HASSO-PLATTNER-INSTITUT
Fachgebiet Computergrafische Systeme

Web-Based Provisioning and Application of Large-Scale Virtual 3D City Models

Dissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Praktische Informatik

eingereicht an der
Digital Engineering Fakultät
der Universität Potsdam

von
Jan Klimke, M.Sc.

Potsdam,
25. April 2019

Datum der Disputation: 8. April 2019

Betreuer

Prof. Dr. Jürgen Döllner
Prof. Dr. Robert Hirschfeld

Gutachter

Prof. Dr. Jürgen Döllner
Prof. Dr. Hartmut Asche
Prof. Dr. Birgit Kleinschmit

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-42805
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-428053

Abstract

Virtual 3D city models represent and integrate a variety of spatial data and georeferenced data related to urban areas. With the help of improved remote-sensing technology, official 3D cadastral data, open data or geodata crowdsourcing, the quantity and availability of such data are constantly expanding and its quality is ever improving for many major cities and metropolitan regions. There are numerous fields of applications for such data, including city planning and development, environmental analysis and simulation, disaster and risk management, navigation systems, and interactive city maps.

The dissemination and the interactive use of virtual 3D city models represent key technical functionality required by nearly all corresponding systems, services, and applications. The size and complexity of virtual 3D city models, their management, their handling, and especially their visualization represent challenging tasks. For example, mobile applications can hardly handle these models due to their massive data volume and data heterogeneity. Therefore, the efficient usage of all computational resources (e.g., storage, processing power, main memory, and graphics hardware, etc.) is a key requirement for software engineering in this field. Common approaches are based on complex clients that require the 3D model data (e.g., 3D meshes and 2D textures) to be transferred to them and that then render those received 3D models. However, these applications have to implement most stages of the visualization pipeline on client side. Thus, as high-quality 3D rendering processes strongly depend on locally available computer graphics resources, software engineering faces the challenge of building robust cross-platform client implementations.

Web-based provisioning aims at providing a service-oriented software architecture that consists of tailored functional components for building web-based and mobile applications that manage and visualize virtual 3D city models. This thesis presents corresponding concepts and techniques for web-based provisioning of virtual 3D city models. In particular, it introduces services that allow us to efficiently build applications for virtual 3D city models based on a fine-grained service concept. The thesis covers five main areas:

1. **A Service-Based Concept for Image-Based Provisioning of Virtual 3D City Models** It creates a frame for a broad range of services related to the rendering and image-based dissemination of virtual 3D city models.

2. **3D Rendering Service for Virtual 3D City Models** This service provides efficient, high-quality 3D rendering functionality for virtual 3D city models. In particular, it copes with requirements such as standardized data formats, massive model texturing, detailed 3D geometry, access to associated feature data, and non-assumed frame-to-frame coherence for parallel service requests. In addition, it supports thematic and artistic styling based on an expandable graphics effects library.

3. **Layered Map Service for Virtual 3D City Models** It generates a map-like representation of virtual 3D city models using an oblique view. It provides high visual quality, fast initial loading times, simple map-based interaction and feature

data access. Based on a configurable client framework, mobile and web-based applications for virtual 3D city models can be created easily.

4. **Video Service for Virtual 3D City Models** It creates and synthesizes videos from virtual 3D city models. Without requiring client-side 3D rendering capabilities, users can create camera paths by a map-based user interface, configure scene contents, styling, image overlays, text overlays, and their transitions. The service significantly reduces the manual effort typically required to produce such videos. The videos can automatically be updated when the underlying data changes.

5. **Service-Based Camera Interaction** It supports task-based 3D camera interactions, which can be integrated seamlessly into service-based visualization applications. It is demonstrated how to build such web-based interactive applications for virtual 3D city models using this camera service.

These contributions provide a framework for design, implementation, and deployment of future web-based applications, systems, and services for virtual 3D city models. The approach shows how to decompose the complex, monolithic functionality of current 3D geovisualization systems into independently designed, implemented, and operated service-oriented units. In that sense, this thesis also contributes to microservice architectures for 3D geovisualization systems—a key challenge of today's IT systems engineering to build scalable IT solutions.

Zuammenfassung

Virtuelle 3D-Stadtmodelle repräsentieren und integrieren eine große Bandbreite von Geodaten und georeferenzierten Daten über städtische Gebiete. Verfügbarkeit, Quantität und Qualität solcher Daten verbessern sich ständig für viele Städte und Metropolregionen, nicht zuletzt bedingt durch verbesserte Erfassungstechnologien, amtliche 3D-Kataster, offene Geodaten oder Geodaten-Crowdsourcing. Die Anwendungsfelder für virtuelle 3D-Stadtmodelle sind vielfältig. Sie reichen von Stadtplanung und Stadtentwicklung, Umweltanalysen und -simulationen, über Katastrophen- und Risikomanagement, bis hin zu Navigationssystemen und interaktiven Stadtkarten.

Die Verbreitung und interaktive Nutzung von virtuellen 3D-Stadtmodellen stellt hierbei eine technische Kernfunktionalität für fast alle entsprechenden Systeme, Services und Anwendungen dar. Aufgrund der Komplexität und Größe virtueller 3D-Stadtmodelle stellt ihre Verwaltung, ihre Verarbeitung und insbesondere ihre Visualisierung eine große Herausforderung dar. Daher können zum Beispiel mobile Anwendungen virtuelle 3D-Stadtmodelle, wegen ihres massiven Datenvolumens und ihrer Datenheterogenität, kaum effizient handhaben. Die effiziente Nutzung von Rechenressourcen, wie zum Beispiel Prozessorleistung, Hauptspeicher, Festplattenspeicher und Grafikhardware, bildet daher eine Schlüsselanforderung an die Softwaretechnik in diesem Bereich. Heutige Ansätze beruhen häufig auf komplexen Clients, zu denen 3D-Modelldaten (z.B. 3D-Netze und 2D-Texturen) transferiert werden müssen und die das Rendering dieser Daten selbst ausführen. Nachteilig ist dabei unter anderem, dass sie die meisten Stufen der Visualisierungspipeline auf der Client-Seite ausführen müssen. Es ist daher softwaretechnisch schwer, robuste Cross-Plattform-Implementierungen für diese Clients zu erstellen, da hoch qualitative 3D-Rendering-Prozesse nicht unwesentlich von lokalen computergrafischen Ressourcen abhängen.

Die *webbasierte Bereitstellung virtueller 3D-Stadtmodelle* beruht auf einer serviceorientierten Softwarearchitektur. Diese besteht aus spezifischen funktionalen Komponenten für die Konstruktion von mobilen oder webbasierten Anwendungen für die Verarbeitung und Visualisierung von komplexen virtuellen 3D-Stadtmodellen. Diese Arbeit beschreibt entsprechende Konzepte und Techniken für eine webbasierte Bereitstellung von virtuellen 3D-Stadtmodellen. Es werden insbesondere Services vorgestellt, die eine effiziente Entwicklung von Anwendungen für virtuelle 3D-Stadtmodelle auf Basis eines feingranularen Dienstekonzepts ermöglichen. Die Arbeit gliedert sich in fünf thematische Hauptbeiträge:

1. **Ein servicebasiertes Konzept für die bildbasierte Bereitstellung von virtuellen 3D-Stadtmodellen** Es wird ein konzeptioneller Rahmen für eine Reihe von Services in Bezug auf das Rendering und die bildbasierte Bereitstellung virtueller 3D-Stadtmodelle eingeführt.

2. **3D-Rendering-Service für virtuelle 3D-Stadtmodelle** Dieser Service stellt eine effiziente, hochqualitative 3D-Renderingfunktionalität für virtuelle 3D-Stadtmodelle bereit. Insbesondere werden Anforderungen, wie zum Beispiel standardisierte Datenformate, massive Modelltexturierung, detaillierte 3D-Geometrien, Zugriff auf assoziierte Fachdaten und fehlende Frame-zu-Frame-Kohärenz bei parallelen Service-Anfragen erfüllt. Der Service unterstützt zudem die thematische und gestalterische Stilisierung der Darstellungen auf Basis einer erweiterbaren Grafikeffektbibliothek.

3. **Layered-Map-Service für virtuelle 3D-Stadtmodelle** Dieser Service generiert eine kartenverwandte Darstellung in Form einer Schrägansicht auf virtuelle 3D-Stadtmodelle in hoher Renderingqualität. Er weist eine schnelle initiale Ladezeit, eine einfache, kartenbasierte Interaktion und Zugang zu den Fachdaten des virtuellen 3D-Stadtmodells auf. Mittels eines konfigurierbaren Client-Frameworks können damit sowohl mobile, als auch webbasierte Anwendungen für virtuelle 3D Stadtmodelle einfach erstellt werden.

4. **Video-Service für virtuelle 3D-Stadtmodelle** Dieser Service erstellt und synthetisiert Videos aus virtuellen 3D-Stadtmodellen. Nutzern wird ermöglicht 3D-Kamerapfade auf einfache Weise über eine kartenbasierte Nutzungsschnittstelle zu erstellen. Weiterhin können die Szeneninhalte, die Stilisierung der Szene, sowie Bild- und Textüberlagerungen konfigurieren und Übergänge zwischen einzelnen Szenen festzulegen, ohne dabei clientseitige 3D-Rendering-Fähigkeiten vorauszusetzen. Das System reduziert den manuellen Aufwand für die Produktion von Videos für virtuelle 3D-Stadtmodelle erheblich. Videos können zudem automatisiert aktualisiert werden, wenn sich zugrunde liegende Daten ändern.

5. **Servicebasierte Kamerainteraktion** Die vorgestellten Services unterstützen aufgabenbasierte 3D-Kamerainteraktionen und deren Integration in servicebasierte Visualisierungsanwendungen. Es wird gezeigt, wie webbasierte interaktive Anwendungen für virtuelle 3D-Stadtmodelle mit Hilfe von Kameraservices umgesetzt werden können.

Diese Beiträge bieten einen Rahmen für das Design, die Implementierung und die Bereitstellung zukünftiger webbasierter Anwendungen, Systeme und Services für virtuelle 3D-Stadtmodelle. Der Ansatz zeigt, wie die meist komplexe, monolithische Funktionalität heutiger 3D-Geovisualisierungssysteme in unabhängig entworfene, implementierte und betriebene serviceorientierte Einheiten zerlegt werden kann. In diesem Sinne stellt diese Arbeit auch einen Beitrag für die Entwicklung von Microservice-Architekturen für 3D-Geovisualisierungssysteme bereit – eine aktuelle Herausforderung in der Softwaresystemtechnik in Hinblick auf den Aufbau skalierender IT-Lösungen.

Acknowledgments

This dissertation is the result of my research work at the Computer Graphics Systems Group at the Hasso-Plattner-Institute at the University of Potsdam. I thank my advisor Prof. Dr. Jürgen Döllner for granting me this opportunity and for the support I received over the years. I would also like to thank Prof. Dr. Birgit Kleinschmit and Prof. Dr. Hartmut Asche who kindly agreed to review this thesis. Thanks also to Prof. Dr. Robert Hirschfeld for his support during the dissertation process.

I thank the Research School for "Service-Oriented Systems Engineering" at the Hasso-Plattner-Institute for providing me with financial support, valuable feedback regarding my work, and the possibility to extend my personal and scientific horizon.

Furthermore, I would like to express my special thanks to my colleague and friend Benjamin Hagedorn for working with me over the years, for pushing me gently towards finishing this thesis, and especially for his detailed feedback.

It is a great pleasure to thank everyone who supported me during the time of my studies and in writing this dissertation, in particular my colleagues at the Computer Graphics Systems group at the Hasso-Plattner-Institute. To name name a few I want to thank Marcel Pursche, Sebastian Hahn, Rico Richter, and Daniel Limberger for fruitful discussions and collaborations.

I am particularly grateful for the assistance of Franziska Krüger, who proofread this thesis and gave comments and ideas to improve its language.

I am grateful to my family for their support and making it possible for me to study computer science. Finally, I would like to thank Nadine Ziemann for her support and for motivating me to finish this thesis.

This work has been partially funded by the German Federal Ministry of Economic Affairs and Energy.

Potsdam, Germany, April 25, 2019 *Jan Klimke*

Acronyms

3D GeoVE 3D Geovirtual Environment

3DPIE 3D Portrayal Interoperability Experiment

3DPS 3D Portrayal Service

ADE Application Domain Extension

API Application Programming Interface

BLC Business Location Center

CAD Computer Aided Design

CityGML City Geography Markup Language

CPU Central Processing Unit

DTM Digital Terrain Model

GIS Geographic Information System

GLSL OpenGL Shader Language

GML Geography Markup Language

GPU Grahpics Processing Unit

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

LOD Level of Detail

OGC Open Geospatial Consortium

POI Point of Interest

SE Symbology Encoding

SLD Styled Layer Descriptor

SOA Service-Oriented Architecture

SOC Service-Oriented Computing

TMS OpenGeo Tiled Map Service

URL Uniform Resource Locator

VE Virtual Environment

VRML Virtual Reality Modeling Language

W3DS Web 3D Service

WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WMS-T	Web Map Tile Service
WPS	Web Processing Service
WVS	Web View Service
X3D	Extensible 3D

Contents

Chapter 1

Introduction

This chapter provides an introduction into the the context of thesis, its motivation, and its main contributions. Furthermore, it presents the structure of the remaining chapters.

1.1 Virtual 3D City Models

This thesis focuses on service-oriented concepts and techniques for a web-based access and visualization of virtual 3D city models. These models denote a fundamental category of 3D geospatial models for a broad range of 3D geospatial applications.

"Virtual 3D city models represent spatial and geo-referenced urban data by means of 3D geovirtual environments that basically include terrain models, building models, vegetation models as well as models of roads and transportation systems. In general, these models serve to present, explore, analyze, and manage urban data. As a characteristic element, virtual 3D city models allow for visually integrating heterogeneous geoinformation within a single framework and, therefore, create and manage complex urban information spaces." [DBB06] The web-based access and visualization of virtual 3D city models represent key challenges for digital transformation processes and for the design, implementation, and deployment of corresponding IT applications, systems, and services.

The areas of application for virtual 3D city models are continuously growing and include the following: city planning [XQ06; Fer$^+$15; PIB17] , urban design [XQ06], disaster and emergency management [KGP05; Mor$^+$11; Dem$^+$16], physically-based simulation (estimation of solar radiation [Wie$^+$15; Cha$^+$17], shadow and lighting simulation [LH05; Bil$^+$16], noise distribution [KL16; Zha$^+$16; LBL17; Kum$^+$17], or air quality), energy application [Düb$^+$11; Str$^+$11; KK12; Nou$^+$14], infrastructure planning [Ros10; Nou$^+$14; SK14], navigation [SR08; Cap$^+$12], mobile network planning [Lie$^+$10], or collaborative urban problem solving and public participation [KD10; WHG10; DK14; Bra$^+$16]. Several studies examine the specific conditions and constraints that affect the successful implementation of virtual 3D city models in specific domains (e.g., urban planning [HC15]). As Biljecki et al. point out, virtual 3D city models "have been predominantly used for visualisation; however, today they are being increasingly employed in a number of domains and for a large range of tasks beyond visualisation." [Bil$^+$15] That is, virtual 3D city models are becoming more and more *computational models*.

All of these applications rely on 3D visualization as a key feature to display, manipulate and evaluate 3D geoinformation. Consequently, visualization is a crucial part of nearly every application, service, and system that uses virtual 3D city models.

In recent years, computing paradigms have shifted towards mobile, distributed applications that run on a variety of devices and platforms. The desktop application as former primary target platform for visualization applications is increasingly challenged by the constantly growing demand for a user experience that spans different devices and platforms (noted as number one strategic trends “Gartner Top 10 Strategic Technology Trends for 2016”[1]). *Service-oriented architectures* (SOA) for visualization provides a key means to address issues of hardware and software heterogeneity. It also addresses the increasing demand for distribution of large-scale data especially common in the context of 3D visualization. Consequently, a future key factor for the success of technology related to virtual 3D city models will be their web-based architecture and deployment.

This thesis focuses on concepts and techniques for web-based provisioning, web-based interaction, and web-based applications of virtual 3D city models. To this end, it considers the following four research areas (Fig. 1.1):

3D Computer Graphics and the implementation of rendering techniques and data pre-processing techniques for 3D datasets;

Geovisualization for the interactive display of 3D geodata and georeferenced data;

Service-Oriented Computing providing tools and concepts for building distributed applications that integrate data from different service sources and requires services to be orchestrated; and

3D User Interaction managing the interaction between the user and the *3D geovirtual environment* (3D GeoVE).

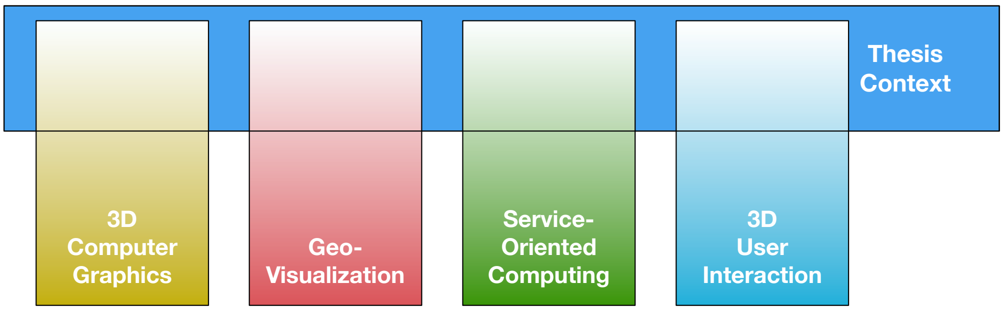


Figure 1.1: *Main research areas of this thesis: a) 3D Computer Graphics, b) Geovisualization, c) Service-Oriented Computing, and d) 3D User Interaction.*

This thesis is inspired by the vision to design and implement geovisualization-oriented service components as small, independently usable and reusable functional components that are connected over a network. It aims to develop an alternative to already existing monolithic approaches and software architectures. Accordingly, it incorporates a growing number of standards set by the *Open Geospatial Consortium (OGC)* that define modeling, encoding, and retrieval of 2D and 3D geodata, such as feature geometry and attributes, maps, and renderable 3D scenes. In particular, CityGML [Kol09] provides a widely accepted standard for semantic, geometric, and appearance modeling of virtual 3D city models. Recently, the OGC released a standard for the retrieval of scene data and image depictions of 3D GeoVEs called *3D Portrayal Service (3DPS)* [Hag+17].

[1] `https://www.gartner.com/doc/3231617`, last accessed 2016/05/06

There are numerous solutions that implement virtual 3D city model preprocessing and visualization using client-side 3D rendering. However, image-based provisioning of virtual 3D city models has not yet received a broad attention in research and application yet, but will certainly become a crucial technology requirement for distributed, web-based or mobile applications and systems. Image-based approaches allow for a robust distribution of depictions of virtual 3D city models. They offer major advantages such as visual quality that is independent of end-user hardware and avoids the transfer of massive raw geometry and texture data to client applications.

In this thesis, the term 'provisioning of virtual 3D city models' is used as follows:

Preprocessing, compacting, distributing, and visualizing virtual 3D city models so that they can be efficiently used within a distributed, web-based environment.

Provisioning also encompasses features such as data fusion, content selection, content authoring, and media encoding in such a way that the depictions of virtual 3D city models can be efficiently transmitted to and effectively used by client applications. It also includes media creation derived from virtual 3D city models such as synthesized digital videos.

1.2 Motivation

The general motivation of thesis is the idea to built a service-based framework that allows us to construct applications that use high-quality 3D visualization of massive, heterogeneous virtual 3D city models in a web-based environment with low software and hardware requirements, e.g., limited 3D rendering capabilities of mobile or low-end desktop clients.

In today's information technology and system architectures, the monolithic standalone systems are slowly disappearing and getting replaced by service-based, decentralized systems. Within those system, each service provides a dedicated, sometimes only small set of features. The provisioning of virtual 3D city models based on a SOA using a network (either intranet or the internet) simplifies the data management of complex data sources as required for virtual 3D city models. It facilitates the construction of application-specific solutions based on a toolbox of services, and leads to scalable approaches for the deployment of such service-based solutions. It also provides a solution to overcoming the hardware diversity regarding 3D rendering capabilities of today's end-user platforms.

A promising approach building such solutions are image-based approaches for provisioning virtual 3D city models using portrayal services where complex model data is kept on the server side. This approach effectively limits the resources required for accessing and displaying those models on the client side. Furthermore, the image-based portrayal as a core functionality of such systems allows for the provisioning of virtual 3D city models in a new and intuitive way by using different media (e.g., video clips) that are generated by higher-level services that rely on robust rendering services,and integrate this rendering functionality from different rendering services.

Taking the above-mentioned motivation into account, the following research questions have been identified that are addressed in this thesis:

1. **Service Decomposition.** How can the functionality of 3D geovisualization systems be decomposed into smaller units that can be operated and maintained individually? How can such systems be orchestrated (e.g., for *geodata infrastructures* (GDI)) and what are the requirements for each of these components (e.g., in terms of performance)?

2. **3D Rendering Services.** How can large-scale virtual 3D city models be rendered by a service-based approach and in a robust and efficient way on different, possibly resource-limited end-user platforms and devices? In particular, efficient 3D rendering must be provided within the constraints of a service and its deployment. How should scalability aspects for serving multiple clients at the same time be addressed?

3. **Interactivity.** How can interaction be supported for clients that are not able to handle the 3D models locally (e.g., due to restricted hardware capabilities or if raw data is not accessible due to privacy reasons)?

4. **Media Generation.** How can services provide features to automatically synthesize standard visual media such as digital videos for virtual 3D city models in an automated manner? And how can those be processed automatically, e.g., if the underlying data changes?

1.3 Contributions

The key contributions of this thesis include:

1. **A service-based concept for image-based provisioning of virtual 3D city models** by a group of geovisualization services that address layered maps, camera control, and video export as key features required by applications and image-based provisioning of virtual 3D city models (Chapter 4).

2. **Design and implementation of a 3D rendering service for virtual 3D city models** that takes into account the specific geometric and graphics characteristics of virtual 3D city models as well as the requirements for building data pipelines between different services operating on images of virtual 3D city models (Chapter 5).

3. **Design and implementation of a layered map service for virtual 3D city models** generating so called *layered maps* as tile-based, map-like oblique views of virtual 3D city models, which allow for exporting the contents of virtual 3D city models by means of standard tile-based map chunks. (Chapter 6).

4. **Design and implementation of a video service** that synthesizes videos for virtual 3D city models (Chapter 7) based on a high-level description of the underlying camera animation.

5. **Design and implementation of a camera interaction service** that supports 3D camera interaction and animations for virtual 3D city models within web-based geovisualization solutions (Chapter 8).

Parts of the work described in this thesis have been previously peer-reviewed and published as scientific papers and journal contributions. Selected key publications include:

- **J. Klimke**, B. Hagedorn, and J. Döllner, “A Service-Oriented Platform for Interactive 3D Web Mapping”, in Service-Oriented Mapping, 2012, pp. 127–139.
- J. Döllner, B. Hagedorn, and **J. Klimke**, “Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps”, in Proceedings of the 17th Int. Conf. on 3D Web Technology, 2012, pp. 97-100.
- **J. Klimke**, B. Hagedorn, and J. Döllner, “A Service-Based Concept for Camera Control in 3D Geovirtual Environments”, in Progress and New Trends in 3D Geoinformation Sciences, 2013, pp. 101-118.
- **J. Klimke**, B. Hagedorn, and J. Döllner, “Scalable Multi-Platform Distribution of Spatial 3D Contents”, Int. J. 3-D Inf. Model., vol. 3, no. 3, pp. 35-49, 2014.
- **J. Klimke**, B. Hagedorn and M. Trapp and J. Döllner “Web-based and Mobile Provisioning of Virtual 3D Reconstructions”, In R. Franken-Wendelstorf and E. Lindinger and J. Sieck, ed., Tagungsband der 12. Konferenz Kultur und Informatik: Reality and Virtuality, pages 17-28, 5 2014 Werner Hülsbusch Verlag.

The following prototypical components and services have been implemented as part of this thesis to show the feasibility of the service-based provisioning concepts:

3D Data Parsing and Optimization This component is responsible for importing virtual 3D city model contents. In particular, 3D geometry, 2D textures, and georeferenced data attributes are reorganized to allow for efficient access and rendering on a GPU. Furthermore, a database back-end has been developed to keep features, their attributes, and their hierarchy accessible and usable for the rendering process as well as for semantics-based styling.

3D Rendering Service This service implements techniques for efficient, high-quality 3D rendering of virtual 3D city models as well as an interface for data selection and retrieval from an underlying database as standardized, self-describing service interface based on the OGC 3DPS. It allows for efficient styling of city model objects based on their attributes (thematic visualization) using an XML-based description language. The implementation and rendering algorithms are specifically designed to handle large-scale data, i.e., large in size and spatial extent), textured 3D geodata, and the geometric and graphics characteristics of typical virtual 3D city models.

Layered Map Service This service exports virtual 3D city model contents by means of a stylized, tiled map-like oblique view (also called “3D map” or “bird’s eye view map”) of the virtual 3D city model. In addition, it allows for combining these views with additional information layers. For each map, contents can be configured, in particular, model elements can be selected such as terrain textures and client-side points-of-interests, or feature geometry.

Mobile and Web Applications based on Layered Maps Three proof-of-concept applications for mobile devices based on Apple iOS and Android operating systems have been implemented that are based on the layered map service. They provide application-specific user interfaces optimized for map usage. Consequently, these applications, consequently, form a kind of blueprint to build application-specific or customer-specific IT solutions based on the proposed services.

Video Service for Virtual 3D City Models This service synthesizes videos of virtual 3D city models; its implementation relies on 3D portrayal services. A service request delivers as result a digital video in a predefined video encoding standard. This service also provides a map-based front-end that allows users to configure the camera related parameters as well as the composition and timing of video elements (e.g., overlays, blendings).

Camera Service This service provides functionality for computing camera positions, viewing directions and camera paths within a virtual 3D city model. Its implementation relies on 3D portrayal services. With this service, applications can build advanced camera controls for virtual 3D city models in a service-oriented way without they themselves having to implement camera models or camera behavior functionality.

1.4 Structure

The remaining part of this thesis is structured as follows.

Chapter 2, *Foundations* introduces foundations and terminology of related topics of this thesis.

Chapter 3, *Related Work* provides an overview of previous research relevant for this thesis.

Chapter 4, *A Service-Based Concept for Image-Based Provisioning of Virtual 3D City Models* presents the overall service-oriented concept related to the image-based provisioning of virtual 3D city models, supporting multiple end-user platforms, devices, and media.

Chapter 5, *3D Rendering Service for Virtual 3D City Models* describes the 3D rendering service for virtual 3D city models and its implementation.

Chapter 6, *Layered Map Service for Virtual 3D City Models* presents the layered map service as well as its implementation and usage by web applications and mobile apps.

Chapter 7, *Video Service for Virtual 3D City Models* introduces the video service and its implementation.

Chapter 8, *Service-Based Camera Interaction* introduces the concept for camera services and its implementation.

Chapter 9, *Case Studies* provides an overview of the use cases that illustrate application and usage of the services introduced in this thesis.

Chapter 10, *Summary and Outlook* summarizes the key contributions of this thesis and provides an outlook of possible future research topics.

Chapter 2

Foundations

This chapter outlines key terminology and concepts for the geospatial area, computer graphics and visualization, service-orientation, and virtual 3D city models that are used throughout this thesis.

2.1 Foundations Related to Geospatial Topics

Geodata According to ISO/TC 211 standard terminology, *geodata* or *geographic data* is "data with implicit or explicit reference to a location relative to the Earth" [ISO14]. The term *geospatial data* is often used in the same way. Data linked to or related to spatial locations is frequently named *georeferenced data.* For example, a transaction for a payment made by credit card in a shop can be spatially referenced to the shop location. Therefore it denotes georeferenced data.

Geospatial Feature *Geospatial features* refer to digital representations of real-world entities or phenomena. "It has a spatial domain, a temporal domain, or a spatial/temporal domain as one of its attributes." [KRT09] For example, a park can be represented as a geospatial feature by its surrounding polygon.

Open Geospatial Consortium The *Open Geospatial Consortium* (OGC) is the international organization dedicated to developing open standards for the global geospatial community. OGC standards are used in industry and science to ensure interoperability and transparency. They are present in all fields of geoinformation systems, applications, and services in a variety of domains such as "Geosciences & Environment; Defense & Intelligence; Smart Cities, including IoT & Sensor Webs, mobile tech, and the 3D & Built Environment; Emergency Response & Disaster Management; Aviation; Energy & Utilities; and many more."[1]

Virtual 3D City Model A virtual 3D city model is a digital representation of geospatial features, geodata and georeferenced data of an urban area. It represents both man-made and natural objects in terms of 3D models such as building models as textured or untextured 3D geometry. See Section 2.4 for a more detailed description of this topic.

CityGML CityGML [Grö+12] "is a common information model for the representation of 3D urban objects" [Mao11]. It serves as modeling and encoding standard for virtual 3D city models defined by the OGC and is based on *Geography Markup*

[1] `https://www.opengeospatial.org` accessed at September 7th 2018

Language (GML) [Por07]. It allows for defining city model objects at different levels of detail, describes the appearance of its components, specifies structures and relationships among these elements, and assigns semantics according to entity categories.

KML *KML* [Bur15] is "an XML language focused on geographic visualization, including annotation of maps and images"; it is part of the technology on top of which GoogleEarth is implemented. KML includes not only the presentation of graphical data on the globe, but also the control of the user's navigation in the sense of where to go and where to look. Since KML is tightly related to virtual 3D globes, it is designed to cover common application cases such as placement of position markers, map images, or 2D and 3D objects.

Google Earth The *Google Earth*[2] desktop application, the most prominent example of a web-based 3D geovisualization application, supports exploring 2D and 3D geodata, fetched from external services or provided by users on their local machine. While fundamental geodata is provided by Google, the solution supports the integration of customer-specific 2D data (e.g., terrain textures or geometry terrain overlays provided by OGC *Web Map Service (WMS)* or as georeferenced image files) and 3D data (e.g., georeferenced 3D models encoded in OGC KML). It also provides a user interface for defining and rendering of virtual 3D tours and their video export. Classified as a medium to thick client application, 3D rendering in Google Earth is implemented on the client side, limiting the amount of data and rendering effects that can be applied to the current system configuration. Nevertheless, there have been efforts to integrate 3D portrayal services with Google Earth in the context of the OGC 3D Portrayal Interoperability Experiment (*3DPIE*) [SHC12], particularly using the *Web 3D Service (W3DS)* [SK10] for geometry-based 3D portrayal and *Web View Service* (*WVS*) [Hag10] for image-based 3D portrayal.

Virtual Environment *Virtual environment* (VE) denotes computer-generated, three-dimensional representations of a virtual 3D scenes. VE generally refers to "interactive, virtual image displays enhanced by special processing and by non visual display modalities, such auditory and haptic, to convince users that they are immersed in a synthetic space" [Ell94]. The users perceive themselves to be within the VE where interaction takes place. For example, the CAVE [Cru$^+$92] represents a strongly immersive virtual environment. Related technology is frequently based on techniques from Virtual Reality (VR) and Augmented Reality (AR), whose 3D virtual scenarios allow for a simulation of a realistic experience for its users. Its contents are typically organized by a 3D scene graph.

3D Geovirtual Environment A *3D Geovirtual Environment* (3D GeoVE) refers to a virtual environment that models physical environments similar to our real world or imaginary spaces. Frequently, virtual 3D city models are the base data used to build 3D GeoVEs. 3D GeoVEs "make use of one or more 'aspects of virtuality' to reflect components of the real world in intuitive ways." [FM01] For example,

[2]`https://www.google.com/earth/`

large-scale CAVE [Cru+92] can be used to provide an immersive experience for urban design projects.

2.2 Foundations Related to Computer Graphics and Visualization

Rendering The term *rendering* refers to processes and techniques that synthesize digital images based on computer graphics models and procedures. Rendering represents the core task of today's computer graphics hardware and is largely implemented using graphics processing units (GPUs). In most interactive applications, the image-generation process needs to be "fast enough" to provide the illusion of a steady image flow, i.e., a frame rate of more than approx. 8-15 frames per second is required to achieve *real-time rendering* [Ake+18].

Image-Based Rendering *Image-based rendering* denotes those rendering approaches using "images rather than geometry as main primitives for rendering novel views" [SCK07] of a specific 3D scene. It allows for optimizing the depiction of complex 3D scene geometry and is well suited for server-side rendering. In general, generated images can be transferred efficiently to clients that reconstruct the original 3D scene based on these images [HHD11]. For example, a rendering server can generate a six-sided cube map of a 3D scene, then transfer it to a client viewer, which displays it [DHK12].

Visualization The notion *visualization* refers to a cognitive activity or process that allows humans to obtain insights and understanding about data and its correlations [Spe14]. In general, it can be seen as "the activity of forming a mental model of something" [Spe14]. Visualization aims at mapping data in a way so that the human eyes can see (*perception*) and the human mind can understand (*cognition*) the data and its underlying characteristics, i.e. its structure, relationships, outliers, and clusters [RF94]. Visualization creates "graphical models and visual representations from data that support direct user interaction for exploring and acquiring insight into useful information embedded in the underlying data." [OL03] The impact and importance of visualization arises from the strong human perceptual and cognitive abilities such as automatic pattern finding: "Visual displays provide the highest bandwidth channel from the computer to the human. Indeed, we acquire more information through vision than through all of the other senses combined" [War12]. Historically, visualization started as *Scientific Visualization* and was motivated by the first applications that visualized scientific data [MDB87; Bro+92; Spe14].

Information Visualization The notion *information visualization* refers to the visualization of abstract data, i.e., it includes those concepts, methods, and techniques that allow us to visualize non-geometric data, either numerical or non-numerical data, in particular high-dimensional information spaces [War12]. For information visualization, it is essential to define a process that maps abstract data to renderable geometry and graphics. It "creates graphical models and visual representations from data that support direct user interaction for exploring and acquiring insight into useful information embedded in the underlying data" [OL03]. The abstract and possibly high-dimensional data needs to be "spatialized", i.e., it is transformed into a lower-dimensional space and into geometric representations using computational algorithms and spatial metaphors. This way, spatialization helps us discover patterns, structures, and relationships within high-dimensional data using our human perceptual and cognitive abilities [Gee+05]. Common techniques for information visualization include scatter plots [Cle93], parallel plots [Ins97], or dust-&-magnet [Yi+05] representations.

Geovisualization The visualization of geospatial or georeferenced data is frequently called *geographic visualization* [DMK05] or, slightly more specific, *geovisualization* [MAC+99]. It was initially defined by MacEachren et al. as "the use of concrete visual representations – whether on paper or by computer displays or other media – to make spatial contexts and problems visible, so as to engage the most powerful of human information-processing abilities, those associated with vision" [Mac+92]. So, the notion generally refers to methods, concepts, and techniques that are based on or make use of the spatial nature of the data and that apply principles found in cartography and map making. The visualization of geospatial data has its roots more than thousand years ago when maps became fundamental tools to understand and explore our physical environment and the world in general [Nol06].

The Visualization Pipeline The *visualization pipeline* describes the technical visualization process, i.e., the process of creating visual representations of data as a data flow model. The model of the pipeline consists of by three major conceptual stages [HM90]:

Preprocessing Stage: Gathers, fuses, and integrates raw data from different data sources. It can filter, complement, convert or prepare data to simplify it for processing in later stages.

Mapping Stage: Transforms preprocessed data into visual representations, i.e., renderable graphic representations by means of geometric primitives and graphics attributes, e.g., visual features, shape, size, color, or texture.

Rendering Stage: Transforms mapped data into displayable images and synthesized visual representations.

Dos Santos and Brodile [SB04] extended this model for improved processing of multivariate data. They split the preprocessing stage into a separate data analysis and a filtering step. The data analysis generates an improved dataset by applying different methods, e.g., error correction, data smoothing, transformation, or clustering algorithms. The filtering stage extracts the portion of the overall data that is to be visualized. Fig. 2.1 shows this extended pipeline model. Here, users are able to configure data processing more precisely by adjusting the data analysis and filtering process separately.

Another key component in the visualization pipeline is the human user. The human perceptual and cognitive system allows us to visually and cognitively process the visualized information. By perception and the ensuing interpretation of the generated images by users, a mental model is built that allows users to recognize

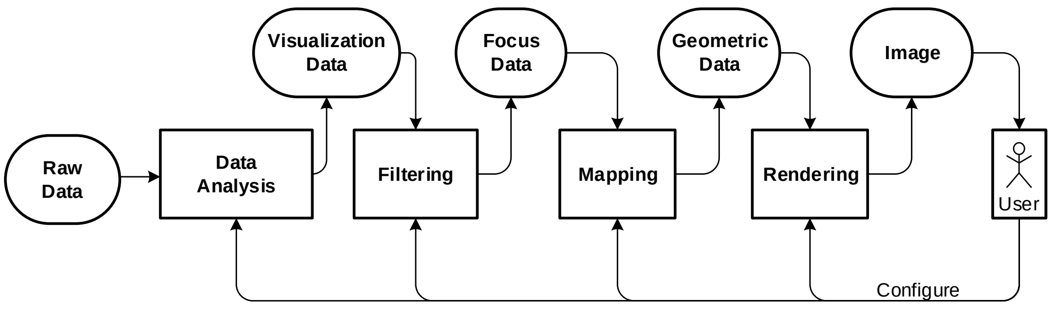


Figure 2.1: *Model of the visualization pipeline describing the stage-wise generation of visual representations. The depiction follows the four parted model for a visualization pipeline introduced by Dos Santos and Brodlie [SB04]. The user can modify each stage of the visualization pipeline to adjust the visualization forming a visualization cycle.*

connections and analyze the dataset. Colin Ware gives a detailed description of the steps and processes of the visualization pipeline [War12]. In connection with the technical stages, the so-called *visualization cycle* is formed, i.e., the user configures data selection and filtering, defines data mappings, and interactively explores the resulting visualization; based on their feedback, the stages can then be reconfigured [Ups^{+}89]. Step by step, users form and update their internal mental model.

Distributed Visualization Pipeline Technically, the stages of the visualization pipeline could be implemented in a distributed manner. Depending on the stages that are implemented on client side and on server side, three different types of systems are distinguished based on the type of data that is transmitted via a network: 1) Thick clients fetching filtered data from thin servers, 2) medium clients fetching mapped data (i.e., geometric primitives or textures) from medium servers, and 3) thin (lightweight) clients fetching rendered images from thick servers (see Fig. 2.2). End-user hardware and software requirements (e.g., processing power, disk and main memory capacity and GPU capabilities) differ based on the distribution type and the corresponding types of client applications.

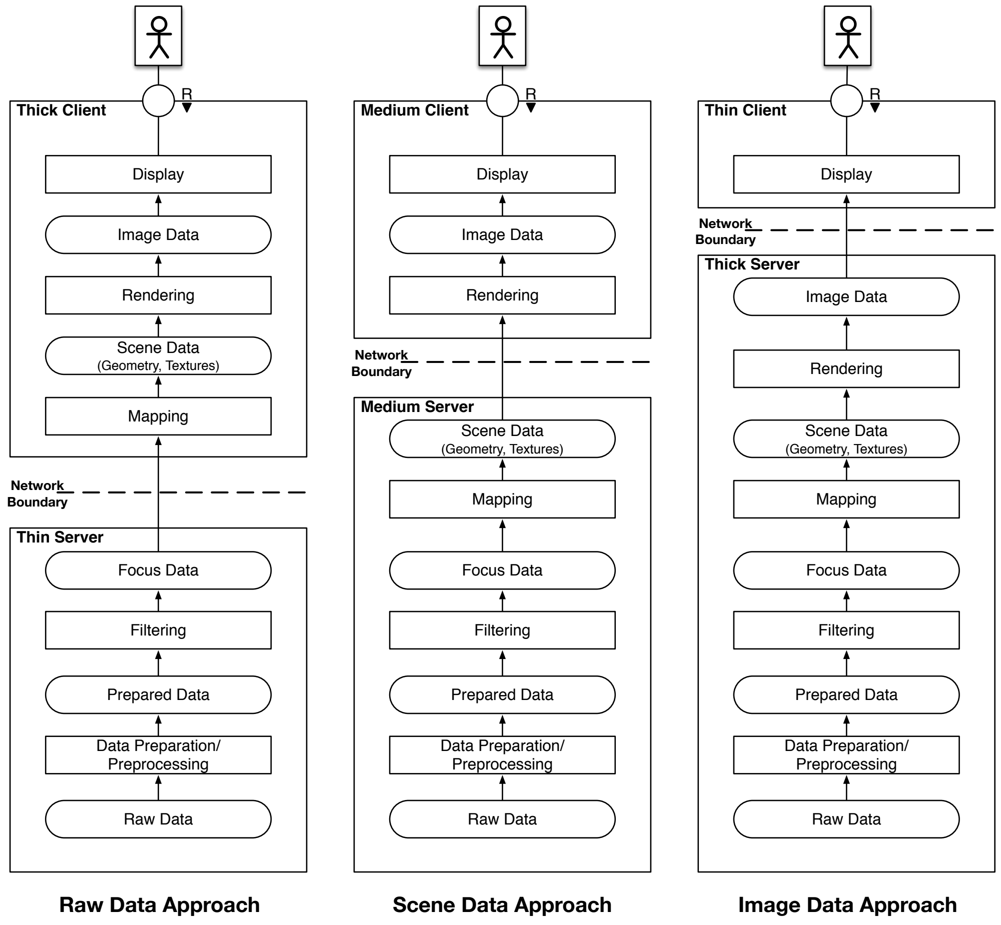


Figure 2.2: *Functional segregation of the visualization pipeline into server and client components for three principal client types [DC98].*

2.3 Foundations Related to Service-Orientation

Service-Oriented Computing The term *service-oriented computing* (SOC) refers to a computing paradigm that uses services as its core building blocks to design, construct and deploy complex software systems in a distributed way and within heterogeneous environments. SOC envisions "cooperating services that are being loosely coupled to flexibly create dynamic business processes and agile applications that may span organizations and computing platforms, and can adapt quickly and autonomously to changing mission requirements." [PG03]

Service-Oriented Architecture The principles and concepts for SOC are investigated in the field of SOAs. They provide "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations." [Mac$^+$06]

Service In the context of software engineering, in particular in the fields of SOC and SOA, the term *service* refers to a concept for software components with specific properties: They "provide autonomous, platform-independent, computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries." [KA07] In the context of geographic information systems, a wide range of services has been defined and standardized by the OGC.

Web Feature Service The *Web Feature Service* (*Web Feature Service (WFS)*) standardized by the OGC, allows access to feature data including their 2D or 3D geometries and their properties, encoded in standardized formats. The WFS standard uses the OGC Filter Encoding Standard [Vre14b] for defining complex spatial and nonspatial queries to a WFS instance. Besides defining a query interface, the WFS standard also defines an optional `Transaction` operation that allows data insertion, update, and deletion.

3D Portrayal Service The *3D Portrayal Service* (3DPS), as defined by the OGC, "is a geospatial 3D content delivery implementation specification" [Hag$^+$17] intended to build interoperable service-oriented 3D geovisualization systems and applications. This standard specifies how 3D geodata portrayals are described, selected, and delivered. The 3DPS provides two delivery schemes: The *Scene Extension* for requesting 3D scene data that is rendered by clients and the *View Extension* for requesting rendered images of 3D scenes. As this thesis primarily handles image-based 3D portrayal, the term 3DPS refers to the View Extension of the 3DPS unless otherwise noted.

Web-Based Provisioning *Web-based provisioning* of virtual 3D city models, in the context of this thesis, refers to the methods, concepts, and techniques that allow for transmitting, rendering, and use of virtual 3D city models in typical web-based computing environments such as web browsers or 3D apps on smartphones. To implement those approaches, SOA and SOC are key concepts used to design corresponding geovisualization services. For example, the OGC 3DPS can be used to build such systems.

2.4 Virtual 3D City Models

Virtual 3D city models are a major category of data models that form the basis for 3D GeoVEs [DBB06]. They represent entities of our built and natural environments by means of data entities that reflect their principal characteristics in terms of geometry, appearance, structure, and semantics. Their use is not limited to visualization applications, i.e., they are a generic computational models for analysis, simulation, visualization, and documentation. Virtual 3D city models are analogous to physical 3D city models that have been used throughout centuries for purposes of urban planning and design.

Various types of 2D geodata, 3D geodata, and georeferenced data are frequently used to assemble a virtual 3D city model. In most cases, virtual 3D city models cannot be created from geodata from a single source alone (e.g., 3D point clouds acquired by laser scanning) since they require several layers of information (e.g., terrain models, building models, infrastructure objects). For that reason, systems and applications dealing with virtual 3D city models must be able to handle different types, qualities, precision, and acquisition times of geodata and georeferenced data. Consequently, virtual 3D city models are usually complex, massive, distributed, and heterogeneous.

2.4.1 CityGML

The *City Geography Markup Language* (CityGML) is an international standard published by the OGC for modelling and encoding virtual 3D city models. CityGML is widely accepted and supported by a variety of tools for 3D modelling, authoring, and checking model consistency [Wag+13; Ala+14; Wen+17]. In its 12 thematic modules, it defines principal components and entities that constitute a virtual 3D city model [Grö+12]. Tab. 2.1 lists these modules and the corresponding entities or concepts that are being modelled therein.

Module	**Description**
Appearance	Textures and materials for other types
Bridge	Bridge-related structures, possibly split into parts
Building	The exterior and possibly the interior of buildings with individual surfaces that represent doors, windows, etc.
CityFurniture	Auxiliary objects in an urban environment (e.g., benches, traffic lights, signs, etc.)
CityObjectGroup	Grouping concepts for groups of objects of other types
LandUse	Areas that reflect different land uses, such as urban, agricultural, etc.
Relief	The shape of the terrain
Transportation	Transportation infrastructure such as roads, railways and squares
Tunnel	Tunnels, possibly split into parts
Vegetation	Areas with vegetation or individual vegetation objects such as trees
WaterBody	Lakes, rivers, canals, etc.
Generics	Other types that are not explicitly covered

Table 2.1: *CityGML modules and the objects covered*

The corresponding model entities are described by CityGML in an object-oriented manner by classes, their attributes, as well as by the relationships between these classes. The model is built as an application schema of the OGC GML [Por07]. It extends the language in terms of additional classes, attributes, constraints and modules. A core feature of CityGML is a concept for *Level of Detail* (LOD) that enables modelling the objects of a virtual 3D city models in different geometric and semantic detail. There are five *levels of detail LOD* levels defined for objects, e.g., buildings:

LOD0 represents buildings by their 2D footprints.

LOD1 provides block models, i.e., 3D extruded footprints without roof structures.

LOD2 provides differentiated boundary surfaces and roof structures.

LOD3 provides architectural models of the outside hull in high geometric detail including openings for windows and doors.

LOD4 adds elements for modelling indoor structures such as rooms, interior doors, stairs, or furniture.

2.4.2 Generation and Management of Virtual 3D City Models

A growing number of administrations and businesses have set up processes for generating and updating virtual 3D city models. However, when quality and precision requirements apply their models, those processes are far from being completely automated. For that reason specifically, generating a virtual 3D city model is still a cost- and time-intensive task.

Today, virtual 3D city models are generally created today from a variety of heterogeneous sources of 2D and 3D data [Ros$^+$09], such as *Geographic Information System (GIS)* datasets, i.e., *Digital Terrain Model* (DTM), road network geometry, land use data, cadastral data, aerial photographs. Additional data sources can also include high-detail, textured 3D building models or architectural models [DBB06]. Latest approaches are based on remote sensing data (e.g., dense 3D point clouds), from which 3D object representations can be derived (e.g., 3D building models) [Bre00; Döl$^+$06; ZN08; RKD13]. However, due to the complexity of man-made and natural environments, it remains difficult, or is nearly impossible to create a complete match for all for objects of interest (e.g., due to temporary construction work, parked cars, or partially built infrastructures). Nevertheless, some existing approaches allow a derivation of operational LOD2 models in a highly automated manner by using a combination of data sources, such as a 2D building cadastre, LiDAR scans, and aerial images [CXL17; RB16]. A comprehensive description of the various approaches to generate virtual 3D city models is outside the scope of this thesis; Musialski *et al.* [Mus$^+$13] provide a good introduction on this topic.

2.4.3 Application Domains

Applications of virtual 3D city models are manifold, since they are used as base data for applications both in administrative areas (e.g., cadastre, urban planning, or simulations) as well as in various industries (e.g., city marketing, real-estate industry, or renewable energies). Different fields of applications take advantage of their steadily increasing availability and quality in terms of resolution, coverage, and precision [MAK10; Bil+15].

Example application fields include:

- Disaster and Emergency Management [KGP05; Mor+11; Dem+16]
- Urban Planning [XQ06; Fer+15] and Public Participation [WHG10; DK14; Bra+16]
- Energy Planning and Simulation [Düb+11; Str+11; KK12; Nou+14]
- Training [RBK07]
- Simulation of Noise Propagation [KL16; Zha+16]
- Visibility Analysis [ED09; AMH13; NF14]
- Archaeology and Heritage [Tra+10; Aga+10; RC15; PIB17]
- Semantic Supported Statistical Analysis [BLS17]

CityGML provides the *Application Domain Extension (ADE)* as "a built-in mechanism of CityGML to augment its data model with additional concepts required by particular use cases" [BKN18]. An overview of numerous ADEs that have been created is provided by Biljecki, Kumar, and Nagel [BKN18].

2.5 Modeling Notation

Two different modeling languages are used in this thesis: a) *The Fundamental Modeling Concepts* (FMC)[3] [KGT06] and b) the *Unified Modeling Language* (UML). FMC is primarily used to describe system structures, architectures, and behavior on a more conceptual level while UML is used more for the implementation specific modeling.

[3]A compact introduction into the FMC modeling notation can be found at `http://www.fmc-modeling.org/download/notation_reference/FMC-Notation_Reference.pdf` (accessed September 8th 2018)

Chapter 3

Related Work

This chapter gives an overview of relevant related work for this thesis. It briefly introduces the concepts and presents how they relate to the contributions of this thesis.

3.1 Web-Based Provisioning of Virtual 3D City Models

There are three major research areas for web-based provisioning of virtual 3D city models that are relevant for this thesis: Service-oriented approaches in GIS, web-based approaches in computer graphics, and the provisioning of virtual 3D city models as oblique image tiles. In the following, related work concerning these areas is briefly described.

3.1.1 Service-Oriented Approaches in GIS

Service orientation has been playing a key role for about two decades in the scope of geoinformation systems and for interoperability among these systems. The term *service* is defined by the OGC as "a distinct part of the functionality that is provided by an entity through interfaces" [Per11]. These interfaces are "a named set of operations that characterize the behavior of an entity" [Per11]. A *service operation* is defined as "a specification of a transformation or query that an object may be called to execute. Each operation has a name and a list of parameters." [Per11] In particular, the OGC has defined a number of influential standards for geodata encoding and service interfaces for data exchange such as the WMS [La 06] for retrieval of map images, the WFS [Vre14a] for retrieval and modification of feature data, the *Web Coverage Service (WCS)* [Bau18] for retrieval of geospatial coverages, the *Web Processing Service (WPS)* [MP15] for implementation of processing functionality. All of these standards provide a common `GetCapabilities` operation that provides service meta information, e.g., available service operations and information about the data provided by the service. Encoding formats standardized by OGC include, e.g., the GML [Por07] describing features, the Filter Encoding [Vre14b] providing means for query encoding, *Styled Layer Descriptor (SLD)* [Lup07] and *Symbology Encoding (SE)* [Mül06] describing map styling, KML [Bur15] for describing geographic annotation and visualization scenarios. Furthermore, more recent works have been investigating how SOC and SOA principles can be applied to building and operating interactive, web-based geovisualization systems [HD10; SHC12; KG15; Hag16].

The software reference architecture for geovisualization relying on image-based representations was introduced by Hildebrandt [Hil14]. It uses per-pixel data for information about depth, object ID, or object class. It aims at building lightweight, interactive geovisualization clients, that use techniques for image-based rendering introduced earlier [HHD11; DHK12]. A recently presented example of a geovisualization system based on the OGC 3DPS has uses the standard as an interface for different rendering back-ends based on ray tracing [Gut+16].

"Typically, within GIS there is a clean cut between raw geodata and visualization properties." [NZ08] In standard OGC WMS, styling is limited to a set of predefined styles that are provided in a proprietary way by the WMS implementation. Accordingly, the client-side styling of features cannot be influenced, e.g., by thematic mapping. The OpenGIS *Styled Layer Descriptor Profile* [Lup07], as an extension to the OGC WMS, "explains how WMS can be extended to allow user-defined symbolization of feature and coverage data. This profile defines how the SE standard can be used with WMS. " [Per11] It can be used to describe rule-based styles that are applied to features, which are selected by filters specified in OGC's Filter Encoding [Vre14b]. The *OpenGIS Symbology Encoding Implementation Standard* [Mül06] (*SE*), "specifies the format of a map-styling language for producing georeferenced maps with user-defined styling. SE is an XML language for styling information used to portray Feature and Coverage data. SE may be used together with SLD. As SE is a grammar for styling map data independent of any service interface specification it can be used flexibly by a number of services that style georeferenced information or store styling information that can be used by other services." [Per11]. The OGC SE defines *Symbolizer* elements that describe "how a feature is to appear on a map. The Symbolizer describes not just the shape that should appear but also such graphical properties as color and opacity." [Mül06] Five types of symbolizers are defined: `LineSymbolizer, PolygonSymbolizer, PointSymbolizer, TextSymbolizer, RasterSymbolizer`. Current OGC standards are limited to 2D styling and do not apply styling in 3D. However, some approaches extend the mechanisms of SLD and SE for styling in 3D portrayal by adding additional symbolizers, as well as symbolizer properties and an extension for 3D portrayal services [NZ08; HFR07; Bas+08; RKT09; NZ09] that is analogous to SLD for WMS.

This thesis is based on the ideas of service orientation for geovisualization. The concepts and techniques are aligned with the aforementioned OGC standards that support the reusing of service components. The contributions of this thesis are targeted at higher-level services for applications using virtual 3D city models, providing a key functionality for their visualization. For thematic styling, a styling language for 3D portrayal is introduced that is aligned with approaches mentioned above.

3.1.2 Web-Based Approaches in 3D Computer Graphics

Web-based approaches for 3D computer graphics are becoming key technologies for the development of a various application areas, e.g., teaching and education [Blu+11], e-commerce [GR15], and medicine [Con+11]. Applications can be separated into client-side and server-side 3D rendering approaches. Generally, approaches that render 3D computer graphics in a web browser are described as either declarative approaches or imperative approaches [Jan+13]. In *declarative approaches*, 3D scenes are described using specialized modeling languages and encoding standards such as the *Extensible 3D (X3D)* [BD07], a successor of the *Virtual Reality Modeling Language (VRML)* that was introduced already in the early 1990s [Pes95]. Other approaches integrate 3D scene descriptions into the HTML5 Document Object Model (*DOM*), such as X3DOM [Beh+09] and XML3D [Son+10]. In contrast, *imperative approaches* for web-based 3D graphics specify the rendering process itself and often provide more flexibility for implementing 3D graphics applications. The WebGL API, standardized by the Khronos Group[1], is today's key implementation tool for web-based 3D rendering, since it is supported by all major mobile and desktop browsers at the time of writing. The Khronos Group also specified the GL Transmission Format (*glTF*), a 3D scene format that "has been designed with modern graphics card and web technologies, especially WebGL, in mind. glTF 1.0 has been launched by Khronos in October 2015 and received support from the industry from the beginning. glTF supports only basic elements of 3D scenes (node hierarchy, materials, textures, animation, cameras, lights), which is sufficient for most applications." [SBN16]. A general overview about web-based visualization approaches is provided by Mwalongo *et al.* [Mwa+16].

In contrast to client-side techniques for 3D rendering, "remote rendering techniques permit streaming of high-quality 3D graphics onto a wide range of devices, and recent years have also seen much research on methods of content delivery for web-based 3D applications" [Eva+14]. This concept shifts complex and computationally expensive 3D rendering to server side and delivers rendered images to client applications. There are several approaches to enabling interactive 3D graphics through remote rendering based on a) video streaming [DHS05; Tiz+11], b) streaming of OpenGL commands [Hum+01; Gla+13], c) transmission and display of rendered framebuffers [Lam+03; Gri+05], or d) image-based rendering [DHK12; WV14]. A comprehensive survey of remote rendering systems is provided by Shi and Hsu [SH15].

Unfortunately, standard approaches for both remote and client-side, browser-based 3D-rendering cannot be directly transferred to the specific demands of virtual 3D city models as they exhibit specific data characteristics such as predominant texture data for facades or large number of relatively small polyhedral 3D building models. When virtual 3D city models become large, general approaches typically fail in real-time application scenarios.

In this thesis, a 3D rendering service implementation is proposed that can handle these challenges of massive, textured, virtual 3D city models. This service provides a lean, technical foundation for other services and modes of provisioning of virtual 3D city models as introduced in Chapter 4. It applies specialized 3D rendering techniques and provides simple-to-use components. A user can easily select objects in virtual 3D city models or configure camera position and orientation himself. This approach does not require the geometry and texture data of a virtual 3D city model to be transferred to the clients.

3.1.3 Provisioning of Virtual 3D City Models Based on Oblique Image Tiles

One major challenge for 2D portrayal services, such as the OGC WMS, is that they are not "scalable to large numbers of users due to a lack of ability to cache requests and this is implicit in the way the standards are written." [Bat+10] In order to avoid this issue, tile-based map services are been used for 2D portrayal. They trade "the flexibility of custom map rendering for the scalability possible by serving of static data (base maps) where the bounding box and scales have been constrained to discrete tiles." [MPN10] Tile-based provisioning of maps is one of the key techniques for easy to use, immediately

[1] `https://www.khronos.org/webgl`

accessible applications with minimal hardware and software requirements. Prominent examples for such services are the OGC *Web Map Tile Service (WMS-T)* [MPN10] and the *OpenGeo Tiled Map Service (TMS)*[2]. By reducing rendered map images to tiles with fixed scales and bounds and named layers, these tiles are commonly addressed by a tuple containing their integer coordinates in x and y direction, their zoom level z and the layer name. This method allows for efficient multi-level caching of map tiles on server side and client side. It significantly reduces the amount of data that has to be transferred and processed, especially for large numbers of clients using such a tiled representation of geodata.

In analogy to the tile-based map services for applications based on 2D maps, Lian et al. stated that tiled "2.5D maps have the potential to augment the value and extend the use of existing virtual 3D city models." [Lia+16]. In 2016, they provided a step by step workflow for generating such 2.5D maps from virtual 3D city model datasets using orthographic camera projections. Christen and Nebiker [CN15] presented an image-based provisioning approach for complex 3D models based on tiled G-Buffer representations. They used color, object id, and normal maps to simulate different lighting situations using client-side deferred lighting. Image-tiles are organized in a quadtree providing tile zoom levels, where only the most detailed zoom level is rendered by the 3D renderer and the remaining levels are derived using image processing.

The approach to generate 2.5D tiled maps from a large-scale virtual 3D city model also uses orthographic projections and rendering. In contrast to the approach presented by Christen and Nebiker, the tile generation process allows for a detailed configuration, which allows for different, scale-dependent contents to be included for each zoom level of generated tiles, as it is common in 2D cartography. In addition, the layered map approach approach supports run-time reconfiguration of maps by using overlay layers that allow for visualization of different planning variants or provide switchable annotations. These additional map layers can be combined with the help of standard 2D blending techniques on a per-pixel basis. Insofar the the layered map approach is able to provide a tile dataset that is designed specifically for the use case to be supported by a layered map application.

3.2 Export of Virtual 3D City Models as Videos

Many applications aim to export contents of virtual 3D city models as common visual media formats, e.g., digital videos. Videos provide an easy to use medium that is well suitable to communicate virtual 3D city model contents. Those contents can be directly fed into usual web content management systems. Consequently they are found on websites, video platforms (e.g., YouTube), social media platforms, and even being incorporated into presentations. Generation of these videos requires software features that either partially or completely automatically produce videos based on corresponding camera animations. Technically, videos are generated by rendering the clip frame by frame and then encoding these frames using video encoding standards such as H.264 MPEG4 [ISO04].

The OGC standardized KML encoding provides the means for defining camera animations, from which videos can be generated. Besides features for description of

[2]`https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification`

visualization scenarios (i.e., scene content and camera orientation), KML provides tours (see `kml:Tour` element at 9.23 in [Bur15]) to model camera animations and scene content over time. The general concept for tour modeling is illustrated in Fig. 3.1. It supports simple camera animations, periods without camera movements, general animations of visualization parameters (e.g., showing/hiding items), and timed sound effects. However, KML does not provide the means to express complex video projects and thus does not support including text overlays or blending between different parts of a video.

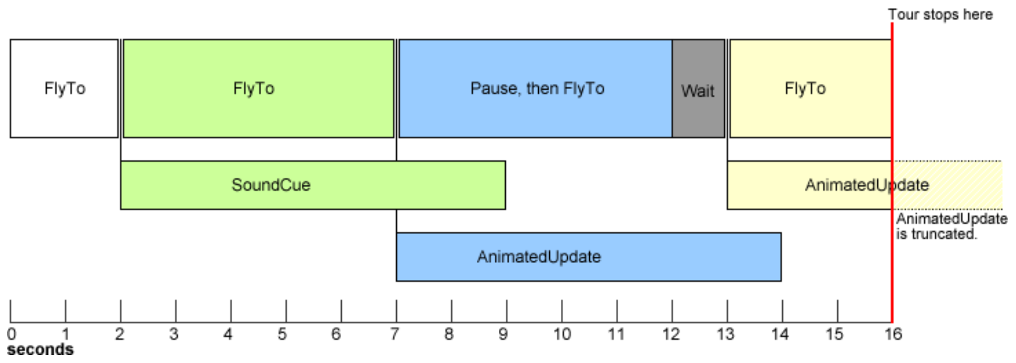


Figure 3.1: *Overview of KML tour modeling. Source: Google KML documentation*[3]

Video authoring with 3D geovisualization as in KML Tours in the Google Earth desktop client[4], mainly defines the positions for camera keyframes and the transitions between them, based on the KML model for camera paths and transitions. These camera paths are then rendered at the selected frame rate and resolution using the local rendering engine for video generation.

There are several other visualization solutions (e.g., Autodesk LandXplorer, ESRI ArcGIS, Viewers by Agency 9) for generating videos from 3D geodata. They all require local 3D capabilities for the videos to be rendered locally. While there are some approaches for 3D visualization that are based on remote rendering, there are currently service-based approaches for web-based authoring and video generation videos from massive 3D geodata. In particular, none of them generate videos at a high abstraction level with thin clients that do not require client-side 3D rendering. Such an innovative approach is presented in this thesis.

Grimstead et al. presented the Resource Aware Virtual Environment [GAW09] targeting heterogeneously sized hardware, wherein high-end devices execute local rendering of the scene, while rendering services serve low-end devices, such as PDAs, an enable mobile usage of the visualization system.

Camera control and video export represent two key challenges for a service-oriented 3D visualization systems: Both are high-level services that are meant to operate on lower-level services to generate images and access data thereby ensuring flexibility, interoperability, reusability, and a consistent body of geodata for all applications based upon these service instances. Most importantly, they are to be decoupled from rendering services, and supposed to handle the model data that is required to compute and design a camera animation across the services in an efficient way. Insofar, they are good candidates to enhance a SOA for image-based web provisioning of virtual 3D city models.

[3] `https://developers.google.com/kml/documentation/touring`
[4] https://earth.google.com

The approach for video generation presented in this thesis, includes a unique solution for the description of video scenes from virtual 3D city models and their standards-based generation on server side. It provides a more holistic approach for web-based video generation, since it integrates a description of the 3D scene to be rendered, its styling, camera paths, and its transition over time. It also includes the features for video authoring that allows allow for scene transitions, text and image overlays, object-specific highlights, or complex image-based rendering styles, often are neglected by many other applications for visualization of virtual 3D city models. Consequently, the video service described in this thesis provides a more comprehensive solution for the video-based communication of virtual 3D city model contents than existing approaches and solutions.

3.3 Camera Control within Virtual 3D City Models

"Camera control, which encompasses viewpoint computation, motion planning and editing, is a component of a large range of applications, including data visualization, virtual walk-throughs, virtual storytelling and 3D games" [CON08]. This section introduces relevant topics within the area of interaction with virtual 3D environments. An introduction and general overview of interaction techniques is provided Jankowski and Hachets [JH15] survey paper.

3.3.1 Challenges of Virtual Camera Controls

"A 3D world is only as useful as the user's ability to get around and interact with the information within it" [TRC01]. Interaction is considered to be a non-trivial task for all kinds of virtual environments [Fit$^+$08; Rus$^+$00] – many negative side effects of camera control, such as the lost-in-space syndrome or motion sickness, are well-known. The manipulation and steering of a virtual camera in 3D represents a complex task especially for non-expert users – "indeed users find it problematic to deal simultaneously with all seven degrees of freedom" [CON08]. Fuhrmann and MacEachren state that "core problems for users of these desktop GeoVEs are to navigate through, and remain oriented in, the display space and to relate that display space to the geographic space it depicts." [FM01].

Chen and Bowman advocate that any design for 3D interaction techniques should be application and domain specific [CB09]. They propose to decompose the interaction tasks into sub tasks that consist of universal interaction tasks (e.g. navigation, selection or manipulation). Camera control in all kinds of virtual environments should assist users in effectively exploring the 3D space and in avoiding confusing or disorienting viewing situations, preventing "getting-lost" moments, and it should provide time-coherent, continuous camera movements [BBD05]. A virtual camera's behavior may also depend on the semantics of the underlying 3D model: "A high degree of usability is achieved because users can trigger complex navigation commands in a task and goal oriented way taking advantage of the navigation properties and affordances inherent to elements of geovirtual environments." [DHS05] For example, strictly task-oriented camera techniques could generate camera animations with respect to high-level navigation intentions such as "from here, go to the closest landmark". Thus, the camera control can be partially automated to achieve a higher degree of effectiveness and reliability for user interactions within a 3D environment.

3.3.2 Automated Camera Control

"Automatic camera control refers to the automated computation of static and dynamic camera parameters according to goal specifications typically without any direct user input." [HT14] Automated camera control techniques try to assist users with camera navigation tasks by providing higher-level, task-driven camera manipulation that adapts to a current situation in a geovirtual environment in terms of scale, geometry, object semantics, or feature attributes, among other. In such advanced approaches, the camera control process can generally a) interpret navigation intentions, b) derive path information, and c) adjust the visualization. This way, users can express their *navigation intentions* by simple inputs to a client application, such as pressing user interface controls, selecting objects in the scene, or sketching paths or gestures [HHD10]. The user input is then interpreted based on heuristics and corresponding rules, and potentially complex camera animations can be automatically generated. With the help of spatial and logical constraints, navigation in 3D geospatial models can be simplified. Hildebrandt and Timm present a three-fold concept based on "users point to navigate, users are lead by suggestions, and the exploitation of semantic, multiscale, hierarchical structurings of city models" [HT14].

3.3.3 Camera Control for Thick Clients and Thin Clients

For distributed applications using thick clients and medium clients, the rendering stage of the visualization pipeline is implemented on client side, where the required data, such as model geometry or points of interest, is directly available [AK03]. For those clients, various camera-control techniques can be implemented in a straightforward way, while thin clients need supplementary services to support their navigation needs, since only very limited model information is available on client side.

There is currently no support for camera interaction in official standards for 3D portrayal services, i.e., in the OGC 3DPS standard. More importantly, in its current version 1.0, it does not provide standardized operations for camera interaction at all. Its predecessor, the Web View Service [Hag10], discussion paper already contained proposed service operations to support, e.g., ray casting or camera position retrieval.

The services presented in this thesis focus on supporting thin client solutions, which do not have local access to a city model's geometry or feature data.

Chapter 4

A Service-Based Concept for Image-Based Provisioning of Virtual 3D City Models

This chapter presents a concept for *image-based provisioning* of virtual 3D city models, supporting multiple end-user platforms, devices, and media.

4.1 Image-Based Provisioning

While most existing approaches for 3D visualization of virtual 3D city models are built upon *geometry-based provisioning*, i.e., clients generate images based on streamed 3D geometry and 2D texture data, our approach for image-based provisioning differs with respect to the following aspects:

Decoupled Complexity Image-based provisioning keeps the original virtual 3D city model on the server side and generates image-based data, which is streamed to clients. This way, model complexity is decoupled from visualization complexity. The server handles the possibly large, massive virtual 3D model, while clients receive and process image-based data of fixed size (e.g., rendered views). Consequently, client applications can be implemented in a lean and efficient way as they do not need to provide a fully featured 3D city model rendering engine and less data has to be transmitted as no geometry and texture data from the original virtual 3D city model has to be streamed.

Model Security The original virtual 3D city model does never leave its safe place on the server side. Only image-based, derived representations are sent out to the clients. In particular, if models contain sensitive data, their secure treatment represents a crucial requirement for applications to get acceptance and to fulfill data privacy policies.

High-Quality Rendering As the image-based approach relies on server-side rendering, advanced 3D rendering techniques can be applied, which generally require extensive rendering resources and features to reach a high degree of rendering quality (e.g., ambient occlusion, shadowing, antialiasing, or illustrative rendering styles). These rendering techniques, if executed on client side, would most likely exceed the GPU resources of mobile clients and would consume significant energy.

Cross-Platform Compatibility It would be difficult to provide cross-platform implementations for advanced rendering techniques as the GPU feature support differs on various devices and platforms, especially on mobile or browser-based clients. These techniques, however, depend on many sophisticated and advanced graphics functionality. A large number of branches in the source codes and, consequently, high costs for software development and maintenance would be the result. The image-based provisioning, in contrast, relies on a server-side rendering implementation, which gets executed in a known server environment with known computer graphics hardware and GPU capabilities.

Software Maintenance The image-based provisioning for virtual 3D city models leads to a high degree of maintainability as it avoids client-specific implementations with respect to the core 3D graphics technology. In the long run, the diversity of mobile clients and web browsers is even more likely to increase and, hence, cross-platform software developments of geometry-based clients becomes increasingly complex and can lead to unforeseen costs.

Robustness The robustness of 3D visualization applications benefits from a known server environment with possibly specific high-performance hardware. The core 3D graphics functionality can be tested in that environment – a similar test on a variety of mobile clients and web browsers would be almost unmanageable due to the enormous variety of hardware and software configurations. In addition, the server-side rendering as the most time and resource consuming process steps can be deployed in distributed environment improving the overall robustness and reliability of the service operation.

Reusability A collection of image-based services for visualizing virtual 3D city models allows us to assemble and configure different applications, systems and high-level services by reusing these components in the sense of service-based building blocks. For example, a set of image-based styling services can be used to configure application-specific visualization applications [Hil16].

Scalability Another key quality of the image-based provisioning of virtual 3D city models is its scalability. This is achieved, e.g., by pre-generating image sets for a priori known or frequently used configurations, applying server-side as well as client-side caching strategies, and generating secondary media such as digital videos, which then can be independently distributed and used by corresponding services (e.g., video platforms) and applications (e.g., media viewers).

This way, in particular, applications using virtual 3D city models can be built with low hardware and software requirements – an essential point for IT applications and IT solutions in industry.

These advantages and characteristics of image-based provisioning are key to a number of today's and future applications. For example, for end-user applications such as public participation tools in the context of city planning and development, it is necessary to allow many different stakeholders — with their diverse, possibly low-cost devices — to access the given virtual 3D city model and keep it in a safe environment (e.g., if copyrighted architecture models are included). Another example represent mashups that require interactive access to virtual 3D city models based on service composition and chaining. As image-based provisioning decouples model complexity from visualization complexity, it can be integrated more seamless and more efficient way compared to geometry-based approaches with significant client-side complexity. To foster the separation of concerns, to enable reuse of functional components, and to ensure cross-platform compatibility, an image-based provisioning concept and its service-based software architecture have been developed in this thesis.

4.1.1 Image-Based Provisioning Process

Conceptually, the image-based provisioning of virtual 3D city models generally involves five steps (see Fig. 4.1):

1. Virtual 3D city model data has to be selected, fetched, decoded, stored, and optimized for efficient 3D rendering.
2. The contents, their mapping to graphical primitives, and their appearance (styling) have to be specified and configured as portrayal configuration.
3. Images have to be synthesized (rendered) from the virtual 3D city model.
4. The images need to be transmitted to clients.
5. The images are received, handled, and displayed by clients; users may interact with these representations, and possibly new images need to be requested.

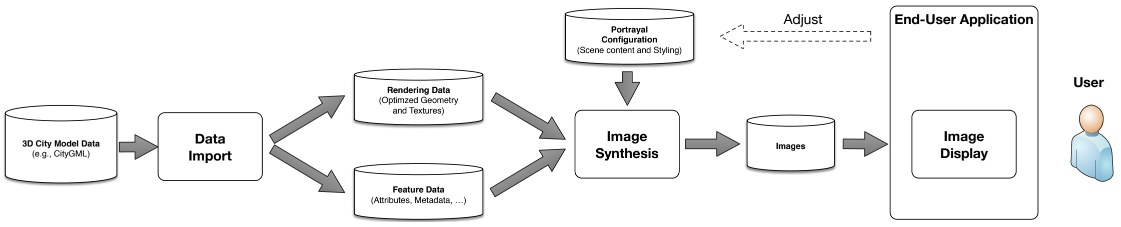


Figure 4.1: *An overview of the image-based provisioning process for virtual 3D city models. Image synthesis takes prepared renderable data and feature data and generates images. Scene content and styling are defined by a separate portrayal configuration. End-user applications fetch and display required images and display them to users. Depending on the type of end-user application, the portrayal configuration can be changed by end-user applications and image-generation adjusts accordingly.*

There are several levels of interactivity for end-user applications, e.g., regarding changes of 3D camera position, camera orientation, or scene contents. The portrayal configuration can therefore either be configured statically for applications or changed on the fly through client interaction. The following sections describe different client types as well as core functional components of the presented approach for image-based provisioning of virtual 3D city models.

4.1.2 Provisioning Schemes

Provisioning schemes for image-based provisioning of virtual 3D city models can be identified based on the type data that is used by clients to portray and interact with a virtual 3D city model. The schemes provide different characteristics based on the interaction features provided, updating processes, and the numbers of clients to be served in parallel using a given amount of rendering service resources:

Live Rendering Clients that retrieve on-demand rendered images from a 3D rendering service allow users to chose camera configurations, scene contents, and styling when retrieving images from a 3D rendering service. As live rendering of such images of virtual 3D city models requires dedicated rendering resources, the number of clients that can be served in parallel tightly depends on the number and the capacity of available rendering service instances. Further, rendering resources need to be provided constantly during the run time of an application.

Pregenerated Images For many applications, an on demand rendering of virtual 3D city models is not always necessary. In contrast to individually rendered images per client during run-time, pregenerated images of virtual 3D city models for a limited set of scene configurations and stylings, and classes of camera parameters, can be served by simple web servers and do not require 3D rendering resources to be available during run-time. Due to this preselection of scene contents, clients are restricted in their interaction capabilities. This scheme is well suited for scaling for a large number of parallel clients as no expensive 3D rendering resources are required during application run time.

Videos Videos represent one of the most important digital media formats; they can be used and processed by countless content management systems, authoring systems, and presentation applications. As an easily accessible and usable type of media, digital videos derived from virtual 3D city models provide effective means for communicating geospatial information. They are key to release the added values virtual 3D city model can unroll in business processes.

Clients managing and viewing derived digital videos are using videos to portray single objects or areas with a virtual 3D city model. Videos are rendered in advance of the application run time showing dedicated scene configuration an paths. They can provide, e.g., in detail tours of selected parts of a virtual 3D city model that are styled according to the application purpose, e.g., a presentation of a project to be built. This scheme also provides limited interactivity in clients. As videos can be hosted on simple web servers or external video portals, the schema scales well for a large number of corresponding clients.

Depending on the application use case, different client types can be applied to provide the required properties for the overall application, i.e., in terms of scalability, interactivity, and compatibility. For most cases within an application, such as portrayal of the spatial context in an urban environment, a client using pregenerated images can be sufficient, but for specialized, maybe less frequently used functions in an application a live rendering client may be the better choice. For building applications, the corresponding advantages of client types can be combined, as all of the mentioned client applications are light weight — they do not have specific requirements.

4.2 Service-Oriented Architecture for Image-Based Provisioning of Virtual 3D City Models

The management and visualization of virtual 3D city models requires a complex set of functionality to be implemented. A SOA is proposed in this thesis that enables reuse of functionality an shows how these services can be orchestrated and configured for different image-based applications for virtual 3D city models. Fig. 4.2 shows the SOA, its services and their relationships of our prototype system. There are five general service layers, which are described in detail in the following:

1. **Data Import**
2. **3D Rendering**
3. **Application Data Provisioning**
4. **End-User Applications**
5. **Configuration and Administration**.

The four services — 3D rendering service, layered map service, video service, camera service — form the core software components developed for image-based provisioning of virtual 3D city models in this thesis.

4.2.1 Data Import

The *data import* layer is responsible for transforming virtual 3D city model data into efficiently renderable internal data structures, called *rendering data.* Data from different formats and sources, e.g., semantics-based virtual 3D city models encoded in CityGML, detailed architectural models in CAD formats or 3D computer graphics formats (e.g., Collada or X3D), together with referenced 2D textures is converted into compact and spatially organized native formats as optimized geometry and optimized textures. In that respect, a data import service is connected to a specific rendering service implementation as the formats and data structure is specific to the rendering service implementation. A relation to the underlying geospatial features and objects is maintained using a feature database where feature data and metadata about features contained in source data are stored. This metadata includes, e.g., 2D geometry footprints, information about the origin of features, and their IDs in the original data sources.

As virtual 3D city models can change over time, the data import can process updated source data and update the rendering data and feature information. It is key to automate related workflows.

4.2.2 3D Rendering

The 3D rendering layer is implemented on the server side as a service based on the *View Extension* of the OGC 3DPS interface. This interface allows us to configure camera

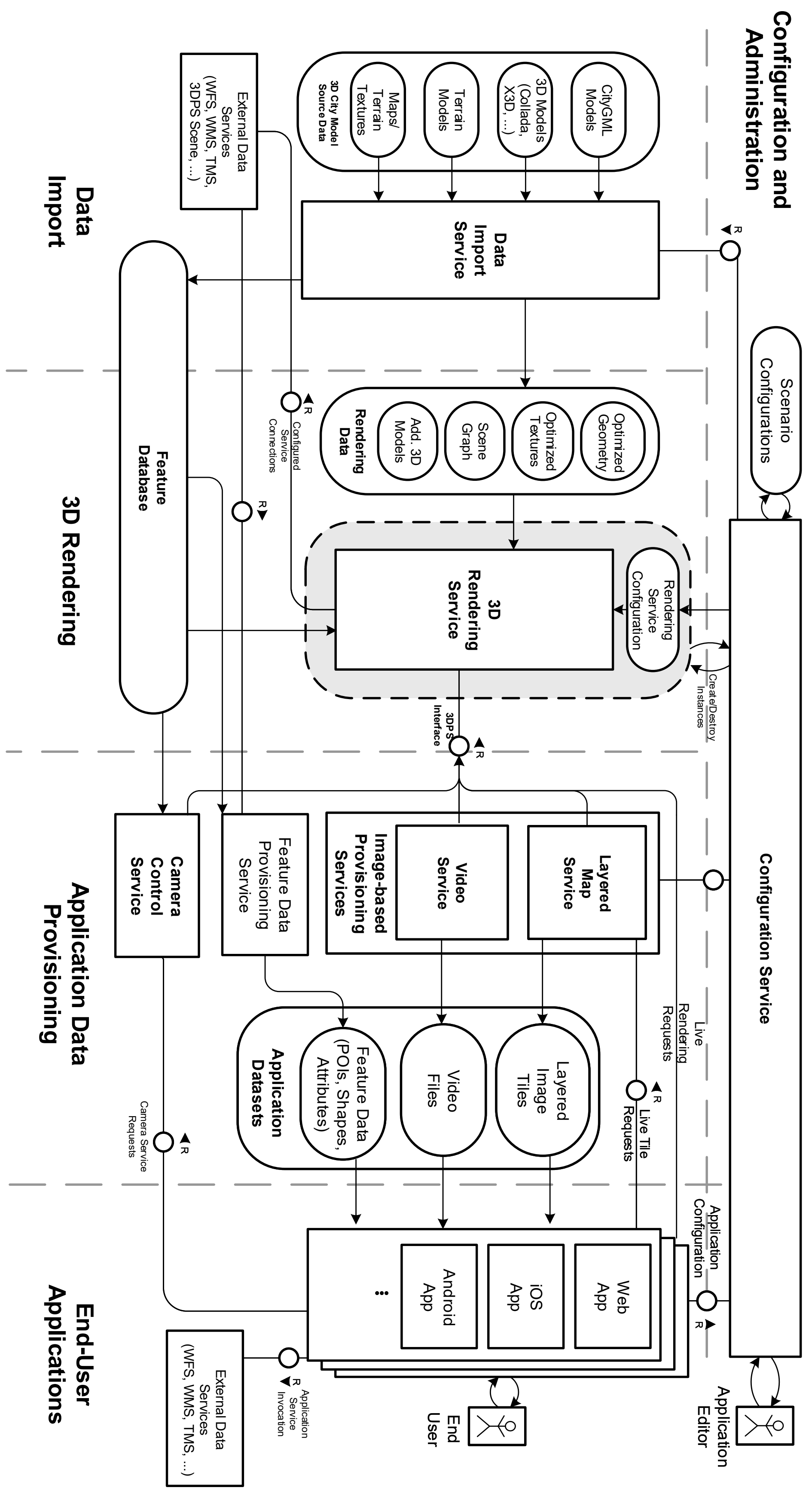


Figure 4.2: *Overview of the service-oriented architecture for image-based provisioning of virtual 3D city models.*

parameters, scene contents and styling using named layers with corresponding named styles and parameterizable image-based styling. Styling can also depend on feature data stored in the feature database that is available to the rendering process. The 3D rendering uses the imported and transformed rendering data to efficiently select, load, and render depictions of virtual 3D city models.

4.2.3 Application Data Provisioning

The *application data provisioning* layer provides two image-based provisioning services, i.e., the layered map service and the video service, which invoke a 3DPS to generate image-based artifacts such as image views, tiled image-sets for layered maps, or videos. It also contains a feature data provisioning service that converts application-specific feature data (e.g., *points of interest* (POIs), 2D feature geometry and feature attributes) into compact formats, called *application data*, that can easily accessed and processed by client applications. This data may be accumulated from the feature database created during data preprocessing or from external services. The layer also hosts the camera service as a utility service that provides camera path computation.

4.2.4 End-User Applications

End-user applications access a given virtual 3D city model data via application data provisioning, including layered image tiles, videos, and feature data. These applications are built to run as browser-based web apps, or native mobile apps for Android or iOS operating systems. Their implementation uses platform-specific software modules and libraries, which provide the virtual 3D city model viewing and interaction functionality. However, they do not have to implement a 3D rendering engine for virtual 3D city models (as this is done by the 3D rendering service), i.e., these apps tend to be lightweight with respect to the computer graphics techniques. This way, the software development efforts to build and maintain such apps are significantly lower compared to thick-client applications. To further reduce software development efforts, an app framework has been created that can be configured for a given scenario, e.g., for a specific area of interest, for combinations of content layers including, e.g., visualization variants for evaluation of different plannings. In many of our test scenarios, viewing virtual 3D city models in a straightforward way, the blueprint configuration is only essential part to implement an application using the image-based provisioning framework. For applications that require an individual scene configuration per user, a live tile endpoint of the layered map service can be used to query the required image tiles that are generated during application run time by a 3D rendering service instance. While this provides greater flexibility for users, it requires server-side 3D rendering resources to be available at run-time to serve these client applications.

Widely compatible applications that work with a minimum of server and client resources can be built by combining the different approaches for image-based provisioning. Users can be enabled to interactively configure their scene views when required, but for many applications and use cases the provisioning method using preprocessed data is usually preferable, since it keeps the required user interaction limited to a simple map-based interaction.

4.2.5 Configuration and Administration

For each given virtual 3D city model, an image-based provisioning requires a scenario description that consists of a description of the virtual 3D city model data, the 3D rendering service configuration (e.g., the available scene contents as named 3DPS layers, available rendering styles, available terrain styles, selectable styling and rendering methods etc.), and the configuration for light-weight end-user applications displaying rendered images together with prepared feature and POI data. Scenario descriptions contain all the information necessary to configure services for generation of condensed application datasets. This way, an automated update of contents for all kinds of clients and media can be performed by repeated invocation of the service-based provisioning process.

The information about available service endpoints for data preprocessing, 3D rendering of preprocessed data (including content to layer mappings and required rendering styles), and data provisioning settings (target format, additional data sources to include, etc.) are managed as scenario configurations as well. They are particularly important to orchestrate the services required for generation of application datasets. The data generation process can be derived and executed by invoking the necessary services using the arguments given by the scenario configuration.

Service administration also includes managing and assigning available hardware resources, i.e., GPU-based server systems for 3D rendering involving creation of 3D rendering service instances that generate image data for provisioning jobs. It specifies, e.g., the number of active rendering server nodes or creates cloud instances, such as amazon EC2 GPU instances, which can be requested an run on demand, bypassing the need to administer and operate the hardware.

4.3 Services

The following services provide the implementation of the described back-end functionality for image-based provisioning of virtual 3D city models. Their specific implementation is described in more detail in the following chapters.

3D Rendering Service

An image-based 3D rendering service has been designed and implemented as one core component. The service is responsible for synthesizing images of the virtual 3D city model and can generate different information layers (e.g., color images, depth image, object-id images) on a per-frame basis. Its implementation takes into account the specific statistical and graphics properties of virtual 3D city models, e.g., a large number of relatively small polyhedral objects (e.g., buildings), a high number of 2D textures (e.g., for facades). Further, the implemented rendering techniques are designed to handle parallel requests that do not provide frame-to-frame coherence, as these occur when multiple clients are using a 3D rendering service. It provides an OGC standardized 3DPS interface that allows for detailed configuration of the resulting images. This also involves styling of virtual 3D city models and feature data access.

Layered Map Service

The layered map service generates a tile-based representation of a given virtual 3D city model from a bird's eye perspective. It uses the 3D rendering service to synthesize the corresponding images. Based on layered maps, clients can blend-in and integrate application-specific georeferenced data by means of superimposed information layers. It is the key service to a broad range of applications that want to access contents of virtual 3D city models by means of this map-related 3D visual representation.

Video Service

The video service provides users with a high-level user interface to author and generate videos from virtual 3D city models. Users can model camera paths and varying scene contents over time, apply blending effects and overlays. Videos are rendered using 3D rendering services for image generation. This way, the service can generate video presentations that combine image data served by different service instances in one video.

Camera Service

Camera control and navigation is a complex and potentially domain-specific subject. It can be refactored and encapsulated by a separate service that computes camera views and camera paths to be used by camera animations. It is intended to serve as a blue-print service for more application-specific or domain-specific camera services. Its implementation can take advantage of the detailed and explicitly defined semantics of components of virtual 3D city models to provide higher-level, assisting 3D navigation and exploration functionality.

Chapter 5

3D Rendering Service for Virtual 3D City Models

In this chapter, design and implementation of a 3D rendering service for large-scale virtual 3D city models is presented.

5.1 Concept and Requirements

This service implements techniques for efficient, high-quality 3D rendering of virtual 3D city models as well as an interface for data selection and retrieval from an underlying database as standardized, self-describing service based on the View Extension defined by the OGC 3DPS specification. It allows for efficient, individual styling of features in a virtual 3D city model based on their attributes using an XML-based description language. The 3D rendering service is designed for the specific geometric and graphics characteristics of large-scale, massive virtual 3D city models as well as for the requirements for building data pipelines between different services operating on images of virtual 3D city models. It constitutes the basic rendering functionality to implement image-based provisioning of virtual 3D city models for the approaches presented in this thesis.

Objects contained in a model can differ significantly, e.g., in terms of geometric detail and precision, the availability and resolution of textures, as well as their semantic structure. For example, CAD-based models usually have a higher geometric complexity and precision than building models that have been automatically generated from remote sensing data. The massive amount of data, for building geometry and textures, terrain data, (map) image layers, and feature attributes, requires specialized rendering techniques to enable the efficient and high-quality image synthesis. In particular, service-based rendering for image-based provisioning of virtual 3D city models demands for rendering techniques that are able to cope with massive virtual 3D city model data and with multiple clients requesting images of different parts of the model in parallel.

Besides the 3D rendering engine, the efficient organization and access of rendering data plays a major role for rendering large-scale virtual 3D city models. Here, a data import service has been created that implements data integration for large-sized virtual 3D city model data into the rendering system, including preprocessing and optimizing input data. While input data for building and terrain models can be provided in a variety of computer graphics and GIS formats, the service implementation primarily targets CityGML-based datasets as it is the international standard for modeling and interchanging virtual 3D city models [Grö$^{+}$12; Kol09; SBN16]. The data import service is aware of feature data and attributes. It processes the models in a way, that the original features and their attributes can still be accessed and mapped to the optimized, renderable geometry generated by the service. Further service operations are provided for accessing feature data and attributes.

5.1.1 Requirements

The requirements for a 3D rendering service for virtual 3D city models differ from the requirements typically found in 3D graphics engines for real-time applications, e.g., computer games or CAD applications. In the following, the functional and non-functional requirements for a web-based 3D rendering service are grouped into two categories: a) requirements for the rendering process and b) requirements for feature data and its applicability to build image-based end-user applications.

Rendering Process Requirements

Robust Rendering of Heterogeneous Datasets Data from heterogeneous sources with varying resolutions and different quality levels, as it is common for virtual 3D city models [Bil+15; Jul+18], should be integrated into the rendering process in a robust way. Services should support the combination of different datasets so that a consistent resulting image can be generated. The rendering implementation and the data import should be able to handle, or even to fix errors in data modelling (geometric and semantic) and encoding as far as possible.

Massive Model Rendering Virtual 3D city models become more and more available and precise, e.g., due to improved remote-sensing technology (e.g., fast, high precision laser-scan devices or high-resolution aerial cameras), geometric processing (e.g., building model reconstruction), and resulting shorter update cycles. In contrast to authored 3D models, as they are used, e.g., in computer games, the geometry and textures of real wold models cannot be reused and are therefore hard to optimize. The generated datasets represent massive models and the corresponding implementation of a 3D rendering service has to handle such complex 3D models using limited hardware resources. For example, the rendering process and data management should adapt to the available hardware resources to avoid, e.g., slowdown of rendering or the service stopping to work due to hardware overload (due depletion of GPU memory, main memory, or disk space).

Configurable High-Quality Rendering The 3D rendering service should be able to maintain high rendering quality even for massive models. Rendering styles and rendering effects should be easily configurable by adjusting request parameters, without prior knowledge of the implementation and without modification of rendering source code. This eases the configuration of scene views and styling parameters by facilitating the easy experimentation with the visual results during development and use of image-based applications and, thereby, can lead to an improved quality of resulting applications and foster a reuse of implemented rendering techniques. This way, also users with no explicit skills in computer graphics can achieve good results by determining styling parameters by experimentation.

Scalability for Multiple Users A 3D rendering service should account for being used by multiple users in parallel. Here, GPU resources are found to be a limiting factor when it comes to service-based rendering. The service implementation should therefore use its hardware resources efficiently and apply parallelization wherever possible to obtain high utilization of GPU and CPU resources and fast execution results.

Missing Frame-to-Frame Coherence The performance of common real-time rendering techniques depends strongly on frame-to-frame coherence. The selection of 3D geometries and textures to render a frame depends strongly on the position and viewing direction of the virtual camera as well on the current scene configuration.

In our service-based approach multiple clients can access the service in parallel. Therefore, frame-to-frame coherence cannot be assumed when building processing and 3D rendering techniques. The 3D rendering service, therefore, must be aware of scattered camera settings and mixed scene configurations.

Requirements for Feature Data and Attributes

For dealing with feature data and attributes in image-based visualization systems the following requirements have been identified to support building applications on top of a 3D rendering service:

Information Access via Service Interface The service interface should provide or link feature data and attributes for any object that is depicted in a generated image. For this, a mapping from (at least) the ID of a feature in its source dataset (e.g., a CityGML virtual 3D city model) or service to pixels in the rendered image needs to be kept throughout the rendering process. Mechanisms are required to map this information in both directions. This way, information from external services can be linked, e.g., using Web Feature Services for retrieving detailed feature data in image-based GIS applications.

Feature Specific Thematic Visualization For feature-specific parameterization of rendering effects, e.g., to create thematic visualizations from 3D feature data, feature identities and attributes should be available in shader programs during the rendering process. This way, shading and rendering algorithms can involve attribute mappings for providing highly efficient attribute-based styling also for a massive amount of features as they are contained in large-scale virtual 3D city models.

5.2 Service Interface

The 3D rendering service provides the following service operations aligned to the 3DPS standard:

`GetCapabilities` The operation provides machine readable information that provide metadata about the service and its operations based on the OGC Web Service Common Standard [WG10]. The `GetCapabilities` response also contains the

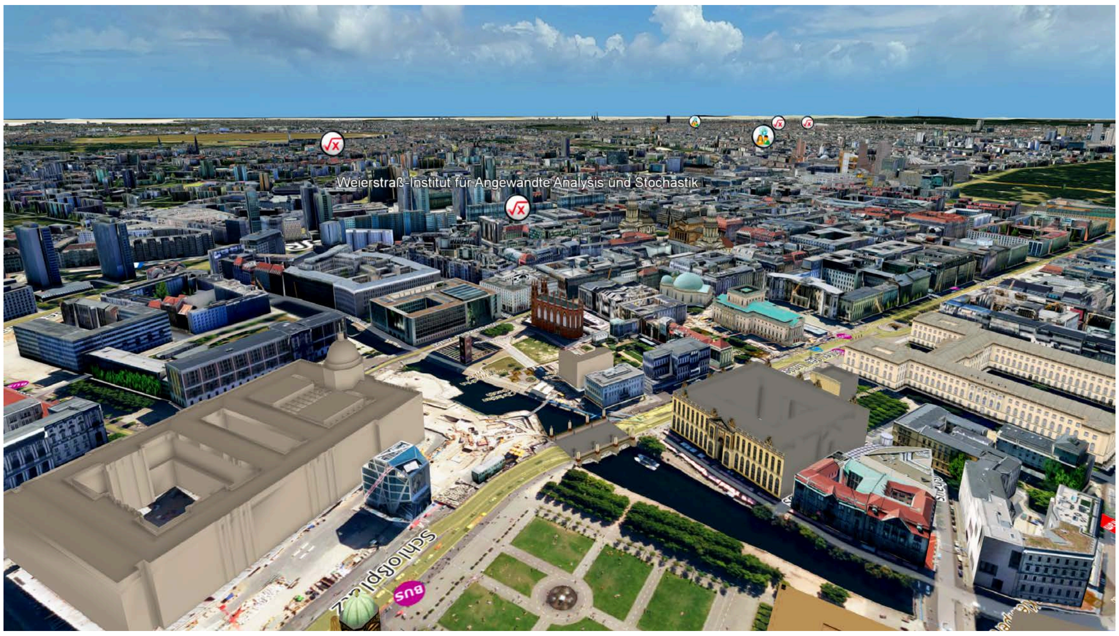


Figure 5.1: *Example of an image generated by a* `GetView` *request. It shows the virtual 3D city model of Berlin, layered terrain textures (road names on top of the aerial image), planning models (palace at the bottom left in shaded style), POI layers with labels. Several rendering effects are applied as image styling (screen space ambient occlusion, anti-aliasing, color improvement).*

additional parameters, e.g., for in-detail configuration of the rendering and styling process for `GetView` operations.

`GetView` The core operation of the image-based 3DPS for retrieving rendered G-Buffers. It allows to configure data, scene styling, image-based styling and other properties that influence the rendering process. The service provides a range of additional parameters, e.g. to enable and configure specialized rendering techniques or to configure image styles. An example of an image generated by a `GetView` operation is provided in Fig. 5.1. The corresponding key-value encoded request can be found at Listing 5.1.

`GetFeatureInfoByRay` Operation for retrieving feature data using a ray cast into the 3D scene. Returns object ID, feature ID, and available feature attributes if an feature was hit by the ray cast.

`GetFeatureInfoByObjectId` Operation for retrieving feature data using a scene configuration and a particular object ID. It returns the same data as the `GetFeatureInfoByRay` request.

`GetPosition` The operation provides functionality for ray-casting in 3D scenes, i.e., to compute 3D scene intersections for 2D image coordinates, a core feature for interaction in image-based visualization applications. Further it allows to project 3D points to 2D image coordinates. For both, the camera projection parameters need to be provided to the operation. Its implementation is based on the specification of the operation in the WVS discussion paper [Hag10]. It requires the scene content and projections to be defined by parameters, in particular content layers and their styles.

```
https://<host:port>/<service_endpoint_path>?
  SERVICE=3DPS&
  VERSION=1.0.0&
  REQUEST=GetView&
  CRS=WGS84&
  PORTRAYALS=
    WIDTH=3200;
    HEIGHT=1800;
    Projections=
      Perspective,            // Projection type
      13.401123986820638,52.52029899107024,216.66666666666669, // POC
      13.399299999999998,52.518100000000004,130.5709,   // POI
      0,0,1, // UP direction
      90,50.62, // FOV x and y
      1,87188;  // distance near and far plane
    IMAGELAYERS=COLOR; // image layer list
    FORMATS=image/jpeg; // image layer formats
    QUALITIES=95&   // encoding quality of image layers
  LAYERS= // selected content layers
        TERRAIN,TERRAIN,BERLIN,PLANDATEN,
        POI_ZWEISPRACHIGE_UND_FREMDSPRACHIGE_SCHULEN,
        POI_MATHEMATIK_UND_IHRE_ANWENDUNGEN&
  STYLES= // named content layer styles
        ORTHOPHOTO,ROAD-NAMES,DEFAULT,DEFAULT,
        DEFAULT,DEFAULT&
  BACKGROUND=SKY4& // selected background
  OVERALLSTYLES=SSDO,FXAA,IMAGE_ENHANCEMENT // selected image-based styling
```

Listing 5.1: *Key value encoded HTTP request for* GetView *service operation for generating the image illustrated in Fig. 5.1. The OVERALLSTYLES parameter represents named styles that are referred as image-based styling effects throughout this thesis.*

5.3 Architecture

The 3D rendering service system consists of two top-level components: a) The data import service implements preprocessing and optimization of source datasets, i.e., virtual 3D city model geometry, textures, and feature data and b) the 3D rendering service implements 3DPS operations for image generation, feature data access, ray casting, and service description. Fig. 5.2 shows the architecture of the 3D rendering service system. The data import service and the 3D rendering service use the same optimized formats of *rendering data*, i.e., optimized texture and binary geometry data, scene graph representation, and feature database schema. The techniques for optimization and organization of virtual 3D city models that are implemented as *data import service* are described in Section 5.5.

The *request handler* provides the interface for the service. It parses and validates requests and enqueues requests for the handling components.

The *G-Buffer generator* implements the 3D rendering process that is used to generate images for the GetView operation. It processes requests from the *rendering request queue* and configures a per-request scene graph and the OpenGL rendering pipeline based on the current rendering request.

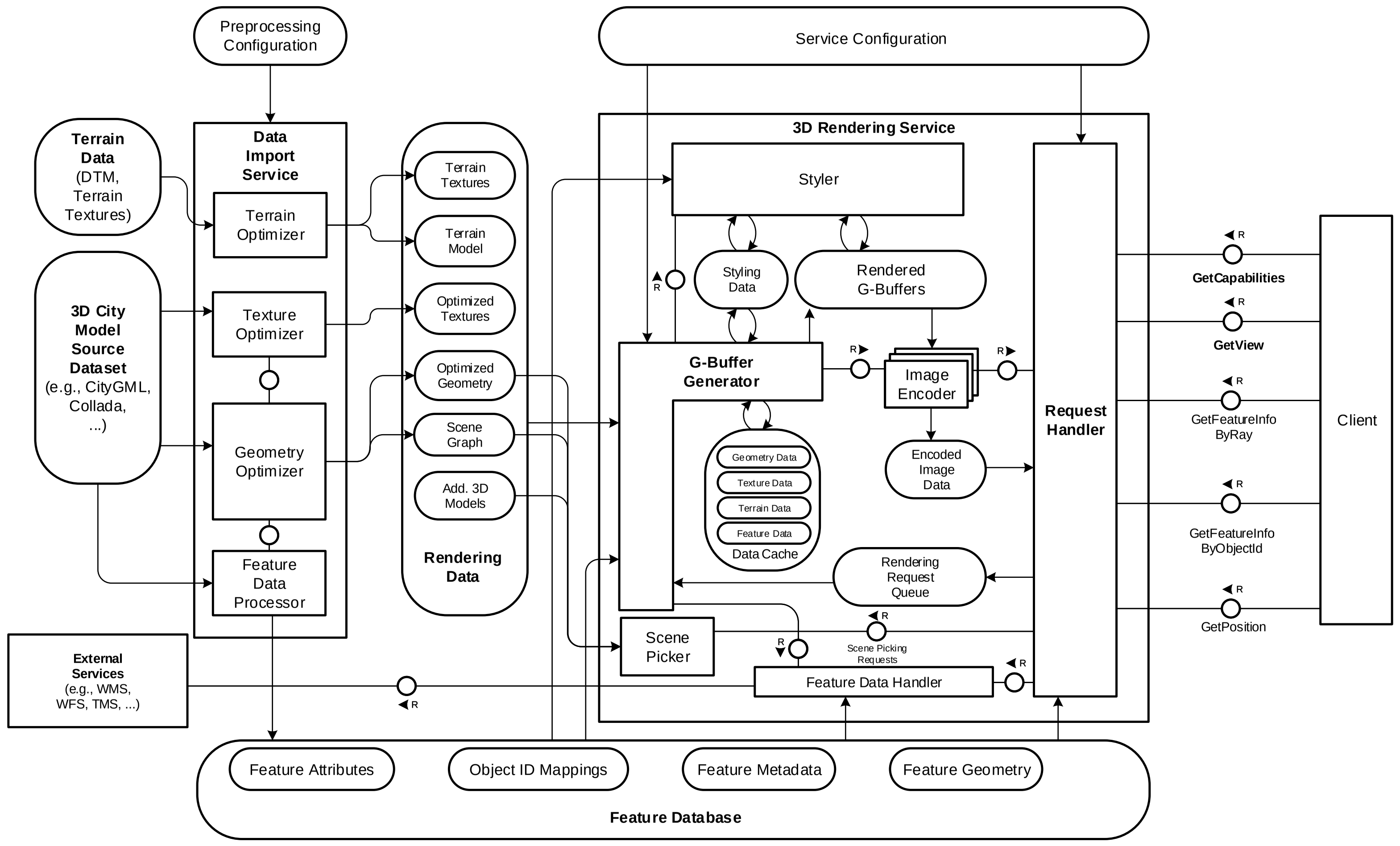


Figure 5.2: *Architecture and interfaces of the 3D rendering service system. The system provides two main components: a) a data import service for import and optimization of large-scale models and b) the 3D rendering service itself for service-based image generation.*

The *styler* component evaluates the styling-specific parameters in rendering request. It generates required data (*styling data*) for the G-Buffer generator to apply the different styling variants supported by the 3D rendering service (see Section 5.6). It can also use and modify generated G-Buffers to apply image-based styling via image postprocessing.

The request processing for `GetView` requests, as the main service operation of the 3D rendering service, is implemented in four major steps:

1. **Request Parsing:** The *request handler* parses requests and checks them for validity. Valid requests are put into the *rendering request queue* to be processed by the G-Buffer generator.

2. **Configuration of 3D Rendering and Data Fetching:** Depending on the requested content and styles, a per-request scene graph is built, the 3D rendering pipeline is configured, and required data for the current request (terrain, geometry, texture, and feature data) is fetched (either from disk or from external services). Additional G-Buffers are added to the request if they are required by requested image-based styling effects.

3. **3D Rendering** The rendering process gets executed on GPU and thematic styling is applied. After generating G-Buffers, image-based styles are applied using the GPU in a postprocessing step. Finally, G-Buffers are downloaded from GPU-memory into main memory.

4. **Image Encoding** For each requested G-Buffer, a response image file encoded in the required format is generated. Image encoders can work in parallel to speed up image encoding.

The G-Buffer generator applies in-memory caching as well as disk-based caching for rendering data to improve rendering times and avoid repeated calls to *external services* that can be used to request map tiles (via WMS, WMTS, TMS), vector data for feature terrain overlays (e.g., a road geometry layer), POIs, or additional feature attributes (e.g., from WFS).

The *scene picker* component handles `GetPosition` requests. It implements ray casting using the scene graph and geometry that is also used for 3D rendering by the G-Buffer generator.

The *feature database* stores data about features of the virtual 3D city model, e.g., a mapping of the rendering data's object ID to features or feature attributes of a source dataset (see Section 5.5.3 for a description of the data model). The data is used by several components in the 3D rendering service, e.g., for thematic styling, for feature ID queries, or for delivering feature attributes to clients.

The *service configuration* defines the datasets that are provided for rendering as well as the location of the corresponding data, either optimized by the data import service or *additional 3D models* in computer graphic formats such as COLLADA or X3D. Further, the scene configuration defines the grouping of datasets into scene layers that advertised in the `GetCapabilities` document.

5.4 Rendering Service Environment

The 3D rendering service implementation is expected to be executed in a server environment, providing specific minimal hardware resources. Any techniques presented in this chapter therefore relies on the following assumptions regarding the available server environment:

CPU The CPU of the host system should provide parallel processing capabilities as the techniques and algorithms are designed to run in parallel using multiple threads. The more CPU cores are available the faster image requests can be handled. A minimum of 4 available CPU cores is expected for hosting the 3D rendering service.

Main Memory The system main memory is used primarily for the scene graph of the virtual 3D city model and corresponding texture and attribute data. The more main memory is available, the more data can be cached in memory and the better the corresponding performance of the overall system. Due to the out-of-core capabilities of the rendering techniques, the minimum requirement for running the service should be 2 GB main memory for smaller models and 8 GB for larger ones. Common server environments are usually providing far more than these amounts of main memory.

GPU 3D Rendering Service instances require access to an OpenGL enabled GPU with sufficient GPU memory to deal with the geometry and texture amount. The more GPU memory is available, the more of the geometry and texture data can be held directly close to the GPU. Further, rendering techniques as well as image-based styling techniques often require a large number of frame-buffers to be created in GPU memory. The larger the required resolution for images, the more GPU memory is required for the techniques to work properly and efficiently. The minimum expected GPU memory is 2 GB. For larger models, a minimum of 4 GB GPU memory would be recommended to handle the geometry and textures in complex rendering techniques.

Storage The server system is expected to provide fast IO capabilities, which means that there is a solid state disk available for storage and fast random access of texture and geometry data of preprocessed virtual 3D city models. This is of particular importance as geometry and textured can be required for consecutive rendering requests that require distinct sets of geometry and texture to be rendered (missing frame-to-frame coherence). Conventional hard drives would provide a limiting factor to the number of nodes that can be loaded simultaneously.

Network Transferring images over a network requires large network bandwidth, depending on the requested image resolution and encoding format. A commonly available server connection is expected to be available for transferring images (starting from 100 MBit/s).

All of these assumptions can today be provided either by physical servers or by virtual machines or services provided by cloud environments such as the Amazon Elastic Compute Cloud EC2 in a cost efficient way. This way, the solution can be provided in a variety of environments, from self-hosted servers within an organization to completely managed remote data centers.

5.5 Optimization of Massive CityGML-Based Virtual 3D City Models for Service-Based Rendering

CityGML as well as other encoding and transmission formats for 2D and 3D geodata are designed to reduce and avoid information loss when exchanging data between systems and organizations. “Although it is possible to render 3D views directly from CityGML this language is more suitable for representation than for visualization.” [RFC13] Because of this, it is necessary to transform and optimize geometry and textures of such a model for OpenGL-based 3D rendering and 3D graphics hardware. There are three main goals of such a process:

Optimization of 3D Geometry GPU hardware is heavily optimized for parallel processing of vertex data. Examples for such vertex data used during 3D rendering are, e.g., vertex positions, texture coordinates, vertex object ids, or material properties. While rendering of geometry is implemented efficiently in hardware on the GPU, modifications to the OpenGL state, e.g., changing material properties, matrices, or applied textures, are expensive operations. Therefore, it is highly favorable to organize vertex data into larger buffers containing data for multiple faces of that geometry. This reduces the number of OpenGL state modification during rendering significantly, since vertex and fragment shaders can access the necessary information for rendering directly from these buffers on a per-vertex basis.

Each batch of vertex data should have a size that takes the IO and processing capabilities of the hardware system into account. Further, the batches should only contain vertex data for geometry that covers a common spatial area to support loading vertex data for a specific area of interest, e.g., defined by the current camera projection.

Texture Optimization and LOD Mechanism Textured virtual 3D city models usually depend on large numbers of individual textures that define the appearance of model elements such as building facades. In common 3D city model datasets, these textures are either provided as single files that correspond to a specific geospatial extent (e.g., each texture file covering a specific building facade or aerial photography provided as 2D texture) or as texture atlases that pack textures for multiple surfaces into a single texture container [SHC12]. Both types are commonly not usable for efficient 3D rendering. For massive virtual 3D city models, textures need to be handled efficiently by the rendering process as texture handling represents one of major bottlenecks for rendering performance. To this end, the corresponding processing, storage and rendering techniques apply level of detail (*LOD*) techniques to handle such texture data [BD05].

Retain Feature Data Mapping The connection between rendered vertex and dataset feature needs to be preserved. Therefore, an efficient mechanism for encoding of this information into frame-buffers and corresponding image formats is required.

The different data types of a virtual 3D city models are handled individually by the data import service to achieve an optimal data organization and encoding for the 3D rendering process. Processes that are implemented to optimize the different data types are illustrated in Fig. 5.3.

5.5.1 Geometry and Feature Data Optimization

Virtual 3D city models are characterized by a large number of individual objects (e.g., buildings, streets, trees) that are spread across a geospatial extent in varying density. Key to rendering optimization for this class of 3D models is the fact that they represent in a sense a 2.5D model, i.e., a model predominantly distributed over a 2D plane segment.

Usually, simple 3D building models represent a large portion of the objects of a virtual 3D city model. They are specified, e.g., as 3D extrusion shapes of their ground polygons together with a (often simple) roof geometry. That is, most of the objects of a virtual 3D city model are not complex in terms of 3D geometry. In particular, this is the case for CityGML LOD2 models, which represent the majority of building models in typical virtual 3D city models. For higher level of details (i.e., CityGML LOD3 and LOD4), the geometric complexity is significantly higher, since these models contain, e.g., individual walls, windows, doors, and also indoor geometry that can also have individual materials and textures assigned. To achieve a representation that facilitates real-time rendering for all of these objects, there are six processing steps applied:

1. **Data Parsing and Analysis:** The source data is parsed, analyzed, objects are identified (e.g., buildings in a CityGML-based city model), and corresponding object IDs, an integer number, are assigned. The object ID refers to the object identity within the original dataset, e.g., a CityGML document, or a cadastral dataset). Further, texture assignments, a mapping between geometric primitive and texture, are extracted for later optimization (see Fig. 5.3). The geometry is extracted from source documents and converted into vertex data with their object ID, and further available properties, such as material or color data, assigned as vertex attributes.

2. **KD-Tree Building:** The vertex data is inserted into a KD-tree [Ben75] spatial data structure to build a queryable set of geometry nodes with a comparable amount of vertices.

3. **Rendering Optimization:** The vertex data nodes are optimized for rendering, e.g., the vertex position is optimized for float representation on GPU by extracting an offset matrix from single nodes and adjusting the vertex positions accordingly.

4. **Compute Feature Properties:** Feature properties, e.g., a 2D bounding box, the maximum and minimum height, and feature attributes are extracted from the source data and inserted into the feature database. Object ID mappings for features are written to the feature database database.

5. **Scene Graph Encoding:** The final scene graph, which structures and organizes the virtual 3D city model for rendering, is built based on the KD-tree. It is written in a compact binary format that can be efficiently addressed and loaded by the rendering engine.

The resulting serialized scene graph provides homogeneously sized leaf nodes with all the geometry for a specific spatial area. This way, loading times per node are predictable for a given hardware configuration.

5.5.2 Texture Optimization

In contrast to 3D models in gaming or CAD, virtual 3D city models are almost entirely based on real-world data about a given physical environment whose parts and objects should reflect their individual appearance. This leads to a massive number of textures that must be stored and efficiently managed by the rendering engine. For example, the virtual 3D city model of Berlin, contains more than 5.7 million individual textures with a resolution of about 10 cm per pixel that are used to texture buildings.

To cope with massive texture data the data import service applies *virtual texturing* [LDN04; Mit08]. To this end, model textures are rearranged into a single large-scale virtual texture atlas, which is physically stored using square image tiles each covering 256 pixels of the original texture. A detailed introduction of the process and texture coordinate calculation is provided by Mayer [May10].

A complete texture atlas that contains all original input textures would require an enormous amount of main memory — resulting texture atlases would a physical dimension of millions of pixels square — it cannot be processed and assembled in one single step. Instead, sets of inhomogeneously sized, rectangular input textures originating from the input data are joined into sub-atlases of maximum 16,384 x 16,384 pixels using a recursive, tree-based texture placement algorithm[1], which ensures an efficient fill rate of these sub-atlases. This avoids sparsely filled image tiles, which is particularly important when tiles are uploaded to the GPU, as empty areas in tiles will never be used by the GPU, but still consume GPU memory. A overview of the steps of the texture optimization process can be found in Fig. 5.3.

Beside the image tiles representing a full-resolution version of the provided source textures, several mipmap levels are generated. The storage of large numbers of texture image tiles and their corresponding mipmap levels requires a large amount of disk space and requires fast IO capabilities during rendering to render images efficiently. To reduce the disk space and IO throughput required per image tile and for improving the overall rendering process, image tiles are compressed twice: In a first step, they are encoded using *DXT5* format[2], a lossy image compression format that supports transparency in textures and can be used directly by GPUs for texturing. In a second step, the resulting DXT5 data is compressed using the *lz4*[3] algorithm, which provides a high compression ratio along with very high performance decompression. The combination of these two compression method provides a very efficient method for storing these large-scale sets of image tiles. For the final storage of the data, the processing creates a SQLite database that offers a fast way for retrieving large numbers of relatively small files.

For retrieving the texture tiles during the rendering process, a tile ID and the corresponding texture coordinate is encoded in a virtual texture coordinate that is stored as vertex attribute. A separate rendering pass is performed during rendering that

[1] `http://blackpawn.com/texts/lightmaps/default.html`
[2] `https://www.khronos.org/registry/DataFormat/specs/1.1/dataformat.1.1.html`
[3] `https://lz4.github.io/lz4/`

produces a frame-buffer containing the IDs for the tiles that are required to render the current frame. The required tiles are fetched from disk and uploaded into a physical texture on the GPU.

The virtual texturing based method for texture preprocessing and rendering provides a very efficient handling of individual textures per building. It minimizes the amount of data to be loaded to the GPU per frame by following a view-based approach. This way, fast rendering can be achieved also if there is little to no frame-to-frame coherence between two consecutive requests. Further in main memory caching strategies for texture tiles are applied to further improve loading times for textures.

5.5.3 Feature Data Modelling and Import

The image-based provisioning approach gives applications access to features, their attributes, and their hierarchies via service interfaces. Feature data is contained in common geodata formats, e.g., in CityGML, Shape files, or GeoJSON. For attribute-based styling and access through the 3DPS interface, the data is stored in the feature database using a unified data model together with their attributes and meta information.

Data Model

Feature data and attributes are modeled by a relational database scheme, which allows for efficient storage and querying of feature attributes of different types, the model specific object IDs and their mapping to feature IDs in source datasets and metadata. Additionally, the database stores feature hierarchies using a parent reference for each feature included in a model instance. A `FeatureGeometry` represents a simplified geometry of a feature. It contains a 2D bounding polygon and a computed or given coordinate. While the model does not store the complete geometry, as it can be accessed in source data, store `minHeight` and `maxHeight` attributes to allow estimations of the spatial extent using database queries.

For every feature, there can be multiple `AttributeSources`. Each of these sources, e.g. a CityGML dataset that contains additional generic feature attributes, can provide different, named feature attributes with corresponding values for a point in time. Here, metadata about the attributes is stored as `AttributeKey` instance. Attribute entries are uniquely identified by these three properties (ID of the feature, attribute key, and timestamp). This way, different versions of the same attribute can be stored and accessed using this model (time series support).

Feature and attribute sources store mappings of the origin dataset per feature and attribute. This is especially important due to legal reasons (e.g., data ownership, data licences that require to display attributions when their data is used). Further it allows to query the original information from 3rd party data sources and to recognize features in case of updates to be performed. The overall data model is illustrated in Fig. 5.4.

Feature Data Import

Feature data is processed during data import. The data import service creates the necessary data structures, such as feature and attribute sources for the imported dataset, features and their basic geometry as well as attribute keys and their values extracted

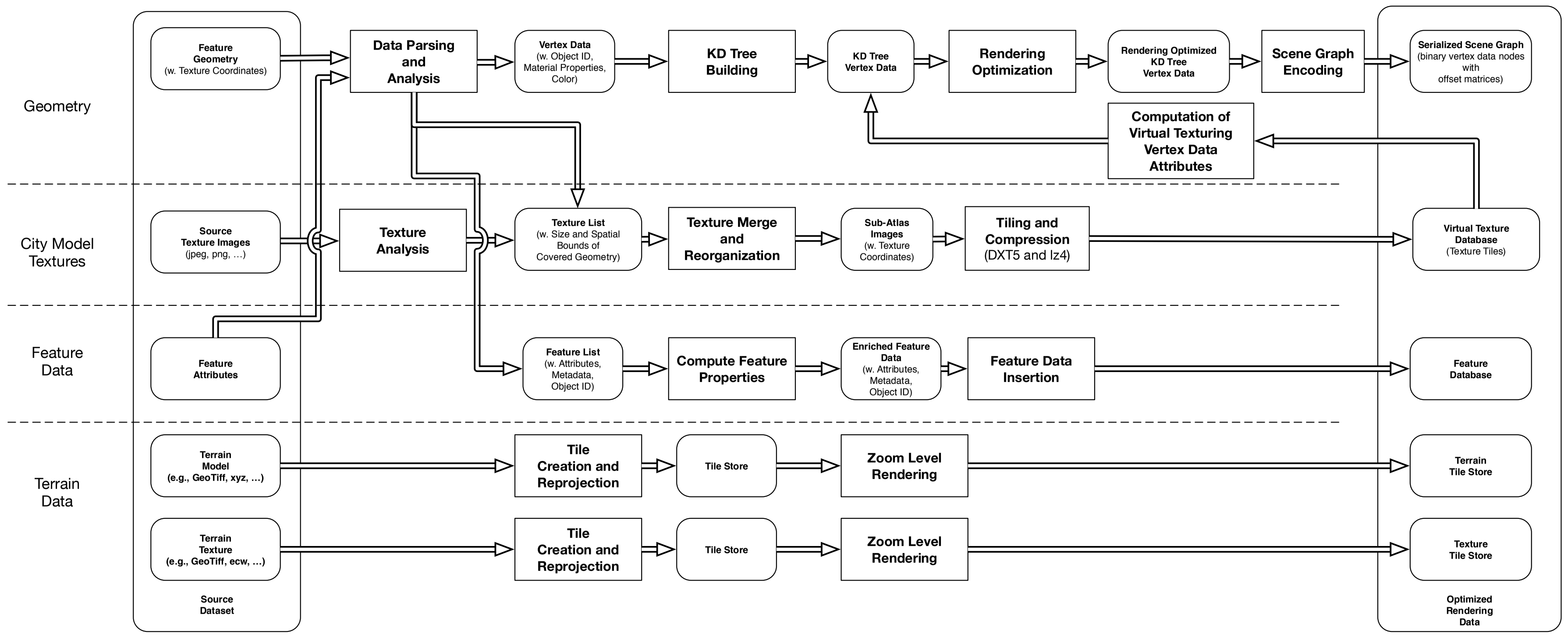

Geometry
City Model Textures
Feature Data
Terrain Data
Feature Geometry (w. Texture Coordinates)
Source Texture Images (jpeg, png, ...)
Feature Attributes
Terrain Model (e.g., GeoTiff, xyz, ...)
Terrain Texture (e.g., GeoTiff, ecw, ...)
Source Dataset
Data Parsing and Analysis
Vertex Data (w. Object ID, Material Properties, Color)
KD Tree Building
KD Tree Vertex Data
Rendering Optimization
Rendering Optimized KD Tree Vertex Data
Scene Graph Encoding
Serialized Scene Graph (binary vertex data nodes with offset matrices)
Computation of Virtual Texturing Vertex Data Attributes
Texture Analysis
Texture List (w. Size and Spatial Bounds of Covered Geometry)
Texture Merge and Reorganization
Sub-Atlas Images (w. Texture Coordinates)
Tiling and Compression (DXT5 and lz4)
Virtual Texture Database (Texture Tiles)
Feature List (w. Attributes, Metadata, Object ID)
Compute Feature Properties
Enriched Feature Data (w. Attributes, Metadata, Object ID)
Feature Data Insertion
Feature Database
Tile Creation and Reprojection
Tile Store
Zoom Level Rendering
Terrain Tile Store
Texture Tile Store
Optimized Rendering Data

Figure 5.3: *Overview of the processes for optimizing virtual 3D city model geometry, textures, and terrain data, and organizing feature records within a relational feature database.*

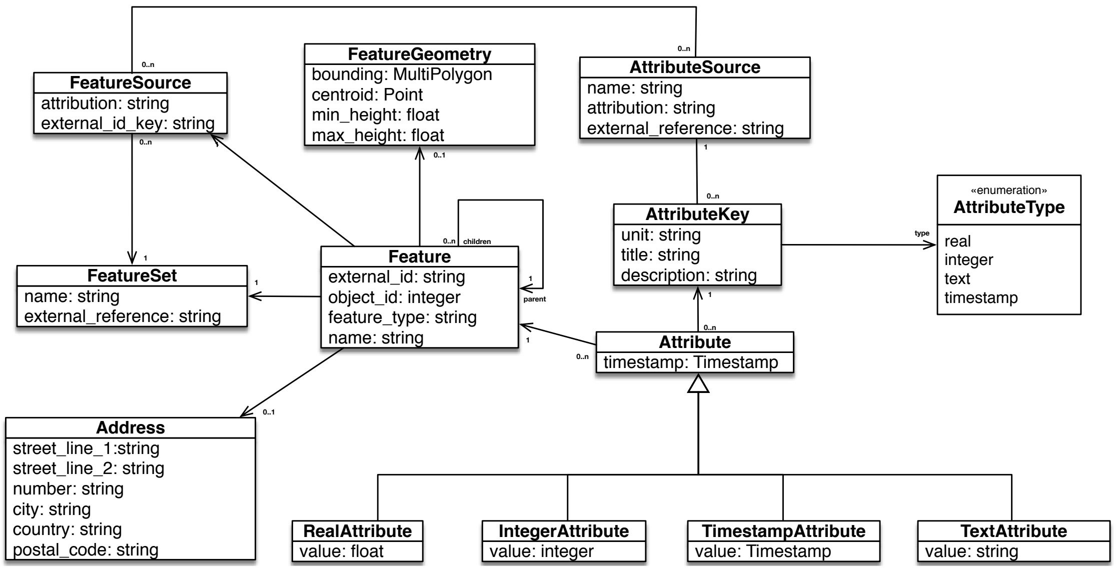


Figure 5.4: *Data model for storing features, their origin, attributes and relations. An object ID is assigned to features that identifies a feature within its feature set. Attribute meta information, such as the unit, type, and name of attributes, are modeled as separate* `AttributeKey` *entities. The model is implemented with a PostgreSQL database management system using PostGIS extensions for handling of spatial data.*

from the dataset. Besides the attributes that are explicitly included in the source data itself, the import can invoke additional algorithms that compute additional attribute values or query additional values from other data sources or services to enrich the feature dataset.

5.5.4 Optimization of Terrain Data

Terrain data for 3D rendering of virtual 3D city models include *digital terrain models* (DTM) as well as terrain textures such as maps or aerial imagery. Most commonly, DTMs are distributed as 2D height fields encoded, either in text-based formats such as xyz or encoded in georeferenced image formats such as GeoTIFF. This format is also frequently used for terrain texture data. Height fields and especially terrain textures can provide very high resolutions, e.g., up to 5 to 10 cm per pixel for aerial imagery. Therefore, rendering of textured terrain models requires reorganization of that data to allow efficient access to the data during 3D rendering. The common approach to deal with such 2D data is to apply tiling to the overall terrain model and textures. Here, source data is split into rendered tiles at fixed scales (*zoom levels*)[OSG]. Tiling of image data is supported well by existing geodata processing tools and libraries, such as the open source Geospatial Data Abstraction Library[4] (*GDAL*), OSGEarth[5] or commercial tools such as Map Tiler[6] or FME[7]. The data import service supports processing of such

[4]`https://www.gdal.org`
[5]`http://osgearth.org`
[6]`https://www.maptiler.com`
[7]`https://www.safe.com/`

file-based terrain data into tile stores using GDAL tools encoded as mbtiles[8] based on the Tile Map Service [OSG] tiling schema. The created tiles can then be efficiently loaded on demand by the 3D rendering service during runtime.

Preprocessing is only applied for file-based datasets. Service-based datasets are integrated during runtime.

5.6 Styling Virtual 3D City Models

Styling generally defines how features and their attributes are mapped onto visual variables (e.g., color, shape, texture) of a corresponding visual representation. Insofar, it forms an essential functionality for most visualization applications because it defines how to express thematic information.

The term *styling* is used frequently in a number of OGC standards. Services use SLD to identify and define the styling. "SLD addresses the important need for users (and software) to be able to control the visual portrayal of the geospatial data. The ability to define styling rules requires a styling language that the client and server can both understand." [9]

The 3D rendering service implements three different approaches for styling for virtual 3D city models:

Layer-Based Styling is used to apply rendering effects based on scene graph modifications during 3D rendering of the virtual 3D city model. Terrain image layer selection is also implemented by layer-based styling.

Thematic Styling by SLD evaluates feature data and attributes based on a user-provided styling description (SLD) to adjust the visual appearance based on styling rules that evaluate attribute values.

Image-Based Styling implements rendering effects by applying image-space operations in a postprocessing step for each generated frame.

These three approaches can be combined to achieve styling effects for virtual 3D city models which provide, e.g., texturing of massive virtual 3D city models implemented through layer-based styling, high-quality ambient occlusion using image-based styling, and rendering of thematic 3D visualization using styled layer descriptions referencing feature attributes. Together they form a styling pipeline implemented in the 3D rendering service. An overview of this pipeline combining the styling approaches shown in Fig. 5.5. First, layer-based styling is applied during the rendering process for G-Buffer generation. Then, the SLD-based thematic styling is applied incorporating the required feature attribute data. Afterwards, the generated G-Buffers are used by image-based styling techniques. Each styling approach can further be configured by additional request parameters that are advertised in the `GetCapabilities` document of the 3D rendering service. The styling techniques are implemented in by the *styler* component within the 3D rendering service architecture (see Fig. 5.2).

[8]MBTiles spec `https://github.com/mapbox/mbtiles-spec/blob/master/1.3/spec.md`
[9]`http://www.opengeospatial.org/pressroom/pressreleases/761`

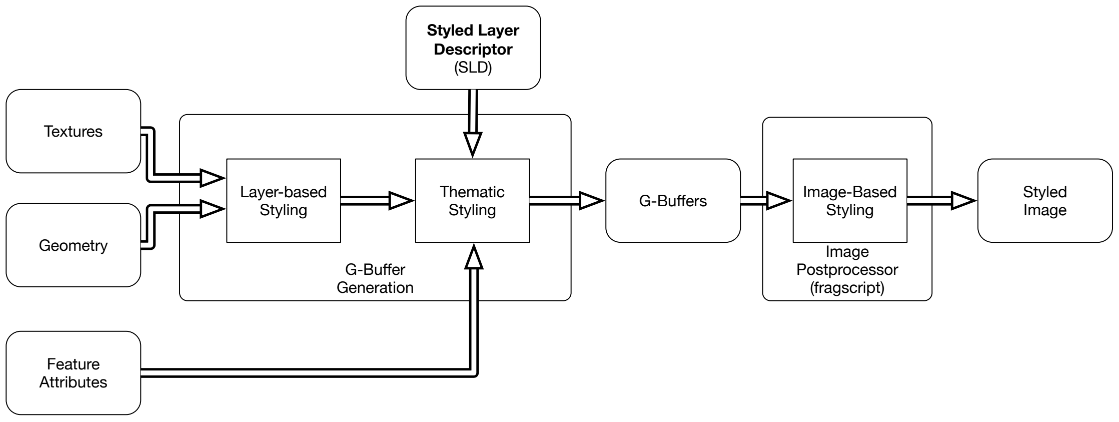


Figure 5.5: *Overview of the styling pipeline that applies the three styling approaches for virtual 3D city models within the image generation process.*

5.6.1 Layer-Based Styling

Following the OGC 3DPS standard, each named layer in an image-based portrayal request has a named style assigned. In our approach, the *layer-based styling* denotes an implementation of this styling strategy. It is based on shaders and rendering passes that are applied to the scene graph that contains the layer geometry, i.e., the style modifies the scene graph used for 3D rendering of the virtual 3D city model, e.g., by exchanging materials or shaders for OpenGL rendering or by introducing additional rendering passes. The implementation of layer-based rendering styles can therefore affect every stage of the overall rendering process, i.e., they can be implemented in geometry shaders, vertex shaders and fragment shaders to influence the rendering result. This way, complex rendering techniques can be implemented using this technique. But it requires complex implementation effort to create and distribute such rendering techniques, as this requires modification of the source code of the 3D rendering service implementation. As one example, the technique for model texturing is implemented as layer-based styling technique named “textured”. An example of a request using layer-based styling is provided in Listing 5.1.

5.6.2 Thematic Styling of Virtual 3D City Models

Thematic styling defines how feature attribute values of city model features are mapped to visual variables. E.g., the mapping a buildings value (as attribute value) to a color value (visual variable) as illustrated in Fig. 5.7. The thematic styling in our approach includes three components:

Styling Language for 3D The styling language encodes the feature attribute selection and their mapping to visual variables within the 3D rendering process.

Attribute Mapping A process that fetches the required feature attributes from a database, encodes these attributes in a way that allows for efficient access of per-feature attribute values in GPU shader programs.

GPU-based Representation and Selection of Rendering Styles The styling implementation uses GPU-friendly implementations to cope with attribute-based styling of large numbers of individual features. The implementation is based on an approach for hardware-accelerated attribute mapping for interactive visualization techniques, which “directly transfers preprocessed input data to the GPU that subsequently performs geometry manipulation and attribute mapping.” [Bus+14]

The implemented styling language is based on the extensions to the OGC SLD to 3D proposed by Neubauer and Zipf [NZ08; NZ09]. It provides capabilities to define styling rules using OGC *Filter* expressions to select features to which defined *symbolizers* are applied, which define the graphical style to be applied. Currently, three types of symbolizers are implemented:

`AppearanceSymbolizer` applies fill colors;

`TextureSymbolizer` applies parameterizable procedural texture patterns;

`GeometrySymbolizer` affect the geometry, e.g., by changing scaling in each dimension.

```xml
<?xml version="1.0" encoding="utf-8"?>
<StyledLayerDescriptor>
    <NamedLayer>
        <Name>BERLIN</Name>
        <UserStyle>
            <Name>Immo has appartment</Name>
            <FeatureStyleType >
                <Rule>
                    <Filter>
                        <PropertyIsEqualTo>
                            <PropertyName>has_appartments</PropertyName>
                            <Literal>true </Literal>
                        </PropertyIsEqualTo>
                    </Filter>
                    <AppearanceSymbolizer>
                        <Fill mix="0.5">
                            <SVGParameter>#0000FF</SVGParameter>
                        </Fill>
                    </AppearanceSymbolizer>
                </Rule>
            </FeatureStyleType>
        </UserStyle>
    </NamedLayer>
</StyledLayerDescriptor>
```

Listing 5.2: *Example of a SLD Descriptor defining one* `Rule` *element with one* `Filter` *and one* `AppearanceSymbolizer` *to be applied. It blends the original color (*`mix` *attribute) with a blue color.*

Symbolizers may either define static values, reference feature attributes via properties, or apply functions using these attributes to specify visual variables, e.g., color or object scaling. Such functions could be interpolation definitions or arithmetic calculations using feature attribute values as arguments.

An example of a SLD is provided in Listing 5.2. It defines one rule to affect the content layer named ’BERLIN’. The `Filter` element defines all features to be matched that have an attribute `has_appartments` with the value `true`. The `AppearanceSymbolizer` provided modifies the appearance of the matched features by mixing their original color (originating either from an assigned texture or material) with the color provided as hexadecimal color code.

To apply thematic styling efficiently for large numbers of unique features to be rendered, e.g., buildings within a large-scale virtual 3D city models, SLD documents are compiled to a representation that can be efficiently uploaded and evaluated by OpenGL compute shaders. The implementation of the thematic styling adds a style evaluation step to the 3D rendering process (i.e., the filtering and mapping stage of the visualization process) implemented as OpenGL compute shaders. In these stages, compiled style definitions are evaluated to create a set of rules that are evaluated during the rendering process to apply a per-feature styling to the resulting image. The SLD documents are parsed and compiled to an efficient OpenGL-based representation on CPU. All stages of the (either at the geometry, rasterization or fragment processing)a preprocessing step for styling rules to the rendering process. Fig. 5.6 shows the process for thematic styling implemented in the 3D rendering service.

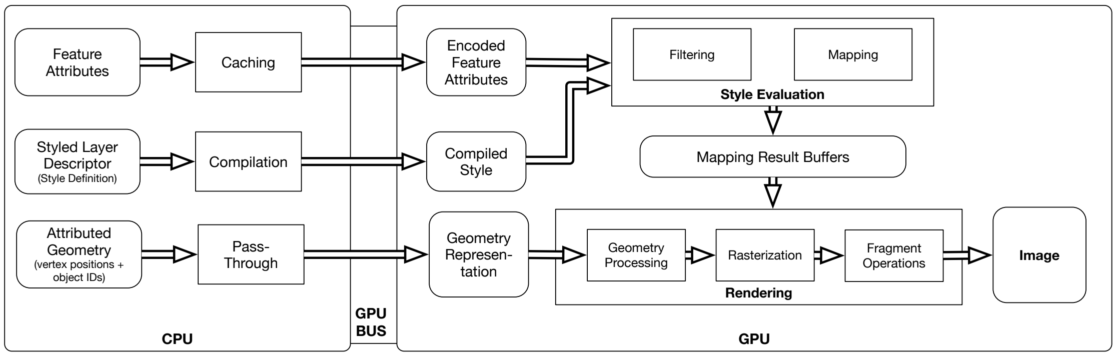


Figure 5.6: *Applied concept for thematic styling in the 3D rendering service. Styled Layer Descriptors are parsed and compiled into compact representations on the GPU. Style evaluation implements the filtering and mapping stage of the visualization pipeline based on OpenGL compute shaders. It generates mapping result buffers that contain the values for visual variables to be applied during the OpenGL rendering process.*

Style evaluation builds a set of mapping result buffers, each containing per-feature values for the target visual variable, e.g., a saturation, vertex color, or scaling vector.

Since these buffers do not depend on the camera projection, i.e., projection type, camera position and orientation, the style evaluation is only required if if the input data or styling changes. Therefore they can be efficiently cached and reused for multiple frames requesting the same style. An example applying thematic styling for multiple visual variables is shown in Fig. 5.7.

5.6.3 Image-Based Styling

Image-based styling implements rendering effects that are not based on 3D geometry and its vertex attributes but on per-pixel information generated during the 3D rendering process (G-Buffers). Different kinds of per-pixel information, e.g., color, depth, object id, or normal direction, are evaluated by fragment shaders. Image-based styling can be seen

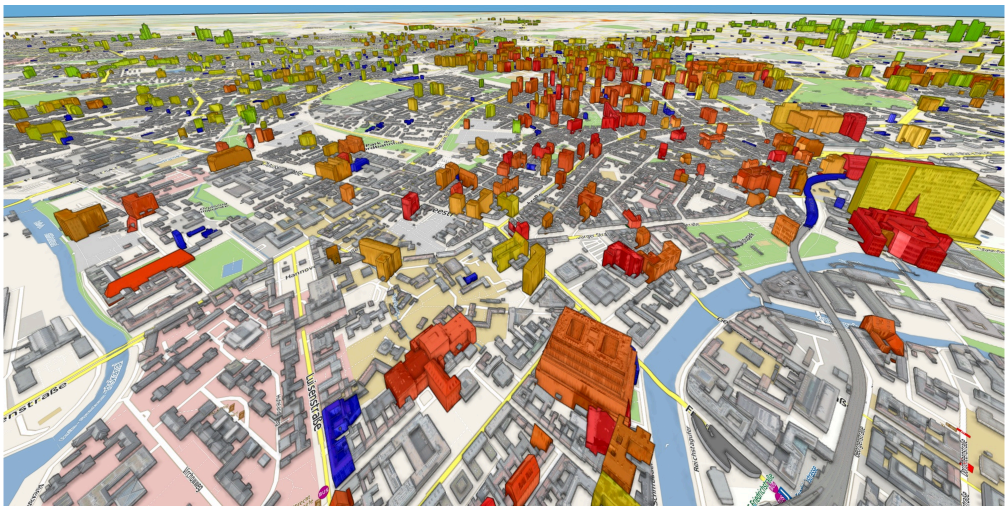


Figure 5.7: *Example of thematic styling: Visualization of available apartments and their pricing based on the Berlin virtual 3D city model. The 3D rendering service applies styling for each of the 600,000 building models using two visual variables: 1) Buildings that are out of scope (no apartments available) are flattened and the building textures are desaturated. 2) Buildings of interest are scaled up and their textures are blended with colors reflecting the rental price per square meter.*

as a postprocessing effect on image data that has been created by the core 3D rendering engine.

In contrast to layer-based styling on a scene graph level, image-based styling is performed in image space at image resolution. This allows us to decouple styling from image generation, providing advantages in terms of a more efficient development process for image based rendering effects and increased reusability of the effect implementations [Hil16].

Image-based styling in our prototype system is implemented in a framework for scripted shader execution named *fragscript*. Fragscript is based on a C++ engine for image-based OpenGL rendering using G-Buffers and Lua [IFF96], a scripting language that is used to describe the parameterization and execution order of OpenGL fragment shaders for effect application. The components of the framework and its invocation by the 3D renderer for virtual 3D city models in illustrated in Fig. 5.8. Fragscript provides an extensible *effect library* that contains self-descriptive image-based rendering effects. The fragscript API is able to publish the effect metadata, such as effect name, required G-Buffers, or necessary as well as optional configuration parameters together with their default values. As the library is integrated into the 3D rendering service, the corresponding entries for 3DPS `GetCapabilities` can be automatically derived from the data provided by the fragscript API. Configuration parameters for image-based styling effects are exposed as additional parameters of the *GetView* operation following the naming scheme `X_<IMAGE_STYLE_NAME>_<PARAMETER_NAME>`. This way, fragscript is used as extensible effect library that provides a framework for implementation .

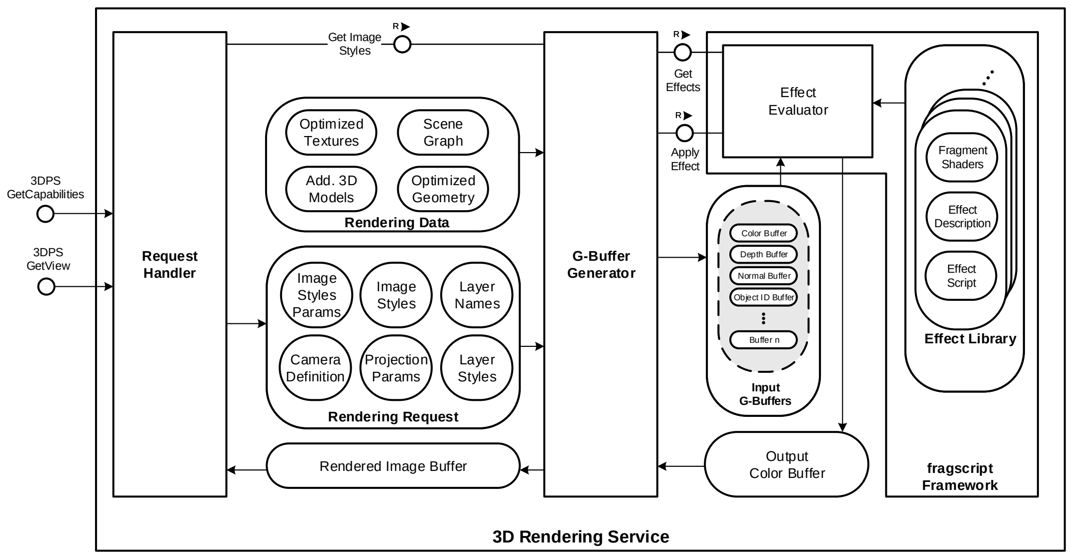


Figure 5.8: *Overview of components for image-based styling based on a library of effects and a generic, scripted effect evaluator. Effect descriptions are provided to the 3D renderer. If an image-based effect should be applied, the renderer generates the necessary G-Buffers during the rendering process and provides the OpenGL handles to the effect evaluator. The evaluator executes the script defining the order of shader application and return the rendered color buffer.*

In rendering effect within the fragscript effect library contains the following artifacts:

Effect Description JSON formatted files containing information about:

- Effect name
- Human readable effect description
- Effect version
- Required margin: As some effect implementations expose undesired artifacts on the edges of images, the margin setting defines how wide this margin is.
- Required input G-Buffer types
- Parameters for configuring the effect from outside the styling framework. For each parameter the following properties are provided:
  - Name
  - Human readable parameter description
  - Data type
  - Value range
  - External manipulable flag (for disabling external exposure of parameters)
- Presets: Predefined named parameter settings. A name for the default preset can be provided.

Fragment Shaders Text files containing valid fragment shader code or functions according to the *OpenGL Shader Language (GLSL)*. The files are preprocessed to assemble the final shader code for single rendering pass. Their file names are referenced within the effect implementation.

Effect Script Implementation of the image-based rendering effect provided as Lua script. The implementation defines the order of application of the fragment shaders. It configures the fragment shaders using the parameters defined in the effect description.

The effect evaluator runs the Lua script provided with each effect. It creates the necessary OpenGL shaders and variables, binds the input buffers provided by the G-Buffer generator of the 3D rendering service, initializes and binds the output buffer, and executes the effect shaders in the given order. An example of an image-based styling is provided in Fig. 5.9.

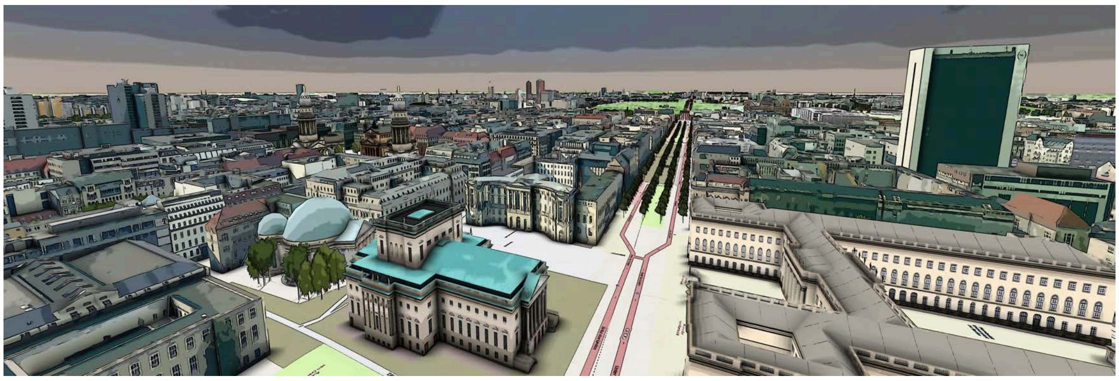


Figure 5.9: *An example of image-based styling applying multiple image styles (a non-photorealistic rendering style combined with screen-space directional occlusion) being applied on a view of the virtual 3D city model of Berlin. Although there is a large number of features portrayed in the image, the execution time for the rendering effect remains in an equal range since execution times only depend on image resolution instead of number of features portrayed.*

Chapter 6

Layered Map Service for Virtual 3D City Models

In this chapter, a service for layered maps and its application for web and mobile applications is introduced. The service enables map authors to export virtual 3D city model contents by means of a stylized, tiled map with oblique 3D view (also called “layered map” or “bird’s eye view map”) on the virtual 3D city model and allows for combining these views with additional information layers. For each use case, an application dataset can be configured and generated that includes, e.g., in particular, model layers and their styling (see Section 5.6), terrain textures, client-side points-of-interests, or feature geometry.

6.1 Motivation and Concept

In common 3D geovisualization systems and services (e.g., Google Earth, Cesium[1]), portrayal of large-scale virtual 3d city models usually requires powerful client applications that implement scene-based rendering for streamed 3D geometry and textures. These applications offer a high degree of interactivity and freedom in 3D navigation. However, this approach also leads, e.g., to long initial loading times and large amounts of data to be transferred before the application is ready to be used. Additionally, the rendering quality and the amount of data that is visible at once depend on the client’s computing and 3D-rendering resources. Consequently, different end-user devices may exhibit very different user experience regarding loading times, interactivity, visual quality, or amount of data visible at once.

The layered map approach presented in this chapter uses oblique 3D views for the provisioning of virtual 3D city models. This way, it addresses these challenges and enables lightweight applications for virtual 3D city models that provide short initial loading times.

The general idea behind the concept is to generate artificial oblique image tiles using the 3D rendering service introduced earlier. These tiles depict the data of the virtual 3D city model for given scene content and styles that are authored especially for the purpose of the application to be built. Several layers of information can be combined in one map, i.e., specifically designed tile layers can augment or overlay parts of the map, e.g., to add planning variants of real estate developments to a map. Each map tile can be identified by the layer id – identifying the scene content and styling, the tile coordinates ($coord_{tile} = \{row, column, zoom\_level\}$), and the map heading in degrees.

[1] `https://cesiumjs.org`

By providing the virtual 3D city model as rendered image tiles, the complexity of data to be transmitted to clients is reduced to image files (compressed representations of bitmaps) that are displayed by lightweight client applications based on standard 2D mapping frameworks. Such applications are small in code size, require only very limited client resources, and are easy to use. This makes applications load faster and provides a low entry barrier to virtual 3D city model applications based on this solution. Due to the small data and code sizes to be initially transferred to clients, the 3D layered map provides an easy and fast access to visual representations of virtual 3D city models.

6.2 Generation Process

The key goal of the layered map service is to synthesize tiled, oblique views for a given virtual 3D city model and scene configuration. The views, generated once, can be stored, distributed, and used similar to classical 2D map tiles. In particular, the complexity related to processing, management, and rendering of virtual 3D city models from a client's perspective is massively reduced. Both, content and styling, can be configured based on the 3DPS capabilities to match the requirements of the corresponding application. This can also include thematic aspects when thematic styling is applied during rendering (see Fig. 6.1).

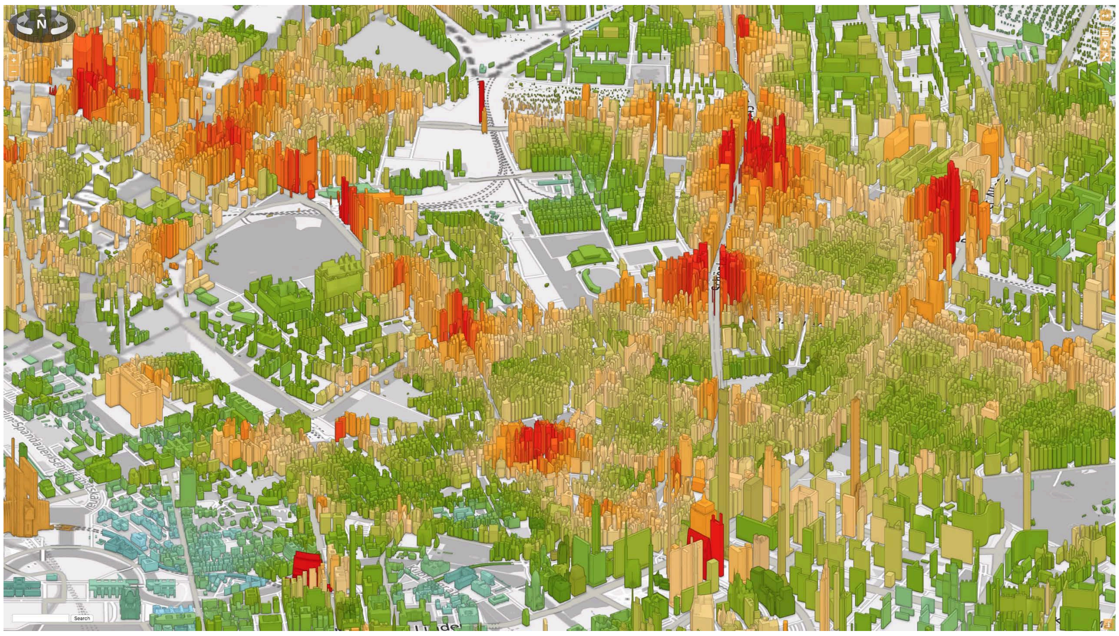
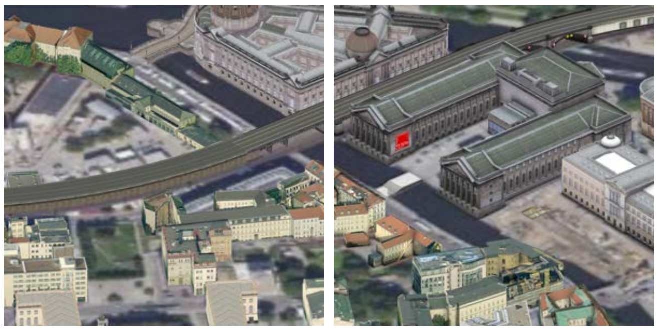


Figure 6.1: *Example of a web-based layered map of the city of Berlin with thematic styling applied. Each depicted building is assigned a specific value for its public transport reachability index (mapped to a color scale) and the number of young people living in that area.*

The process of generating such layered maps for a given virtual 3D city models can be completely automated, e.g., to provide automated updates to packaged application data if contents change (e.g., in case of city model updates, or additions of planning variants to be included in an application). To define , the geographical space of a virtual 3D city model is subdivided into a regular grid of tiles at different zoom levels in analogy to existing tiling schemes such as TMS [OSG] or OGC WMS-T. Each tile has a unique


(a) *Color Image Layer*

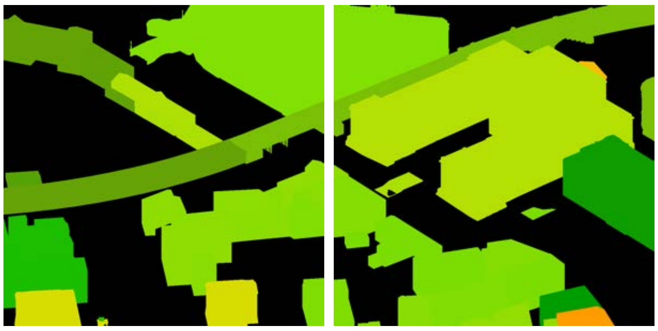

(b) *Object ID Image Layer*

Figure 6.2: *Example of two different layers of image data in two adjacent tiles generated for the virtual 3D city model of Berlin using South to North viewing direction with an viewing angle of 30°. While tiles in sub figure (a) are used for user display, tiles shown in sub figure (b) contain color encoded object IDs.*

key that is composed of zoom level and grid indices; each tile covers a spatial extent that can be computed using its key.

Oblique views for each layer are generated for different viewing directions (e.g., the four principal directions) with supports the step-wise rotation of a view by clients by exchanging the underlying tile sets as illustrated in Fig. 6.2.

To change the contents that are used in an application using a layered map, the application configuration is modified and the build process for the datasets is triggered again. Map contents include, e.g., points of interest (POI) and 2D shapes (e.g., named roads, or polygons describing areas of interest). Additional image layers can be packed and superimposed onto a map. In general, features depicted in image tiles can be linked to layered map on a per-pixel basis, enabling feature data retrieval or client-side attribute-based styling implemented using image-based 2D rendering APIs, e.g., HTML5 canvas in case of browser-based applications. For this, an object ID layer is also transferred to clients, which will not be visible to users, but which is used as background data for rendering and feature identification. This way, client applications can assign a feature to each image pixel displayed. If there is feature data available, clients can use the attribute information together with the object ID layer to implement a simple client-side thematic visualization, e.g., by mapping these feature attributes to visual variables, such as color value or its saturation.

The layered map service has been applied in different application scenarios ranging from real estate marketing, city planning, to cultural heritage visualization [Kli+14].

6.3 Architecture

The architecture for layered map applications (see Fig. 6.3) focuses on efficient distribution of image and feature data to a possibly large number of clients. Here the *layered map service* provides the central component as it encapsulates the image tile generation, i.e., the computation of scene content, styling, and projection parameters from tile coordinates and layer ids. The layered map service allows for generation of complete image tile datasets as named layers for applications that can easily be served by *application data providers* that provide the same `GetTile` operation but serve tiles from pregenerated image datasets only. This way, clients can be configured to either use pregenerated tiles from application data providers or live-rendered tiles from a layered map service instance.

Tile dataset generation is initiated by the *configuration service* using the `CreateTiles` service operation which generates tiles for specific map layers (i.e., 3DPS `GetView` parameter sets) for given spatial extents and map orientations.

Feature data, i.e, POI data, is prepared and optimized for applications by the *feature data provisioning service*. Clients fetch the POI layers and connected information using the *GetPOIs* operation of an application data provider.

Application data providers are generally very lightweight components that scale well, e.g., lightweight service implementation that encapsulate keyed data stores such as cloud storage services (e.g., amazon S3) or file system based stores. This allows the approach to scale to a large number of concurrent users fetching tiles in parallel, since scaling the network-based delivery of files to client applications is a solved problem, e.g., in cloud-based environments [SI10; WHP16]. The same applies for application optimized POI and feature datasets.

Client applications provide the user interface for applications using virtual 3D city models based on layered maps. They are implemented for Android, iOS, and web browsers. Clients can access *runtime external data providers* to provide specific business functionality or to provide access to dynamic data, i.e., data that changes frequently.

6.4 Client Applications

A minimalist implementation of a client application for layered maps does only require capabilities to fetch tiles from a service and display these images. For that reason, these applications tend to be very lightweight from a software development's point of view. Such client applications can be easily implemented on mobile platforms (i.e., smartphones or tablets) or on web browsers.

Their design includes a map-like user interface (also called *Slippy Map* in OpenStreetMap terminology) used to explore the virtual 3D city model by shifting the map center, changing zoom levels and rotating the view. Many tasks, such as exploring a specific area, do not require full 3D camera navigation, which forces users to handle six degrees of freedom.

Users can adapt displayed contents by configuring the image, POI, and feature layers to be displayed on the map. This way, e.g., different variants of an area, either having different 3DPS layers or points of interest selected or feature geometry displayed, can be selected and presented to users in a robust way with low latency. Users can further interact with the feature items on the map, since to additional information (i.e., detailed

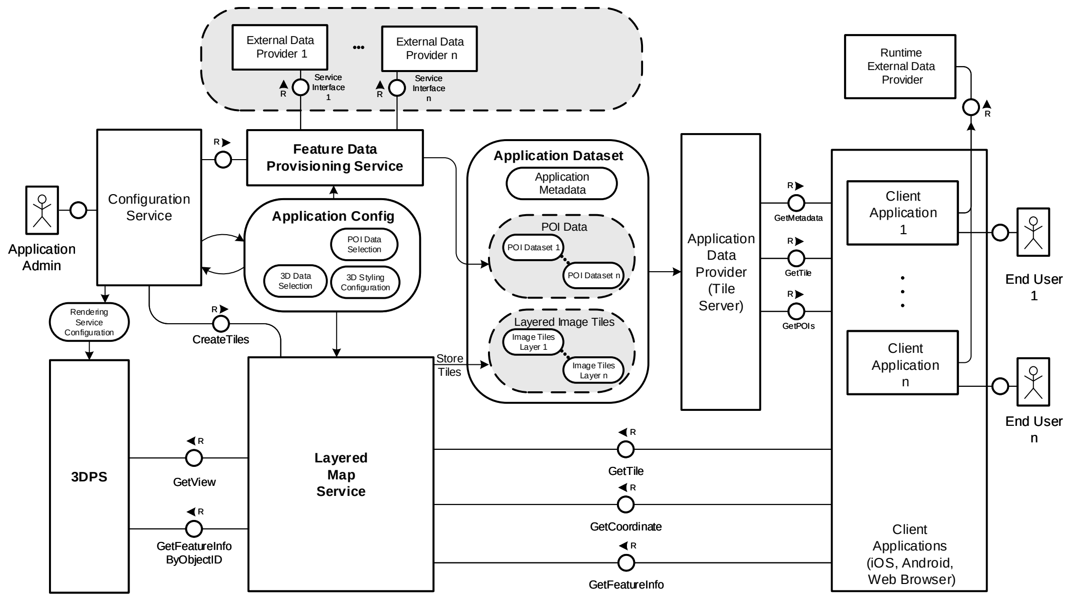


Figure 6.3: *Overview of the architecture for layered map applications. The layered map service generates image tiles by using 3DPS GetView operations and provides access to 3DPS feature data and picking functionality. Clients can either access prepared application datasets or use live rendering provided by the layered map service using an GetTile operation.*

descriptions, web links, contact information, tags, and categories) can be contained in the packaged POI and feature data format. Client applications can use the features of the target platform, e.g., map-popups, information overlays, in-app dialogs, or external web-browsers, to display the content provided or to forward to the URL provided in the POI data, where additional functionality, e.g., commenting, for the corresponding item can be implemented. This way, additional services can be connected to the map view that implement starting points for business processes related to the objects displayed on the layered map.

For retrieval of image tiles, client applications use the HTTP-based service interface. Requests contain the layer name, the tile coordinate and the view orientation. The same service interface is provided by tile servers, which provide tiles that were previously rendered by the layered map service, and by the layered map service itself, which is also provided by the layered map service for rendering tiles using a 3DPS. This way, client applications can also include image layers that are generated on the fly by 3DPS instances without having to change the client implementation. Live rendering of image tiles allows users to modify content selection and styling, and to include up-to-date data in their layered map applications.

Combination of Map Layers Using Information Lenses

Client applications can use interactive information lenses to present a second map configuration, e.g., with different scene contents or additional information layers, for a specific area of interest at the same time. Such lenses provide provides a focus & context

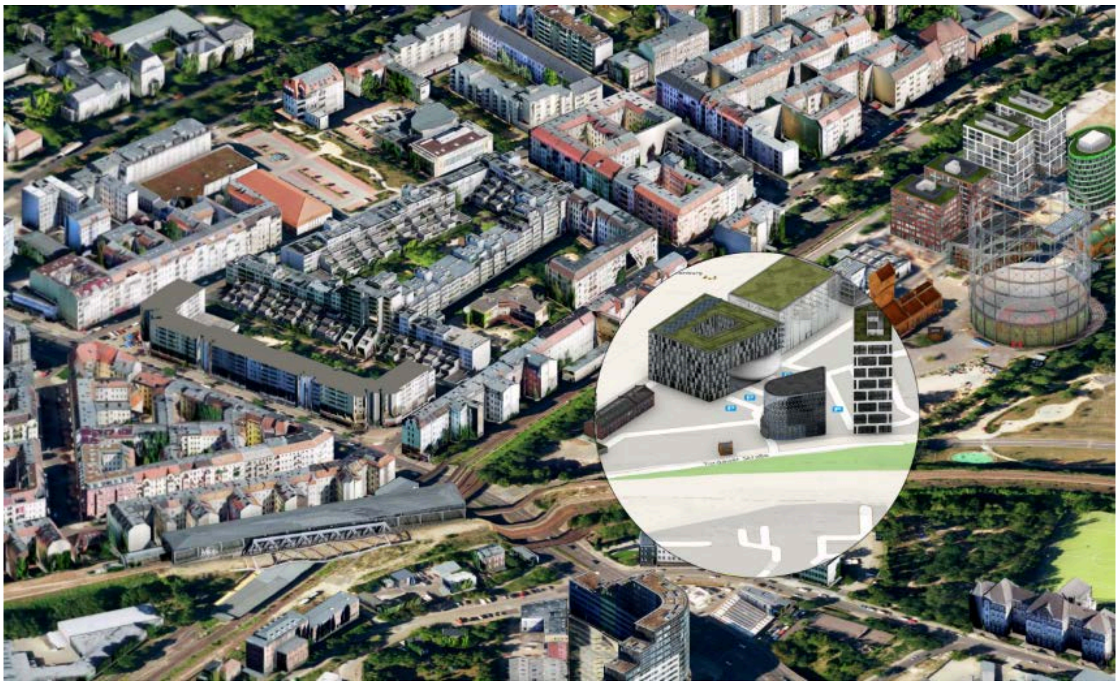


Figure 6.4: *Example of a web-based layered map using an information lens to combine two map layers in one view. The context layer shows the textured virtual 3D city model and an orthophoto as terrain texture. The focus map is configured to show a different configuration, i.e., a planning model and a street map as terrain texture.*

mechanism for use with virtual 3D city models. This way, users can explore and connect multiple content layers selectively without manually switching the overall map content. Here, a region of the map, e.g., a circle around a point of interest or the current cursor position, shows an alternative scene configuration, e.g., a planning model (Fig. 6.4 shows an example).

The implementation of lenses relies on blending of rendered layered maps where the content of the context map (displayed outside of the lens area) can either be replaced or combined with the content of the focus map (inside the information lens area) on client side. Here, the different information layers such as object IDs can be used to influence the blending result, e.g., to apply the blending effects only to specific objects in the virtual 3D city model. As 2D rendering and blending is supported well by all target platforms, information lenses can be rendered efficiently also on lightweight end-user devices.

6.5 Cloud-Based Deployment

With the increasing availability of high performance GPU-supported hardware, also in major virtualized cloud-based environments (e.g., Amazon AWS, Microsoft Azure, or Google Cloud), it becomes feasible to scale rendering infrastructures for image generation for virtual 3D city models on demand. Most notably, pre-configured 3D rendering service instances (e.g., based on container technology like docker [Nic16]) are easy to deploy and to instantiate since they encapsulate the complexity of heterogeneous operating systems and hardware configurations. The major challenge for the cloud-based deployment of rendering infrastructure is the data distribution since rendering instances need to have access to large-scale dataset in a way that allows to access data very fast. By using, e.g., in memory caches, as already implemented in the the presented 3D rendering service implementation, the overhead implied through remote fetching of geometry and texture data can be minimized. Nevertheless, it requires a certain amount of time until a 3D rendering service instance becomes ready within a cloud environment.

Chapter 7

Video Service for Virtual 3D City Models

In this chapter, design and implementation of a video service that synthesizes videos for virtual 3D city models based on a high-level description of a camera animation is introduced. For a given service request, containing a video description, the service generates a video file encoded in a standard video format. The video service also provides a map-based front-end that allows users to configure the camera-related parameters, the data to render and its styling, as well as the composition and timing of additional video elements. It applies concepts and techniques that have been presented earlier in this thesis to enable users to create videos with a minimal set of client-side hardware requirements.

7.1 Motivation

Common visualization applications for virtual 3D city models offer a rich set of software features, e.g., to interactively view, explore and analyze geospatial features, but they also require corresponding hardware and software resources and, most importantly, the 3D model data to be available on client side for 3D rendering. However, for many application scenarios, a lightweight, "read-only" visual access to virtual 3D city models is required and sufficient. Videos as key visual media type, hence, represent an alternative way for provisioning of virtual 3D city models. Editing, 3D rendering and encoding of videos generated from virtual 3D city models typically involves a lot of manual work that makes video production a relatively expensive, slow, and not very efficiently repeatable process. For example, videos depicting urban development projects are often generated on a contract basis by professional graphics agencies. For that reason, the proposed video service aims at automating many of the steps of the video generation process. The characteristics of videos as medium for communication of virtual 3D city models include:

Low Requirements Video delivery and playback does not require any special hardware or software on the client side since they are commonly supported by all devices and by a variety of different video players.

Ease of Use Video usage and handling is straightforward compared to interactive applications for virtual 3D city models as they do not require users to get into the details of 3D navigation and 3D interaction techniques.

Standardized Formats Video encoding is based on standardized video formats, which are supported by almost all end-user devices, systems, and platforms.

Efficient Management and Distribution Videos are easy to store, manage, and distribute, e.g., using common video portals or content management systems for digital media.

The following exemplary application scenarios for the video service have been identified that benefit from video-based provisioning of virtual 3D city models:

AS.1 Videos for Project Communication Videos can be used as content for communication of urban planning projects to different groups of stakeholders. Automating video production facilitates creating individualized, up-to-date variants for different stakeholders (e.g., versions for the general public, possible investors, administration, politics).

AS.2 Videos as Documentation of Decision Processes Videos serve as documentation artifacts about discussions and decision finding processes. Textual annotations and object highlighting can be used to enhance the contents of videos.

AS.3 Videos for Location Finding Videos can be used to produce visual media that communicates how and where to find locations in a city. For example, these videos could complement the web information of an event website.

AS.4 Videos for Operation Briefings Videos can be used for briefing operations for events, e.g., demonstrations or festivals. Here, videos could be used to demonstrate possible scenarios to communicate the spatial situation in the operation area.

Requirements

The following requirements for the web-based video editing and generation have been identified:

R.1 Repeatability Virtual 3D city models tend to change over time. To keep related video contents up to date, it should be possible to regenerate videos from a video description automatically, i.e., without additional manual effort.

R.2 Content Integration The service is supposed to include contents from different sources into the visualization that is recorded as video, e.g., virtual 3D city models, architectural 3D models, results of simulations or analysis, or points of interest.

R.3 High-Quality Rendering High-quality rendering and previews during video editing should be available regardless of the virtual 3D city model's size and complexity. The hardware and software used by the video service front-end should not have any impact on the visual quality of the rendered videos.

R.4 Non-Expert User Interface The video editor front-end should be designed towards users that are not specially trained in computer animations. It should provide assistance to users to achieve good results already with a minimum of effort.

R.5 Cross-Platform Compatibility The video service front-end should run across all platforms and devices, e.g., in a standard browser environment.

R.6 Robustness The video generation should be robust regarding weaknesses in 3D camera paths designed by the users. The system should automatically "heal" or support users to fix quality issues, e.g., regarding discontinuous changes in the camera's position, orientation or velocity.

R.7 Scalability A video generation system should be able a) to efficiently use rendering resources, such as GPU hardware, and b) to distribute workloads to multiple computing nodes for horizontal scaling. The factors that must be regarded concerning the performance for video generation are:

a) **User Count** The number of users that edit or render videos in parallel defines the required server resources, e.g., for processing video rendering request or rendering of preview images. The system implementation should be able to cope with growing user numbers and provide a nearly constant response time, especially for preview images since they are required to provide an interactive user experience.

b) **Model Data Size and Complexity** The video generation should be based on a 3D rendering service that is able to handle models of nearly arbitrary size and complexity. Further, the data quality and precision should not influence on the video generation process.

c) **Number of Generated Videos** Video generation is a very resource-intensive process. Hence, a video rendering service should be able to scale horizontally (also know as *scale-out*) [Mic+07] to allow videos to be rendered in parallel by adding additional rendering nodes. Video files require major storage capacity. Therefore, storage systems that are used for generated videos should be extendable to store large volumes of generated videos.

7.2 Concept

In this section, a novel approach is presented that applies service-oriented concepts to video editing and generation from virtual 3D city models. The general idea is to decompose a video authoring process (see Section 7.2.1) into components that can be implemented by independent services that provide a well defined service interface. For the specification of videos, a video description model is developed that covers the aspects of video generation, i.e., structure, timing, scene content and styling, encoding properties, etc. To this end, a video service system has been designed that consists of three major parts (Fig. 7.3):

Video Editor Front-End: A browser-based *video editor front-end* that allows for editing video specifications.

Video Editor Back-End: A web-based *video editor back-end* that provides user management and project management as well as database access and persistence functionality.

Video Generation Service: A *video generation service* that generates, composes, and encodes the frames and additional graphics elements (e.g., titles, text descriptions, overlays) as defined by the video description.

The video editor front-end allows expert users as well as non-expert users to specify and generate videos of virtual 3D city models based on 3D camera paths and using different information layers regarding 2D and 3D geodata. The resulting videos are assembled from one or more video scenes whose contents (i.e., 3D models, terrain textures, rendered POIs) and corresponding rendering styles (i.e., applied rendering technique, highlights, image postprocessing) can be configured individually. The user interface for camera path definition is primarily based on 2D maps for authoring camera paths (Fig. 7.10), which are specified by using 3D control points of BSpline curves. While the map-based portion of the video editor front-end supports the definition of 2D curves for camera positions with corresponding points of interest for camera animation, the 3D component (heights of point of camera POC and point of interest POI) of the camera path components can be configured using a lightweight image-based 3D client. For this, preview panoramas are generated by a 3DPS. The service requests are generated using the settings from the scene and camera configuration at a selected control point of the overall camera path. Through combining camera paths on the map, fine-tuning of the camera's height and online previews, the video editor front-end provides the means to configure complex 3D camera paths and video scenes that are rendered as videos by the video generation service.

The result of each video editing process is a video description document, which is structured according to the video model described in Section 7.4. This model supports a broad range of video effects and time-dependent interpolation descriptions of differently typed parameters (e.g., vector positions, vector directions, colors, or number parameters).

7.2.1 Automating the Video Generation Process

Commonly, creating videos of virtual 3D city models involves two manual steps (Fig. 7.1):

Fly-Through Creation Creation of single video sequences each showing a camera fly-through in a virtual 3D city model using a conventional 3D visualization application, e.g., Google Earth that renders screencasts of live camera navigation or predefined camera paths, often based on key-frames.

Video Postproduction Complex video sequences require manual work for postprocessing videos to achieve a visually appealing result since complex video sequences usually contain image or text overlays, blending effects between difference scene content, or specialized techniques for image styling. These steps are usually performed using standard video authoring software that implements the compositing and final encoding of the video sequence, e.g., Adobe Premiere.

Both steps require a specially trained user to operate the tools for 3D city model visualization, video generation and postprocessing. Video authors are required to cope with 3D camera animation, mechanisms for scene configuration, styling and with video editing tools. Hence, users need a special training for those specialized tools, limiting the user group that is able to actually perform these tasks. Further, video editing tools

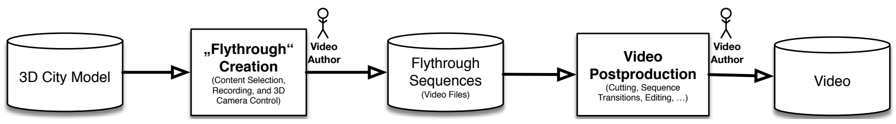


Figure 7.1: *The manual video generation process involves fly-through creation and video post-production. A repeated creation of videos is commonly as expensive as its first generation as the parts of the video scene need to be recreated and recombined in video postproduction.*

as well as 3D visualization applications require powerful hardware to render large-scale 3D city models and to efficiently edit, process, and encode high resolution videos. Many times, licenses are required for commercial solutions for each of the tools used in the process. This also involves licensing fees that increase costs for video generation.

To address these issues, the proposed video service provides a different workflow for video authoring with the following two steps:

Web-Based Video Editing A video author creates a description document that specifies the video to be generated. The description is stored persistently.

Video Generation Video generation that performs frame rendering, video compositing, and postprocessing. The input for the video generation step is an instance of a video description model (see Section 7.4) that includes all information that is necessary to generate a video sequence from a virtual 3D city model using 3DPSs (Fig. 7.2).

Hence, the video generation itself (frame rendering, compositing, video encoding, etc.) does not require any manual steps. As a further major advantage of the approach, a video description can be rendered automatically multiple times, e.g., for different resolutions, variants, or if the underlying virtual 3D city model changed.

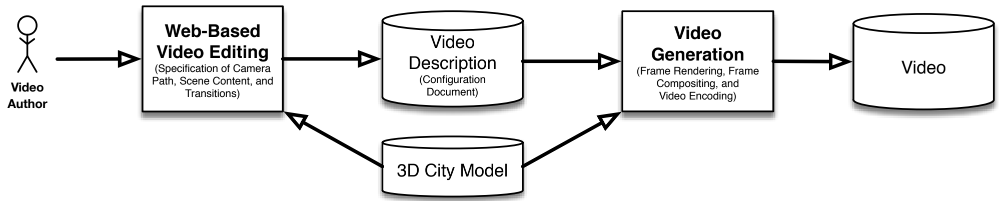

Video Author
Web-Based Video Editing
(Specification of Camera Path, Scene Content, and Transitions)
Video Description
(Configuration Document)
Video Generation
(Frame Rendering, Frame Compositing, and Video Encoding)
Video
3D City Model

Figure 7.2: *The automated video generation process using the video service. With the editor component, users specify the parameters for video generation such as camera position and orientation over time (i.e., the camera path), the scene configuration, transitions between scene configurations, and overlays. Instead of creating intermediate video scenes and doing the combination in a separate manual postproduction step (Fig. 7.1), the final videos are rendered automatically from the configured virtual 3D city model. A repeated generation of videos or variants with different scene configurations, e.g., additional planning models, can be automated as well.*

A video description, which is created by the video editor front-end, is stored in a database. Due to this explicit representation of video specifications, videos can be incrementally refined. Additionally, a collaborative editing of videos by different stakeholders or contributors is possible.

To generate the actual video file, the *video generation service* implements interpolation of camera paths as well as scene contents, generation of images and their compositing to video frames. The process is executed by a service-based system in server environments providing sufficient hardware for 3D rendering of large-scale 3D city models (CPU, GPU, main memory, hard disk, and network resources) as well as video composition and encoding. In contrast, video descriptions can be edited on the client side without any special requirements regarding 3D hardware or software resources, since 3D portrayal services are used for 3D rendering.

7.2.2 Video Editing and Generation Stages

Technically, the automated video generation process requires the following six stages to be implemented for the service-based video editing and generation:

Editing A user specifies camera path and camera velocity, the scene configurations (data layers and styling), and the transitions between different scene configurations. Video descriptions are the output of that stage.

Interpolation and Request Generation Camera position, camera orientation, and scene contents are interpolated to match the specific time index of a video frame to be rendered; the results are stored by means of 3DPS requests.

Video Frame Rendering For each video frame, images of the virtual 3D city model (generated by 3DPS requests) and possibly additional images containing the text and image overlays are generated.

Image Compositing The final video frame is composed of one or multiple image layers.

Encoding The composite frames are encoded into a video stream or video file using a video codec, e.g., H.264 [ISO04].

Publishing and Distribution Encoded videos can be distributed to web sites for downloading or viewing. In most cases, video platforms will be used for these distribution tasks.

7.3 System Architecture

The system design separates video editing, video generation, and 3D rendering into distinct services that operate independently. Fig. 7.3 shows an overview of the system architecture. The principal components of the architecture include:

Video Editor The *video editor* is a web application for creating and managing video projects, in particular authoring of video descriptions, management of the video generation process, user management and authentication, and video publication. It is implemented as Ruby on Rails [RCT17] web-application using a HTML5 standard-based video editor front-end implemented in TypeScript [Roz15]. It contains two major parts:

Video Editor Front-End Users can edit camera paths, scene contents, styling, video timings (e.g., scene order, transition effects) as well as video postprocessing effects (e.g., image or text overlays, etc.). Further, it provides a preview of a scene for each camera position as well as first video previews, generated by the associated 3DPS.

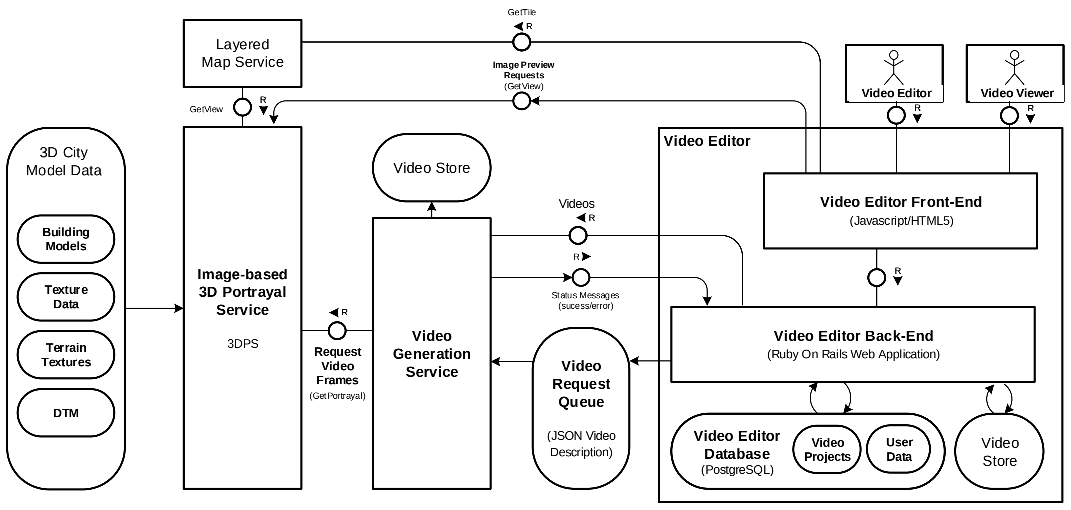


Figure 7.3: *The architecture of the video service and its three main components: A web-based video editor providing the user interface and data management for editing and distribution of videos. A video generation service assembles single images, retrieved from one or multiple 3D portrayal service instances. The 3D portrayal service encapsulates image synthesis for virtual 3D city models.*

Video Editor Back-End The video editor back-end provides API endpoints used by the front-end. It implements management of projects, user data, video data, and video jobs.

Video Editor Database Data persistence is implemented using a PostgreSQL database based on a relational database model (*video editor database*). It also copes with storage and access of rendered video files in its *video store* (e.g., a local file store or a cloud-based storage).

Video Request Queue Users can create and edit video projects and submit the corresponding video descriptions (*video jobs*) into the *video request queue*, implemented by the `createVideo` service operation (see Section 7.5.2).

Video Generation Service The *video generation service* synthesizes videos from video descriptions by interpolation of definitions for single video frames. Source images are rendered using a 3DPS instance, combined into video frames, and encoded into video files. It is split into two components a) a NodeJS[1]-based web application providing the service endpoint, request checks and process control and b) a C++ application that implements video frame retrieval, composition, and encoding of the final video file. The service is described in detail in Section 7.5.

3DPS 3DPS instances are used to render video frames. They encapsulate rendering and management of virtual 3D city models. Multiple 3DPS instances can be combined, e.g., to integrate different geodata infrastructures based on provided 3DPS.

[1]`https://nodejs.org`

7.4 A Data Model for Video Authoring and Generation

For the description of videos to generate, a data model was designed that defines the entities that configure the displayed scenes (i.e., selected models and their styling, position and orientation of the camera, etc.), the timing and compositing of videos from single frames. To address the requirements in Section 7.1, this description should support features such as:

- Visual editing and composition of video descriptions
- Persistent storage and management of videos
- Generation of video files from virtual 3D city models using a video generation service.

The following aspects of a data model for videos have been identified:

Time Modeling Videos from virtual 3D city models can consist of different scenes, e.g., when exploring two different locations. Therefore, a time-wise segmentation of the overall video is necessary. Scene contents are defined for particular time ranges, i.e., which content is displayed in which time range and how it is composed from different image sources. This information allows video generation services to compute the required service parameters for every time index within the video, e.g., projection parameters or parameters that influence scene styling.

Scene and Style Configuration The 3D contents and styling for a video sequence (or its parts) define which features or model elements are included in the visualization and how they are styled. Further, means for configuring the image-based styling of the rendered video frames need to be provided, i.e., image-based rendering effects as well as highlighted and other rendering service parameters can be configured to provide a wide range of creative means for definition of video scene content. So, the video model must be capable of defining those parameters and their interpolation over time.

Video Transitions For visually appealing video sequences, e.g, including transition effects between scene configurations and to avoid abrupt changes of scene content, model features for describing scene transitions (e.g., source and target scene contents, the time range of the transition, the transition type, and the used transition technique) are necessary.

Image Sources and Image Generation Different types of images can be used for rendering a video frame (see Section 7.5.3 “ Image Sources”). They can be generated, e.g., by a remote 3D portrayal service, a local (in-process) 3D portrayal service, a text renderer (e.g., for text overlays), static image files, remote image files, or 2D renderers (e.g., for 2D vector graphics). The model needs to define which image sources, endpoints, and configurations should be used for a specific time index.

Image Compositing The final video frame at any point of a video can be composed from different source images (*layers*). These image layers can be text or image overlays, rendered images of virtual 3D city models, or transition effects between

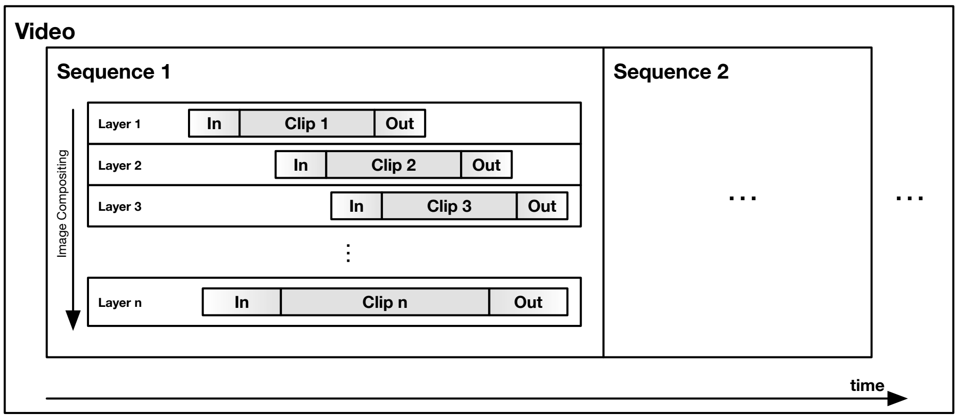


Figure 7.4: *Overview of the video model from a time, contents, and image compositing perspective. A video is divided into sequences each defining a specific range in time for the overall video. Each sequence has a notion of image layers, each containing one or more clip definitions. A video clip defines the video content for a duration (e.g., a 3D video scene, an image or font overlay, or a static color) together with definitions for type and timing for video transitioning.*

different parts of the video sequence with different content or styling. Therefore, the composition order and technique must be specified in a video description model as well.

Video Encoding Properties Finally, technical parameters, such as the file format, video encoding type, resolution, and aspect ratio, form also part of the video description.

The designed data model addresses these aspects. Here, the naming of the model elements and classes is aligned with the wording used for existing desktop-based solutions for video authoring, such as de facto standard applications such as Adobe Premiere or Apple iMovie. There are three dimensions to consider when structuring a video:

- Time
- Content
- Image Compositing

The general structure of a video regarding these dimensions is illustrated in Fig. 7.4. While a *video* contains information about the video encoding and other parameters that are constant throughout the complete video, *sequences* create a partitioning of an overall video into scenes of a specific duration that can be described independently. Here, *clips* describe a specific scene content to be portrayed during a time period within a video sequence. Multiple clips can cover the same time span. They are arranged as *layers* that describe the order of frame compositing. In general, properties that are defined for parent entities, such as the video video or specific sequences are inherited to the lower level entities. Properties can be redefined in these lower level entities, e.g., video sequence properties can overwrite properties defined for the video. This reduces the complexity and the size of model instances since most of the properties become optional in low level elements, as long as they should not be changed or interpolated for a specific part of the video.

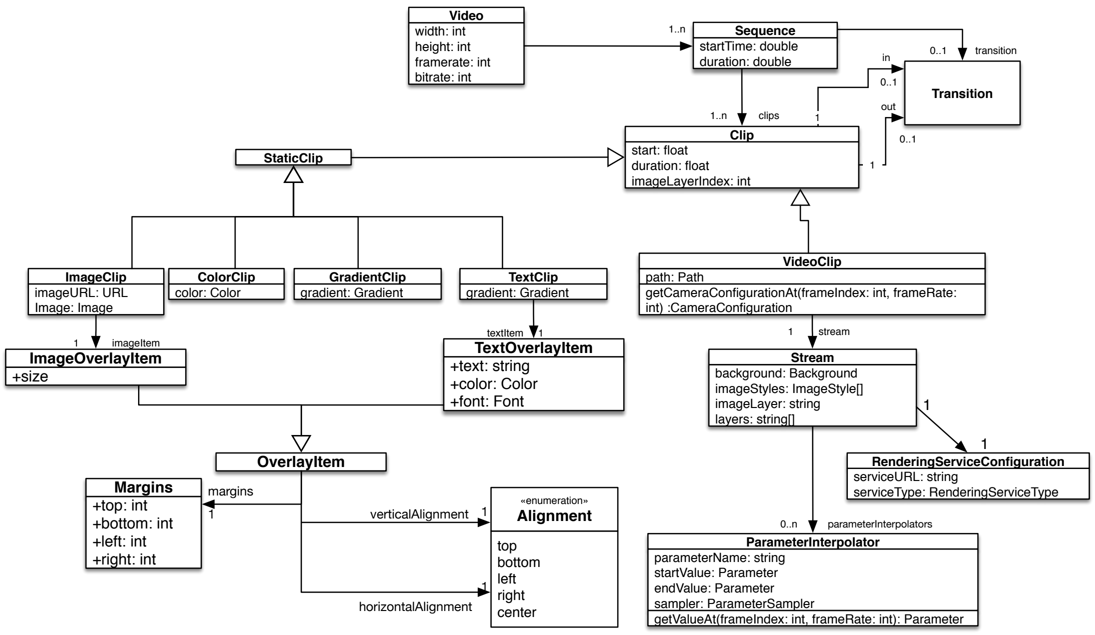


Figure 7.5: *A class diagram providing an overview of the modeling of video sequences and clips.*

7.4.1 Scene Content and Timing

The video editor and the video generation service use the same model entities to describe videos (see Fig. 7.5). This assures consistency in data exchange and common internal structures in both components. A video is composed of *sequences* that contain *clips*. In general, a clip provides a description how to synthesize rendered frames for a specific time interval, specified as start time and duration in seconds. Clips are implicitly grouped to image layers (using *imageLayerIndex*) defining the order for frame composition and blending. In general, there are two subclasses of *Clip*:

StaticClip providing a static content over their complete definition period (fixed image overlays, text overlays, colors, or gradients).

VideoClip require frames to be generated by interpolation of parameters, such as camera position and orientation, scene content, or styling parameters for a specific point in time.

Commonly, a video clip describes camera flights through a virtual 3D city model. The configuration of the video clip frames is encoded as *Stream* entity. Streams contain the necessary information for time dependent image generation using a 3D rendering service. They define scene contents and styling (named content layers and their selected styles, background configuration, service endpoints) as well as additional parameters that affect the portrayal result. The parameters of portrayal requests for a single frame is adjusted by interpolation over time using *ParameterInterpolators*. This way, the model provides a generic mechanism that allows for a variety of animated effects.

Assigned *transition*s connect two clips. They define the effect that is applied at the beginning (*in* transition) or the end (*out* transition) of a clip, e.g., to apply a blending effect.

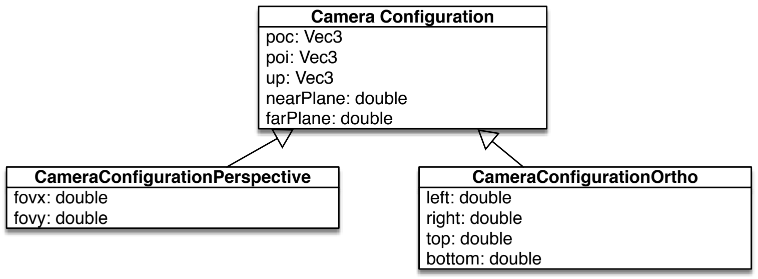


Figure 7.6: *Camera model for perspective projections and orthographic projections.*

7.4.2 Virtual Cameras and Projections

The camera projection and additional camera related parameters have to be defined to configure the properties of the virtual camera. The current implementation provides support for conventional perspective and orthographic camera projections (Fig. 7.6), specified by the point of camera (look-from position *poc*) and the point of interest (look-to position *poi*). Both points are specified in geocoordinates in a spatial reference system, usually defined on a per-path or per-video basis. Projection-dependent parameters such as camera opening angles (field of view in x and y direction *fovx* and *fovy*) for perspective projections and top, bottom, left, and right distances for view frustum planes of orthographic projections are specified in subclasses.

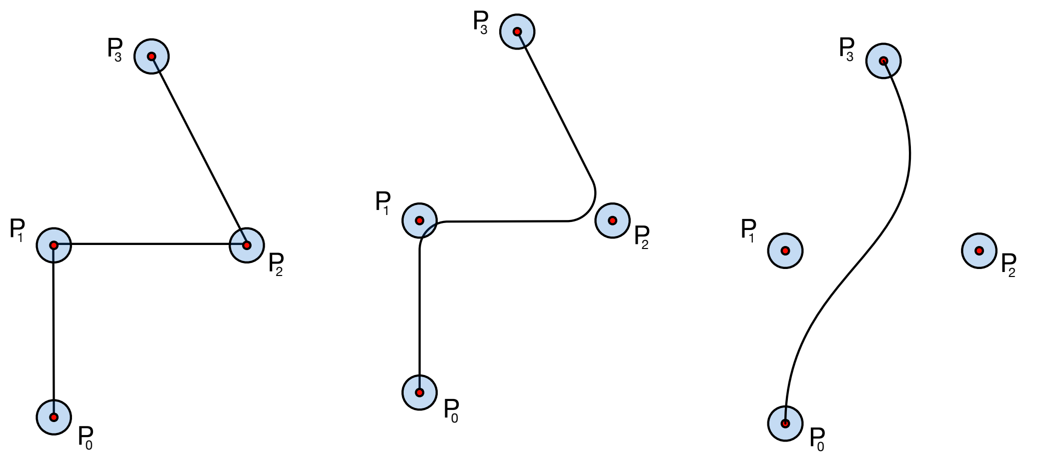
Advanced projection techniques such as multi-perspective views [AZM00; PD13; PTD11] or projections for interactive 360 degree videos can also be supported by extending the model and corresponding implementations in 3DPS that is used for video generation.

7.4.3 Camera Paths

A camera path is specified by a parameterized function used to calculate camera parameters depending on the normalized time interpolation parameter $t \in [0, 1]$. The interpolation method depends on the type of path and camera definition. There are currently three path types specified that the video generation service is able to interpolate (see Fig. 7.8 for modeling details):

Linear Keyframe Path Each component of a camera configuration gets linearly interpolated between the camera configuration defined for keyframes (Fig. 7.7(a)). While being easy to define and simple to evaluate, this approach creates notably discontinuous changes in camera position and camera orientation, especially when rapidly changing camera moving directions or camera orientations. The camera path interpolation strictly follows the polyline defined by the keyframe camera positions and orientations since no adjustment or adaptations in terms of smoothing camera paths are performed with this interpolation method.

Smoothed Linear Keyframe Path Smoothed linear keyframe paths are designed to compensate abrupt camera orientation and position changes by applying a smoothing at timestamps within a specified interval of camera configuration keyframes (Fig. 7.7(b)). The position for a time index is calculated from two points around a



(a) *Linear Keyframe Path* (b) *Smoothed Linear Keyframe Path* (c) *Beziér Path*

Figure 7.7: *Example of different types of camera paths. For a linear keyframe path given points of a camera path describe a linear connection between the given points. Smoothed linear keyframe paths apply smoothing around the control points to avoid abrupt changes of camera movement. For Beziér paths the given points are interpreted as control points of a cubic Beziér spline providing a smooth camera movement.*

path keyframe: 1) The linear interpolation at the current time index on the polyline given by the keyframes and 2) an interpolated position on a future time index. The final interpolated position on the path is then given by the center between these two points. This way, camera positions around keyframes are following a quadratic Bézier curve, which generates smoother transitions of camera direction and orientation changes compared to a pure linear interpolation.

Beziér Path Beziér curves provide intuitive means to design and describe continuous and smooth curves. The Beziér path is based on interpolating cubic Bézier splines to minimize discontinuous camera animations (Fig. 7.7(c)) . Four adjacent points of a path are interpreted as control points of a cubic Bézier curve, which constitute a single curve segment. For adjacent curve segments, the last control point of the prior curve segment is equal to the first control point of the following segment. This way, a continuous transition between curve segments is established.

Concepts for Simplified Path Modeling

Detailed specification of camera animations commonly requires a significant amount of manual adjustment and optimization to create a smooth camera animation, i.e., to prevent unsteady changes in camera orientation (e.g., sharp turns), camera velocity (e.g., sudden or strong acceleration or deceleration), or camera movement direction. To simplify the specification from a user's perspective, there are two basic concepts implemented in the model specification and in the overall video service system.

Presets Most properties are optional to allow users to define usable path descriptions with a minimum of properties set. Parameters are either provided as defaults in the implementation, taken transitively from parent elements, or calculated from given properties, e.g., a default field of view angle in y direction that was computed from the given field of view in x direction and the aspect ratio of the video file. Further, there is a temporal relation between camera path specifications. If previous camera paths define parameters, these parameters are kept as defaults until they are overridden by setting the parameter in later path specifications.

Gaze Assistance Second, the model provides an option that allows the video editor and video generation service to automatically derive camera points of interest from camera position path definitions. The `lookInPathDirection` flag causes the camera point of interest for the current frame poi_n to follow the path direction by setting the poi for frame n using $poi_n = poc_{n+1}$. In case of the last frame with index m of a path segment, the direction vector for frame n $\vec{v}_n = poi_n - poc_n$ of the last i frames is averaged and applied to the last point of camera poc_m:

$$poi_m = poc_m + \frac{\sum\limits_{j=m-i}^{m-1} \vec{v}_j}{m-i-1} \text{ for } i < m \text{ and } i > 1$$

This way, paths can be specified by providing a path for the camera position only. The other parameters can be derived using defaults or calculated from the given camera positions, providing a plausible first path that can be incrementally refined by users. So the model is able to cover both: simplicity for path specification and the possibility to create more fine tuned camera paths.

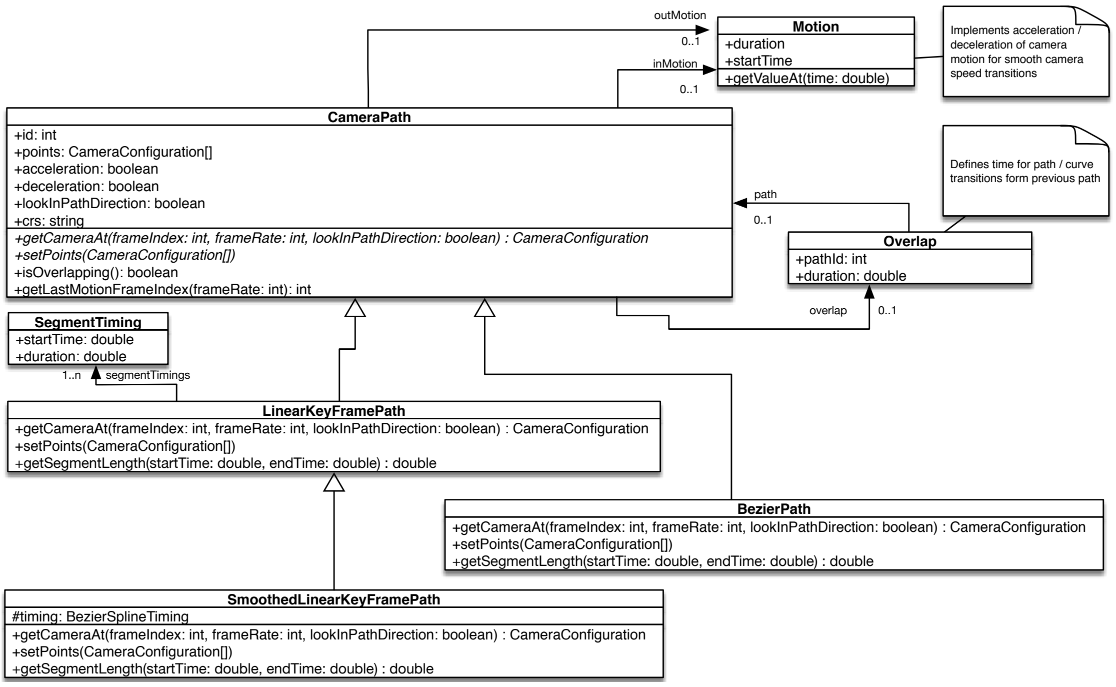

outMotion
0..1
inMotion
0..1
Motion
+duration
+startTime
+getValueAt(time: double)
Implements acceleration / deceleration of camera motion for smooth camera speed transitions
CameraPath
+id: int
+points: CameraConfiguration[]
+acceleration: boolean
+deceleration: boolean
+lookInPathDirection: boolean
+crs: string
+getCameraAt(frameIndex: int, frameRate: int, lookInPathDirection: boolean) : CameraConfiguration
+setPoints(CameraConfiguration[])
+isOverlapping(): boolean
+getLastMotionFrameIndex(frameRate: int): int
Defines time for path / curve transitions form previous path
path
0..1
Overlap
+pathId: int
+duration: double
overlap
0..1
SegmentTiming
+startTime: double
+duration: double
1..n
segmentTimings
LinearKeyFramePath
+getCameraAt(frameIndex: int, frameRate: int, lookInPathDirection: boolean) : CameraConfiguration
+setPoints(CameraConfiguration[])
+getSegmentLength(startTime: double, endTime: double) : double
BezierPath
+getCameraAt(frameIndex: int, frameRate: int, lookInPathDirection: boolean) : CameraConfiguration
+setPoints(CameraConfiguration[])
+getSegmentLength(startTime: double, endTime: double) : double
SmoothedLinearKeyFramePath
#timing: BezierSplineTiming
+getCameraAt(frameIndex: int, frameRate: int, lookInPathDirection: boolean) : CameraConfiguration
+setPoints(CameraConfiguration[])
+getSegmentLength(startTime: double, endTime: double) : double

Figure 7.8: *Overview of path types with interpolation facilities to create smoothed paths in terms of velocity and general steadiness of motion.*

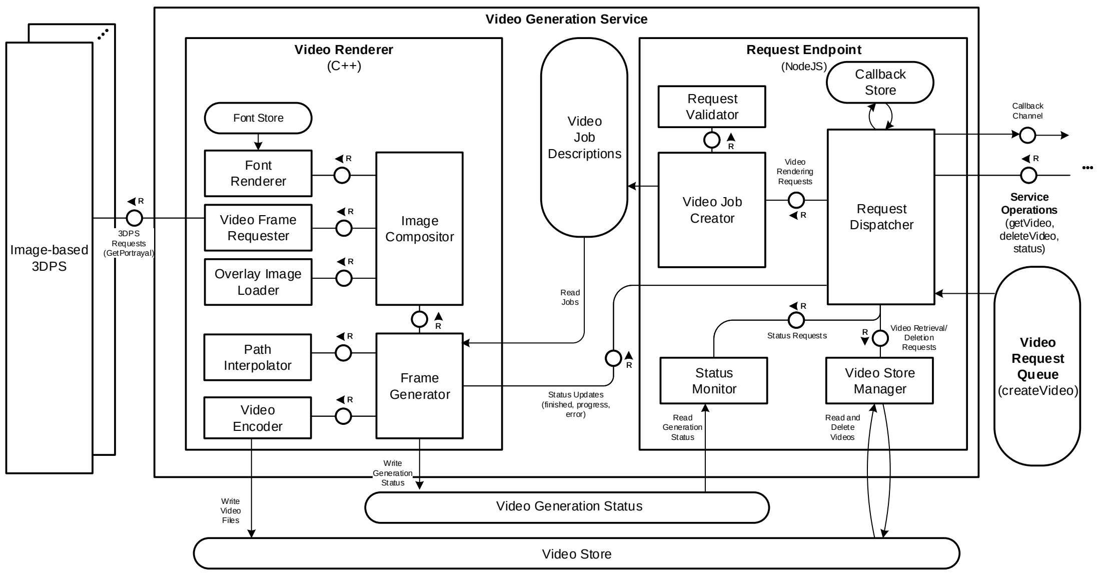


Figure 7.9: *Architecture of the video generation service.*

7.5 Video Generation Service

The video generation service processes video job descriptions based on the data model introduced in the previous sections. It interpolates camera path and scene parameters, fetches video frames from different services, composes these images to final video frames and encodes the video file. Generally, video generation of virtual 3D city model is a resource intensive process as large numbers of video frames need to be configured, synthesized, processed, composed, and encoded. The performance of rendering video frames is limited by the throughput of 3D rendering services since they have to manage limited rendering resources to cope with large-scale virtual 3D city models. To this end, it is necessary to decouple request management, video composition and encoding, and frame rendering to allow the system to scale by adding either rendering service instances or video generation service instances to the overall system (Fig. 7.9).

7.5.1 Architecture

The video generation service is implemented as two major components:

Request Endpoint The *request endpoint* service provides management functionality, e.g., to start, stop, and delete video processing jobs.

Video Renderer The *video renderer* processes video jobs, generates or retrieves video frames, decodes and composites frames, and encodes the video files.

Both components share a *video store* with rendered video files that can be retrieved by service clients, i.e., the video-editor-backend to make video files accessible by clients. A *video generation status* is written by a video renderer. It contains the current state of the video generation process (queued, started, finished, error) and the progress of the video generation for video jobs. The status is read an processed by the *status monitor*.

Request Endpoint

The video generation service request endpoint manages the job queue of the video renderer. It implements validity checks for requests and maintains status information for video generation jobs. Further, it provides access to finished video files.

The *request dispatcher* stores per-request callbacks to notify the initial caller of the service operation about events regarding the video generation process, such as generation status changes, finish of the generation process, or errors during generation.

The request dispatcher is notified of events in the video generation process, i.e., generation progress changes, completion of a generation process, and in case of errors during the generation process.

Clients can provide URL callbacks to notify or as email addresses that receive notifications. This way, the components can operate independently and event-based following the idea of micro service architectures increasing scalability and robustness of the overall system.

Video Renderer

The video renderer is implemented in C++ to improve performance characteristics and use computing resources efficiently. It implements the camera position and orientation interpolation (*path interpolator*), and frame generation from different sources (see Section 7.5.3), compositing (*image compositor*), and video encoding (*video encoder*). Depending on the video description multiple 3D portrayal services can be invoked in a single video generation process.

7.5.2 Service Interface

The video generation service provides a service interface based on JSON encoded requests. For this, the request endpoint component implements processing of these requests and dispatches them to video renderer instances for video generation. Further, it monitors the video rendering process and manages status updates. It exposes the following service operations via a REST-like[Fie00] interface.

`CreateVideo` The `CreateVideo` operation is called via HTTP POST method. It requires a JSON video description to be sent in the request body. A video job is created from the parsed request and added to the execution queue. Jobs are then processed as soon as there is a free instance of the video renderer available. The video renderer returns a video ID, which is used by the remaining requests for job management and video retrieval. On completion of the request, a given HTTP callback is called to notify submitting clients, i.e., the video editor web-application.

`Status` The `Status` request is issued using the HTTP GET method. It does not involve any changes in a database or other state changes. The operation returns the processing status of a video with a specific ID as JSON-formatted response.

`GetVideo` Implements video access and download via HTTP GET request.

`DeleteVideo` Deletes the video from the video service internal storage using a HTTP DELETE request with the video ID as parameter.

7.5.3 Image Sources

An image source delivers image data to be included into the video rendering process. There are different clip types, each describing a different content type for video frames, the images can originate from different sources, e.g.:

Remote 3DPS for rendering images of virtual 3D city models on servers and transferring encoded images over a network.

Local 3DPS are process-internal 3D rendering services integrated into the video generation service process using a C++ library interface.

Text renderers , e.g., for generating text overlays.

Static image files that are loaded from a local store identified, e.g., by an alias name in a video description or remotely hosted image files via URL.

2D graphics renderers , e.g., for rendering 2D vector graphics or maps (either service-based or integrated into video renderer).

Due to the image compositing performed by the video renderer, multiple images from different sources (external services or local renderers) can be necessary for a single video frame to be rendered. Using external services for image rendering or retrieval in the video generation process provides interoperability between different geodata infrastructures, data owners, or stakeholders. This allows us to integrate data from different sources in one video, but it also causes a major challenge: External 3DPS cause a significant performance overhead per image, e.g., for request processing, image encoding, image transfer, and decoding on client side, which massively affects the overall time for retrieving single frames for video generation. This way, application of external (network-based) 3DPSs for video frame rendering creates a major performance challenge for implementation of a video generation service. Further, video scenes generally cope with a specific geographic area of interest and, therefore, they usually require only a subset of data for that particular area in sequential order. That is, most likely, two consecutive frames require a similar subset of geometry and texture data for rendering since they have a similar camera position and orientation. An implementation of the rendering service can take advantage hereof using these characteristics (*frame-to-frame coherence*) to apply optimized out-of-core rendering techniques, e.g., caching of geometry and texture data, managing available resources (in particular GPU, IO and memory resources). If a portrayal operation of a 3DPS is invoked over a network, there is no guarantee about the order of incoming requests and their processing due to the statelessness of such services. Requests originating from different clients might request images of completely distinct spatial areas, resulting in no frame-to-frame coherence as a worst case scenario.

To optimize video generation for most common cases of sequential portrayal requests, e.g., when a camera follows a path for traversing a specific area of interest, there is an internal renderer for virtual 3D city models implemented within the video render that encapsulates an instance of the C++ rendering service (presented in Chapter 5) for processing 3DPS requests. The internal rendering module uses the C++ interface provided by the 3D rendering service library, which is aligned to the 3DPS interface provided by the network based service. This allows us to determine the order of rendering requests by the calling video renderer. It aligns portrayal requests for video clips in a way that provides a high frame-to-frame coherence and, therefore, an optimized rendering performance for video generation. Further, rendered frames are not encoded into transmission formats, but returned as internal image representations. This way, the overhead for image generation of virtual 3D city models is reduced significantly, but external 3DPS instances can still be invoked and resulting images can be combined into one video, either through image-based operations during frame compositing or as clips or sequences for different time ranges.

7.6 Video Editor Front-End

A *video editor front-end* has been designed and implemented that supports users to create video descriptions in a web-based environment with a minimal set of hardware requirements. All components that use 3D capabilities, i.e., WebGL-based clients for image-based preview rendering, are optional and are activated only if there is support on the client machine. In the case that WebGL rendering is not available on client browsers, there is a replacement using 3DPS image retrieval. As a consequence, the video editor front-end does not require any browser-plugins or 3rd-party software to be installed on end-user systems.

The web-application provides four main functionality that are available in dedicated interaction areas in the user interface (see corresponding numbers in Fig. 7.10):

(1) **Camera path editing** specifies the 2D camera movement and orientations on a 2D map,

(2) **Video editing** specifies video composition and timing using a timeline representation

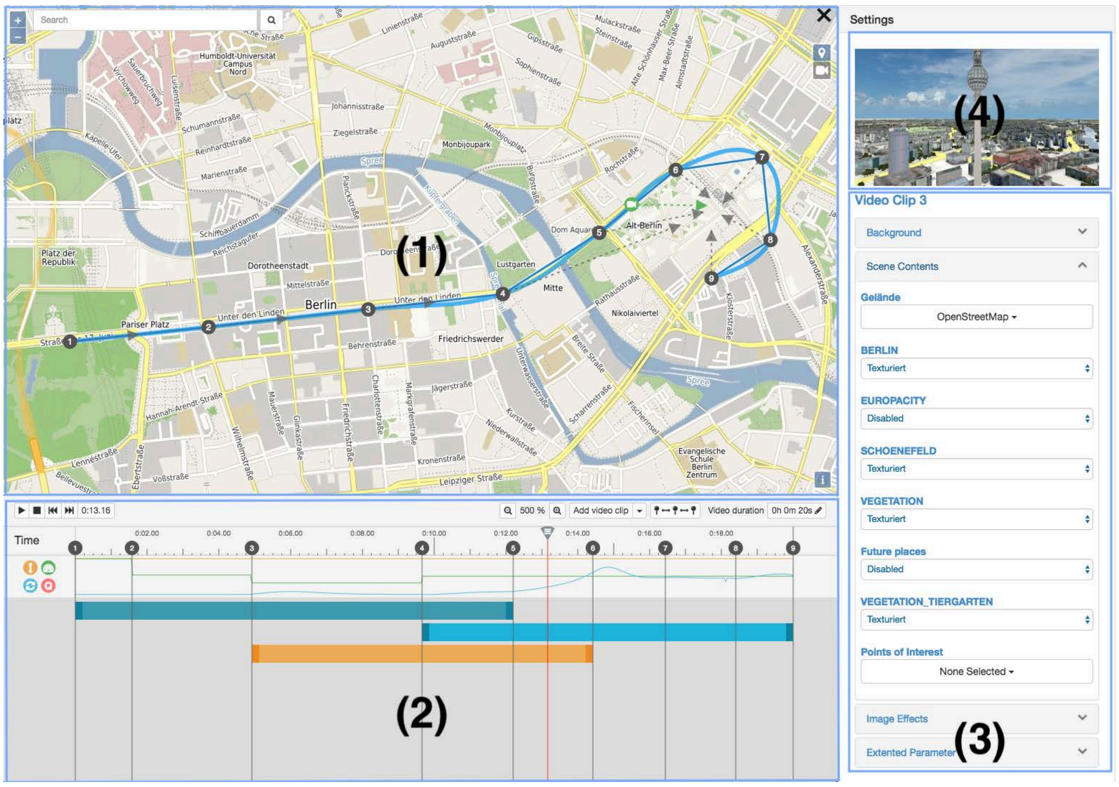
(3) **Clip editing** configures the contents and styling for currently selected clips

(4) **Image preview** shows the currently selected scene configuration using the current camera configuration

7.6.1 Camera Path Editor

Specifying a camera path in 3D represents a complex task, especially for inexperienced users. For that reason, a primary objective of the user interface design for the video editor front-end has been to reduce the complexity of the camera path specification. The process of defining 3D camera paths has been split into two separate interaction modes: Camera path definition, i.e., the camera position over time, and specification of camera orientations. To this end, the camera path editor allows users to “draw” camera paths based on control points of a B-Spline in 2D using a map (see Section 7.4.3 for camera path modelling). The 3D component of camera positions and orientations are kept unchanged during this map-based interaction.

The orientation as well as the camera position are defined as point of camera and point of interest at specific camera viewpoints (waypoints). All intermediate positions are then interpolated by the system following the curve. Users specify only the P_0 and



0:13.16 | 500 % | Add video clip | Video duration 0h 0m 20s

Settings

Video Clip 3

Background

Scene Contents

Gelände: OpenStreetMap

BERLIN: Texturiert

EUROPACITY: Disabled

SCHOENEFELD: Texturiert

VEGETATION: Texturiert

Future places: Disabled

VEGETATION_TIERGARTEN: Texturiert

Points of Interest: None Selected

Image Effects

Extented Parameter

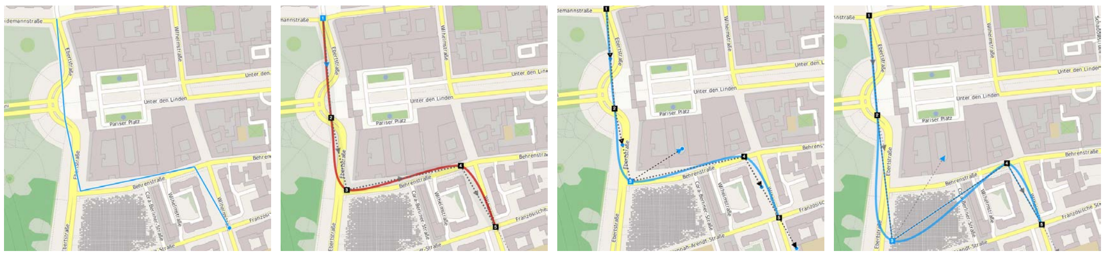
Figure 7.10: *The user interface of the video service front-end. 2D camera paths are configured using a B-Spline in connection with focus points on a map (1). The video itself is structured into clips that may have overlapping time frames to specify transitions from one clip configuration (i.e., selected layers and their styling, as well as applied image styles) to another (2). Each clip configuration is edited in the clip editor panel (3). An image preview is provided that shows the current scene configuration using the currently selected camera position and orientation and the current scene configuration (4).*

P_3 of a cubic Beziér path segment. The missing control points P_1 and P_2 are calculated by the system to ease camera path specification (see Fig. 7.7(c) for an example of such a path segment). When n waypoints are defined on the map, there are $n-1$ of such path segments created by the system – one between each pair of neighboring waypoints.

The camera path editor provides three interaction modes on the 2D map:

Path Creation An initial curve is defined by placing a number of waypoints on the map (Fig. 7.11(a)). The camera spline and its control points are derived from these waypoints. It provides a good initial approximation of a smooth camera path, i.e., it avoids discontinuous movements or the the camera in terms of orientation and position (Fig. 7.11(b)). Initial camera orientations for the control points are calculated as the tangential vectors of the B-spline at the single control points of the spline.

Path Editing Waypoints can be added to or removed from to a camera path to further adjust the curve. In addition, the waypoints can be moved. Immediate feedback is provided by updating the curve displayed on the map as well as the image preview (Fig. 7.11(d)) providing immediate feedback.



(a) *Initial path created by setting multiple points.* (b) *Path derived from initial input.* (c) *Edit view direction by dragging lookTo point.* (d) *Editing existing path points by dragging.*

Figure 7.11: *Example of a camera path drawn on the map of the camera path editor. The camera position interpolation is shown continuously as 2D B-Spline curve, while the camera orientation is visible at the control points of the camera path.*

Camera Orientation Users can shift the point of interest for a selected waypoint by dragging a corresponding circle to the desired direction (Fig. 7.11(c)). A dashed line indicates the line-of-sight with from the camera's center of projection.

Camera velocity, defined by the the time span and the spatial distance between two waypoints, can be configured by shifting a waypoint on the timeline. A duration of a path segment and the resulting velocity during that segment is then derived from the time between neighboring waypoints. The system provides users with graphical feedback about the quality of the configured camera paths to support them with camera path editing. In that respect, it helps users to create continuous camera movements in terms of camera position, orientation, and velocity. The more detail about the feedback types are provided in Section 7.6.7.

The configured path covers the complete duration of the edited video. For configuration of time segments with possibly changing video contents, the complete path is separated into several video clips represented as bars between waypoints in the video editor widget.

7.6.2 Video Clip Configuration

The clip editor widget allows users to specify the properties for the different clip types (i.e., 3D city model scenes and different kinds of overlays). For video clips rendered from virtual 3D city models, the scene content and styling is selected, as well as other clip parameters can be configured. The selectable layers, their styles, image styles and extended parameters are retrieved from the `GetCapabilities` operation of a 3DPS. Here, a filter mechanism is applied in the back-end limiting the options that are available in the front-end. The filter is configured by rules that are stored as video editor scenario in the back-end database.

Users can configure scene layers by selecting an available layer style or a "disabled" option otherwise. The "terrain" layer is handled separately. It can be contained multiple times in a rendering request to enable combination of maps with 2D overlays, e.g., to apply a road overlay over an aerial photograph. This particular layer is reserved for terrain data by convention of the 3D rendering service implementation.

Image styles can also be activated or deactivated using a UI element. Further, the

3DPS may define extended parameters, which vary depending on the underlying service implementation. These parameters can represent, e.g., image styling effects or the 3D rendering service execution in other ways, e.g., enable or disable an optional rendering technique. These user-configurable, extended parameters are defined in the scenario configuration of the video editor front-end and are exposed by the user interface.

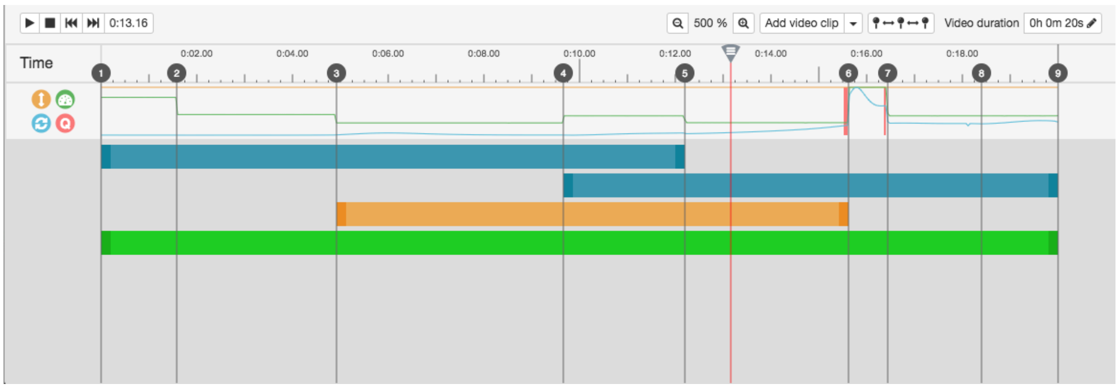


Figure 7.12: *User interface component for configuration of video composition and camera path timing. Each defined camera configuration is presented as a numbered vertical line on a timeline scale. Timings of video clips (blue bars), text overlay clips (orange bars), or image overlay clips (green bars) can be configured to cover certain time spans by modifying their position and length on the timeline. A line plot above the timeline provides feedback on specified camera path and its timing.*

7.6.3 Timeline Arrangement of Clips, Transitions, and Waypoints

Clips structure the video content (Fig. 7.4). They are represented in the user interface by bars covering a specific time span. Here, start and end of video clips are bound to waypoints (e.g., the blue bars in Fig. 7.12). As static clips do not require camera path interpolation, these are not bound to waypoints for their start and end time but can be arranged freely on the timeline (e.g., the orange or green bars in Fig. 7.12). On creation of the initial path, a video clip covering the complete video time is added automatically. To refine the video description, additional video clips, overlays or other static elements can be added to the timeline on different image layers. If two or more video clip configurations cover the same waypoint intervals, the final video frame is created by blending the generated images from the different clips covering a time index. This way, different scene contents can be blended together by creating a time overlap between multiple video clips on different layers. The video editor automatically creates the corresponding blend-in and blend-out effects for the video description.

The time distance between waypoints can be adjusted by shifting waypoints on the timeline. During this shift operation, the camera path and the path length between waypoints remain constant. This way, the camera velocity can be specified individually for each path segment between two waypoints.

7.6.4 Adjustment of Camera Orientation and Scene Preview

Editing camera paths based on 2D maps proves to be useful as it facilitates the rapid creation of feasible camera paths. Nevertheless, when it comes to the definition of the height component of waypoints or points of interest the map does not reflect the camera path sufficiently due to their 2D nature. Consequently, camera height and height of points of interest need to be configured separately either numerically by entering the camera height or visually using simple image-based clients for camera adjustment. Particularly for small scale scenarios, such as indoor scenarios or pedestrian level navigation, the adjustment of camera position and orientation need to be fine tuned in 3D to provide the user an impression of the final view at a certain position.

A 3DPS instance renders preview images depending on the currently selected video clip and the waypoint with its camera position and orientation. The preview image is updated when any changes on the video clip configuration or the camera path (i.e., moving waypoints or changing or camera orientation) are performed. There are several ways to fine tune the camera position and orientation: a) modification of camera orientation using arrow buttons that are displayed beneath the preview image, b) by changing the numerical values for height of *lookFrom* and *lookTo* positions, or c) by using a WebGL-based client that renders a cube map from 6 images retrieved from the 3DPS (a JavaScript based implementation of a service-based G-Buffer cube map client presented in [DHK12]). Here, the cube map rendering-client enables users to rotate the camera freely into a desired position.

7.6.5 Object Selection and Highlighting Using Layered Maps

Selections of feature ids for object highlighting in synthesized videos is implemented using a layered map. Its tiles are retrieved from a layered map service (see Fig. 7.3) that encapsulates a 3DPS for live rendering of the scene configuration of the currently selected video clip. The map allows to select highlighted features by picking object IDs (Fig. 7.13). As the object ID of a feature is dependent on the current scene configuration (enabled layers, number of features in the selected layers, version of the underlying city model, etc.), the object ID cannot be used for persistent identification of highlighted objects, e.g., in video descriptions. For this purpose, the object ID is translated to a unique feature ID using a service method of the 3DPS. It is called on object ID selection returning feature IDs, which are stored in the video description. During video generation, highlighting is implemented using an image-based highlighting effect provided by the rendering service implementation.

7.6.6 Text and Image Overlay Editing

To each video clip in the timeline, users can add a text or image overlays. For this, an overlay editor has been implemented based on HTML5 2D canvas rendering applied on top of a preview image generated by the 3DPS. For text overlays, it allows to define text, its styling (typesetting, alignment, and font), and placement (in terms of numeric offsets) (see Fig. 7.14). Image overlays can be provided using an image upload mechanism and placed using offsets. Further transition animations for text and image overlays can be defined within the time bounds of the image or text clip to allow for transitions that are

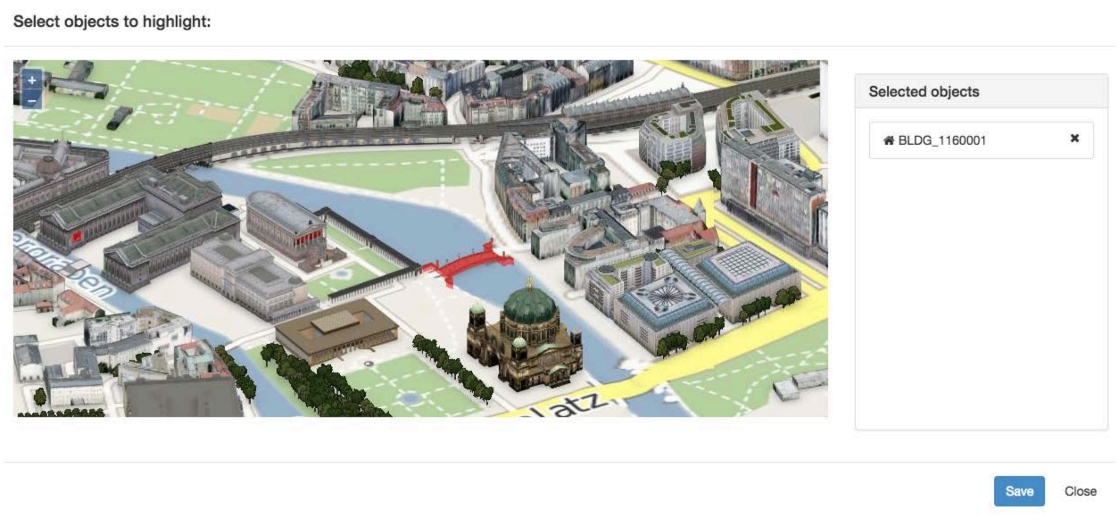


Figure 7.13: *Objects to highlight can be selected in the integrated the layered map client. The color tiles and object ID tiles are retrieved from a layered map service based on the scene contents and styling configurations of the currently selected video clip.*

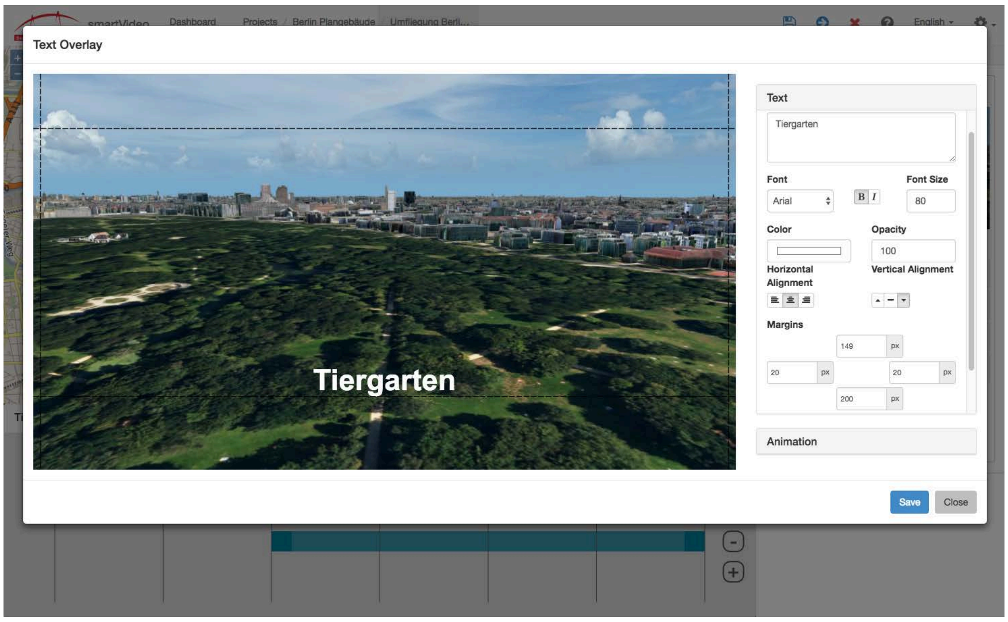
independent from the transitions of the video clip (i.e., 3D city model scene), which is currently selected in the user interface.

7.6.7 User Feedback for 3D Camera Path Authoring

The map-based specification approach provides a simplified way of editing camera paths in 3D. There are still challenges to be addressed to help users to understand the spatial and timing properties of the 3D path they are creating. To this end, the video editor front-end provides three kinds of feedback mechanisms that are built to help users to evaluate and improve their path and scene configuration iteratively:

Image Previews Image previews are provided for any point in time of the video and the corresponding camera position and orientation and the currently active video clip configuration. The time index and the corresponding camera position and orientation are selected either using the time slider in the video editor widget or by selecting a specific waypoint on the map. To provide a more detailed view of the scene, the preview image can be requested in high resolution, e.g., to evaluate the visibility of objects in the rendered video.

Animation Previews The configured camera animation can be rendered by continuously issuing 3DPS `GetView` requests. The camera configuration for the current timestamp is interpolated on client side and the corresponding image is displayed in the preview area. The two dimensions time and space are indicated individually: The current camera position is indicated by a camera icon on the map (see Fig. 7.10) and the time of the current video frame is indicated in the timeline (red vertical line in Fig. 7.12). The preview animation provides users with the means to estimate the camera path and velocity during the overall camera animation without local rendering of a large-scale model.



Text: Tiergarten
Font: Arial
Font Size: 80
Color
Opacity: 100
Horizontal Alignment
Vertical Alignment
Margins: 149 px, 20 px, 20 px, 200 px

Figure 7.14: *Text overlay editor used to define text, its style and position.*

Metrics Diagrams for Camera Paths At the top of the timeline area (Fig. 7.12) the following metrics of a configured camera path are plotted over video time:

Camera Height The camera height defines the height above ground of the camera along its path and corresponding timing. As it is displayed separately from the is separated from the 2D camera path, specified by a 2D map, to simplify the specification process. Further, large changes of camera height in a short amount of time can impact the orientation of viewers of videos.

Camera Velocity A velocity indicator is computed from the camera path to provide feedback concerning velocity changes that result in strong acceleration or deceleration in the rendered video. The resulting line (plotted in green) allows users to recognize acceleration characteristics during path traversal and optimize their path by adjusting the distribution of waypoints over time.

Camera Rotation Speed The camera rotation speed provides information about the change of the camera looking angle. The plot (blue) enables users to identify possible sudden changes in camera orientation that can complicate the orientation in 3D for viewers of rendered videos.

Issue Indicator Indicators for possibly critical issues of a camera path are drawn as red background for time ranges if the system identifies possible issues, derived, e.g., from the mentioned indicators. The indicator metric is currently based on a combination of thresholds for changes of camera height, rotation speed, and camera velocity. Metrics used for these warnings can be extended to reflect other properties and algorithms evaluate camera paths.

All of these mechanisms reflect adjustments to scene and path configurations immediately, by adjusting preview request parameters or updating metrics diagrams. This way, users are supported in having short iteration cycles to build their optimal camera paths and video configurations.

7.7 Discussion

Performance and Workload Adaptation

The current implementation provides several possibilities for scaling horizontally and vertically for larger user bases and changing workloads respectively, but the rendering of images remains the major bottleneck regarding scalability. Since the throughput of a video service, in terms of rendered video frames per second, strongly depends on the capacity of the underlying 3DPS implementations, a more parallel approach for 3D portrayal could be applied. The current 3DPS implementation is designed to optimize utilization of the GPU since it is the most expensive hardware resource for servers running the service (in terms of total cost of ownership including, e.g., price and power consumption during operation). Nevertheless, the rendering system implementation does currently not allow for using multiple GPUs for rendering the same geodata in one process. A major advantage of this would be the more efficient use of main memory and CPU resources per rendering server instance. Shared caches, e.g., for terrain tiles, textures, and scene geometry, could be used by different parallel pipelines which would statistically reduce the effort for loading and decoding the data from disk or network resources.

Generation of videos could also be cascaded if there are videos with multiple clips defined. Since each video clip has an independent description of scene contents and camera path, the cascading video services could generate video data per video clip and another service could combine the source data to produce the final result. From an architectural perspective, the extraction of the video combination or postprocessing capabilities would further improve the separation of concerns in the overall system architecture and provide another separate reusable building block following the approach of microservice architectures.

Generation of High-Quality 360° Videos

The development of hand-held devices, i.e., smartphones and end-user hardware as the Oculus Rift, becomes more and more oriented towards virtual reality applications. This way, 3D contents can increasingly be perceived in an immersive manner. Nevertheless, the production of video contents powering these solutions is still quite challenging since it demands for specialized hardware and needs to consider new objectives like an improved prevention of motion sickness [Gol+12] or specialized projection techniques for optimal usage of image space for video capture and rendering [AL13]. The projections used for rendering, e.g., of spherical 360° panorama videos, could be adapted to support the generation of low-distortion videos. These video types provide an intermediate step between static generated video and interactive 3D rendering and interactive 3D camera control in 3D GeoVEs since users have a limited amount of freedom to explore the environment themselves by rotating the virtual camera while moving on a predefined path of camera positions. To support this, the video generation service as well as the rendering service implementation could be extended to support definition and rendering of such camera projections.

Standardization

The current system is built on 3DPS as 3D rendering components whose service API is standardized by the OGC. The standardization makes the service implementations exchangeable and provides greater separation of concerns in terms of functionality that has to be implemented in the different components. The current modeling specification, language, and video service API could serve as a basis for a standardized description format and API for video generation from 3D contents.

Authorization and Permissions

In the current implementation, there is no explicit authorization mechanism when it comes to video rendering of scene contents or specific rendering features, e.g., image styles. The rendering back-end is currently protected by administrative measures from being used by unauthorized users. An explicit authorization mechanism is important for both: a) security - an unknown user might not be authorized to view a specific planning or model - and b) monetization - different subscription plans for a video service account might not include all models or image styles. For implementation, e.g., of a multi-tenant system provided as a Software as a Service (*SaaS*), the data of each tenant need to be reliably separated from each other. There are several points to address these issues. First, the video service front-end requires a way to model these dependency in the database that is evaluated for every request and explicitly for every video generation task. Further, the video service and the image-based 3DPS could also extended to support access rights for specific layers or styling options to assure delivery of videos and images, e.g., with confidential contents cannot be created or downloaded by unauthorized users, e.g., using a session based authentication mechanism that assigns access rights per session that can be checked by executing services.

Chapter 8

Service-Based Camera Interaction

This chapter introduces a concept for services supporting 3D camera interaction in 3D GeoVEs and, particularly, for virtual 3D city models. The introduced services can serve as a building blocks to encapsulate interaction specific functionality in future service-based 3D geovisualization systems. Therefore, the camera interaction process is disassembled into distinct reusable functionality that can be implemented as separate services. A 3D camera service serves as core component that supports generation of camera definitions and camera paths based on different input types and granularity.

8.1 Motivation

Geovirtual environments serve as means for exploring virtual 3D city models. As Fuhrmann and MacEachren state that "core problems for users of these desktop GeoVEs are to navigate through, and remain oriented in, the display space and to relate that display space to the geographic space it depicts." [FM01] "Efficient navigation strategies and techniques are required, which take account of the users and their goals and avoid problems of general navigation methods, such as "getting-lost" situations and confusing view configurations." [HD08] Due to the complex nature of virtual 3D city models and their provisioning, the design and implementation of such strategies for camera control in distributed 3D visualization systems for virtual 3D city models is a challenging task.

In current research, the area of camera control in service-based visualization systems is still in its beginnings. There have been approaches to integrate camera position calculation into 3D portrayal services using dedicated service operations, as defined in the OGC *Web View Service (WVS)* [Hag10] discussion paper, as the precursor of the OGC 3DPS standard. However, these operations only support simple interactions in terms of camera positioning, but do not cover more complex techniques for user interaction or camera path generation.

For a application of camera control in distributed, service-based visualization systems, the following goals for system design and implementation have been identified:

Decoupling Visualization and interaction should be decoupled and implemented by independent services to facilitate reuse of camera control functionality as independent components of 3D geovisualization systems.

Feedback Feedback about available and pending camera movements as well as system state information should be provided by a distributed system for 3D camera control.

Network Adaption 3D camera control should be designed to deal with limitations of wireless communication networks in connection with mobile clients, e.g., connection loss, network latency, and varying bandwidth.

Multi-Channel Input The system should support different types of user input. Beside conventional input for user interaction, such as mouse pointer or touch positions, button events, or keyboard events, other input types could serve as input to user interaction techniques. With increasing distribution of mobile devices, there is a strong focus towards touch-based interaction and sensor input for all types of applications. For example, device location, orientation, speed or other data delivered by device sensors could influence the way the virtual camera behaves, e.g., to include a user's actual position and the position and orientation for navigation within a virtual 3D city model.

Flexible Deployment The separation of camera-control functionality should support its implementation as web services, but also as integrated part of client applications. This way, the decision of the location of network boundaries in concrete system architectures can be made for every client application.

8.2 A Conceptual Framework for 3D Camera Services

The following sections introduce the conceptual framework for service-based camera interaction.

8.2.1 Quality of Scene Views

A measurement of the quality of camera specification in 3D GeoVE strongly depends on the usage scenario of a virtual 3D city model and is also a rather subjective matter in terms of personal preferences. As minimal definitions for "good views", Vázquez *et al.* state that a "good view must help us to understand as much as possible the object or scene represented." [Váz+01] Several metrics and techniques exist that target to quantify view quality using measures such as number of degenerated faces in orthogonal projections [RM98], viewpoint entropy [Váz+01; Váz+03], or image features such as silhouettes or crease lines [Goo+01]. Many methods exist that use those metrics to compute and optimize scene views and corresponding camera parameters (see [Bon+18; Sec+11] for an overview). These approaches for deriving "good views" for diverse viewing situations and application scenarios using rather complex computational techniques. The proposed framework for 3D camera services should allow us to reuse such approaches and implementation by providing a conceptual framework for implementation of camera interaction and viewpoint generation techniques. In the following, no particular notion of viewpoint quality from these works is used. Instead, the definition of "good views" can generally be defined by service or algorithm implementations. An example of a prototypical implementation for computation of good views and camera paths is provided in Section 8.3.

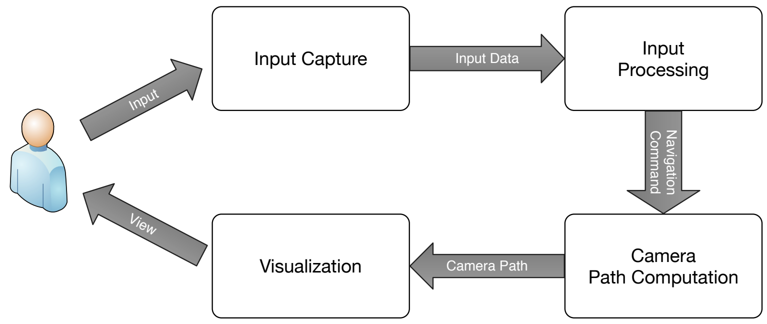


Figure 8.1: *General camera interaction cycle for service-based 3D visualization systems.*

8.2.2 Interaction Process

End-user hardware, software, and network bandwidth for interactive visualization applications are usually very heterogeneous. That causes the development of robust, compatible, and efficient applications that provide interactive access to virtual 3D city models to be a complex software architecture problem. 3D geovisualization systems based on thin clients using server-side 3D rendering are one approach to deal with such limitations. Typical characteristics of thin client solutions are that only minor parts of the overall data of a virtual 3D city model is available on client side and can therefore be considered for camera path computation or for supporting user interaction in general. To address this limitation, we propose to move major parts of functionality for 3D camera control away from client applications to services that can incorporate geometry and feature data of virtual 3D city models on server side. This would enable reuse of camera interaction functionality across different application. Services can be run in a scalable, controlled server environment and, therefore, can be maintained and optimized more efficiently. Furthermore, server-side access to geodata is usually faster and less expensive due to faster network connections and professional grade and high-performance hardware.

To structure the interaction cycle of service-based 3D visualization systems, the process for 3D camera control is divided into four core tasks: (Fig. 8.1):

Input Capture Input provided by a user has to be captured and encoded in a way that allows for efficient evaluation.

Input Processing User input is preprocessed, e.g., converted, transformed, smoothed, or patterns are recognized and a navigation command is derived from the resulting data. This command is used to select the 3D camera service for camera path computation.

Camera Path Computation Camera positions and orientations, and transitions between them are computed. Specifications for camera paths are the result of this stage.

Visualization The computed camera specifications have to be applied for the client-side visualization of the virtual 3D city model. Visual or non-visual (e.g., audible) feedback has to be generated and integrated to complete a 3D camera-control cycle.

While input capture has to be implemented by a client application, input processing, camera path computation and visualization can be implemented by one or multiple

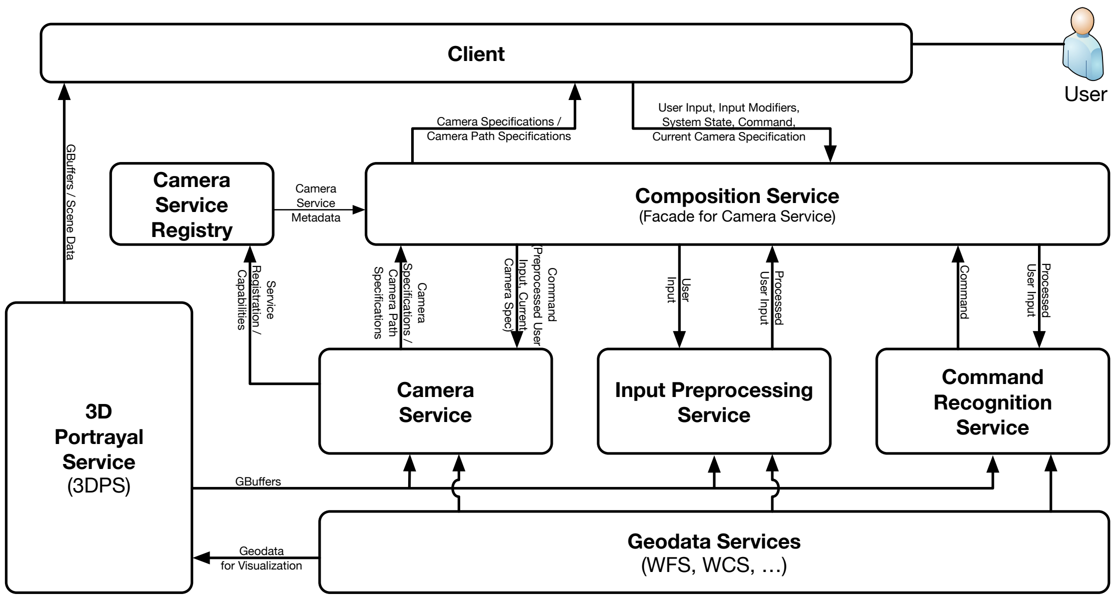

Client
User
Camera Specifications / Camera Path Specifications
User Input, Input Modifiers, System State, Command, Current Camera Specification
GBuffers / Scene Data
Camera Service Registry
Camera Service Metadata
Composition Service
(Facade for Camera Service)
Service Registration / Capabilities
Camera Specifications / Camera Path Specifications
Command Preprocessed User Input, Current Camera Spec.
User Input
Processed User Input
Command
Processed User Input
Camera Service
Input Preprocessing Service
Command Recognition Service
3D Portrayal Service (3DPS)
GBuffers
Geodata for Visualization
Geodata Services
(WFS, WCS, ...)

Figure 8.2: *An overview of the service-based concept for camera interaction in distributed 3D visualization applications.*

services. Fig. 8.2 illustrates the concept for decomposing the core tasks for 3D camera control into functional independent *3D interaction services* and depicts their collaboration and the types of data exchanged between them. In the following, the following major 3D camera interaction services are introduced: Input preprocessing services, command recognition services, 3D camera services and composition services.

8.2.3 Input Capture

User input for camera interaction can have different levels of abstraction. For example, low-level user input, such as mouse positions or mouse button events, does not express the expected action immediately. They need to be evaluated first taking the current application context into account. On the other hand, there are more explicit user inputs provided by end-user client-applications that explicitly describe the action to be performed. Examples for inputs that provide an explicit description of the task to be performed are, e.g., buttons that have an explicit function assigned in the user interface of a client application, e.g., a home button for resetting the virtual camera. Thus, a specification of user inputs is required that supports a variety of inputs. User inputs are generally captured as data annotated with timestamps to allow, e.g., for segmenting user input in time and (display-) space (to capture and evaluate, e.g., a series of sketches or cursor positions) and computing velocities of movements.

Additional information that is not directly originating from user actions can be required for service-based camera interaction. Therefore, the client state, such as input modifiers (e.g., pressed mouse buttons or keyboard modifiers) or previous navigation commands (i.e., navigation history), may affect the mapping from input parameters to camera navigation commands to be executed. Further, the current scene configuration (i.e., camera specification, projection parameters, visible scene layers, etc.) specifies the interaction context of the user input, which affect the evaluation of a user input.

8.2.4 Input Processing

Input processing is divided into two steps: input preprocessing and command recognition.

Depending on the type of input captured by the client, an *input preprocessing step* may be necessary a) to improve the quality of the input, e.g., by filtering or smoothing and b) to convert the input data to a higher-level, analytical representation, e.g., recognizing geometry from a series of 2D input samples.

In a *command recognition step*, navigation commands are derived from the preprocessed input. These represent a more abstract description of a user's intention and include all the parameters required for their execution. There are three categories of navigation commands that define the camera-control task [HD08]:

Direct Camera Manipulation A command directly influences the values of camera parameters, such as position or orientation vectors, which specify the current view. Commands like 'turn by 30 degrees' or 'move 100 meters into camera direction' are examples for such direct camera manipulation commands.

Path Oriented Navigation Command A command includes a path description that has to be followed by a camera path. The desired path has been computed in the input processing step or has been specified by the client directly (explicit or implicitly, e.g., by providing a target name).

Task Oriented Navigation Command A command contains a description of a task, which has to be fulfilled by a 3D camera service. Commands like "go to the next feature of class X" or "inspect feature X" belong to this command category.

To describe a command and to support command recognition, a generic, structured command schema is required that specifies, e.g., command parameters (types and possible values). The command recognition step can involve retrieval of additional geoinformation, e.g., from geodata or geovisualization services such as WFS, WCS, or 3DPS.

Input preprocessing as well as command recognition are optional steps in the 3D camera control process. Simple camera-control tasks can be transmitted by a client as navigation command, e.g., "move one meter to north" for a stepwise camera control. Such commands may be handled directly by an appropriate 3D camera service.

The functionality of input preprocessing and command recognition steps are encapsulated by respective service types, *input preprocessing services* and *command recognition services.*

8.2.5 Camera Path Computation

Camera path computation is the core task for camera-control in virtual 3D city models. *3D camera services* compute camera paths from navigation commands and their parameters. Conceptually, one 3D camera service implements one technique for camera path computation. This includes the generation of camera path components, e.g., camera positions, orientations, or other information that can be associated with a camera transition, e.g., textual annotations. Each of those can be computed by distinct functional components that apply specific algorithms and navigation constraints per path component. For example, a specific position component could determine camera positions only along a street network, while a specific orientation component could aim to keep nearby landmarks visible.

A 3D camera service may request additional (geo-) data, e.g., from a WFS, WCS, or 3PDS (image-based or scene-based) to provide, e.g., a higher-level, semantics-based camera control or to fulfill constraints for camera parameters. To ensure a consistent behavior, these additional services have to be based on the same geodata as the visualization services themselves.

Based on a navigation command, a 3D camera service is selected using a 3D camera-service registry, which holds information about the available 3D camera service instances, the navigation commands they support, and additional metadata.

8.2.6 Camera Path Specification

Camera paths, generated by 3D camera services, represent transitions from one set of camera parameters to another. Parameters required for the specification of a view of a 3D virtual environment are its projection parameters, commonly the camera position, its orientation in 3D space and the view frustum definition.

For camera services, two types of representations of camera paths are distinguished:

Sampled representations of camera paths include of a series of camera-specification samples. Those can be created either using fixed or variable sample times. An adaptive sampling rate of camera specifications allows for more efficient representations of camera paths by applying more dense sampling for time periods where camera parameters change more rapidly, e.g., because of increased movement speed or sharp camera turns.

Analytical representations provide a function definition for each camera parameter to compute their values for an arbitrary point in time during a camera transition. For simplicity, these path are often defined by piecewise functions, e.g., multiple Bézier splines, linear or even constant functions for stopping the camera for a specific amount of time. Furthermore, this enables the specification of storyboard-like camera transitions. The overall time for the complete camera path animation is normalized. A camera path specification contains a recommended overall animation time, which would produce a comfortable camera motion.

There are several path-related computational tasks in distributed 3D camera control systems that can be implemented as reusable utility services, e.g., for path manipulation (conversion, transformation, smoothing, composition) or camera orientation computation. Further, existing standards-based implementations of services, e.g., for geocoding of locations or conventional 2D routing using street networks may be used for camera path generation in 3D GeoVEs.

8.2.7 3D Camera Service

To be managed in a service registry and allowing consumers to bind correctly to their operations, 3D camera services are required to express general service information as well as functional and additional non-functional metadata. 3D camera service capabilities should include metadata regarding the following aspects:

Service identification Type and version of the service, service description.

Camera control metadata Available path description formats, covered geospatial region, supported spatial reference systems, available navigation commands including command parameters.

Quality of service metadata Information such as expected computation time, result accuracy, available level of collision avoidance (e.g., guaranteed, best-effort, or no avoidance).

Application context Information regarding, e.g., user information, usage information, network conditions, and device properties.

8.2.8 Service Operations

The 3D camera service exposes the following service operations:

`GetCapabilities` A camera service provides a `GetCapabilities` operation that provides service metadata. The format of the `GetCapabilities` request and response is aligned to the same operation operation In particular, the `GetCapabilities` response includes the available service operations. It further contains the metadata for each supported operation and which computation modes are available for clients to call. This also includes a list of available and required parameters for these operations.

`GetCamera` Calculates a set of camera positions and orientations, optionally with further projection parameters assigned. If several camera specifications are returned, a conviction score is assigned to each entry of the list. The higher the assigned score, the "better" the selected algorithm expects the camera position to fit quality demands. Algorithms promoted in the `GetCapabilties` operation response can be selected using a `MODE` parameter. For viewpoint calculation of features and sets of features, a list of feature IDs must be provided as parameter. The request should then return a "good view" for all the features contained in the request.

`GetCameraPath` Calculates a camera path for a given set of a) 3D points to visit during the camera transition or b) a set of features to inspect during the camera path. As with the `GetCamera` operation, the `GetCameraPath` operation provides a `MODE` parameter to select specific path generation algorithms.

While the `GetCapabilities` operation is mandatory, a service must provide at least one of the operations `GetCamera` or `GetCameraPath`.

8.3 3D Camera Service Implementation

As proof of concept, a 3D camera service has been designed and implemented that generates camera configurations for single features as well as collision-free camera paths for a series of given objects of interest. It provides a novel building block for visualization applications, which is specifically useful for thin client implementations that are usually unable to access global geometric and semantic information needed to perform complex, assisting camera interaction.

The implementation of this service does not focus on advanced techniques for collision avoidance but serves as a proof of concept for service-based 3D camera interaction in virtual 3D city models. Its objective is to demonstrate applicability for large software systems and environments, e.g., as utility service for the video service front-end described in Chapter 7.

The current implementation of the camera service provides a single mode of computation for each of the `GetCamera` and `GetCameraPath` operations (see Section 8.2.8). As a simple definition of a "good view" of objects and resulting good camera paths, algorithms should provide camera specifications and camera animation paths that are a) collision free and b) show the requested features within the view frustum. Further, the checks and computations that are necessary for collision avoidance should scale to very large virtual 3D city models. The implemented algorithms target in particular outdoor navigation in virtual 3D city models. It exploits properties of such models for efficient computation of camera paths, e.g., the 2.5D distribution of building within an urban area that is modelled by a virtual 3D city model.

8.3.1 Data Sources and Precision

The camera service uses two different data sources:

Feature Data Feature data is fetched from the spatial feature database introduced in Chapter 5. The service uses 2D footprints of objects as well as other feature attributes such as maximum and minimum height of a feature. Further, spatial queries are used to determine possible 2D intersections with other features that could possibly obscure the line of sight towards a feature of interest or cause collisions on a camera path.

Additionally, the feature data can serve as a basis for determining the camera view direction, e.g., by trying to include landmark buildings into camera view frustums that could ease orientation within for users within the urban environment.

Depth Images For an area of interest the service requests depth images from a 3DPS using an orthographic projection with a top-down perspective with north as top direction. In this way, a rasterized height map is retrieved that covers the area of interest whose complexity and data size is independent of the number and complexity of features within this area. An example of a height map generated from the virtual 3D city model of Berlin can be found in Fig. 8.3. Distances from the projection plane of the virtual camera are encoded as normalized values of the interval $[0..d_{far}]$ where d_{far} is the distance of the camera far plane. Within such a height map, the spatial bounding box of each pixel can be calculated using the camera projection parameters, i.e., camera position and view frustum specification. This enables the service to efficiently sample height values at specific positions or even determine minimum and maximum heights along a 2D path or area efficiently.

While feature data queries provide a consistently high accuracy based on the geometry stored in the feature database, the accuracy of queries based on depth images depends on the resolution of the fetched depth image (resulting in a specific size of a pixel within the height map) and the numerical resolution of the depth values (16bit or 32bit

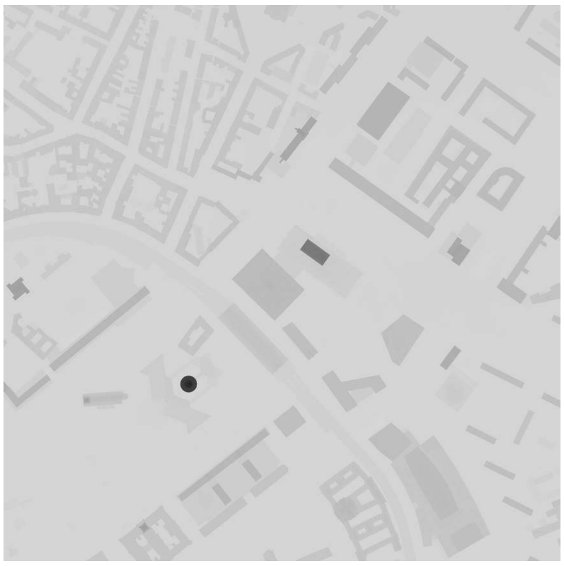


Figure 8.3: *Example of a depth map covering* $1km^2$ *around the Alexanderplatz in Berlin. The image was generated by a 3DPS from the virtual 3D city model of Berlin. It contains gray values encoding the normalized distance from camera near (black) to far plane (white).*

floating numbers that cover the interval between the near and far plane). The depth map precision in each dimension defines a minimum size of objects to be considered for collision avoidance and path planning. The service implementation deals with that by adaptation of the requested resolution of depth images to maintain a minimum precision during camera viewpoint transition. For common outdoor camera navigation, a 16 bit encoding precision of floating point depth values provides a sufficient overall height precision of about $2cm$ in case of a view frustum depth of $1000m$.

8.3.2 GetCamera Implementation

The `GetCamera` implementation currently supports a single mode that computes views on selected features in a way, that a) the features of interest fits into the view frustum – the distance to the feature is derived from its 3D bounding box stored in the feature database – and b) the four cardinal directions are used as 2D viewing directions. First, a check for possible obstacles along the viewing axis is performed using a database to identify 2D intersections between the Point of Camera (POC) and the centroid of the features of interest. If there are possible intersections, a height field retrieved from a 3DPS is sampled along the line of sight originating from the feature centroid towards the initially determined camera position. If an obstacle has been identified, the camera position and the corresponding line of sight is shifted towards the top direction in a way that there is an unobstructed view towards the feature of interest is possible. See Fig. 8.4 for an illustration of the principle.

8.3.3 GetCameraPath Implementation

The `GetCameraPath` implementation uses a similar technique to derive camera paths that provide a collision free transition of one origin camera position to a set of targets specified

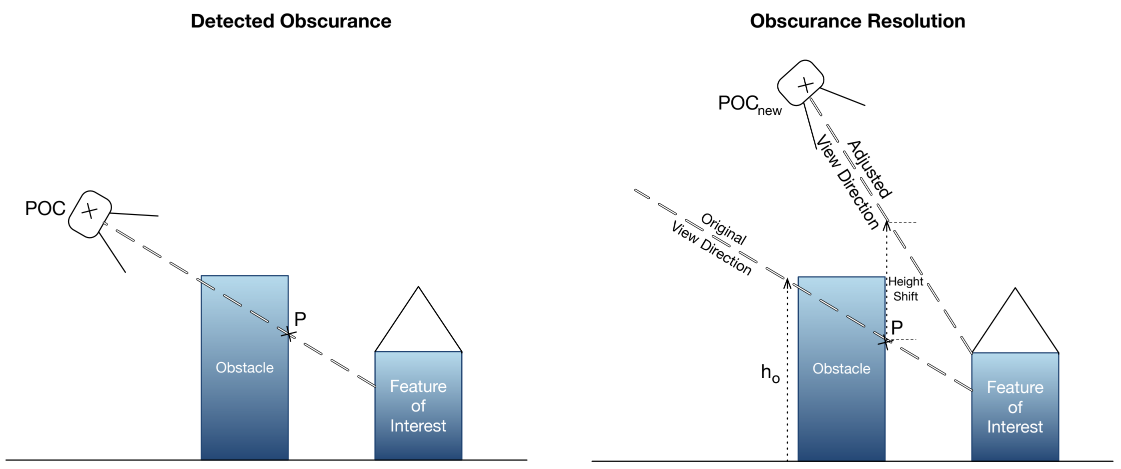


Figure 8.4: *Mechanism for obscurance resolution using sampled heights h_o from height maps. Possible obscurances are detected in two steps. First, a 2D intersection is performed using the feature database. Second, the height of an object (h_o) is sampled using a height map retrieved from a 3DPS. If an obscurance was detected, the viewing direction is shifted up adjusting the initial POC to POC_{new}. The amount of this shift is computed from h_o.*

as feature of interest (using a feature id) or target camera specification. In case of a feature target, the `GetCamera` implementation is used to derive the camera viewpoint for each intermediate target. The algorithm creates path segments each connecting a pair of adjacent viewpoints. Here, the same method for image-based collision detection and avoidance is applied that is also used in the `GetCamera` implementation with a minor modification: Instead elevating the camera position for a viewpoint, in case of a detected collision with a city model feature, an additional path segment is inserted into the path that provides a collision free transition between the beginning of the original path segment and the newly inserted intermediate camera viewpoint. This process is repeated incrementally until a collision free path is found, or stops if no collision free path can be determined.

Chapter 9

Case Studies

This chapter discusses a number of case studies to illustrate how the concepts and techniques described in this thesis can be applied to real-world use cases. The following cases studies for various model datasets evaluate and validate their functionality and investigate how to improve their compatibility with data formats and modeling variants as well as the robustness of the presented preprocessing and rendering techniques. The case studies generally show the applicability of this thesis' concepts and their implementations. The use cases represent a selection of representative scientific and commercial projects that were realized.

9.1 Datasets

Datasets used to differ significantly in terms of size (e.g., covered area, number of features, number of textures, and data volume), data format (i.e., well structured 3D source data in CityGML, scene based 3D graphics formats, GIS feature data in formats and 2.5D data generated from OpenStreetMap geometry and attributes), modelled detail (LOD), and data quality (e.g., correctness and precision of geometry).

The following list provides an overview of the datasets and the projects that have been conducted to validate the service functionality and to prove the applicability of the thesis concepts. An overview of the datasets and their properties is provided in Tab. 9.1.

DS.1 **Berlin Virtual 3D City Model** Between 2003 and 2005, the Senate Department for Economics and Urban Development of Berlin initiated the development of an official virtual 3D city model of Berlin [Döl+06]. It was one of the first 3D models of its scale, as it covered the entire Berlin area featuring fully textured 3D building models, most of them with feature attributes assigned. Throughout the years, this virtual 3D city model has been regularly adjusted and extended. Data is recaptured on a regular basis and the 3D building stock is continuously updated and a selection of high-detail 3D architectural models have been added to the model over the years, such as the Berlin Central Station in CityGML LOD4 and hundreds of buildings in LOD3. In addition to the CityGML building models, other types of data sources were included to further enrich the model:

- High-resolution aerial images and different map layers (e.g., street map, or a standard ground value map)
- High-resolution terrain model
- Vegetation models, i.e., a street tree cadastre covering the overall city and automatically derived tree positions for the Tiergarten with a total of several

Dataset	Format	# Objects	Textures	LOD	Covered Area km^2	Terrain and Maps	Model Data Size	Datal Quality	Special Properties
Berlin	CityGML 2.0	~534, 000	~5,750,000	LOD2 LOD3 LOD4	~ 900	separate terrain model, aerial photo and map layers	31 GB	good	heterogeneous LOD, including indoor models, multiple appearances, semantic attributes included planning models / variants
Paris	CityGML 1.0	~14,000	~ 500 (texture atlases)	LOD2	~ 100	CityGML Terrain as TIN	62 GB	medium	terrain model included, homogeneous LOD, semantic attributes included errors in CityGML encoding
Nuremberg	CityGML 2.0	~ 15,500	~ 100,000	LOD1 LOD2	~ 10	separate terrain model, aerial photo and map layers	3 GB	good	multiple appearance types mixed textured / untextured
Geneva	CityGML 2.0	~ 1400	87 (2 Buildings)	LOD2 LOD3	~ 1.5	separate tiff terrain model, aerial photo	0.5 GB	good	untextured LOD2 model, complex roof structures, single textured high-detail LOD3 buildings
Roman Cologne	Collada[a]	~ 187	157 high resolution texture atlases	LOD3 LOD4	~ 5	Textured 3D terrain model	3 GB	good	highly detailed 3D geometry models, high resolution textures, Collada as standard CG format
Rotterdam	CityGML 1.0	~ 195000	~ 26200	LOD2	~ 320	High resolution terrain model, Map and aerial photos as files and OGC services	3.7 GB	medium	Homogeneous massive model, Open Data, CityGML Quality

Table 9.1: *Overview of datasets used in different projects to approve the concepts and implementations.*

[a]Collada is a XML-based format for interchange of 3D assets defined by Khronos Group. See `https://www.khronos.org/collada/`

ten thousands vegetation objects.

- Several planning models in different detail levels (from simple block models to detailed architectural planning)
- Numerous POI layers available from web services

With the amount of 3D building models that are included here, as well as the overall size and richness of information of the dataset, this virtual 3D city model is an excellent test case for the services and techniques for web-based provisioning of virtual 3D city models described in this thesis.

DS.2 Paris 3D The Paris virtual 3D city model was provided by IGN France for use within the OGC *3D Portrayal Interoperability Experiment (3DPIE)* [SHC12]. The model covers the inner city of Paris in textured CityGML LOD2. The entire terrain model and its texture are encoded in CityGML, significantly increasing the amount of texture data included in this model. This type of model challenges the texture preprocessing, management, and rendering of the presented 3D rendering service. Furthermore, textures contained in the model are prepackaged texture atlas files that are referenced multiple times within one or even multiple CityGML.

DS.3 Nuremberg Virtual 3D City Model

The virtual 3D city model of Nuremberg combines model parts with different LODs and characteristics. It contains the following datasets:

- Textured LOD2 building models encoded in CityGML for the inner city area
- LOD1 building models encoded in CityGML for remaining city parts
- Terrain textures, i.e., orthophotos and an official map layer
- Medium resolution terrain model in $5m$ raster
- POI extracted from OpenStreetMap

As the model was provided by the Bayerische Vermessungsverwaltung (Bavarian surveying office) it is a good example of virtual 3D city models that are currently used in administrative contexts.

DS.4 Geneva Virtual 3D City Model The Geneva virtual 3D city model was used the context of a technology demonstration session during the OGC Technical Committee meeting in June 2014 in Geneva, Switzerland. It contains several data sources that should be combined into a single visualization:

- CityGML building models in LOD2 and textured LOD3 for a small number of textured buildings
- Feature data for these buildings accessible via OGC WFS
- Vegetation data Tree position
- Geneva terrain model (high resolution) and general Swiss terrain model (low resolution)
- High resolution orthophotos (summer and winter)

This dataset is an excellent example to demonstrate the different types of data sources that can provide data for a virtual 3D city model.

DS.5 Roman Cologne The virtual 3D city model of Cologne[1] during the Roman period was created in course of a collaboration of archaeologists, designers, and software experts from the Archaeology Institute at the University of Cologne, the Köln International School of Design (KISD), the Cologne University of Applied Sciences, the Hasso-Plattner-Institute (HPI), and Cologne's Romano-Germanic Museum. Those institutions aimed to create a detailed reconstruction of the Roman settlements in Collogne period based on archaeological findings. The model is extraordinarily precise resulting in a high degree of geometrical complexity and large amounts of high resolution texture data (see Tab. 9.1 for figures).

The Roman Cologne dataset is not encoded as a semantics-driven 3D model, but given as a collection of COLLADA [BF08] files that reference high-resolution texture atlases for texturing. As this is a standardized format for exchanging of general-purpose 3D models, the Roman Collogne model represents a well suited test case for processing and visualization of general purpose, massive 3D computer graphic models.

DS.6 Rotterdam Open Data The city of Rotterdam initiated one of the first large-scale open data initiatives, thus releasing an (semi-) automatically derived virtual 3D city model that includes CityGML LOD2 parts. The Rotterdam data poses a number of challenges regarding the processing and rendering. of geometry, structure and texturing since the CityGML documents contained issues in geometric modeling and texturing, such as falsely oriented or unclosed polygons, malicious texture coordinates, or CityGML encoding errors. The errors are caused mainly by the remote sensing, processing and the applied modeling techniques, are not uncommon. They are inherent in the semi-automatic and automatic generating process for virtual 3D city models. As such a model, the Rotterdam virtual 3D city model creates an interesting test case for robustness of preprocessing and rendering implementation.

9.2 Case Studies and Implemented Applications

The different test cases confirm the applicability of the services and implementations introduced in this thesis, in particular their compatibility with data formats and modeling variants and the robustness of preprocessing, rendering, and provisioning techniques for virtual 3D city models.

All of the aforementioned datasets were tested with the rendering service implementation and selected approaches for provisioning. A selection of datasets were also used in practical real-world application contexts. The following sections outline the case studies, their applied concepts, technical solutions, and applications that have been implemented to test and evaluate the approaches and datasets.

9.2.1 Layered Maps for Location Marketing and

[1]http://www.colonia3d.de/

Business Development in Berlin

The *Business Location Center (BLC)* as a division of the Berlin Partner for Business and Technology[2] is concerned with city marketing and especially with business consulting for company relocation to Berlin. During a long-term cooperation conducted over several years, the BLC provided the official virtual 3D city model of Berlin and valuable feedback and requirements for testing the concepts and techniques introduced in this thesis. During this collaborations, several testing scenarios have been identified and implemented as mobile and web-based applications. They are introduced in this section and Section 9.2.2 and 9.2.3.

Geodata and geoinformation play a major role in the consulting process. One major issue for their customers is to find a location that matches the specific needs of their company, e.g., in terms of office space, production space, connections to the public transportation network or other technical and social conditions. Commonly, the consulting takes place in a designated presentation room that is equipped with the technology for large-scale projections of the virtual 3D city model of Berlin as rendered by local high-performance desktop computers. Here, the interactive 3D visualization of the Berlin model is used in conjunction with a variety of curated POIs and other information from internal and external databases, e.g., available business real estate, available apartments to rent or buy, social infrastructure, research infrastructure, and a wide range of background information that is linked to these geospatial data. In more recent years, the consulting process had to accommodate the increasing use of mobile devices in remote locations, e.g., at trade fairs or meetings in remote locations where no large-scale presentation technology is available, e.g., due to missing space, or hardware infrastructure, or bad network connection on-site.

To address these challenges, a project for an application was set up to provide selected BLC information based on the virtual 3D city model to different stakeholders, such as BLC customers, BLC staff, or the general public using mobile devices with the following requirements:

Connection Quality and Stability The application is supposed to work in environments with restricted internet access, i.e., restricted in terms of speed and stability. Especially at events like fairs or with increased security requirements, the application is required to generally work offline and to update its data once a connection to the internet can be established.

Visual Quality The BLC model provides highly diverse building models (from textured geometry in CityGML LOD2 to LOD4 including precisely modeled indoor geometry and materials). Any application should be able to handle these characteristics while at the same time maintaining high visual quality for the visualization. Thus, it should apply realistic shading techniques that handle the given textures and material properties during rendering as well as to apply a global illumination for non-textured models.

Data from Different Sources The application is supposed to integrate data from already existing systems that provide information necessary for conducting consulting

[2]`https://www.berlin-partner.de`

sessions. The most important requirement here was to integrate the Berlin Partner Real Estate Portal, which informs about business real estate offers for the city of Berlin. Additionally, the BLC hosts a curated database for points of interest covering different topic. It also includes planning variants of real estate developing projects in form of additional architectural or landscape planning models.

Applied Concepts

In principle, mobile applications can be developed technically in three different ways [Dal+13; ElK+17]: a) As native application that can uses all the capability of the end-user platform (mainly iOS or Android driven devices), b) as mobile web application that is exclusively based on common web technologies (JavaScript, CSS, and HTML5), or c) as a hybrid approach where the data is kept on server side and only a lightweight front-end is implemented on the client platform.

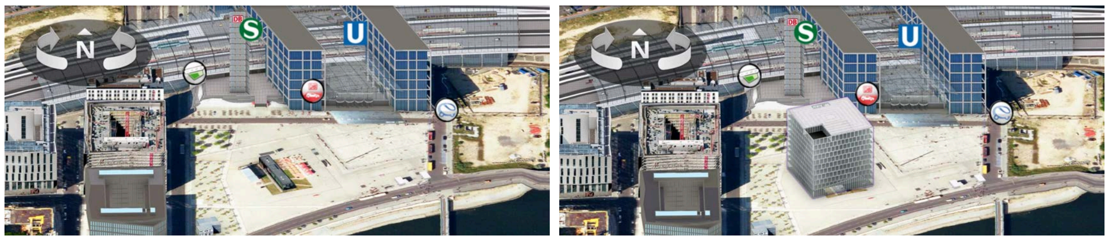
To meet the requirements, an application based on the layered map approach has been implemented using the configurable client framework on the different target platforms, i.e., Android and iOS. As is built based on platform-specific native operating system frameworks, technologies, and languages it provides a native user experience. The decision for a native implementation that includes (or: can handle) more fine-grained control of data caching mechanisms and user interactions was driven by the need for a core application functionality in scenarios where the client is offline or has to work with an unreliable network.

The applications for the different platforms were designed with data and configuration following the provisioning concept for layered maps. They integrate basically four types of data:

1. Layered map tiles depict the virtual 3D city model from the four cardinal directions and different information layers can be combined and overlaid. They offer a view of the virtual 3D city model buildings with different variants regarding 2D maps (a aerial photo or a custom generated 2D map layer, a transparent overlay adding road names and routes) and two 3D plannings planning layers that replace relevant parts of the as-is model tiles for planning area showing currently planned but not yet built projects highlighted with a dedicated styling (see Fig. 9.1 for an example view).

2. Business real estate offers that are accessed from an external service and processed by a map layer plugin. These offers are expected to change frequently and are required to be up-to-date for the consulting purpose of the mobile application.

3. POI data is integrated in compact format generated by a feature data provisioning service (see Fig. 4.2) from different sources, e.g., an information database maintained by the BLC. As the POI data change less frequently, caches for this data type are configured to have long validity periods to save data transmission and facilitate offline use. POIs provide contact data and web URLs that refer to external information systems that for further in-depth information about the displayed POIs.



(a) *As-is situation.* (b) *With optional planning layer*

Figure 9.1: *Example view of the Berlin central station with different information layers. An optional planning layer was generated and can be dynamically added at run time of the application. Detailed information about the single items on the map is provided as link when item is selected. The interaction item in the top left rotates the map view.*

4. Public transport route geometry is obtained as GeoJSON[3] features with polyline geometry. The mechanism can also be used to add other 2D geometry to the map, e.g., to mark areas under development or other areas of interest.

The map tiles for the layered map are updated using the layered map service whenever revisions of the virtual 3D city model or one of the planning layers are performed. The necessary processes for data generation and their provisioning to web servers are automated so the update can be performed efficiently with a minimal amount of manual effort. This procedure allows the mobile applications to operate on the same version of the original 3D city model data while still ensuring the advantages of a layered map approach for provisioning of virtual 3D city models.

Scaling

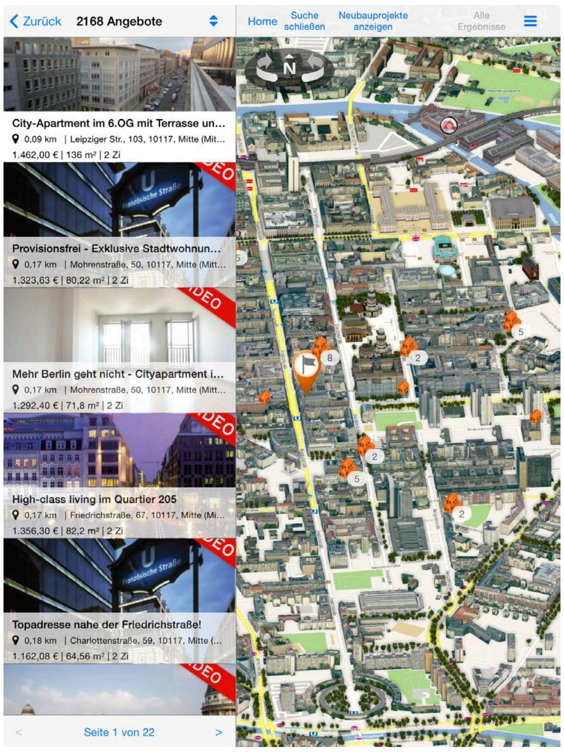
When the business consulting app was released to the public, it ways downloaded over 10,000 times within a 2 month period. A single server was used to host the required application data, i.e., app configuration files, POI data, and layered map tiles. The system could handle the large number of parallel users with no user-noticeable slowdown during map interaction. The combination of efficient tile organization and serving on server side and caching mechanisms on client side allowed a large number of users to browse the application in parallel.

Depending on the number of parallel tile requests, scaling of the server infrastructure for tile serving was achieved by applying simple techniques such as server-side DNS load balancing. More sophisticated techniques, e.g., a distribution of tile requests based on spatial coverage of tiles or dynamic scaling of request processing based on cloud computing instances could be implemented easily on server side when these simple techniques do not provide the necessary performance.

9.2.2 Layered Maps as Berlin Welcome Guide

A second case study in the context of the the BLC project was the implementation of a mobile welcome guide for people moving to Berlin. Instead of the business real estate API that was used in the app for business relocation, mobile application integrates data

[3]IETF RFC 7946 `https://tools.ietf.org/html/rfc7946`



City-Apartment im 6.OG mit Terrasse un...
0,09 km | Leipziger Str., 103, 10117, Mitte (Mit...
1.462,00 € | 136 m² | 2 Zi

Provisionsfrei - Exklusive Stadtwohnun...
0,17 km | Mohrenstraße, 50, 10117, Mitte (Mitt...
1.323,63 € | 80,22 m² | 2 Zi

Mehr Berlin geht nicht - Cityapartment i...
0,17 km | Mohrenstraße, 50, 10117, Mitte (Mitt...
1.292,40 € | 71,8 m² | 2 Zi

High-class living im Quartier 205
0,17 km | Friedrichstraße, 67, 10117, Mitte (Mi...
1.356,30 € | 82,2 m² | 2 Zi

Topadresse nahe der Friedrichstraße!
0,18 km | Charlottenstraße, 59, 10117, Mitte (...
1.162,08 € | 64,56 m² | 2 Zi

Figure 9.2: *Example of the application running on an iOS tablet integrating real estate offers into a layered map. It combines several data sources visually into the virtual 3D city model using a layered map.*

on apartments to rent or to buy from an external real estate portal in conjunction with the same feature data service mentioned before. It targets employees of companies that are planning to relocate or expand their businesses to Berlin. The application provides tools, in the form of layered maps and it rovides in-depth information about moving to Berlin as structured content that supplies information, links, and contacts.

Since the availability of real estate offers varies frequently, the offers and corresponding features are retrieved directly from the REST-API provided by the real estate portal. Their processing and display is implemented directly on client side. A filter mechanism was configured to reduce the real estate offers displayed on the layered map. The application used the same mechanism and framework implementation that was also used for the business consulting app. This example shows that the framework can be used for different apps that require different data to be integrated and displayed on the layered map. In particular, the layered maps can help users to gain an impression of a neighborhood and its social, economical, and educational services. The creation of app datasets could be performed efficiently by creating app configurations with the relevant contents.

9.2.3 Video Service for the Berlin Virtual 3D City Model

In addition to the provisioning of the Berlin virtual 3D city model for mobile apps, an instance of a video service as presented in Chapter 7 has been set up for the Berlin Partner BLC. The option to generate videos was an important feature for BLC. Videos are used to communicate information about locations, e.g., potential locations where in Berlin an investor's business could be set up. They are used in meetings with customers and also in presentations at trade fairs. As such video presentations of the virtual 3D city model had to be generated by trained operators in a manual process, their creation of used to be an expensive effort. Additionally, existing videos could not be regenerated easily if data has been updated in the meantime, i.e., if new buildings or more detailed models had been added.

Therefore, an instance of the system for video generation was deployed for the virtual 3D city model of Berlin for the BLC. The automated generation and their repeatability in different variants without requiring too many skills in 3D navigation has reduced the costs for video generation significantly. Thus, video assets can now be applied more easily and more frequently. For example, it is now possible for consultants to prepare videos prior to meetings or they can to provide videos to customers after a consulting session that recaps important parts of the meeting.

Implementation and Deployment

For the service, the server system for the service was configured to meet the special requirements of a rendering server for large-scale virtual 3D city models. Two distinct rendering server instances were created, one responsible for rendering the video frames in the video generator process and another one for rendering scene previews. Both service instances use the same configuration to assure that the same layers and image styles are available for preview and rendering. CPU, main memory, and IO resources are shared between these two instances. However, each instance has its own GPU assigned, since a parallel usage of the GPU for both processes results in insufficient GPU memory and affect the rendering result. The server hardware is operated in a data center to assure a high speed internet connection and general physical security and availability of the service. The amount of parallel users for this system was expected to be quite low (a total of approximately 500 users), yielding an infrequent usage pattern, so that a parallel use of more than 3 users is improbable. Therefore, the currently selected system capacity – a single system for video editor front-end and video generation back-end and datastore – is expected to be sufficient. Nevertheless, if necessary, the application could be easily scaled over a larger number of video generation nodes.

The system integrates POI data from a custom web service of the BLC. These POIs can be used to enrich videos with domain specific information. Furthermore, several planning models and maps (e.g., standard ground value maps, aerial photographs, and different street maps) are integrated into the rendering service that generates video frames and preview images as described in Chapter 7. This way, complex spatial information can be combined in video presentations using a simple web front-end.

Usage of the Service

The service was provided in a real-world context and was potentially accessible for the employees of Berlin Partner für Wirtschaft und Technologie GmbH and some of their selected partners. During the time of observation there were two larger use cases performed with this service.

- A video presentation introducing 10 investment locations was authored and rendered for a presentation at a real estate trade fair. The video provided a complex camera path covering the complete area of the virtual 3D city model. It included acceleration and deceleration phases and text overlay for the single locations.
- The Berlin police was provided with access to the service. It was used to generate a briefing video for a police operation during a demonstration. The officer was able to edit the camera path and to render corresponding videos without requiring any further instructions.

These applications show that the video service can be handled by untrained users to efficiently create complex video presentations after a short learning phase.

9.2.4 Health Prevention and Education for Berlin Neukölln

The Berlin district of Neukölln has a very heterogeneous population in terms of origin and culture. It has developed a variety of preventive health care options, child and parental education programs to support parents living in the district during their offsprings' childhood and adolescence. when compared to other districts of Berlin, Neukölln presents with a weaker social structure, especially in terms of income, level of education, unemployment rate, and health-related factors [Mei13]. Since many of the inhabitants have a relatively low educational level or inadequate German language skills, they cannot easily be reached with conventional outreach strategies, such as newspaper advertisements or specialized school programs. The district has a third party set up a system to gather and manage information about available offers for preventive health care and parenting support. It disseminates available offers using a web service API. To enable users to access this information about available offers, a mobile application that enables users to find those offers based on their location, e.g., in their direct neighborhood. For effective communication of these offers, a mobile application was designed that incorporates the user position and the web service API to allow users to find offers that fit their current needs.

Mobile Application

The mobile application based on layered maps was created using the framework presented in Chapters 4 and 6. End-user applications are based on the same client application framework used in the layered map applications for BLC. A different configuration for the framework and a different set of packaged application data was created. It contains layered map tiles, POIs, and informational content to serve the specific needs of the district of Neukölln regarding content availability and presentation.

The following functionality was implemented:

Offer Management and Presentation Offer records are obtained from a remote third party service. Their content and contacts are made available in a geospatial context using the layered map client framework. Offers can be searched, filtered, and arranged into lists.

3D City Model layered maps The project partner describe the target group of citizens as closely connected to their neighborhood. For that reason, the offer location of available offers plays a major role in their presentation.

Multimedia Content The application contains contextual information, e.g., financial support for families, general children health-care, or medical emergencies. The information was provided in form of text, structured content (contacts, web links, locations), and multimedia content (i.e., videos). Since the content changes frequently and should ideally be available offline, it is bundled within the app package an can only be updated together with the application.

9.2.5 OGC 3D Portrayal Interoperability Experiment

The OGC 3DPIE "is designed to test and demonstrate different approaches for service-based 3D portrayal based on the drafts for the candidate standards for 3D portrayal" [SHC12]. It was initiated by the Hasso-Plattner-Institut at the University of Potsdam, GIScience research group at the University of Heidelberg, and the Fraunhofer Institute for Computer Graphics Research. It involved different companies and organizations that applied and combined their particular solutions for service-based 3D portrayal to test and analyze the compatibility of data formats and service interfaces for different exemplary virtual 3D city models. Some of the experiments conducted during the experiment involved a combination of different visualization services that work on the candidate standard specifications for OGC W3DS and OGC WVS.

The IGN France[4], as one of the participating organizations, pro vided the Paris 3D dataset for conducting integration and visualization experiments. The dataset was analyzed and processed to be used by the 3D rendering service. A 3D rendering service instance was set up and the service endpoint was provided to the other participants of the interoperability experiment. This way, other parties could integrate images of the model in their own applications, e.g., by displaying these images as overlay in a 3D client at specific camera positions. Thus, they could apply image-based styling techniques that were not available in the particular client implementation by temporarily exchanging the rendered image with the one generated by the 3D rendering service.

The result of this experiments showed, that the service interface and implementation were capable of processing and rendering of the Paris model that contains textured terrain as well as building models within one CityGML dataset. The 3D rendering service was successfully used with client applications from different vendors, e.g., to apply styling effects or to integrate data on visualization level.

9.2.6 Nuremberg Demonstrator

As a research project collaboration with the Bayerische Vermessungsverwaltung (Bavarian surveying office), the Nuremberg virtual 3D city model was integrated into the 3D rendering service for testing web-based provisioning. A second aspect this project was to apply image-based techniques for image abstraction to textured virtual 3D city models. The technique was implemented using the tools and methods introduced by Semmo et al.

[4]Institut national de l'information géographique et forestière - National Institute of Geographic and Forest Information of France, `http://www.ign.fr`

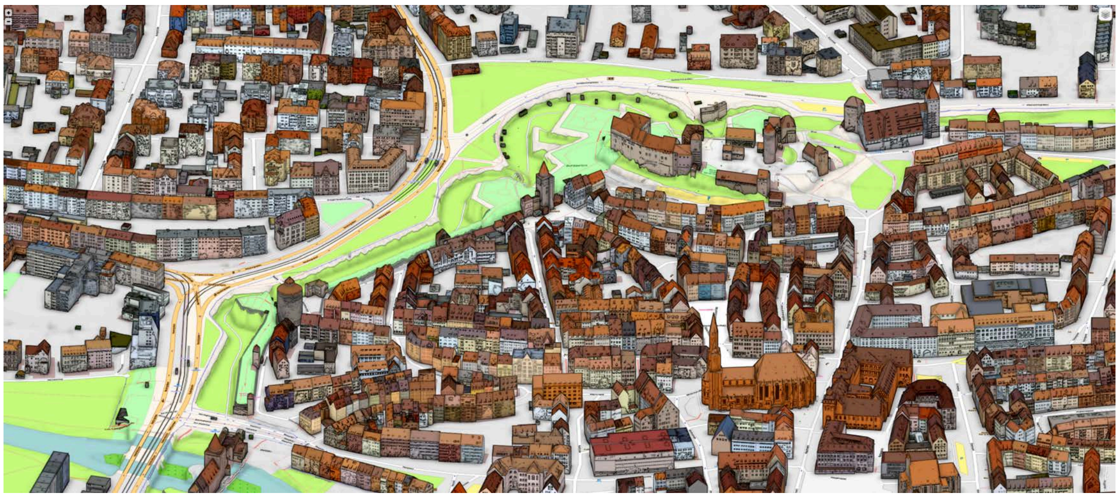


Figure 9.3: *Web-based layered map showing the virtual 3D city model of Nuremberg with texture abstraction applied.*

in 2015 [Sem+15] for processing virtual 3D city model textures. Image-based abstraction has been implemented as preprocessing step for texture files and CityGML documents creating an additional CityGML appearance definition for the buildings of the virtual 3D city model based on the abstracted image textures. For the purpose of demonstrating it to the surveying office, the provisioning system was used to create a layered map, a mobile application, and a cube-map based rendering application as technology demonstration. An example of a web-based layered map showing the buildings with texture abstraction applied can be found in Fig. 9.3.

9.2.7 Geneva Layered Map

In 2014, an example of web-based provisioning as layered maps was developed for the 2014 "3D Geospatial Geneva Showcase"[5], held in the context of the June 2014 Technical Committee Meeting of the OGC. The foundation of the application data was a heterogeneous virtual 3D city model (see Tab. 9.1) along with a WFS that provided feature information. The virtual 3D city model was imported into the 3D rendering service, and a layered map service was used to generate a tile dataset for the Geneva area with a color layer as well as an object id layer. As key developmental step, an Android mobile application based on the layered map framework (see Fig. 9.4) was created. It was configured to use a set of POI features, which had previously been extracted from OpenStreetMap data using the feature data provisioning service. Additionally, the application used an object id map layer to fetch object ids for specific positions and query feature data and attributes from the given WFS if the user touches a building depiction on the map. This demonstration, in particular, illustrated that background feature data for those features portrayed on the map can be easily accessed using standard service APIs.

The complete processing, configuration, and provisioning of the Android application was performed in a fast and cost efficient way by using the implemented services and the

[5] `https://www.perey.com/3DGVA/event/`

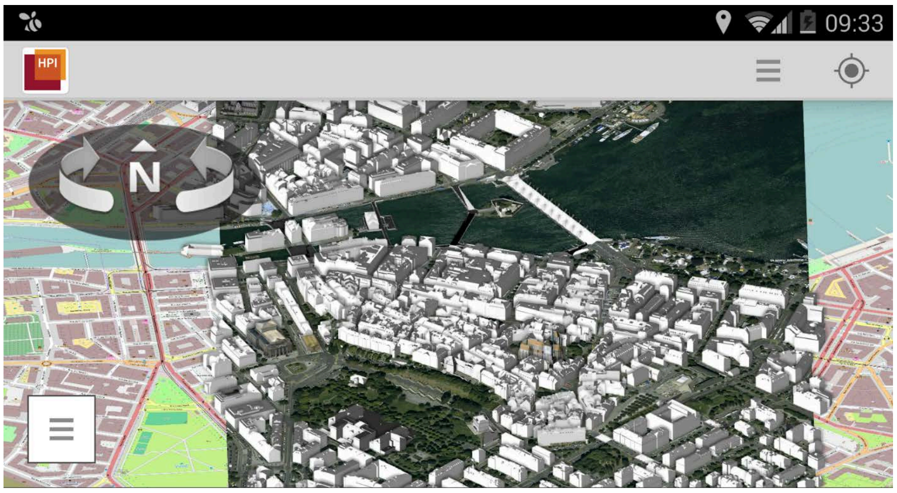


Figure 9.4: *Layered map application for the virtual 3D city model of Geneva running on an Android phone.*

client application framework for layered maps. Manual configuration and administrative effort is thereby limited to just a few hours. This shows how the service concepts and techniques introduced in this thesis can be used to create low-cost, robust, and standards-based applications for communication of 3D geoinformation based on the presented service concepts.

9.2.8 Roman Cologne

The model of the Roman Cologne was imported into the 3D rendering service to confirm its applicability for high-detail models that were not provided in semantics-driven formats. The 3D geometry was analyzed and optimized by the data import service, its assigned textures were extracted and reorganized, and feature data (object names, ids, and their position within the overall scene hierarchy) were extracted and stored in the feature database.

A 3D rendering service instance was set up to serve generally two visualization applications:

1. An image-based client for exploration of the model on mobile devices [Kli^{+}14] enabling in-situ exploration of the complex 3D model. See Fig. 9.5 for an example.

2. A 3D layered map-based client for web browsers.

The service was able to handle this type of general-purpose computer graphic models even though it contains massive geometry and texture data and does not provide a spatial reference system. Importing and 3D rendering of this complex model created the possibility of also accessing it on mobile devices and web-browsers which had not been possible before.

9.2.9 Rotterdam

The 3D rendering service was also evaluated with the virtual 3D city model of Rotterdam as it provides a prototypical example of an open data dataset. This dataset

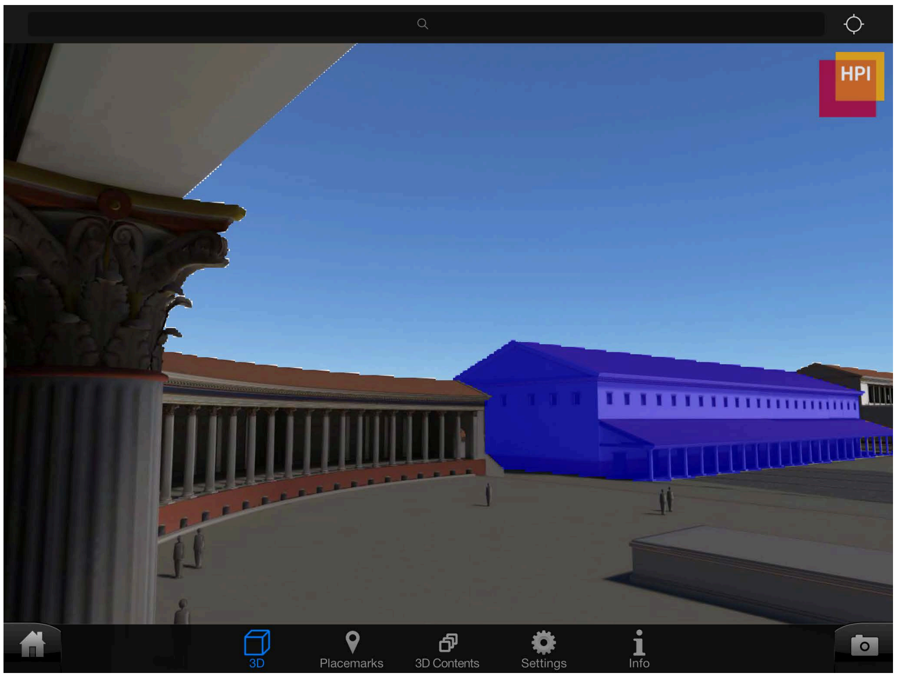


Figure 9.5: *The image-based application showing the highly detailed model of the Roman Cologne running on an iOS tablet. A particular feature is selected and highlighted using the rendered object ID layer.*

was particularly challenging for the data import service implementation as the model contained flaws in terms of geometry and data modelling. A consistent result of model preprocessing and optimization is crucial for providing homogeneous rendering result. For example, an inconsistent orientation of surface normals or degenerated faces in the geometry can easily lead to graphical artifacts during 3D rendering with many shading techniques. Thus, model preprocessing needs to handle model flaws in a way that assures a homogeneous rendering result with different models and use cases. For the test use case, the 3D rendering service was extended with repair functionality to cope with such issues in imperfect datasets. With this functionality, the robustness of the service could be improved and validated.

9.3 Hardware Requirements

All of the aforementioned case studies were successful on a range of scientific and commercial projects. Image generation was practically feasible using a rendering server system equipped with moderate server hardware. A typical system includes an 8 core CPU (4 cores with hyper-threading), 64 GB RAM, 3 GB consumer grade GPU, and SSD-based storage). The amount of main memory that is required for running a 3D rendering service instance varied and can be configured to match the available main memory by reducing cache sizes for texture and geometry caching. In general, the virtual texturing approach for efficient texture handling and the compact geometry representation generated by the geometry preprocessing proved to be applicable up to the largest model within the range of tested models. Accordingly, the are currently no imminent limitations for the size of models that can be rendered by the rendering service, as the most immediate restriction of model sizes would arise from memory and processing capacity of today's GPUs, which, however are continuously increasing. Consumer GPUs with 12 GB and professional-grade hardware with more than 16 GB of GPU memory is already available. The Berlin 3D city model, which is one of the largest and most detailed virtual 3D city model datasets currently available, could be successfully processed and rendered with the above-mentioned hardware specs. The concepts for image-based provisioning, layered maps, and service-based video generation have proven their ability to massively reduce the client-side complexity and requirements for client applications.

9.4 Conclusions from Case Studies

The case studies and applications presented in this chapter proof the applicability of the general image-based provisioning approach based on a common framework. Several examples illustrated that the approach and corresponding services can handle the provisioning of virtual 3D city models of different data size, complexity, and quality. Furthermore, it was shown that the approach supports efficient, automated provisioning of applications using virtual 3D city models with only minor preparation time.

The applications were distributed to a large number of end users since there were no specific requirements for 3D rendering hardware on the target devices. Their platform-specific implementation, currently iOS, Android, and HTML5 environments, can be used to communicate spatial and also non-spatial information for different application purposes.

Chapter 10

Summary and Outlook

10.1 Summary

Virtual 3D city models are evolving into a fundamental category of geospatial models. In particular, they form a unique source of complex geoinformation in many application fields in industry and administration. From a technical perspective, these models have become increasingly more complex and massive, e.g., due to improved remote sensing technologies and 3D reconstruction technologies. The image-based provisioning approaches presented in this thesis substantially extend common visualization approaches for virtual 3D city models. In particular, these services provide key benefits such as decoupling the complexity of a model from its visualization complexity or the increased reusability of implementation for rendering techniques and interaction techniques. The approach supports the key IT paradigm of SOC and leads to a software architecture that can be implemented straightforward with a SOA.

This thesis presents concepts and techniques for web-based provisioning of and interaction with virtual 3D city models. The corresponding service implementations show the feasibility of these concepts and offer a foundation for developing future service-based applications and systems. In particular, the thesis provides a 3D rendering service (based on the OGC 3DPS standard) specifically designed to support massive virtual 3D city model with their typically large amounts of geometry data and texture data. Furthermore, the rendering service provides extensive styling capabilities that support thematic as well as artistic visualization of virtual 3D city models by taking advantage of general purpose GPU computing capabilities. The web-based provisioning techniques allow us to build IT systems and applications that provide thematic visualization for large-scale virtual 3D city models without transferring the complete set of required geometries, textures, and thematic attributes to clients. This approach saves a significant amount of data to be transferred and processed on client side and thus provides an efficient, robust mechanism for thematic visualization applications can be provided for a diversity of end user devices.

This thesis also introduces layered maps, which provide an efficient and robust approach for web-based 3D maps to handle massive virtual 3D city models. They provide a map-like user interface, offer simple means to navigate through these models, have short initial loading times, avoid performance problems regarding 3D rendering, and can be deployed as simple web services for nearly any user device. Layered maps allow us to integrate virtual 3D city models into existing standard web-based applications or portals in a lightweight manner. In particular, the object ID layer can be used to establish a bidirectional communication on a feature basis with other components of the a web-page. Feature selection events can be emitted by the layered map client component or selection events from other components can be consumed, and selected features can then be rendered differently. This way, the system is able to connect selected features within the 3D space of a virtual 3D city model with additional information, e.g., provided by external services.

The illustrated service for video authoring and video generation addresses a practical need that has been observed for many applications using virtual 3D city models. Rendering videos usually requires large efforts and manual labor. The automated video service reduces these costs significantly and enables us to efficiently render video assets that can be automatically regenerated when necessary to reflect changes in the underlying virtual 3D city model.

As a novel category of services in the area of service-based visualization of virtual 3D city models, interaction services offer building blocks for a service-based interaction support that allows us to implement reusable, component-based camera interaction techniques. Consequently, 3D clients as well as higher level services (e.g., the web-based video editor presented in Chapter 7) can make use of existing techniques for camera path planning, input evaluation, or navigation command recognition leading to reduced development costs for application development and, additionally, to a more homogeneous user experience across different applications.

Design and implementation of geo-oriented systems and applications are continuously evolving. Scalable IT solutions based on microservices, as "small, autonomous services that work together" [New15], are gaining growing attention in research and commercial applications [Krä18; Erl16; FDA18; Che+17; Bra+17]. "Currently, microservice architecture (MSA) is maturing as an architectural style for distributed software systems with high requirements for scalability and adaptability" [RSS18]. This thesis contributes to this objective by introducing a functional decomposition of geovisualization systems into components for visualization, provisioning, and user interaction that can be implemented as independent microservices.

The concepts and techniques introduced in this thesis extend the application scope of virtual 3D city models as a tool for communication of spatial and non-spatial information. They were evaluated and their practical applicability was demonstrated in a variety of visualization projects using virtual 3D city models.

10.2 Outlook

The approaches for web-based 3D rendering, provisioning of and interaction with virtual 3D city models represent a key step for future image-based technology for virtual 3D city models.

Virtual 3D city models can be used as a central information base for city-related, georeferenced data in general. For example, base data and real-time data related to infrastructure status, traffic flow, or environmental conditions could be integrated. Such sensors will become a vital data source that will drive virtual 3D city models. Real-time, streamed data has equally large potential as a basis for thematic visualization. Future research could address this issue to connect 3D visualization of virtual 3D city models with corresponding real-time data sources. Integrating this into an image-based visualization process would require the definition and implementation of new, standardized communication protocols, server-side and client-side rendering and data integration techniques as well as user interfaces for configuring service requests to include this data into the visualization process.

In recent years, developments in virtualization techniques for GPU computing in recent years have significantly reduced the complexity of building remote rendering solutions. GPU-enabled servers or virtual machine instances are available at all major vendors and data centers, such as Amazon Web Services, Microsoft Azure, or Google Cloud. This suggests a promising development for remote rendering solutions, as the complexity for deployment and scaling (i.e., creating new rendering service instances) is massively reduced. Those developments also makes it significantly easier and more cost-efficient to setup, deploy, and scale image-based solutions, e.g., for service-based live rendering of virtual 3D city models, for generation of layered map tiles, or for generation of videos. Consequently the hurdles for implementing image-based approaches, such as the 3D rendering and provisioning approaches suggested in this thesis, are significantly lower. This opens up new areas of application that could be addressed by such image-based solutions.

Another research direction would be to identify and classify specific use cases for the different technological approaches for visualization of virtual 3D city models, i.e., the geometry-based approach (e.g., implemented in the Cesium framework or by scene-based 3DPS), and the image-based approach used in this thesis. Future work could also compare the usability and interaction methods for specific classes of use case or evaluate the effectiveness of communication of spatial and non-spatial information in the different approaches. Future research could also explore which type of presentation of a virtual 3D city model meets the requirements for a specific use case best, e.g., in terms of accessibility, comprehensibility, or user orientation.

The services presented in this thesis, in particular the 3D rendering service and the video service, provide feature-rich implementations of high-level visualization components for virtual 3D city model applications, making them are a good starting point to add value to existing geospatial infrastructures and services. For example, a service that creates multimedia presentations for virtual 3D city models and their connected data could be built that combines video fragments, images generated by a 3DPS, maps, charts, etc. and incorporates them into multimedia presentations that can be easily distributed.

Finally, the services for image-based provisioning of complex 3D models can also support the increasing digitization in different industries. As more and more products, machines, and buildings are designed, produced, and maintained using digital 3D models, e.g., as a part of a “digital twin”, the use of these models in all stages of the product life cycle is increasingly gaining importance. In this context, image-based provisioning services can help to handle, visualize, and communicate the increasing amount of data created, e.g., for smart products or digital production processes. Furthermore, future research activities could focus on transferring the concepts and techniques proposed for the provisioning of virtual 3D city models to other areas, such as product design and construction, facility management, machine engineering, or big data visualization and analytics in general.

Bibliography

[Aga$^+$10] S. Agarwal, Y. Furukawa, N. Sna, B. Curless, S. M. Seitz, and R. Szeliski. “Reconstructing Rome”. In: *Computer* 43.6 (June 2010), pp. 40–47. ISSN: 0018-9162. DOI: 10.1109/MC.2010.175. URL: http://ieeexplore.ieee.org/document/5481934/.

[AK03] A. Altmaier and T. Kolbe. “Applications and Solutions for Interoperable 3d Geo-Visualization”. In: *Proceedings of the Photogrammetric Week*. 2003, pp. 253–267. URL: http://www.ifp.uni-stuttgart.de/publications/phowo03/kolbe.pdf.

[Ake$^+$18] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire. *Real-Time Rendering*. 4rd. Natick, MA, USA: A. K. Peters, CRC Press, 2018, p. 1200. ISBN: 9781-1386-2700-0.

[AL13] J. Ardouin and A. Lécuyer. “Navigating in Virtual Environments with 360 ∘ Omnidirectional Rendering”. In: *3Dui 2013* (2013), pp. 95–98. URL: http://hal.archives-ouvertes.fr/hal-00818277/.

[Ala$^+$14] N. Alam, D. Wagner, M. Wewetzer, J. von Falkenhausen, V. Coors, and M. Pries. “Towards Automatic Validation and Healing of CityGML Models for Geometric and Semantic Consistency”. In: *Lecture Notes in Geoinformation and Cartography*. Vol. II. November. 2014, pp. 77–91. ISBN: 9783319005157. DOI: 10.1007/978-3-319-00515-7_5. URL: http://link.springer.com/10.1007/978-3-319-00515-7_5.

[AMH13] F. Albrecht, J. Moser, and I. Hijazi. “Assessing Façade Visibility in 3D City Models for City Marketing”. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* XL-2/W2.November (2013), pp. 1–5. ISSN: 16821750. DOI: 10.5194/isprsarchives-XL-2-W2-1-2013.

[AZM00] M. Agrawala, D. Zorin, and T. Munzner. “Artistic Multiprojection Rendering”. In: *Eurographics Rendering Workshop 2000* (2000), pp. 1–13. DOI: http://doi.acm.org/10.1145/647652.732127.

[Bas$^+$08] J. Basanow, P. Neis, S. Neubauer, A. Schilling, and A. Zipf. “Towards 3D Spatial Data Infrastructures (3D-SDI) based on open standards - experiences, results and future issues”. In: *Advances in 3D Geoinformation Systems*. Ed. by P. van Oosterom, S. Zlatanova, F. Penninga, and E. M. Fendel. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 65–86. ISBN: 978-3-540-72134-5. DOI: 10.1007/978-3-540-72135-2. URL: http://www.springerlink.com/index/10.1007/978-3-540-72135-2.

[Bat+10] M. Batty, A. Hudson-Smith, R. Milton, and A. Crooks. “Map mashups, Web 2.0 and the GIS revolution”. In: *Annals of GIS* 16.1 (Apr. 2010), pp. 1–13. ISSN: 1947-5683. DOI: 10.1080/19475681003700831. URL: http://www.tandfonline.com/doi/abs/10.1080/19475681003700831.

[Bau18] P. Baumann. *OGC Web Coverage Service (WCS) 2.1 Interface Standard - Core*. 2018. URL: http://docs.opengeospatial.org/is/17-089r1/17-089r1.html.

[BBD05] H. Buchholz, J. Bohnet, and J. Dollner. “Smart and Physically-Based Navigation in 3D Geovirtual Environments”. In: *9th International Conference on Information Visualisation (IV’05)*. IEEE, 2005, pp. 629–635. ISBN: 0-7695-2397-8. DOI: 10.1109/IV.2005.117. URL: http://ieeexplore.ieee.org/document/1509140/.

[BD05] H. Buchholz and J. Dollner. “View-Dependent Rendering of Multiresolution Texture-Atlases”. In: *IEEE Visualization 2005 - (VIS’05)*. IEEE, 2005, pp. 28–28. ISBN: 0-7803-9462-3. DOI: 10.1109/VIS.2005.112. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1566010.

[BD07] D. Brutzman and L. Daly. *X3D: Extensible 3D Graphics for Web Authors*. Elsevier, 2007. ISBN: 978-0-12-088500-8.

[Beh+09] J. Behr, P. Eschler, Y. Jung, and M. Zöllner. “X3DOM – A DOM-based HTML5/ X3D Integration Model”. In: *Proceedings of the 14th International Conference on 3D Web Technology - Web3D ’09*. Vol. 1. 212. New York, New York, USA: ACM Press, 2009, p. 127. ISBN: 9781605584324. DOI: 10.1145/1559764.1559784. URL: http://portal.acm.org/citation.cfm?doid=1559764.1559784.

[Ben75] J. L. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 00010782. DOI: 10.1145/361002.361007. URL: http://portal.acm.org/citation.cfm?doid=361002.361007.

[BF08] M. Barnes and E. L. Finch, eds. *COLLADA – Digital Asset Schema Release 1.5.0*. Aug. 2008. ISBN: 978-1565921610. DOI: 10.1109/MM.1996.526933. URL: http://ieeexplore.ieee.org/document/526933/.

[Bil+15] F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. “Applications of 3D City Models: State of the Art Review”. In: *ISPRS International Journal of Geo-Information* 4.4 (2015), pp. 2842–2889. ISSN: 2220-9964. DOI: 10.3390/ijgi4042842. URL: http://www.mdpi.com/2220-9964/4/4/2842.

[Bil+16] F. Biljecki, K. A. Ohori, H. Ledoux, R. Peters, and J. Stoter. “Population estimation using a 3D City Model: A multi-scale country-wide study in the Netherlands”. In: *PLoS ONE* 11.6 (2016), pp. 1–22. ISSN: 19326203. DOI: 10.1371/journal.pone.0156808. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0156808.

[BKN18] F. Biljecki, K. Kumar, and C. Nagel. “CityGML Application Domain Extension (ADE): overview of developments”. In: *Open Geospatial Data, Software and Standards* 3.1 (Dec. 2018), p. 13. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0055-6. URL: https://opengeospatialdata.springeropen.com/articles/10.1186/s40965-018-0055-6.

[BLS17] F. Biljecki, H. Ledoux, and J. Stoter. “Does a Finer Level of Detail of a 3D City Model Bring an Improvement for Estimating Shadows?” In: *Advances in 3D Geoinformation: Lecture Notes in Geoinformation and Cartography.* Ed. by A. Abdul-Rahman. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 31–47. ISBN: 978-3-642-12669-7. DOI: 10.1007/978-3-319-25691-7_2. URL: http://link.springer.com/10.1007/978-3-642-12670-3.

[Blu+11] A. Blume, W. Chun, D. Kogan, V. Kokkevis, N. Weber, R. W. Petterson, and R. Zeiger. “Google Body: 3D Human Anatomy in the Browser”. In: *ACM SIGGRAPH 2011 Talks on - SIGGRAPH ’11.* New York, New York, USA: ACM Press, 2011, p. 1. ISBN: 9781450309745. DOI: 10.1145/2037826.2037852. URL: http://dl.acm.org/citation.cfm?doid=2037826.2037852.

[Bon+18] X. Bonaventura, M. Feixas, M. Sbert, L. Chuang, and C. Wallraven. “A Survey of Viewpoint Selection Methods for Polygonal Models”. In: *Entropy* 20.5 (May 2018), p. 370. ISSN: 1099-4300. DOI: 10.3390/e20050370. URL: http://www.mdpi.com/1099-4300/20/5/370.

[Bra+16] M. Brasebin, S. Christophe, F. Jacquinod, A. Vinesse, and H. Mahon. “3D Geovisualization & Stylization to Manage Comprehensive and Participative Local Urban Plans”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1.October (2016), pp. 83–91. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-2-W1-83-2016. URL: http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W1/83/2016/.

[Bra+17] E. Braun, C. Düpmeier, D. Kimmig, W. Schillinger, and K. Weissenbach. “Generic Web Framework for Environmental Data Visualization”. In: *Advances and New Trends in Environmental Informatics.* Ed. by V. Wohlgemuth, F. Fuchs-Kittowski, and J. Wittmann. Cham: Springer, 2017, pp. 289–299. ISBN: 978-3-319-44711-7. DOI: 10.1007/978-3-319-44711-7_23. URL: http://link.springer.com/10.1007/978-3-319-44711-7_23.

[Bre00] C. Brenner. “Towards fully automatic generation of city models”. In: *International Archives of Photogrammetry and Remote Sensing.* Vol. XXXIII Par. 2000, pp. 85–92. URL: http://www.isprs.org/proceedings/XXXIII/congress/part3/84_XXXIII-part3.pdf.

[Bro+92] K. W. Brodlie, J. R. Gallop, C. D. Osland, L. A. Carpenter, R. J. Hubbold, P. Quarendon, R. A. Earnshaw, and A. M. Mumford, eds. *Scientific Visualization.* Springer Berlin Heidelberg, 1992. ISBN: 978-3-642-76944-3. DOI: 10.1007/978-3-642-76942-9. URL: http://link.springer.com/10.1007/978-3-642-76942-9.

[Bur15] D. Burggraf. *OGC KML 2.3*. 2015. URL: http://www.opengeospatial.org/standards/kml.

[Bus+14] S. Buschmann, M. Trapp, P. Luhne, and J. Dollner. “Hardware-Accelerated Attribute Mapping for Interactive Visualization of Complex 3D Trajectories”. In: *International Conference on Information Visualization Theory and Applications (IVAPP) 2014*. January. 2014, pp. 356–363. ISBN: 9789897580055.

[Cap+12] C. Cappelle, M. E. El Najjar, F. Charpillet, and D. Pomorski. “Virtual 3D City Model for Navigation in Urban Areas”. In: *Journal of Intelligent & Robotic Systems* 66.3 (May 2012), pp. 377–399. ISSN: 0921-0296. DOI: 10.1007/s10846-011-9594-0. URL: http://link.springer.com/10.1007/s10846-011-9594-0.

[CB09] J. Chen and D. a. Bowman. “Domain-Specific Design of 3D Interaction Techniques: An Approach for Designing Useful Virtual Environment Applications”. In: *Presence: Teleoperators and Virtual Environments* 18.5 (Oct. 2009), pp. 370–386. ISSN: 1054-7460. DOI: 10.1162/pres.18.5.370. URL: http://www.mitpressjournals.org/doi/abs/10.1162/pres.18.5.370.

[Cha+17] K. Chaturvedi, B. Willenborg, M. Sindram, and T. H. Kolbe. “Solar Potential Analysis and Integration of the Time-Dependent Simulation Results for Semantic 3D City Models using Dynamizers”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. IV-4/W5. 4W5. Oct. 2017, pp. 25–32. DOI: 10.5194/isprs-annals-IV-4-W5-25-2017. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W5/25/2017/.

[Che+17] G. Cherradi, A. El Bouziri, A. Boulmakoul, and K. Zeitouni. “Real-Time HazMat Environmental Information System: A micro-service based architecture”. In: *Procedia Computer Science* 109.June (2017), pp. 982–987. ISSN: 1877-0509. DOI: 10.1016/j.procs.2017.05.457. URL: http://linkinghub.elsevier.com/retrieve/pii/S1877050917311420.

[Cle93] W. Cleveland. *Visualizing Data*. Hobart Press, 1993. ISBN: 0963488406.

[CN15] M. Christen and S. Nebiker. “Visualisation of Complex 3D City Models on Mobile Webbrowsers Using Cloud-Based Image Provisioning”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3/W5 (2015), pp. 517–522. ISSN: 2194-9050. DOI: 10.5194/isprsannals-II-3-W5-517-2015. URL: http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-3-W5/517/2015/.

[Con+11] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz. “Interactive visualization of volumetric data with WebGL in real-time”. In: *Proceedings of the 16th International Conference on 3D Web Technology - Web3D ’11*. New York, New York, USA: ACM Press, 2011, p. 137. ISBN: 9781450307741. DOI: 10.1145/2010425.2010449. URL: http://dl.acm.org/citation.cfm?doid=2010425.2010449.

[CON08] M. Christie, P. Olivier, and J.-M. Normand. “Camera Control in Computer Graphics”. In: *Computer Graphics Forum* 27.8 (Dec. 2008), pp. 2197–2218. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2008.01181.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2008.01181.x.

[Cru+92] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. “The CAVE: Audio Visual Experience Automatic Virtual Environment”. In: *Communications of the ACM* 35.6 (June 1992), pp. 64–72. ISSN: 00010782. DOI: 10.1145/129888.129892. URL: http://portal.acm.org/citation.cfm?doid=129888.129892.

[CXL17] K. Chen, F. Xue, and W. Lu. “Development of 3D Building Models Using Multi-Source Data: A Study of High-Density Urban Area in Hong Kong”. In: *Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction.* July. Edinburgh: Heriot-Watt University, July 2017, pp. 609–616. ISBN: 978-0-9565951-6-4. DOI: 10.24928/JC3-2017/0252. URL: http://itc.scix.net/cgi-bin/works/Show?_id=lc3-2017-252.

[Dal+13] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein. “Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools”. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC).* IEEE, July 2013, pp. 323–328. ISBN: 978-1-4673-2480-9. DOI: 10.1109/IWCMC.2013.6583580. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6583580.

[DBB06] J. Döllner, K. Baumann, and H. Buchholz. “Virtual 3D City Models as Foundation of Complex Urban Information Spaces”. In: *CORP, Vienna.* Ed. by M. Schrenk. Vol. 2. 11. 2006, pp. 107–112. ISBN: 978-3-9502139-0-4.

[DC98] A. Doyle and A. Cuthbert. *OpenGIS Project Document 98-061: Essential Model of Interactive Portrayal.* 1998.

[Dem+16] E. Demir Ozbek, S. Zlatanova, S. Ates Aydar, and T. Yomralioglu. “3D GEO-INFORMATION REQUIREMENTS FOR DISASTER AND EMERGENCY MANAGEMENT”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences.* Vol. XLI-B2. July. June 2016, pp. 101–108. DOI: 10.5194/isprsarchives-XLI-B2-101-2016. URL: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B2/101/2016/isprs-archives-XLI-B2-101-2016.pdf.

[DHK12] J. Doellner, B. Hagedorn, and J. Klimke. “Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps”. In: *Proceedings of the 17th International Conference on 3D Web Technology - Web3D ’12.* Web3D ’12. New York, New York, USA: ACM Press, 2012, p. 97. ISBN: 9781450314329. DOI: 10.1145/2338714.2338729. URL: http://dl.acm.org/citation.cfm?doid=2338714.2338729.

[DHS05] J. Döllner, B. Hagedorn, and S. Schmidt. “An Approach towards Semantics-Based Navigation in 3D City Models on Mobile Devices”. In: *Location Based Services and TeleCartography.* Berlin, Heidelberg: Springer Berlin

Heidelberg, 2005, pp. 357–368. DOI: 10.1007/978-3-540-36728-4_27. URL: http://link.springer.com/10.1007/978-3-540-36728-4_27.

[DK14] J. Dambruch and M. Krämer. "Leveraging public participation in urban planning with 3D web technology". In: *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*. New York, New York, USA: ACM Press, 2014, pp. 117–124. ISBN: 9781450330152. DOI: 10.1145/2628588.2628591. URL: http://dl.acm.org/citation.cfm?id=2628588.2628591.

[DMK05] J. Dykes, A. MacEachren, and M.-J. Kraak, eds. *Exploring Geovisualization*. Elsevier, 2005, p. 730.

[Döl+06] J. Döllner, T. H. Kolbe, F. Liecke, T. Sgouros, and K. Teichmann. "The Virtual 3D City Model of Berlin - Managing, Integrating, and Communicating Complex Urban Information". In: *25th Urban Data Management Symposium (UDMS)*. May. Aalborg, 2006.

[Düb+11] S. Dübel, M. Röhlig, C. Tominski, and H. Schumann. "Visualizing 3D Terrain, Geo-Spatial Data, and Uncertainty". In: *HVAC&R Research*. Vol. 17. 4. Feb. 2011, pp. 526–539. DOI: 10.3390/informatics4010006. URL: http://www.mdpi.com/2227-9709/4/1/6.

[ED09] J. Engel and J. Döllner. "Approaches Towards Visual 3D Analysis for Digital Landscapes and Its Applications". In: *Digital Landscape Architecture Proceedings 2009*. Wichmann, 2009, pp. 33–41. ISBN: 978-3-87907-491-4.

[ElK+17] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba. "Taxonomy of Cross-Platform Mobile Applications Development Approaches". In: *Ain Shams Engineering Journal* 8.2 (June 2017), pp. 163–190. ISSN: 20904479. DOI: 10.1016/j.asej.2015.08.004. URL: http://linkinghub.elsevier.com/retrieve/pii/S2090447915001276.

[Ell94] S. Ellis. "What are virtual environments?" In: *IEEE Computer Graphics and Applications* 14.1 (Jan. 1994), pp. 17–22. ISSN: 0272-1716. DOI: 10.1109/38.250914. URL: http://ieeexplore.ieee.org/document/250914/.

[Erl16] T. Erl. *Service-Oriented Architecture: Analysis and Design for Services and Microservices*. Ed. by G. Wiegand. 2nd. Prentice Hall, 2016. ISBN: 9780133858587.

[Eva+14] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, and J. Blat. "3D Graphics on the Web: a Survey". In: *Computers & Graphics* 41 (June 2014), pp. 43–61. ISSN: 00978493. DOI: 10.1016/j.cag.2014.02.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0097849314000260.

[FDA18] A. Fricke, J. Döllner, and H. Asche. "Servicification – Trend or Paradigm Shift in Geospatial Data Processing?" In: *Computational Science and Its Applications – ICCSA 2018*. Ed. by D. Taniar, O. Gervasi, B. Murgante, E. Pardede, and B. O. Apduhan. Vol. 6018. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 339–350. ISBN: 978-3-642-12178-4. DOI: 10.1007/978-3-319-95168-3_23. URL: http://link.springer.com/10.1007/978-3-319-95168-3_23.

[Fer$^+$15] N. Ferreira, M. Lage, H. Doraiswamy, H. Vo, L. Wilson, H. Werner, M. Park, and C. Silva. “Urbane: A 3D framework to support data driven decision making in urban development”. In: *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, Oct. 2015, pp. 97–104. ISBN: 978-1-4673-9783-4. DOI: 10.1109/VAST.2015.7347636. URL: http://ieeexplore.ieee.org/document/7347636/.

[Fie00] R. T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis. University of California, Irvine, 2000, p. 162. ISBN: 0-599-87118-0. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[Fit$^+$08] G. Fitzmaurice, J. Matejka, I. Mordatch, A. Khan, and G. Kurtenbach. “Safe 3D navigation”. In: *Proceedings of the 2008 symposium on Interactive 3D graphics and games - SI3D ’08*. Vol. 1. 212. New York, New York, USA: ACM Press, 2008, p. 7. ISBN: 9781595939838. DOI: 10.1145/1342250.1342252. URL: http://portal.acm.org/citation.cfm?doid=1342250.1342252.

[FM01] S. Fuhrmann and A. MacEachren. “Navigation in Desktop Geovirtual Environments: Usability Assessment”. In: *Proceedings 20th ICA/ACI International Cartographic Conference*. 2001, pp. 2444–2453.

[GAW09] I. J. Grimstead, N. J. Avis, and D. W. Walker. “RAVE: The Resource-Aware Visualization Environment”. In: *Concurrency and Computation: Practice and Experience* 21.4 (2009), pp. 415–448. ISSN: 15320626. DOI: 10.1002/cpe.1327. URL: http://doi.wiley.com/10.1002/cpe.1327.

[Gee$^+$05] A. G. Gee, M. Yu, H. Li, and G. G. Grinstein. *Dynamic and Interactive Dimensional Anchors for Spring-Based Visualizations*. Tech. rep. Lowell, MA: University of Massachusetts, Lowell, 2005, pp. 1–11. URL: http://www.cs.uml.edu/~agee/publications/cs2005/cs2005.pdf.

[Gla$^+$13] T. Glander, A. Moreno, M. Aristizabal, J. Congote, J. Posada, A. Garcia-Alonso, and O. Ruiz. “ReWeb3D”. In: *Proceedings of the 18th International Conference on 3D Web Technology - Web3D ’13*. 1. New York, New York, USA: ACM Press, 2013, p. 147. ISBN: 9781450321334. DOI: 10.1145/2466533.2466535. URL: http://dl.acm.org/citation.cfm?doid=2466533.2466535.

[Gol$^+$12] J. F. Golding, K. Doolan, A. Acharya, M. Tribak, and M. A. Gresty. “Cognitive Cues and Visually Induced Motion Sickness”. In: *Aviation, Space, and Environmental Medicine* 83.5 (May 2012), pp. 477–482. ISSN: 00956562. DOI: 10.3357/ASEM.3095.2012. URL: http://openurl.ingenta.com/content/xref?genre=article&issn=0095-6562&volume=83&issue=5&spage=477.

[Goo$^+$01] B. Gooch, E. Reinhard, C. Moulding, and P. Shirley. “Artistic Composition for Image Creation”. In: *Eurographics Workshop on Rendering Techniques*. Ed. by S. Gortler and K. Myszkowski. Eurographics. Springer, 2001, pp. 83–88. ISBN: 978-3-7091-6242-2. DOI: 10.1007/978-3-7091-6242-2_8. URL: http://www.springerlink.com/index/10.1007/978-3-7091-6242-2_8.

[GR15] J. Geelhaar and G. Rausch. “3D Web Applications in E-Commerce - A Secondary Study on the Impact of 3D Product Presentations Created with HTML5 and WebGL”. In: *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*. IEEE, June 2015, pp. 379–382. ISBN: 978-1-4799-8679-8. DOI: 10.1109/ICIS.2015.7166623. URL: http://ieeexplore.ieee.org/document/7166623/.

[Gri+05] I. J. Grimstead, N. J. Avis, D. W. Walker, and R. N. Philp. “Resource-Aware Visualization Using Web Services”. In: *In Proceedings of the UK e-Science All Hands Meeting 2005 - Innovation through e-Science*. 2005. ISBN: 1904425534. URL: http://www.allhands.org.uk/2005/proceedings/papers/306.pdf.

[Grö+12] G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele. *OGC City Geography Markup Language (CityGML) Encoding Standard*. 2012. URL: http://www.opengeospatial.org/standards/citygml.

[Gut+16] R. Gutbell, L. Pandikow, V. Coors, and Y. Kammeyer. “A framework for server side rendering using OGC’s 3D portrayal service”. In: *Proceedings of the 21st International Conference on Web3D Technology - Web3D ’16*. New York, New York, USA: ACM Press, 2016, pp. 137–146. ISBN: 9781450344289. DOI: 10.1145/2945292.2945306. URL: http://dl.acm.org/citation.cfm?doid=2945292.2945306.

[Hag+17] B. Hagedorn, S. Thum, T. Reitz, V. Coors, and R. Gutbell. *OGC® 3D Portrayal Service 1.0*. 2017. URL: http://www.opengeospatial.org/standards/3dp.

[Hag10] B. Hagedorn. “Web View Service Discussion Paper Version 0.3.0”. 2010. URL: http://portal.opengeospatial.org/files/?artifact_id=37257.

[Hag16] B. Hagedorn. “Konzepte und Techniken zur servicebasierten Visualisierung von geovirtuellen 3D-Umgebungen”. PhD thesis. University of Potsdam, 2016.

[HC15] G. Herbert and X. Chen. “A comparison of usefulness of 2D and 3D representations of urban planning”. In: *Cartography and Geographic Information Science* 42.1 (Jan. 2015), pp. 22–32. ISSN: 1523-0406. DOI: 10.1080/15230406.2014.987694. URL: http://www.tandfonline.com/doi/full/10.1080/15230406.2014.987694.

[HD08] B. Hagedorn and J. Döllner. “Sketch-Based Navigation in 3D Virtual Environments”. In: *Proceedings of the 9th international symposium on Smart Graphics*. Vol. 5166. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 239–246. ISBN: 978-3-540-85410-4. DOI: 10.1007/978-3-540-85412-8. URL: http://www.springerlink.com/index/10.1007/978-3-540-85412-8.

[HD10] D. Hildebrandt and J. Döllner. “Service-oriented, standards-based 3D geovisualization: Potential and challenges”. In: *Computers, Environment and Urban Systems* 34.6 (Nov. 2010), pp. 484–495. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2010.05.003. URL: http://linkinghub.elsevier.com/retrieve/pii/S0198971510000426.

[HFR07] J. Haist, H. M. Figueiredo Ramos, and T. Reitz. “Symbology Encoding for 3D GIS - An Approach to Extending 3D City Model Visualization to GIS Visualization”. In: *Urban Data Management: Urban Data Management Society Symposium* December (2007). DOI: 10.13140/2.1.1896.3523.

[HHD10] B. Hagedorn, D. Hildebrandt, and J. Döllner. “Towards Advanced and Interactive Web Perspective View Services”. In: *Developments in 3D Geo-Information Sciences*. Ed. by T. Neutens and P. Maeyer. Berlin/Heidelberg: Springer, 2010, pp. 33–51. ISBN: 978-3-642-04791-6. DOI: 10.1007/978-3-642-04791-6_3. URL: http://link.springer.com/10.1007/978-3-642-04791-6_3.

[HHD11] D. Hildebrandt, B. Hagedorn, and J. Döllner. “Image-based strategies for interactive visualisation of complex 3D geovirtual environments on lightweight devices”. In: *Journal of Location Based Services* 5.2 (June 2011), pp. 100–120. ISSN: 1748-9725. DOI: 10.1080/17489725.2011.580787. URL: http://www.tandfonline.com/doi/abs/10.1080/17489725.2011.580787.

[Hil14] D. Hildebrandt. “A Software Reference Architecture for Service-Oriented 3D Geovisualization Systems”. In: *ISPRS International Journal of Geo-Information* 3.4 (2014), pp. 1445–1490. ISSN: 2220-9964. DOI: 10.3390/ijgi3041445. URL: http://www.mdpi.com/2220-9964/3/4/1445/.

[Hil16] D. Hildebrandt. “Image-based styling”. In: *The Visual Computer* 32.4 (Apr. 2016), pp. 445–463. ISSN: 0178-2789. DOI: 10.1007/s00371-015-1073-3. URL: http://link.springer.com/10.1007/s00371-015-1073-3.

[HM90] R. B. Haber and D. A. McNapp. “Visualization ldioms : A Conceptual Model Visualization for Scientific Systems”. In: *Visualization in scientific computing* 74 (1990), pp. 74–93. URL: http://cose.math.bas.bg/Sci_Visualization/visConcepts/concepts/idio_sh.htm.

[HT14] D. Hildebrandt and R. Timm. “An assisting, constrained 3D navigation technique for multiscale virtual 3D city models”. In: *GeoInformatica* 18.3 (July 2014), pp. 537–567. ISSN: 1384-6175. DOI: 10.1007/s10707-013-0189-8. URL: http://link.springer.com/10.1007/s10707-013-0189-8.

[Hum+01] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. “WireGL: A Scalable Graphics System for Clusters”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’01*. New York, New York, USA: ACM Press, 2001, pp. 129–140. ISBN: 158113374X. DOI: 10.1145/383259.383272. URL: http://portal.acm.org/citation.cfm?doid=383259.383272.

[IFF96] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. “Lua—An Extensible Extension Language”. In: *Software: Practice and Experience* 26.6 (June 1996), pp. 635–652. ISSN: 0038-0644. DOI: 10.1002/(SICI)1097-024X(199606)26:6<635::AID-SPE26>3.0.CO;2-P. URL: http://doi.wiley.com/10.1002/%28SICI%291097-024X%28199606%2926%3A6%3C635%3A%3AAID-SPE26%3E3.0.CO%3B2-P.

[Ins97] A. Inselberg. "Multidimensional detective". In: *Proceedings of VIZ '97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium.* Ed. by J. Dill and N. Gershon. IEEE Computer Society, 1997, pp. 100–107. ISBN: 0-8186-8189-6. DOI: 10.1109/INFVIS.1997.636793. URL: http://ieeexplore.ieee.org/document/636793/.

[ISO04] ISO. *ISO/IEC 14496-10 Information technology — Coding of audio-visual objects Part 10: Advanced Video Coding.* Geneva, Switzerland, 2004. URL: https://www.iso.org/standard/75400.html.

[ISO14] ISO/TC 211 Terminology Maintenance Group. *ISO/TC 211 Glossary of Terms - English.* 2014. URL: www.isotc211.org/Terminology.htm.

[Jan+13] J. Jankowski, S. Ressler, K. Sons, Y. Jung, J. Behr, and P. Slusallek. "Declarative integration of interactive 3D graphics into the world-wide web". In: *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13.* New York, New York, USA: ACM Press, 2013, p. 39. ISBN: 9781450321334. DOI: 10.1145/2466533.2466547. URL: http://dl.acm.org/citation.cfm?id=2466533.2466547.

[JH15] J. Jankowski and M. Hachet. "Advances in Interaction with 3D Environments". In: *Computer Graphics Forum* 34.1 (Feb. 2015), pp. 152–190. ISSN: 01677055. DOI: 10.1111/cgf.12466. URL: http://doi.wiley.com/10.1111/cgf.12466.

[Jul+18] A. Julin, K. Jaalama, J.-P. Virtanen, M. Pouke, J. Ylipulli, M. Vaaja, J. Hyyppä, and H. Hyyppä. "Characterizing 3D City Modeling Projects: Towards a Harmonized Interoperable System". In: *ISPRS International Journal of Geo-Information* 7.2 (Feb. 2018), p. 55. ISSN: 2220-9964. DOI: 10.3390/ijgi7020055. URL: http://www.mdpi.com/2220-9964/7/2/55.

[KA07] J. Koehler and G. Alonso. "Service-Oriented Computing". In: *European Research Consortium for Informatics and Mathematics* 70 (2007), pp. 14–15. ISSN: 0926-4981.

[KD10] J. Klimke and J. Döllner. "Combining Synchronous and Asynchronous Collaboration within 3D City Models". In: *Sixth International Conference, GIScience 2010, Zürich, Switzerland, Sep. 14-17 2010, Proceedings.* Ed. by S. Fabrikant, T. Reichenbacher, M. van Kreveld, and C. Schlieder. Zürich: Springer, 2010, pp. 115–129. DOI: 10.1007/978-3-642-15300-6_9. URL: http://link.springer.com/10.1007/978-3-642-15300-6_9.

[KG15] M. Krämer and R. Gutbell. "A case study on 3D geospatial applications in the web using state-of-the-art WebGL frameworks". In: *Proceedings of the 20th International Conference on 3D Web Technology - Web3D '15.* Vol. 2775303. March. New York, New York, USA: ACM Press, 2015, pp. 189–197. ISBN: 9781450336475. DOI: 10.1145/2775292.2775303. URL: http://dl.acm.org/citation.cfm?doid=2775292.2775303.

[KGP05] T. Kolbe, G. Gröger, and L. Plümer. “Geo-information for Disaster Management”. In: *Geo-information for Disaster Management*. Berlin/Heidelberg: Springer-Verlag, 2005, pp. 883–899. ISBN: 3-540-24988-5. DOI: 10.1007/b139115. URL: http://www.springerlink.com/index/10.1007/b139115.

[KGT06] A. Knopfel, B. Grone, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2006, p. 352. ISBN: 978-0-470-02710-3.

[KK12] T. Krüger and T. Kolbe. “Building analysis for urban energy planning using key indicators on virtual 3D city models—the energy atlas of Berlin”. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. XXXIX. September. 2012, pp. 145–150. URL: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXIX-B2/145/2012/isprsarchives-XXXIX-B2-145-2012.pdf.

[KL16] J. Kubiak and R. Lawniczak. “The propagation of noise in a built-up area (on the example of a housing estate in Poznan)”. In: *Journal of Maps* 12.2 (2016), pp. 231–236. ISSN: 1744-5647. DOI: 10.1080/17445647.2014.1001801. URL: http://www.tandfonline.com/doi/abs/10.1080/17445647.2014.1001801.

[Kli+14] J. Klimke, B. Hagedorn, M. Trapp, and J. Döllner. “Web-based and Mobile Provisioning of Virtual 3D Reconstructions”. In: *Kultur und Informatik: Reality and Virtuality*. Ed. by R. Franken-Wendelstorf, E. Lindinger, and J. Sieck. 5. Werner Hülsbusch Verlag, 2014, pp. 17–28. ISBN: 978-3-86488-064-3.

[Kol09] T. H. Kolbe. “Representing and Exchanging 3D City Models with CityGML”. In: *3D Geo-Information Sciences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 15–31. ISBN: 978-3-540-87395-2. DOI: 10.1007/978-3-540-87395-2_2. URL: http://link.springer.com/10.1007/978-3-540-87395-2_2.

[Krä18] M. Krämer. “A microservice architecture for the processing of large geospatial data in the Cloud”. PhD thesis. 2018, p. 186. URL: http://tuprints.ulb.tu-darmstadt.de/6956/.

[KRT09] C. Kottman, C. Reed, and O. G. C. T. C. (TC). *The OpenGIS abstract specification, topic 5: features*. 2009. URL: http://www.opengeospatial.org/standards/as.

[Kum+17] K. Kumar, H. Ledoux, T. J. F. Commandeur, and J. E. Stoter. “Modelling Urban Noise in CityGML ADE: Case of the Netherlands”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-4/W5.October (Oct. 2017), pp. 73–81. ISSN: 2194-9050. DOI: 10.5194/isprs-annals-IV-4-W5-73-2017. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W5/73/2017/.

[La 06] J. de La Beaujardiere. *OpenGIS Web Map Service (WMS) Implementation Specification*. 2006. URL: http://www.opengeospatial.org/standards/wms.

[Lam+03] F. Lamberti, C. Zunino, A. Sanna, A. Fiume, and M. Maniezzo. “An accelerated remote graphics architecture for PDAS”. In: *Proceeding of the eighth international conference on 3D web technology - Web3D ’03*. New York, New York, USA: ACM Press, 2003, p. 55. ISBN: 1581136447. DOI: 10.1145/636593.636602. URL: http://dl.acm.org/citation.cfm?id=636593.636602.

[LBL17] L. Lu, T. Becker, and M.-O. Löwner. “3D Complete Traffic Noise Analysis Based on CityGML”. In: ed. by T. H. Kolbe, G. König, and C. Nagel. Lecture Notes in Geoinformation and Cartography October. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 265–283. ISBN: 978-3-642-12669-7. DOI: 10.1007/978-3-319-25691-7_15. URL: http://link.springer.com/10.1007/978-3-319-25691-7_15.

[LDN04] S. Lefebvre, J. Darbon, and F. Neyret. *Unified Texture Management for Arbitrary Meshes*. Tech. rep. May. INRIA RhôneAlpes, 2004. URL: http://hal.inria.fr/inria-00070783/.

[LH05] E. Lange and S. Hehl-Lange. “Combining a participatory planning approach with a virtual landscape model for the siting of wind turbines”. In: *Journal of Environmental Planning and Management* 48.6 (Nov. 2005), pp. 833–852. ISSN: 0964-0568. DOI: 10.1080/09640560500294277. URL: http://www.tandfonline.com/doi/abs/10.1080/09640560500294277.

[Lia+16] J. Liang, J. Gong, J. Liu, Y. Zou, J. Zhang, J. Sun, and S. Chen. “Generating Orthorectified Multi-Perspective 2.5D Maps to Facilitate Web GIS-Based Visualization and Exploitation of Massive 3D City Models”. In: *ISPRS International Journal of Geo-Information* 5.11 (Nov. 2016), p. 212. ISSN: 2220-9964. DOI: 10.3390/ijgi5110212. URL: http://www.mdpi.com/2220-9964/5/11/212.

[Lie+10] B. Liesenfeld, M. Lustig, A. Weller, E. Oppermann, and J. Beyer. “LTE-Radio Network Planning with PegaPlan”. In: *Wissen Heute* 63.6 (2010).

[Lup07] M. Lupp. *Styled Layer Descriptor Profile of the Web Map Service Version 1.1.0*. 2007. URL: http://portal.opengeospatial.org/files/?artifact_id=22364.

[Mac+06] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. “Reference Model for Service Oriented Architecture”. In: *OASIS* August (2006), pp. 1–28. URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[Mac+92] A. MacEachren, B. Buttenfield, J. Campbell, D. DiBiase, and M. Monmonier. “Visualization”. In: *Geography’s Inner Worlds: Pervasive Themes in Contemporary American Geography*. Ed. by R. Abler, M. Marcus, and J. Olson. New Brunswick, NJ: Rutgers University Press, 1992. Chap. Visualizat, pp. 99–137. ISBN: 978-0813518305.

[MAC+99] A. M. MACEACHREN, M. WACHOWICZ, R. EDSALL, D. HAUG, and R. MASTERS. “Constructing knowledge from multivariate spatiotemporal data: integrating geographical visualization with knowledge discovery in database methods”. In: *International Journal of Geographical Information Science* 13.4

(June 1999), pp. 311–334. ISSN: 1365-8816. DOI: 10.1080/136588199241229. URL: http://www.tandfonline.com/doi/abs/10.1080/136588199241229.

[MAK10] J. Moser, F. Albrecht, and B. Kosar. “Beyond Visualisation – 3D GIS Analyses for Virtual City Models”. In: *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* XXXVIII (2010), pp. 143–146. ISSN: 16821750. URL: http://misc.gis.tu-berlin.de/3dgeoinfo/ISPRS_Conference_CD/Paper_ISPRS/Poster/14_3DGeoInfo2010_147_Moser_3D_GIS_Analyses.pdf.

[Mao11] B. Mao. “Visualization and Generalization of 3D City Models”. PhD thesis. Royal Institute of Technology, 2011. URL: http://kth.diva-portal.org/smash/record.jsf?pid=diva2%5C%3A456906&dswid=-6575.

[May10] A. J. Mayer. *Virtual Texturing.* 2010. URL: https://www.cg.tuwien.ac.at/research/publications/2010/Mayer-2010-VT/.

[MDB87] B. McCormick, T. A. DeFanti, and M. Brown, eds. *Visualization in scientific computing.* Vol. 21. 6. Mar. 1987. DOI: 10.1145/43965.43966. URL: http://portal.acm.org/citation.cfm?doid=43965.43966.

[Mei13] G. Meinlschmidt, ed. *Handlungsorientierter Sozialstrukturatlas Berlin 2013.* Berlin: Senatsverwaltung für Gesundheit und Soziales, Berlin, 2013, p. 284. URL: https://www.gesundheitliche-chancengleichheit.de/pdf.php?id=5d11b3c2b30720ff9cac6e3ad2b8bef0.

[Mic$^+$07] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski. “Scale-up x Scale-out: A Case Study using Nutch/Lucene”. In: *2007 IEEE International Parallel and Distributed Processing Symposium* (2007), pp. 1–8. DOI: 10.1109/IPDPS.2007.370631. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4228359.

[Mit08] M. Mittring. “Advanced virtual texture topics”. In: *ACM SIGGRAPH 2008 classes on - SIGGRAPH ’08.* Ed. by N. Tatarchuk. New York, New York, USA: ACM Press, 2008, pp. 23–51. DOI: 10.1145/1404435.1404438. URL: http://portal.acm.org/citation.cfm?doid=1404435.1404438.

[Mor$^+$11] A. Moreno, R. J. Segura, A. Korchi, J. Posada, and O. Otaegui. *Interactive Urban and Forest Fire Simulation with Extinguishment Support Aitor.* Ed. by T. H. Kolbe, G. König, and C. Nagel. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 73–93. ISBN: 978-3-642-12669-7. DOI: 10.1007/978-3-642-12670-3. URL: http://link.springer.com/10.1007/978-3-642-12670-3.

[MP15] M. Mueller and B. Pross. *OGC® WPS 2.0.2 Interface Standard.* 2015. URL: http://www.opengeospatial.org/standards/wps.

[MPN10] J. Masó, K. Pomakis, and J. Núria. *OpenGIS® Web Map Tile Service Implementation Standard Version 1.0.0.* Tech. rep. Open Geospatial Consortium, 2010. URL: http://portal.opengeospatial.org/files/?artifact_id=35326.

[Mül06] M. Müller. *Symbology Encoding Implementation Specification.* 2006. URL: http://www.opengeospatial.org/standards/symbol.

[Mus+13] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. van Gool, and W. Purgathofer. "A Survey of Urban Reconstruction". In: *Computer Graphics Forum* 32.6 (Sept. 2013), pp. 146–177. ISSN: 01677-055. DOI: 10.1111/cgf.12077. URL: http://doi.wiley.com/10.1111/cgf.12077.

[Mwa+16] F. Mwalongo, M. Krone, G. Reina, and T. Ertl. "State-of-the-Art Report in Web-based Visualization". In: *Computer Graphics Forum* 35.3 (June 2016), pp. 553–575. ISSN: 01677055. DOI: 10.1111/cgf.12929. URL: http://doi.wiley.com/10.1111/cgf.12929.

[New15] S. Newman. *Building Microservices*. 1st. O'Reilly Media, Inc., 2015. ISBN: 9781 4919 50357.

[NF14] G. Navratil and P. Fogliaroni. "Visibility Analysis in a 3D Cadastre". In: *4th International Workshop on 3D Cadastres*. November 2014. 2014, pp. 183–196. ISBN: 978-87-92853-28-8.

[Nic16] J. Nickoloff. *Docker in Action*. 2016, p. 306. ISBN: 978-1-935182-49-8.

[Nol06] M. Nollenburg. "Geographic Visualization". In: *Human-Centered Visualization Environments*. Ed. by A. Kerren, A. Ebert, and J. Meyer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 257–294. ISBN: 978-3-540-71948-9. DOI: 10.1007/978-3-540-71949-6_6. URL: http://link.springer.com/10.1007/978-3-540-71949-6_6.

[Nou+14] R. Nouvel, M. Zirak, H. Dastageeri, V. Coors, and U. Eicker. "Urban Energy Analysis Based on 3D City Model for National Scale Applications". In: *IBPSA Germany Conference*. Vol. 8. 2014. URL: http://architecture.mit.edu/sites/architecture.mit.edu/files/attachments/lecture/p1117v1%20Ursula%20Eicker-1.pdf.

[NZ08] S. Neubauer and A. Zipf. "Suggestions for extending the OGC styled layer descriptor (SLD) specification into 3D–towards visualization rules for 3D city models". In: *Urban and Regional Data Management: UDMS 2009 Annual*. Ed. by M. Rumor, V. Coors, E. Fendel, and S. Zlatanova. Taylor & Francis, 2008, pp. 133–142.

[NZ09] S. Neubauer and A. Zipf. *3D-Symbology Encoding Discussion Draft*. 2009. URL: http://portal.opengeospatial.org/files/?artifact_id=32904.

[OL03] M. de Oliveira and H. Levkowitz. "From visual data exploration to visual data mining: A survey". In: *IEEE Transactions on Visualization and Computer Graphics* 9.3 (July 2003), pp. 378–394. ISSN: 1077-2626. DOI: 10.1109/TVCG.2003.1207445. URL: http://ieeexplore.ieee.org/document/1207445/.

[OSG] OSGeo. *Tile Map Service Specification*. URL: https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification.

[PD13] S. Pasewaldt and J. Döllner. "Interacting with multi-perspective views". In: *Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction - MapInteract '13*. New York, New York, USA: ACM Press, 2013, pp. 44–47. ISBN: 9781450325363. DOI: 10.1145/2534931.2534942. URL: http://doi.acm.org/10.1145/2534931.2534942.

[Per11] G. Percivall. *OGC Reference Model Version 2.1.* 2011. URL: http://www.opengis.net/doc/orm/2.1.

[Pes95] M. Pesce. *VRML: Browsing and Building Cyberspace.* New Riders, 1995. ISBN: 978-1562054984.

[PG03] M. P. Papazoglou and D. Georgakopoulos. “Introduction: Service-oriented computing”. In: *Communications of the ACM* 46.10 (Oct. 2003). Ed. by D. Georgakopoulos and M. P. Papazoglou, p. 24. ISSN: 00010782. DOI: 10.1145/944217.944233. URL: http://portal.acm.org/citation.cfm?doid=944217.944233.

[PIB17] I. Prieto, J. L. Izkara, and R. Béjar. “Web-Based Tool for the Sustainable Refurbishment in Historic Districts Based on 3D City Model”. In: *Springer.* August. 2017, pp. 159–169. ISBN: 978-3-540-72134-5. DOI: 10.1007/978-3-319-25691-7_9. URL: http://link.springer.com/10.1007/978-3-319-25691-7_9.

[Por07] C. Portele. *OpenGIS Geography Markup Language (GML) Encoding Standard, Version 3.3.0.* July 2007. URL: https://portal.opengeospatial.org/files/?artifact_id=46568.

[PTD11] S. Pasewaldt, M. Trapp, and J. Döllner. “Multiscale visualization of 3D geovirtual environments using view-dependent multi-perspective views”. In: *Journal of WSCG* 19.1-3 (2011), pp. 111–118. ISSN: 12136972.

[RB16] R. Roschlaub and J. Batscheider. “An Inspire-Konform 3D Building Model Of Bavaria Using Cadastre Information, LiDAR And Image Matching”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences.* Vol. XLI-B4. July. June 2016, pp. 747–754. DOI: 10.5194/isprsarchives-XLI-B4-747-2016. URL: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B4/747/2016/isprs-archives-XLI-B4-747-2016.pdf.

[RBK07] B. Randt, F. Bildstein, and T. H. Kolbe. “Use of Virtual 3D-Landscapes for Emergency Driver-Training”. In: *Proc. of the Int. Conference on Visual Simulation IMAGE.* 2007. URL: http://www.redaktion.tu-berlin.de/fileadmin/fg227/Publications/IMAGE2007-RDE-DrivingSim-5-Letter.PDF.

[RC15] P. Rubinowicz and K. Czyńska. “Study of City Landscape Heritage Using Lidar Data and 3D-City Models”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-7/W3 (Apr. 2015), pp. 1395–1402. ISSN: 2194-9034. DOI: 10.5194/isprsarchives-XL-7-W3-1395-2015. URL: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-7-W3/1395/2015/.

[RCT17] S. Ruby, D. Copeland, and D. Thomas. *Agile Web Development with Rails 5.1.* O’Reilly UK, 2017, p. 464. ISBN: 978-1680502510.

[RF94] W. Ribarsky and J. Foley. *Next Generation Data Visualization Tools.* Tech. rep. Georgia Institute of Technology, 1994, pp. 103–126. URL: http://hdl.handle.net/1853/3594.

[RFC13] J. I. J. Rodrigues, M. J. G. Figueiredo, and C. P. Costa. "Web3DGIS for City Models with CityGML and X3D". In: *2013 17th International Conference on Information Visualisation*. IEEE, July 2013, pp. 384–388. ISBN: 978-0-7695-5049-7. DOI: 10.1109/IV.2013.102. URL: http://ieeexplore.ieee.org/document/6676590/.

[RKD13] R. Richter, J. E. Kyprianidis, and J. Döllner. "Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds". In: *Transactions in GIS* 17.5 (Jan. 2013), pp. 724–741. ISSN: 13611682. DOI: 10.1111/j.1467-9671.2012.01362.x. URL: http://doi.wiley.com/10.1111/j.1467-9671.2012.01362.x.

[RKT09] T. Reitz, M. Krämer, and S. Thum. "A processing pipeline for X3D earth-based spatial data view services". In: *Proceedings of the 14th International Conference on 3D Web Technology - Web3D '09*. Vol. 1. 212. New York, New York, USA: ACM Press, 2009, p. 137. ISBN: 9781605584324. DOI: 10.1145/1559764.1559786. URL: http://portal.acm.org/citation.cfm?doid=1559764.1559786.

[RM98] D. R. Roberts and A. D. Marshall. "Viewpoint Selection for Complete Surface Coverage of Three Dimensional Objects". In: *Procedings of the British Machine Vision Conference 1998*. British Machine Vision Association, 1998, pp. 1–74. ISBN: 1-901725-04-9. DOI: 10.5244/C.12.74. URL: http://www.bmva.org/bmvc/1998/papers/d061/h061.htm.

[Ros+09] L. Ross, J. Bolling, J. Döllner, and B. Kleinschmit. "Enhancing 3D City Models with Heterogeneous Spatial Information: Towards 3D Land Information Systems". In: *Lecture Notes in Geoinformation and Cartography*. 2009, pp. 113–133. ISBN: 9783642003172. DOI: 10.1007/978-3-642-00318-9_6. URL: http://link.springer.com/10.1007/978-3-642-00318-9_6.

[Ros10] L. Ross. "Virtual 3D City Models in Urban Land Management". PhD thesis. Technische Universität Berlin, 2010, p. 101. DOI: 10.14279/depositonce-2744. URL: http://www.planen-bauen-umwelt.tu-berlin.de/fileadmin/fg242/Dissertationen/Lutz_Ross_Dissertation.pdf.

[Roz15] N. Rozentals. *Mastering TypeScript*. Packt Publishing, 2015, p. 259. ISBN: 978-1784399665.

[RSS18] F. Rademacher, J. Sorgalla, and S. Sachweh. "Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective". In: *IEEE Software* 35.3 (May 2018), pp. 36–43. ISSN: 0740-7459. DOI: 10.1109/MS.2018.2141028. URL: https://ieeexplore.ieee.org/document/8354426/.

[Rus+00] C. Russo dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J. Paris. "Metaphor-aware 3D navigation". In: *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*. IEEE Comput. Soc, 2000, pp. 155–165. ISBN: 0-7695-0804-9. DOI: 10.1109/INFVIS.2000.885104. URL: http://ieeexplore.ieee.org/document/885104/.

[SB04] S. dos Santos and K. Brodlie. “Gaining understanding of multivariate and multidimensional data through visualization”. In: *Computers & Graphics* 28.3 (June 2004), pp. 311–325. ISSN: 00978493. DOI: 10.1016/j.cag.2004.03.013. URL: http://linkinghub.elsevier.com/retrieve/pii/S0097849304000251.

[SBN16] A. Schilling, J. Bolling, and C. Nagel. “Using glTF for streaming CityGML 3D city models”. In: *Proceedings of the 21st International Conference on Web3D Technology - Web3D ’16*. October 2015. New York, New York, USA: ACM Press, 2016, pp. 109–116. ISBN: 9781450344289. DOI: 10.1145/2945292.2945312. URL: http://dl.acm.org/citation.cfm?doid=2945292.2945312.

[SCK07] H.-Y. Shum, S.-C. Chan, and S. B. Kang. *Image-Based Rendering*. 1st. Boston, MA: Springer US, 2007, p. 408. ISBN: 978-0-387-21113-8. DOI: 10.1007/978-0-387-32668-9. URL: http://link.springer.com/10.1007/978-0-387-32668-9.

[Sec+11] A. Secord, J. Lu, A. Finkelstein, M. Singh, and A. Nealen. “Perceptual models of viewpoint preference”. In: *ACM Transactions on Graphics* 30.5 (Oct. 2011), pp. 1–12. ISSN: 07300301. DOI: 10.1145/2019627.2019628. URL: http://dl.acm.org/citation.cfm?id=2019628%20http://dl.acm.org/citation.cfm?doid=2019627.2019628.

[Sem+15] A. Semmo, M. Trapp, M. Jobst, and J. Döllner. “Cartography-Oriented Design of 3D Geospatial Information Visualization – Overview and Techniques”. In: *The Cartographic Journal* 52.2 (Apr. 2015), pp. 95–106. ISSN: 0008-7041. DOI: 10.1080/00087041.2015.1119462. URL: http://www.tandfonline.com/doi/full/10.1080/00087041.2015.1119462.

[SH15] S. Shi and C.-H. Hsu. “A Survey of Interactive Remote Rendering Systems”. In: *ACM Computing Surveys* 47.4 (May 2015), pp. 1–29. ISSN: 03600300. DOI: 10.1145/2719921. URL: http://dl.acm.org/citation.cfm?doid=2775083.2719921.

[SHC12] A. Schilling, B. Hagedorn, and V. Coors. *OGC 3D Portrayal Interoperability Experiment - Final Report*. 2012. URL: http://www.opengis.net/doc/ie/3dpie.

[SI10] J. T. Sample and E. Ioup. “Tile Storage”. In: *Tile-Based Geospatial Information Systems*. Boston, MA: Springer US, 2010, pp. 117–131. DOI: 10.1007/978-1-4419-7631-4_7. URL: http://link.springer.com/10.1007/978-1-4419-7631-4_7.

[SK10] A. Schilling and T. H. Kolbe. “Draft for Candidate OpenGIS® Web 3D Service Interface Standard, Version 0.4.0”. 2010. URL: http://portal.opengeospatial.org/files/?artifact_id=36390.

[SK14] M. Sindram and T. H. Kolbe. “Modeling of urban planning actions by complex transactions on semantic 3D city models”. In: *7th International Congress on Environmental Modelling and Software*. Ed. by D. Ames, N. Quinn, and A. Rizzoli. International Environmental Modelling and Software

Society. 2014, pp. 848–855. ISBN: 978-88-9035-744-2. URL: https://mediatum.ub.tum.de/doc/1224665/file.pdf.

[Son+10] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov, and P. Slusallek. "XML3D". In: *Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10*. Vol. 1. 212. New York, New York, USA: ACM Press, 2010, p. 175. ISBN: 9781450302098. DOI: 10.1145/1836049.1836076. URL: http://portal.acm.org/citation.cfm?doid=1836049.1836076.

[Spe14] R. Spence. *Information Visualization*. 3rd. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-07340-8. DOI: 10.1007/978-3-319-07341-5. URL: http://link.springer.com/10.1007/978-3-319-07341-5.

[SR08] A. Slingsby and J. Raper. "Navigable Space in 3D City Models for Pedestrians". In: *Advances in 3D Geoinformation Systems*. Vol. 23. 2008, pp. 49–64. ISBN: 9783540721352. DOI: 10.1007/978-3-540-72135-2_3. URL: http://link.springer.com/10.1007/978-3-540-72135-2_3.

[Str+11] A. Strzalka, J. Bogdahn, V. Coors, and U. Eicker. "3D City modeling for urban scale heating energy demand forecasting". In: *HVAC&R Research* 17.4 (2011), pp. 526–539. ISSN: 1078-9669. DOI: 10.1080/10789669.2011.582920. URL: https://www.tandfonline.com/doi/abs/10.1080/10789669.2011.582920.

[Tiz+11] N. Tizon, C. Moreno, M. Cernea, and M. Preda. "MPEG-4-based adaptive remote rendering for video games". In: *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11*. Vol. 1. 212. New York, New York, USA: ACM Press, 2011, p. 45. ISBN: 9781450307741. DOI: 10.1145/2010425.2010433. URL: http://dl.acm.org/citation.cfm?doid=2010425.2010433.

[Tra+10] M. Trapp, A. Semmo, R. Pokorski, C.-D. Herrmann, J. Döllner, M. Eichhorn, and M. Heinzelmann. "Communication of Digital Cultural Heritage in Public Spaces by the Example of Roman Cologne". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6436 LNCS. 2010, pp. 262–276. ISBN: 3642168728. DOI: 10.1007/978-3-642-16873-4_20. URL: http://link.springer.com/10.1007/978-3-642-16873-4_20.

[TRC01] D. S. Tan, G. G. Robertson, and M. Czerwinski. "Exploring 3D navigation: Combining Speed-coupled Flying with Orbiting". In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '01*. Vol. pp. New York, New York, USA: ACM Press, 2001, pp. 418–425. ISBN: 1581133278. DOI: 10.1145/365024.365307. URL: http://portal.acm.org/citation.cfm?doid=365024.365307.

[Ups+89] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. "The application visualization system: a computational environment for scientific visualization". In: *IEEE Computer Graphics and Applications* 9.4 (July 1989), pp. 30–42. ISSN: 0272-1716. DOI: 10.1109/38.31462. URL: http://ieeexplore.ieee.org/document/31462/.

[Váz+01] P. P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. “Viewpoint Selection Using Viewpoint Entropy”. In: *Proceedings of Vision, Modeling, and Visualization 2001* (2001), pp. 273–280. ISSN: 0278-3649. DOI: 10.1177/0278364911410755.

[Váz+03] P. P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. “Automatic View Selection Using Viewpoint Entropy and its Application to Image-Based Modelling”. In: *Computer Graphics Forum* 22.4 (2003), pp. 689–700. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2003.00717.x.

[Vre14a] P. Vretanos. *OGC Web Feature Service Implementation Specification Version 2.0.2*. 2014. URL: http://docs.opengeospatial.org/is/09-025r2/09-025r2.html.

[Vre14b] P. Vretanos. *OpenGIS Filter Encoding Implementation Specification*. 2014. URL: http://docs.opengeospatial.org/is/09-026r2/09-026r2.html.

[Wag+13] D. Wagner, M. Wewetzer, J. Bogdahn, N. Alam, M. Pries, and V. Coors. *Geometric-Semantical Consistency Validation of CityGML Models*. Ed. by J. Pouliot, S. Daniel, Z. Hubert, and A. Zamyadi. 2013. DOI: 10.1007/978-3-642-29793-9_10. URL: http://link.springer.com/10.1007/978-3-642-29793-9_10.

[War12] C. Ware. *Information Visualization: Perception for Design*. Ed. by M. Dunkerley. 3rd. Morgan Kaufmann, 2012, p. 536. ISBN: 978-0-12-381464-7.

[Wen+17] J. Wendel, A. Simons, A. Nichersu, and S. M. Murshed. “Rapid development of semantic 3D city models for urban energy analysis based on free and open data sources and software”. In: *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics - UrbanGIS'17*. New York, New York, USA: ACM Press, 2017, pp. 1–7. ISBN: 9781450354950. DOI: 10.1145/3152178.3152193. URL: http://dl.acm.org/citation.cfm?doid=3152178.3152193.

[WG10] A. Whiteside and J. Greenwood. *OGC Web Services Common Standard*. 2010.

[WHG10] H. Wu, Z. He, and J. Gong. “A virtual globe-based 3D visualization and interactive framework for public participation in urban planning processes”. In: *Computers, Environment and Urban Systems* 34.4 (July 2010), pp. 291–298. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2009.12.001. URL: http://linkinghub.elsevier.com/retrieve/pii/S0198971509000945.

[WHP16] L. Wan, Z. Huang, and X. Peng. “An Effective NoSQL-Based Vector Map Tile Management Approach”. In: *ISPRS International Journal of Geo-Information* 5.11 (Nov. 2016), p. 215. ISSN: 2220-9964. DOI: 10.3390/ijgi5110215. URL: http://www.mdpi.com/2220-9964/5/11/215.

[Wie+15] M. Wieland, A. Nichersu, S. Murshed, and J. Wendel. “Computing Solar Radiation on CityGML Building Data”. In: *Proceedings of the 18th AGILE conference on Geographic Information Science*. 2015, pp. 2–5. URL: https://agile-online.org/conference_paper/cds/agile_2015/shortpapers/107/107_Paper_in_PDF.pdf.

[WV14] E. Walia and V. Verma. "A Framework for Interactive 3D Rendering on Mobile Devices". In: *International Journal of Computer Vision and Image Processing* 4.2 (July 2014), pp. 18–31. ISSN: 2155-6997. DOI: 10.4018/ijcvip.2014040102. URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/ijcvip.2014040102.

[XQ06] Z. Xia and Z. Qing. "Applications of 3D City Models Based Spatial Analysis To Urban Design". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2006). URL: http://www.isprs.org/proceedings/xxxv/congress/comm2/papers/148.pdf.

[Yi+05] J. S. Yi, R. Melton, J. Stasko, and J. A. Jacko. "Dust & magnet: Multivariate information visualization using a magnet metaphor". In: *Information Visualization* 1.18 (2005), pp. 239–256. ISSN: 14738724. DOI: 10.1057/palgrave.ivs.9500099.

[Zha+16] W.-j. Zhao, E.-x. Liu, S.-p. Gao, H. J. Poh, K. W. Li, and S. T. Tan. "Traffic noise prediction and mapping with virtual 3D city models". In: *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*. 2016, pp. 576–581.

[ZN08] Q.-Y. Zhou and U. Neumann. "Fast and extensible building modeling from airborne LiDAR data". In: *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems - GIS '08*. New York, New York, USA: ACM Press, 2008, p. 1. ISBN: 9781605583235. DOI: 10.1145/1463434.1463444. URL: http://portal.acm.org/citation.cfm?doid=1463434.1463444.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Potsdam, 25. April 2019	
(Ort, Datum)	(Unterschrift)