Urban Data Visualization

Vojtěch Tomas[*]
Supervised by: David Sedláček [†]

Department of Computer Graphics and Interaction
Czech Technical University in Prague
Prague / Czech Republic

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 paper presents a design and implementation of a modular urban data visualization system. The modular design allows extending the visualization system, which helps answer questions as they arise during problem exploration. The paper explores the role of visualization platforms in urban development. Technical topics such as possible geometry representations and ways to process large geospatial datasets are discussed. A simple and extensible styling language is proposed to enable visualization customization based on the object metadata. The system is available as two Python/C++ packages. The first package focuses on data processing, utilizing spatial and temporal acceleration data structures, while the second encapsulates a WebGL-based visualization application. An iterative qualitative user study validated the proposed solution's performance and accessibility.

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

1 Introduction

The city is a socio-technical system that evolves naturally in time. On one side, decision-makers plan changes and influence citizens' daily lives. On the other side, citizens adjust and feed new data into the system, effectively influencing future planning. The effectiveness of this loop can be increased by enabling cooperation throughout the participating groups. 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. This paper presents principles and tools that could help find common ground and provide a unified view of the city's state.

[*]tomasvo1@fit.cvut.cz
[†]david.sedlacek@fel.cvut.cz

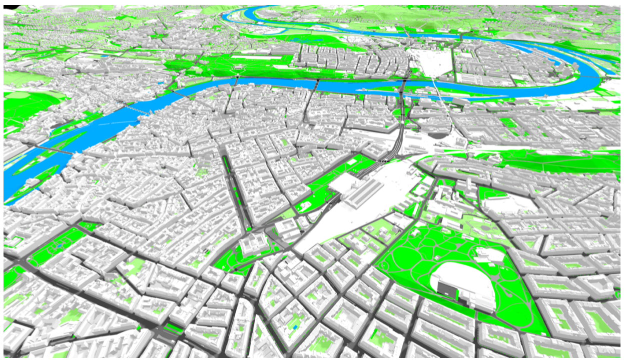

Figure 1: Render of the Prague city center displaying buildings and terrain overlayed with land use dataset, styled using a custom styling language based on the object metadata

The primary goal is to design and implement a visualization system that supports both virtual and physical presentation media, and acts as an environment for collaboration of city planners and stakeholders. 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.

2 Related Work

This section maps the current state of urban data sources and approaches to visualization. As data alone is not always a sufficient source of information for decision-making, alternative approaches using simulations are discussed.

2.1 Data Platforms

The topic of urban data processing and visualization is a focus area of several scientific fields, including Urban Informatics as presented by Foth et al. [1]. The sources of data are outlined by Robinson et al. [2] in a classification into three categories: Open Data, Remote Sensing, and Mobile Social Applications.

As presented by Barns [3], the data can be published on open platforms which work as data repositories or dashboards. Goldsmith and Crawford argued [4] 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"* [4]. However, according to [5], it is possible to observe barriers to effective data exchange and the reluctance of the participants to share data on open data platforms. 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 [6] proposes that one of the main reasons why the smart city transformation has stalled so far is the lack of cohesion between the public and private sectors.

2.2 Geospatial Information Systems

In the field of geospatial information systems, there is a range of both open-source and commercial projects, which focus on urban data visualization. Some offer a complex ecosystem of applications, such as ArcGIS [7], QGIS [8], or Cesium [9]. Additionally, there is a range of frameworks for urban data visualization, such as Mapbox [10], 3DCityDB [11], or LuciadRIA [12]. Most of these tools support visualization on the web, which makes the content easily accessible and collaboration with other users simpler. Typically, these systems are proprietary, complex and their extensibility is limited.

2.3 Urban Modeling Tools

Urban data alone is not always sufficient input for city planning, the principles of visual analytics [13] can be applied. Prior research in the area of geovisual analytics for decision making has been conducted by Andrienko et al. [14]. The practical applicability is well-illustrated by Winder [15]. The MIT City Science Research Group has taken a similar approach with the CityScope platform [16].

The CityScope is particularly interesting from the software architecture perspective. It allows integration of virtually any simulation software with the visualization frontend [17]. It enables the usage of both agent-based [18, 19] and cellular-automata-based [20] simulation tools, which are becoming increasingly popular.

Underkoffler and Ishii [21, 22] conducted several experiments and introduced a framework called Urp — a system for urban planning — overcoming the inherent incompatibility of 2D digital media, 3D physical models, and dynamic simulations. It enabled the projection of information onto objects' surfaces while the objects acted both as the projection plane and as a controller. Winder [15] 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.

3 Analysis

One of the initial goals was to devise principles on which it would be possible to base the design of the visualization framework. These principles draw from the existing work, as well as from the analysis of existing urban data.

3.1 Available Urban Data

An analysis of the available open urban data of the city of Prague [23] was conducted. The data is divided into 38 categories; some contain more than one dataset. Weighted by the number of datasets in each category, the vast majority of the publicly available data utilizes vector representation. Without exception, all vector data is available in the Shapefile [24] format.

The available datasets usually contain about 10^6–10^8 geometrical primitives. The count increases when several datasets are overlayed; therefore, it is desirable to design the system around the utilization of out-of-core memory.

The data is also available in CityGML [25], and DXF [26] formats. 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 [27], although adopting this standard is not yet observable in the open data field.

The output of the simulation software such as MATSim or GAMA is usually an XML file with the description of the dynamic data, which needs to be parsed further.

3.2 Existing Applications

Using dashboards for data presentations is reasonably straightforward. While these platforms can help answer simple questions, they alone cannot improve communication between stakeholders. It is undoubtedly possible to gradually extend the dashboards and 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. A viable alternative is to employ the principles of visual analytics in combination with physical media as the user interface.

From the software architecture perspective, most of the existing solutions are similar. 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 ability to share the visualization is crucial, which is apparent from the popularity of web-based solutions.

The aim is not to create a photo-realistically accurate visualization; the focus is the presentation of object metadata and the ability to share the visualization with collaborators. A shortcoming of the physical user interfaces is that they requires the on-site presence of the participants.

3.3 Design Principles

Based on the previous research, the design of the urban data system should follow these principles:

1. use a format-independent data model,
2. separate semantic information and geometry,
3. enable sharing with web-based visualization,
4. focus on metadata presentation,
5. and use a modular architecture to help answer questions as they arise during problem exploration.

4 Framework Design

The system is divided into three main components — a user interface, a server component, and a processing toolkit, illustrated in Figure 2. The data processing toolkit is designed to be a standalone tool utilized by the server component. The user interface (visualization engine) is a web application communicating with the processing toolkit through the server component.

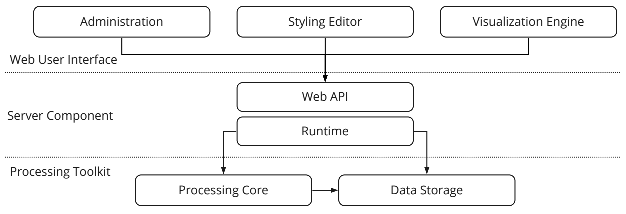

Figure 2: System Structure

4.1 Data Processing

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.

Data Model Previous research revealed that most available datasets are currently available in the Shapefile format. This observation indicates that narrowing the focus field to vector data is possible. The proposed data model is presented in Figure 3. 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*. When an overlay of two *Layers* 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.

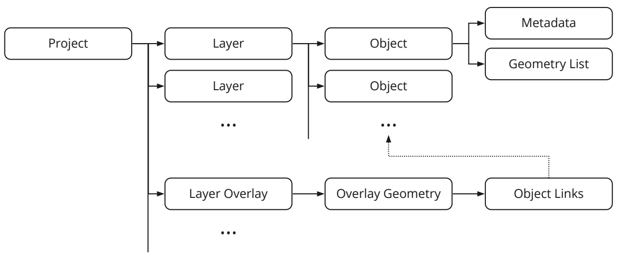

Figure 3: Proposed Data Model

Data Storage Utilizing a database for data storage has several advantages (e.g., scalability, performance, easy integration with web services); it also adds licensing issues, a system dependency, and requires each user to install the database software first if they wish to use the toolkit locally. The proposed alternative utilizes a local file system. The directory hierarchy follows the design of the data model, see figure 3. Geometry and metadata are stored in separate *Data Storage Directories*, which speeds up the data retrieval when only one component is required.

As illustrated in Figure 4, all data are clustered into data chunks, allowing to move only a portion of the dataset into memory at a time. The data management remains transparent to the user, and the storage appears as an ordinary array of objects. An important parameter is the granularity of the data chunks — 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.

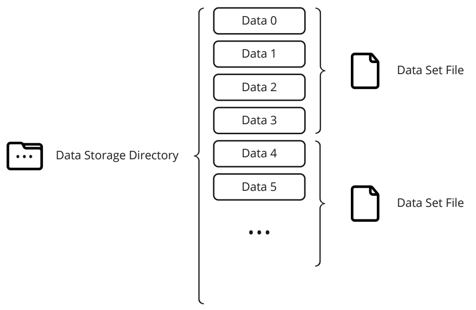

Figure 4: Individual data items in *Data Storage Directories* are packed into chunks and stored in separate Data Set Files

Data Preprocessing for Streaming None of the concepts so far directly addressed the spatial layout of the data. The array-like approach utilized by data storage al-

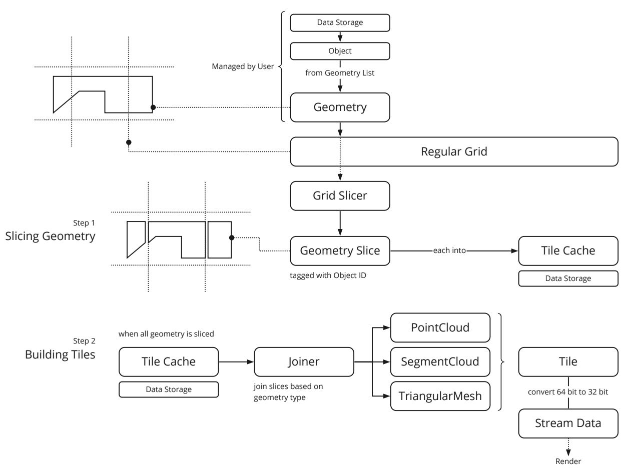

Figure 5: Grid Construction

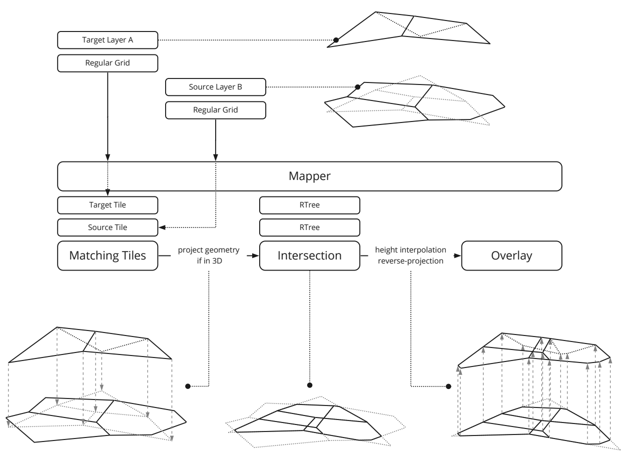

Figure 6: Layer Overlay Mapping

lows pre-sorting the objects based on their position; however, utilizing a regular grid would further speed up frequent spatial queries. Upon construction of the grid, 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 effectively load the entire dataset into memory and then process it, which is not always possible. A proposed memory-efficient approach works in two steps:

1. for each tile, building a cache containing sliced primitives inside the tile

2. rebuilding the tile geometry from the cache

When inserting the sliced geometry into the cache, it is possible to store the data in chunks into out-of-core memory, minimizing the memory footprint of each tile. The entire process is illustrated in Figure 5.

A similar approach can be taken with the dynamic data. An analogy of a regular grid for dynamic data is slicing the data into intervals and storing each interval individually. Depending on the size of the input data, the construction process can be similarly memory-optimized using disk cache.

Dataset Mapping The goal is to compute an overlay of two datasets. The focus is 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. Two approaches were considered: either using the classical plane sweep `MapOverlay` algorithm [28] for the overlays of two doubly connected edge lists (DCEL), or computing an overlap of two triangulated meshes using a parallel traversal of two R-trees built with top-down greedy split construction strategy [29]. The latter was chosen since the polygonal geometry needs to be triangulated to be displayed using the visualization engine. Also, the strategy using R-trees can efficiently work with point, line, or mesh data, and it is easy to extend for overlays of 3D and 2D datasets, as illustrated in Figure 6.

4.2 Visualization Engine

The architecture of the visualization engine takes into account the limitations imposed by web technologies. The engine architecture mirrors the layout of the processing toolkit components and the data model structure. The engine prioritizes processing tiles close to the viewer to those further away. Since the individual geometries are independent, it is possible to process them in parallel (data parallelism approach). All metadata is stored on the server only and presented on-demand.

Styling The key idea is to change the geometry appearance based on the object metadata. For demonstrational purposes, this research focuses only on color as a stylable property common among all supported geometry types. Each object represents a virtual structure and is assigned a color based on its properties. 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 1 is proposed describing the designed styling language. The language supports both static colors as well as the definition of colormaps. Upon submission, a context-free grammar parser generates a styling table based on the predefined grammar and provided user styles. 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.

Exports The geometric data has to be exported to support the creation of physical user interfaces. Two ap-

```
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 1: Styling language grammar, terminals in black; nonterminals in grey, asterisk as any number of occurrences; plus sign as one or more occurrences; round brackets in grey denote a group of symbols.

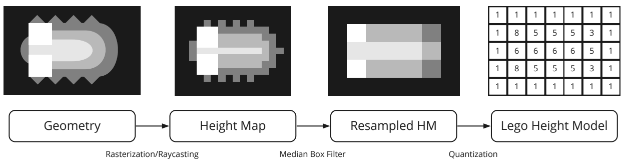

Figure 7: Transforming Geometry into LEGO

proaches were considered — 3D printing and assembly of LEGO models.

Printing the 3D models requires exporting the data into printer-compatible formats (e.g., STL, OBJ) and slicing large models into smaller tiles. 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.

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 7. First, a heightmap is rendered with a resolution larger than the brick resolution of the output. The heightmap is then filtered using a box filter. Minimum, maximum, median, and average box filters were tested and the median filter seems to yield the best results. The filtered heightmap is quantized to match the scale of the bricks to the original geometry.

5 Results

The 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 and use WebGL for 3D graphics. Shapefile and GeoJSON input formats are supported. The deployed application runs behind an Nginx server and uses the Nginx Unit for FastAPI apps.

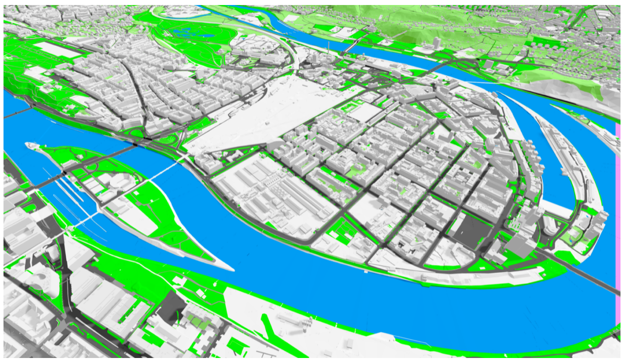

Figure 8: Render of the area Prague-Holešovice

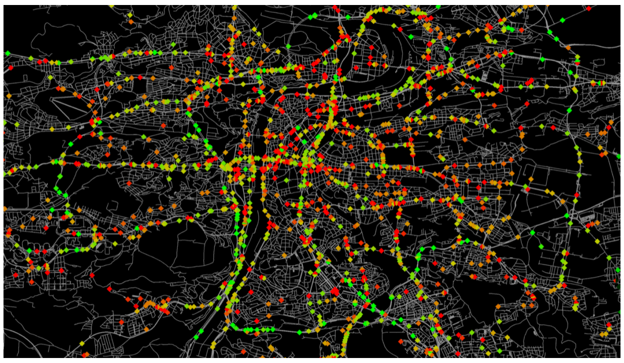

Figure 9: Visualization of traffic simulation, the color indicates the speed at which the agents are moving

The implementation was tested and validated using a series of static and dynamic datasets, see Table 1. The Road 3D and Land Use 3D datasets are actually only Layer Overlays. 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 GeForce GTX 1060 6GT with Chrome 95.0.4638. The utilization of the out-of-core memory enabled predictable memory consumption, see Figure 12. The performance of the visualization was tested in a series of tests where utilized memory and frame rate were monitored. The implementation behaves as expected, maintaining 60 frames per second for average available datasets (see Table 1).

An iterative qualitative user study with three participants was conducted during the development. All participants were familiar with geospatial information systems (ArcGIS and QGIS). The user interface was briefly presented at the testing site to the participants, who were later asked to locate a particular building and interact with the system to obtain metadata for various targets. The participants appreciated the possibility of seeing the city in both 2D and 3D, as the 3D environment provided a richer context. Several issues connected to navigation were identified and removed in the later versions of the system.

Figures 1, 8–11 showcase selected examples of the visualization outputs, presenting both static and dynamic data. User-selected regions can be exported as OBJ file or as a

static dataset general information					grid statistics					
					primitive count per tile					
layer	primitive type	object count	geometry size [MB]	metadata size [MB]	min	max	avg	median	stdev	primitive count
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					
					primitive count per interval					
layer	primitive type	object count	geometry size [MB]	metadata size [MB]	min	max	avg	median	stdev	primitive count
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 1: Data storage, regular grid and timeline statistics for test datasets; the geometry is tiled into 1×1 km tiles or 60-second intervals

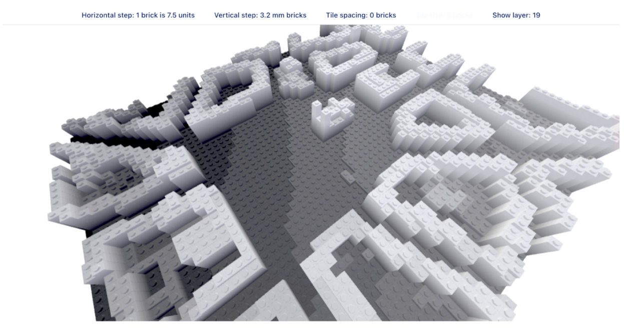
Horizontal step: 1 brick is 7.5 units
Vertical step: 3.2 mm bricks
Tile spacing: 0 bricks
Show layer: 19

Figure 10: The presentation of the exported LEGO model

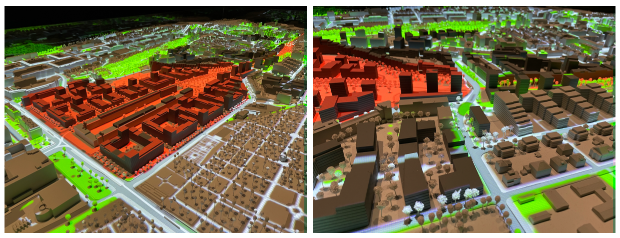

Figure 11: Visualization output projected onto a scale model of the Žižkov Freight Railway Station area, courtesy of CAMP Prague. The red zone highlights the location to be redeveloped into a new residential area

LEGO model. The export page of the LEGO model offers an interactive presentation (Figure 10). It is possible to divide the model into tiles and layers. The projection mapping of the visual outputs was tested using a scale model located at the Center for Architecture and Metropolitan Planning in Prague (see Figure 11).

6 Conclusion

The designed and implemented visualization framework enables efficient data processing and accessible visualization on the web. The internal data model is designed to facilitate a wide range of geometry types, supporting both static and dynamic data. The processing of large datasets is enabled by using out-of-core memory and spatial and temporal data structures and features a custom styling language allowing customization of the geometry appearance based on the object's metadata.

The system was implemented as two Python packages. 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 future objectives include extending the styling system and adding support for other input formats. The complexity of the geometry increases with the application of transformations such as overlay mapping, and it is desirable to adjust the geometry level of detail and further improve the system's performance. The most challenging objective is integrating the system with a simulation modeling tool. As the initial research suggests, pursuing this goal could significantly contribute to sustainable city development.

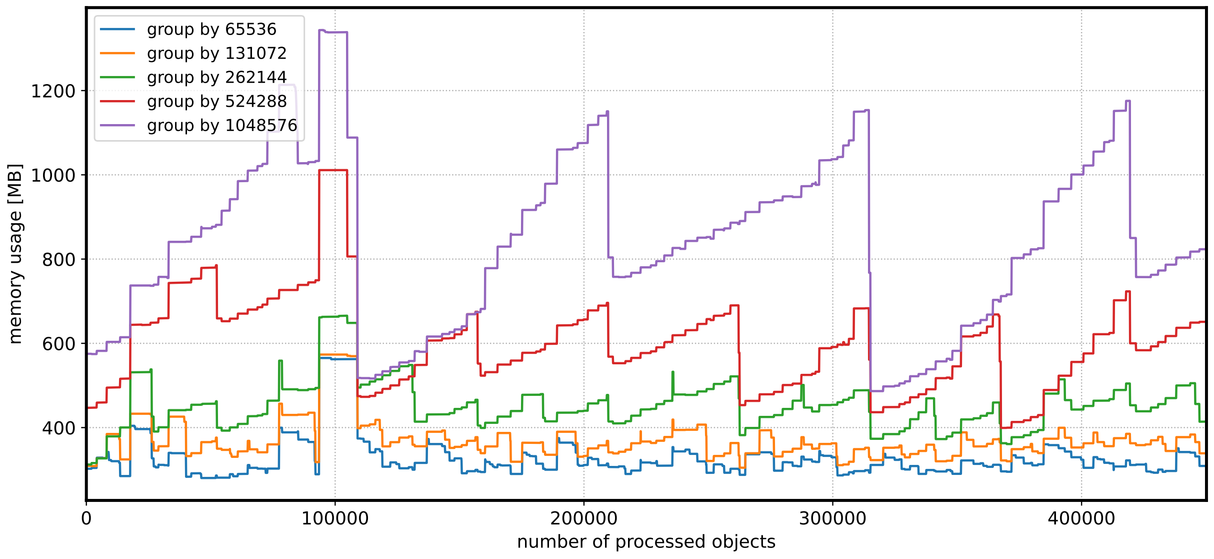

Figure 12: Profiling memory consumption during datasets Terrain (4502183 objects) loading; different sizes of data storage sets are tested

References

[1] Marcus Foth, Jaz Hee-jeong Choi, and Christine Satchell. Urban informatics. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 1–8, 2011.

[2] Ricky Robinson, Markus Rittenbruch, Marcus Foth, Daniel Filonik, and Stephen Viller. Street computing: Towards an integrated open data application programming interface (API) for cities. *Journal of Urban Technology*, 19(2):1–23, 2012.

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

[4] Stephen Goldsmith and Susan Crawford. *The responsive city: Engaging communities through data-smart governance*. John Wiley & Sons, 2014.

[5] Municipality of Copenhagen and Capital Region of Denmark. City data exchange - lessons learned from a private/public data collaboration [online]. 2018 [cit. 15. 10. 2021]. Available from: https://cphsolutionslab.dk/en/projekter/data-platforms/city-data-exchange.

[6] Rick Robinson. Why smart cities still aren't working for us after 20 years. And how we can fix them. [online]. 2016 [cit. 18. 10. 2021]. Available from: https://theurbantechnologist.com/2016/02/01/.

[7] 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.

[8] QGIS Development Team. *QGIS Geographic Information System [online]*. 2021. [cit. 20.11.2021]. Available from: https://www.qgis.org.

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

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

[11] Zhihang Yao, Claus Nagel, Felix Kunde, György Hudra, Philipp Willkomm, Andreas Donaubauer, Thomas Adolphi, and Thomas H Kolbe. 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*, 3(1):1–26, 2018.

[12] 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.

[13] J.J. Thomas and K.A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, 2006.

[14] Gennady Andrienko, Natalia Andrienko, Daniel Keim, Alan M. MacEachren, and Stefan Wrobel.

Challenging problems of geospatial visual analytics. *Journal of Visual Languages & Computing*, 22(4):251–256, 2011.

[15] James Ira Winder and Cambridge Ma. Tangible Interactive Matrix for Real-time Computation and 3D Projection Mapping. *Future Technologies Conference (FTC)*, page 4, 2017.

[16] Mohammad Hadhrawi and Kent Larson. 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*, pages 105–108. ACM, 2016.

[17] CityScope ecosystem | here we build CityScope [online]. 2021 [cit. 10. 9. 2021]. Available from: https://cityscope.media.mit.edu.

[18] Patrick Taillandier, Benoit Gaudou, Arnaud Grignard, Quang-Nghi Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon, and Alexis Drogoul. Building, composing and experimenting complex spatial models with the GAMA platform. 23(2):299–322, 2019.

[19] K. W Axhausen, Andreas Horni, and Kai Nagel. *The multi-agent transport simulation MATSim*. 2016. OCLC: 1116114273.

[20] Inés Santé, Andrés M. García, David Miranda, and Rafael Crecente. Cellular automata models for the simulation of real-world urban processes: A review and analysis. *Landscape and Urban Planning*, 96(2):108–122, 2010.

[21] John Underkoffler and Hiroshi Ishii. Urp: a luminous-tangible workbench for urban planning and design. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 386–393. ACM Press, 1999.

[22] John Underkoffler, Brygg Ullmer, and Hiroshi Ishii. Emancipated pixels: real-world graphics in the luminous room. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, pages 385–392. ACM Press, 1999.

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

[24] ESRI. Esri shapefile technical description. *An ESRI white paper*, 4:1, 1998.

[25] Gerhard Gröger, Thomas H Kolbe, Claus Nagel, and Karl-Heinz Häfele. OGC city geography markup language (CityGML) encoding standard. 2012.

[26] Autodesk, Inc. Dxf reference. Technical report, Autodesk, Inc., San Rafael; CA 94903; USA, 2011.

[27] Llewellyn Tang, Chao Chen, Shu Tang, Zhuoqian Wu, and Polina Trofimova. Building information modeling and building performance optimization. In *Encyclopedia of Sustainable Technologies*, pages 311 – 320. Elsevier, Oxford, 2017.

[28] Mark d. Berg, Otfried Cheong, Marc v. Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, third edition, 2008.

[29] Yván J. García R, Mario A. López, and Scott T. Leutenegger. A greedy algorithm for bulk loading R-trees. In *Proceedings of the sixth ACM international symposium on Advances in geographic information systems - GIS '98*, pages 163–164. ACM Press, 1998.