*: p. 4.

[19] Tactile Matrix. In:Tactile Matrix [online]. Ira Winder, Andy Ryan, © 2013–2021. [cit. 10. 10. 2020]. Available from: https://ira.mit.edu/tactile-matrix.

[20] Electronics components for Tactile Matrix. Drawing by Ira Winder. In: Tactile Matrix SDK [online]. Ira Winder, © 2013–2021. [cit. 10. 10. 2020]. Available from: https://ira.mit.edu/sdk.

[21] Hadhrawi, M.; Larson, K. Illuminating LEGOs with Digital Information to Create Urban Data Observatory and Intervention Simulator. In *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems*, ACM, ISBN 978-1-4503-4315-2, pp. 105–108, doi:10.1145/2908805.2909400. Available from: https://dl.acm.org/doi/10.1145/2908805.2909400

[22] MIT Media Lab City Science group. CityScope Ecosystem | Here We Build CityScope [online]. Available from: https://cityscope.media.mit.edu

[23] Pumain, D.; Reuillon, R. *Urban Dynamics and Simulation Models*. Lecture Notes in Morphogenesis, Springer International Publishing, ISBN 978-3-319-46495-4 978-3-319-46497-8, doi:10.1007/978-3-319-46497-8. Available from: http://link.springer.com/10.1007/978-3-319-46497-8

[24] Taillandier, P.; Gaudou, B.; et al. Building, composing and experimenting complex spatial models with the GAMA platform. volume 23, no. 2: pp. 299–322, ISSN 1384-6175, 1573-7624, doi:10.1007/s10707-018-00339-6. Available from: http://link.springer.com/10.1007/s10707-018-00339-6

[25] Axhausen, K. W.; Horni, A.; et al. *The multi-agent transport simulation MATSim*. ISBN 978-1-909188-75-4, OCLC: 1116114273.

[26] Robinson, D.; Haldi, F.; et al. CITYSIM: Comprehensive Micro-simulation of Resource Flows for Sustainable Urban Planning. 2009: p. 8.

[27] Santé, I.; García, A. M.; et al. Cellular automata models for the simulation of real-world urban processes: A review and analysis. volume 96, no. 2: pp. 108–122, ISSN 01692046, doi:10.1016/j.landurbplan.2010.03.001. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0169204610000472

[28] Torrens, P. M.; Benenson, I. Geographic Automata Systems. volume 19, no. 4: pp. 385–412, ISSN 1365-8816, 1362-3087, doi:10.1080/13658810512331325139. Available from: http://www.tandfonline.com/doi/abs/10.1080/13658810512331325139

[29] Underkoffler, J.; Ishii, H. Urp: a luminous-tangible workbench for urban planning and design. In *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI '99*, ACM Press, ISBN 978-0-201-48559-2, pp. 386–393, doi:10.1145/302979.303114. Available from: http://portal.acm.org/citation.cfm?doid=302979.303114

[30] Underkoffler, J.; Ullmer, B.; et al. Emancipated pixels: real-world graphics in the luminous room. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, ACM Press, ISBN 978-0-201-48560-8, pp. 385–392, doi:10.1145/311535.311593. Available from: http://portal.acm.org/citation.cfm?doid=311535.311593

[31] Ishii, H.; Underkoffler, J.; et al. Augmented urban planning workbench: overlaying drawings, physical models and digital simulation. In *Proceedings. International Symposium on Mixed and Augmented Reality*, IEEE Comput. Soc, ISBN 978-0-7695-1781-0, pp. 203–211, doi:10.1109/ISMAR.2002.1115090. Available from: http://ieeexplore.ieee.org/document/1115090/

[32] Alonso, L.; Zhang, Y. R.; et al. CityScope: A Data-Driven Interactive Simulation Tool for Urban Design. Use Case Volpe. In *Unifying Themes in Complex Systems IX*, edited by A. J. Morales; C. Gershenson; D. Braha; A. A. Minai; Y. Bar-Yam, Springer International

Publishing, ISBN 978-3-319-96660-1 978-3-319-96661-8, pp. 253–261, doi:10.1007/978-3-319-96661-8_27, series Title: Springer Proceedings in Complexity. Available from: http://link.springer.com/10.1007/978-3-319-96661-8_27

[33] Yurrita, M.; Grignard, A.; et al. Dynamic Urban Planning: an Agent-Based Model Coupling Mobility Mode and Housing Choice. Use case Kendall Square. 2106.14572. Available from: http://arxiv.org/abs/2106.14572

[34] Pře(d)stav si Prahu [online]. OFICINA studio, 2019. [cit. 10. 10. 2020]. Available from: https://oficina.design/cs/predstav-si-prahu/.

[35] Turba, M.; Berdichová, E.; et al. Strategic Plan for Prague. *Bulletin of the City of Prague*, 2008.

[36] Rohanský ostrov: nový Karlín? [online]. IPR Praha, 2019. [cit. 25. 11. 2020]. Available from: https://www.iprpraha.cz/rohanskyostrov.

[37] Rohanský ostrov: nový Karlín? Martin Vronský, IPR Praha, © 2020. [cit. 25. 11. 2020].

[38] Telea, A. C. *Data visualization: principles and practice*. Boca Raton: CRC Press, Taylor & Francis Group, second edition, 2015, ISBN 9781466585263.

[39] Moreland, K. A Survey of Visualization Pipelines. volume 19, no. 3: pp. 367–378, ISSN 1077-2626, doi:10.1109/TVCG.2012.133. Available from: http://ieeexplore.ieee.org/document/6212499/

[40] Vitter, J. S. External Memory Algorithms and Data Structures: Dealing with Massive Data. volume 33, no. 2: p. 63.

[41] Law, C.; Schroeder, W.; et al. A multi-threaded streaming pipeline architecture for large structured data sets. In *Proceedings Visualization '99 (Cat. No.99CB37067)*, IEEE, ISBN 978-0-7803-5897-3, pp. 225–232, doi:10.1109/VISUAL.1999.809891. Available from: http://ieeexplore.ieee.org/document/809891/

[42] Ahrens, J. P.; Desai, N.; et al. A modular extensible visualization system architecture for culled prioritized data streaming. p. 64950I, doi:10.1117/12.706325. Available from: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.706325

[43] Ahrens, J. P.; Woodring, J.; et al. Interactive remote large-scale data visualization via prioritized multi-resolution streaming. In *Proceedings of the 2009 Workshop on Ultrascale Visualization - UltraVis '09*, ACM Press, ISBN 978-1-60558-897-1, pp. 1–10, doi:10.1145/1838544.1838545. Available from: http://portal.acm.org/citation.cfm?doid=1838544.1838545

[44] Ahrens, J.; Law, C.; et al. A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets: p. 9.

[45] Thomas, J.; Cook, K. A visual analytics agenda. volume 26, no. 1: pp. 10–13, ISSN 0272-1716, doi:10.1109/MCG.2006.5. Available from: http://ieeexplore.ieee.org/document/1573625/

[46] Andrienko, G.; Andrienko, N.; et al. Challenging problems of geospatial visual analytics. volume 22, no. 4: pp. 251–256, ISSN 1045926X, doi:10.1016/j.jvlc.2011.04.001. Available from: https://linkinghub.elsevier.com/retrieve/pii/S1045926X11000280

[47] Andrienko, G.; Andrienko, N.; et al. Geovisual analytics for spatial decision support: Setting the research agenda. volume 21, no. 8: pp. 839–857, ISSN 1365-8816, 1362-3087, doi:10.1080/13658810701349011. Available from: http://www.tandfonline.com/doi/abs/10.1080/13658810701349011

[48] Kraak, M. J.; Ormeling, F.; et al. *Cartography: visualization of geospatial data*. Routledge, third edition, ISBN 9781138138261.

[49] Veverka, B. Křovák's projection and its use for the Czech Republic and the Slovak Republic. *Holota, P., Slaboch, V.(eds.)*, volume 50, 2004: pp. 173–179.

[50] Berg, M. d.; Cheong, O.; et al. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, third edition, ISBN 3540779744, 19–41 pp.

[51] García R, Y. J.; López, M. A.; et al. A greedy algorithm for bulk loading r-trees. In *Proceedings of the sixth ACM international symposium on Advances in geographic information systems - GIS '98*, ACM Press, ISBN 978-1-58113-115-4, pp. 163–164, doi:10.1145/288692.288723. Available from: http://portal.acm.org/citation.cfm?doid=288692.288723

[52] Munzner, T. *Visualization analysis & design*. ISBN 978-1-4665-0893-4 978-1-4987-0776-3, OCLC: 904266418.

[53] Institut plánování a rozvoje hlavního města Prahy. Opendata | Geoportál hl. m. Prahy [online]. [cit. 14. 1. 2021]. Available from: https://www.geoportalpraha.cz/cs/data/otevrena-data/seznam

[54] ESRI. ESRI Shapefile technical description. Technical report, 1998.

[55] GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library [online]*. Open Source Geospatial Foundation, 2020. Available from: https://gdal.org

[56] Gröger, G.; Kolbe, T. H.; et al. OGC city geography markup language (CityGML) encoding standard. 2012.

[57] Ledoux, H.; Ohori, K. A.; et al. CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, volume 4, no. 1, 2019: p. 4.

[58] CityJSON: Software [online]. CityJSON, 2019. [cit. 30. 8. 2020]. Available from: https://www.cityjson.org/software/.

[59] citygml4j [online]. Citygml4j.org, 2020. [cit. 28. 8. 2020]. Available from: https://github.com/citygml4j/citygml4j.

[60] Stadler, A.; Kolbe, T. H. Spatio-semantic coherence in the integration of 3D city models. In *Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality ISSDQ 2007 in Enschede, The Netherlands, 13-15 June 2007*, 2007.

[61] cjio, or CityJSON/io [online]. 3D geoinformation research group at TU Delft, 2021. [cit. 14. 1. 2021]. Available from: https://github.com/cityjson/cjio.

[62] Alliance, O. D. Open Design Specification for. dwg files Version 5.2 [online]. 1998.

[63] RealDWG Developer Center [online]. Autodesk Inc., 2020. [cit. 12. 9. 2020]. Available from: https://www.autodesk.com/developer-network/platform-technologies/realdwg.

[64] Autodesk, Inc. DXF Reference. Technical report, Autodesk, Inc., San Rafael; CA 94903; USA, 2011.

[65] Moitzi Manfred, c. *EZDXF - A Python package to create and modify DXF drawings [online]*. 2021. Available from: `https://ezdxf.readthedocs.io`

[66] Tang, L.; Chen, C.; et al. Building Information Modeling and Building Performance Optimization. In *Encyclopedia of Sustainable Technologies*, edited by M. A. Abraham, Oxford: Elsevier, 2017, ISBN 978-0-12-804792-7, pp. 311 – 320.

[67] Esri GIS Products | ArcGIS Mapping Software for Desktop, SaaS & Enterprise Apps [online]. ESRI, 2021. [cit. 14. 1. 2021]. Available from: https://www.esri.com/en-us/arcgis/products/index.

[68] Deployment patterns for ArcGIS Enterprise [online]. Esri, 2021. [cit. 14. 12. 2021]. Available from: https://enterprise.arcgis.com/en/get-started/10.7/windows/additional-server-deployment.htm.

[69] QGIS Development Team. *QGIS Geographic Information System [online]*. QGIS Association, 2021. Available from: `https://www.qgis.org`

[70] kepler.gl [online]. Uber, 2021. [cit. 14. 1. 2021]. Available from: https://kepler.gl.

[71] vis.gl [online]. Vis.gl, 2021. [cit. 14. 1. 2021]. Available from: https://vis.gl/.

[72] OpenStreetMap Foundation [online]. OpenStreetMap Foundation, 2020. [cit. 14. 1. 2021]. Available from: https://wiki.osmfoundation.org/.

[73] Mapbox [online]. Mapbox, 2020. [cit. 14. 1. 2021]. Available from: https://mapbox.com/.

[74] Movement Uber [online]. Uber, 2021. [cit. 14. 1. 2021]. Available from: https://movement.uber.com/.

[75] CesiumJS-Geospatial 3D mapping and virtual globe platform [online]. Cesium Consortium and others, 2019. [cit. 14. 1. 2021]. Available from: https://cesiumjs.org/.

[76] Yao, Z.; Nagel, C.; et al. 3DCityDB-a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, volume 3, no. 1, 2018: pp. 1–26.

[77] LuciadRIA-Provides Situational Awareness in the Browser [online]. Hexagon AB and/or its subsidiaries, 2021. [cit. 14. 12. 2021]. Available from: https://www.hexagongeospatial.com/products/luciad-portfolio/luciadria.

Appendix B

Dependencies and Installation

Significant dependencies that are not included in the packages are:

CGAL intersection computation in the overlay mapping algorithm
CMake building the C++ code
GDAL Shapefile parsing

Dependencies included in the packages are:

pybind11 integration of the C++ code into the python interface
gml vector operations
numpy fast array operations
lark context-free grammar parsing
orjson fast JSON parsing
fastapi web API library
react user interface framework
evergreenUI user interface elements
monaco-editor editor framework utilized for style editor
three.js rendering

Deployment dependencies:

Nginx serving static files
Nginx Unit supervisor for FastAPI apps

Installation

This is a guide for the Metacity System setup. This document will guide you through setting up a local environment and creating a Metacity Workspace. This guide is for Linux only; however, the system can be deployed on Windows and macOS. Please find appropriate alternatives to the presented commands for your system.

1. Install GDAL

```
sudo add-apt-repository ppa:ubuntugis/ppa
sudo apt-get update
```

```
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
```

2. Install CGAL

```
sudo apt-get install libcgal-dev
```

3. Install CMake

```
sudo apt-get install cmake
```

4. Install python virtualenv

```
sudo apt-get install virtualenv
```

5. Create python environment

```
virtualenv --python=python3.9 env
```

6. Activate the environment with

```
. ./env/bin/activate
```

7. Install the python package

```
pip install metaworkspace
or
pip install metacity
```

8. Continue only if you are installing the `metaworkspace` package for visualization. If you installed the `metacity` package for data processing, you can leave this guide.

9. Create Metacity Workspace

```
python -m metaworkspace --install [name of your workspace]
```

10. Create new Workspace user

```
python -m metaworkspace --createuser [name of your workspace]
```

11. See other command-line options

```
python -m metaworkspace --help
```

12. Run the application

```
python -m metaworkspace --run [name of your workspace] --serve
```

Hints

- You need locally installed python headers, install it with command `sudo apt-get install python3.9-dev`, make sure the version of python used in the environment matches the headers you install here
- You need a local C++ compiler, we recommend clang, install it with command `sudo apt-get install clang`, g++ works too

Appendix C

Style Examples

```
@layer ("Budovy") {
    @color: #F8F8F8;
}

@layer ("Využití - Terén") {
    @source {
        @meta(CTVUK_POPI) {
            "komunikace - silnice": #999999;
            "komunikace - parkoviště": #999999;
            "komunikace": #999999;
            "komunikace - chodník nebo parková cesta": #BBBBBB;
            "zeleň v zástavbě - veřejná zeleň": #78ff47;
            "zeleň v zástavbě - ostatní zeleň": #bfffa8;
            "zeleň v zástavbě": #bfffa8;
            "louky, zahrady": #bfffa8;
            "vodní plochy - řeka, potok": #47bcff;
            "vodní plochy - rybník, jezírko":  #47bcff;
        }
    }
}

@legend {
    "roads": #999999;
    "nature, trees": #78ff47;
    "water": #47bcff;
}
```

Style 1. Buildings and Land Use 3D

```
@layer (Bubny) {
    @color: #C5283D;

    @meta(autocad_la) {
        "U- 02 Model - terén - silnice": #2F3737;
        "U- 02 Model - terén - parkovací stání": #292e2e;
        "U- 02 Model - Kralupská trať": #1d3333;
        "U- 02 Model - Kladenská trať": #1d3333;
        "U- 02 Model - Trať": #1d3333;
        "U- 02 Model - terén - zeleň": #C8ED51;
        "U- 02 Model - terén - chodník": #FFFFEE;
        "U- 02 Model - terén": #8a4900;
        "U- 02 Model - opěrné zdi": #DDDDDD;
        "U- 02 Model - objekty - schodiště": #FFFFDD;
        "U- 02 Model - střechy - navrhované": #d11800;
        "U- 02 Model - objekty - navrhované": #d11800;
        "U- 02 Model - vltava": #A1E3FF;
        "U- 02 Model - střechy - původní": #ffe8b8;
        "U- 02 Model - objekty - původní": #ffe8b8;
    }
}

@legend {
    "Původní budovy": #ffe8b8;
    "Nová výstavba": #d11800;
    "Zeleň": #C8ED51;
}
```

Style 2. Style for buildings

```
@layer ("Využití území") {
    @color: #000000;

    @meta(CTVUK_POPI) {
        "komunikace - silnice": #FFFFFF;
        "komunikace - chodník nebo parková cesta": #EEEEEE;
        "zeleň v zástavbě - veřejná zeleň": #00FF00;
        "vodní plochy - řeka, potok": #8888FF;
        "železnice": #FF0000;
    }
}

@legend {
    "Železnice": #FF0000;
    "Zeleň": #00FF00;
    "Vodní plochy": #8888FF;
}
```

Style 3. Land Use with a legend, utilized for video-mapping on Žižkov Freight Railway Station scale model in figure 6.13

Appendix D

Outputs

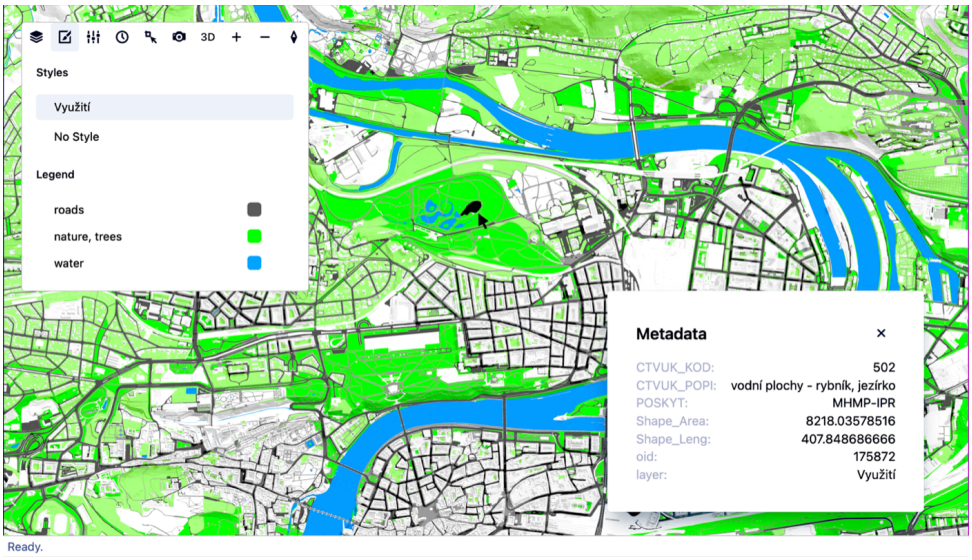


Figure D.1: The final design of the visualization user interface; floating toolbox is on the top left and dialog containing metadata for the selected element on the bottom right.

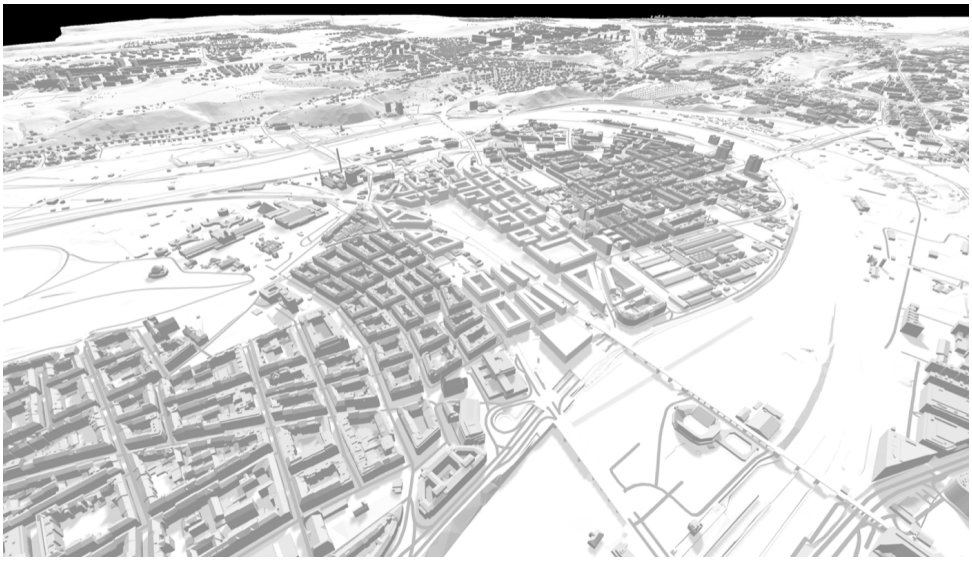


Figure D.2: Bubny-Zátory, datasets Roads, Buildings, and Terrain, no styles; an additional dataset with the currently constructed buildings in Prague Bubny-Zátory is inserted into the visualization, courtesy of IPR Prague.

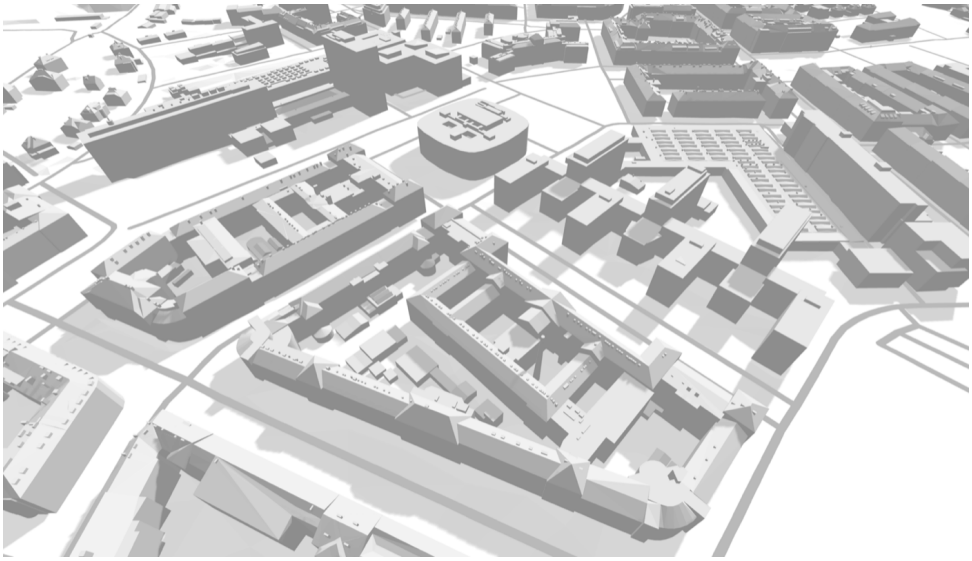


Figure D.3: Campus Dejvice closeup, no styles

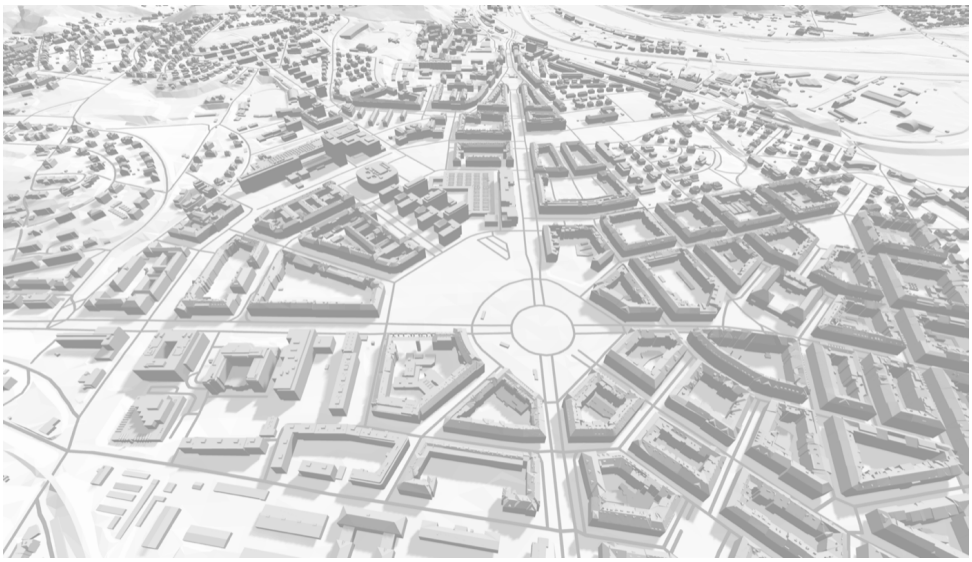


Figure D.4: Vítězné náměstí, no styles

Figure D.5: Prague overview, no styles

Figure D.6: Prague overview, styled with Style 1 in appendix C

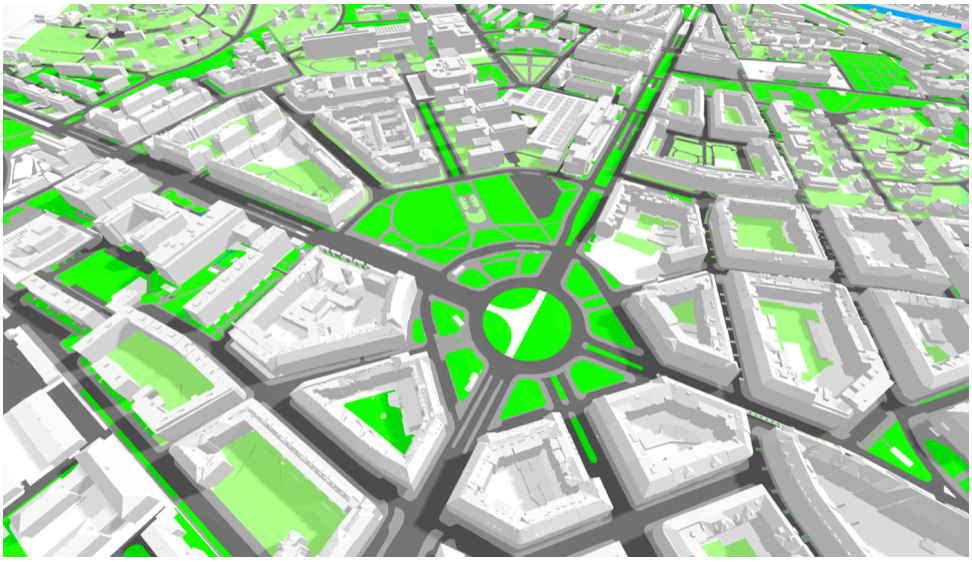


Figure D.7: Vítězné náměstí, styled with Style 1 in appendix C

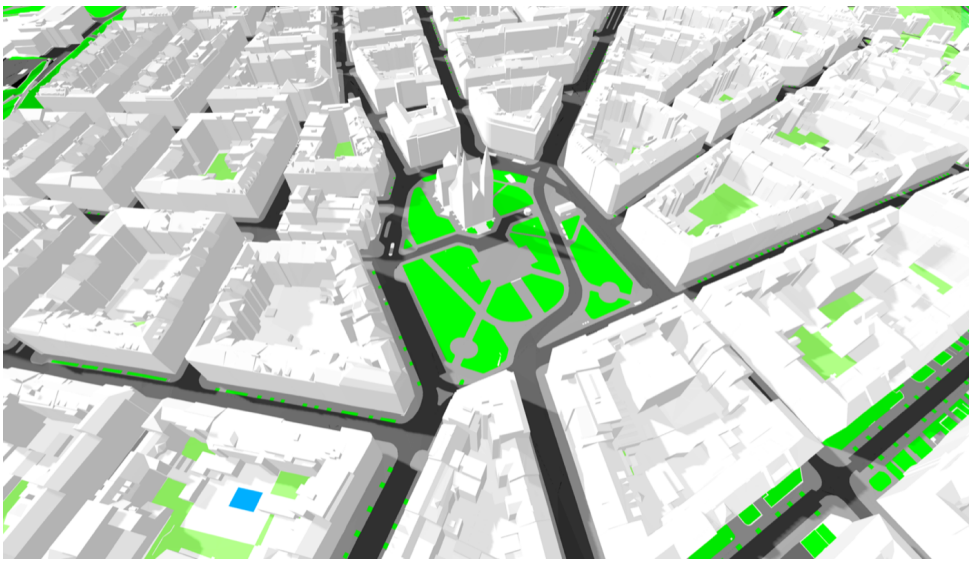


Figure D.8: Náměstí Míru, styled with Style 1 in appendix C

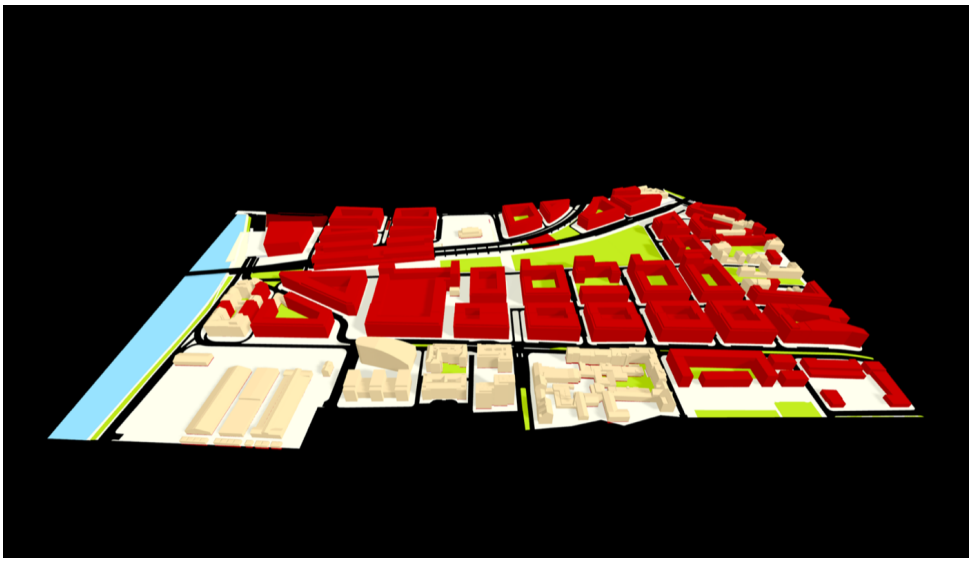


Figure D.9: Model Bubny-Zátory, courtesy of IPR Prague, styled with Style 2 in appendix C

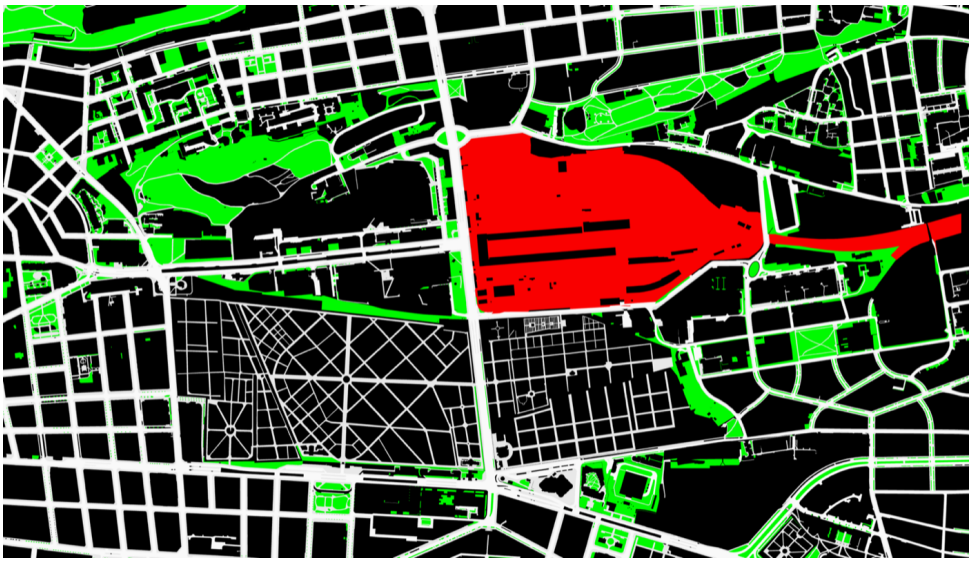


Figure D.10: Žižkov, styled with Style 3 in appendix C

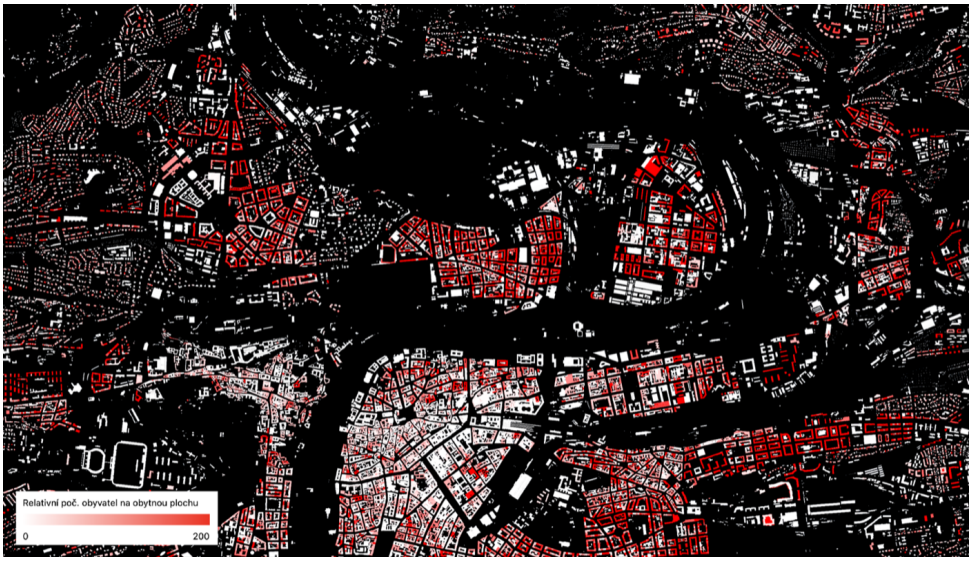


Figure D.11: Distribution of population

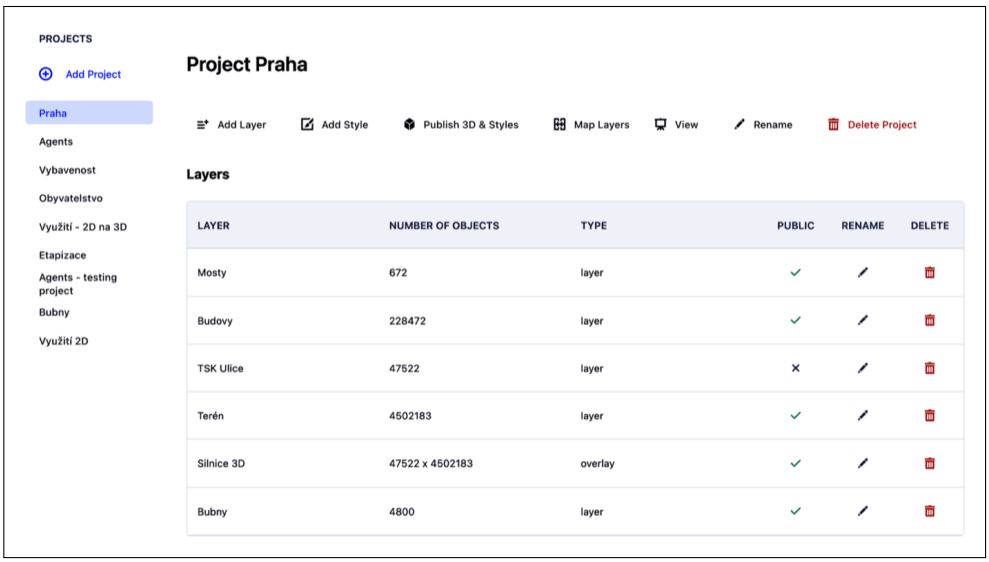


Figure D.12: The user interface of project administration allows adding layers, styles, and overlays to the project; each layer can be made public or remain hidden for the unauthenticated users

Figure D.13: The style editor utilizes the `monako-editor` library developed by Microsoft; the editor supports defining custom rules for code completion.

Appendix E

Contents of Enclosed CD

- `readme.txt` ..... description of CD contents
- `imgs` ..... the directory containing example image outputs
- `metacity` ..... source code of the processing toolkit
- `metacity-workspace` ..... source code of the visualization package
- `thesis` ..... the directory containing LaTeX source codes of the thesis
- `thesis.pdf` ..... thesis in PDF format