Open Geospatial Consortium

Approval Date: 2012-06-23

Publication Date: 2012-08-27

External identifier of this OGC® document: http://www.opengis.net/doc/ie/3dpie

Reference number of this document: OGC 12-075

Category: Public Engineering Report

Editors: Arne Schilling, Benjamin Hagedorn, Volker Coors

OGC 3D Portrayal Interoperability Experiment

FINAL REPORT

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type: OGC® Engineering Report
Document subtype: Interoperability Experiment Report
Document stage: Approved for public release
Document language: English

License Agreement

Preface

This report describes the setup, experiments, results and issues generated by the 3D Portrayal Interoperability Experiment (3DPIE), carried out as an activity of the 3D Information Management Domain Working Group (3DIM). The 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, Web 3D Service (W3DS) and Web View Service (WVS).

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Submitting organizations

This Interoperability Experiment Report was submitted to the OGC Interoperability Program by the following organizations:

- Hasso-Plattner-Institut at the University of Potsdam (HPI)
- GIScience at the University of Heidelberg (GIScience)
- Fraunhofer Institute for Computer Graphics Research (IGD)
- Bitmanagement Software GmbH
- CACI
- Institute for Geodesy and Geoinformation Science at Technical University Berlin (IGG)
- Institut Geographique National (IGN)
- Laboratoire des Sciences de l'Information et des Systémes (LSIS)
- Monterey Bay Aquarium Research Institute (MBARI)
- Naval Postgraduate School (NPS)
- Virginia Tech (VT)

Contents

Page

Figures

Page

Tables

Page

OGC® 3D Portrayal Interoperability Experiment Final Report

1 Introduction

1.1 Scope

This document describes the results of an OGC Interoperability Experiment (IE) on the portrayal of 3D geospatial information. It contains technical details on processing 3D information in an OGC service environment as well as best practices on how to portray large data sets in urban planning scenarios, taking into account architectures and capabilities of interactive 3D graphics. Especially Web 3D Service and Web View Service, two draft standards (published as OGC discussions paper), have been in the focus of 3DPIE.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editors or the contributors:

Name	Organization
Volker Coors	Fraunhofer Institute for Computer Graphics Research (IGD)
Jens Dambruch	Fraunhofer Institute for Computer Graphics Research (IGD)
Gilles Gesquire	LSIS at Aix-Marseille University
Benjamin Hagedorn	Hasso-Plattner-Institut (HPI) at the University of Potsdam
Javier Herreruela	Institute for Geodesy and Geoinformation Science (IGG) at the Technical University of Berlin
Jan Klimke	Hasso-Plattner-Institut (HPI) at the University of Potsdam
Thomas Kolbe	Institute for Geodesy and Geoinformation Science (IGG) at the Technical University of Berlin
Nicola F. Polys	Virginia Tech (VT), Advanced Research Computing (ARC)
Peter Schickel	Bitmanagement
Arne Schilling	GIScience at the University of Heidelberg
Peter Sforza	Virginia Tech (VT), Center for Geospatial Information Technology (CGIT)
Ankit Singh	Virginia Tech (VT), Center for Geospatial Information Technology (CGIT)
Simon Thum	Fraunhofer Institute for Computer Graphics Research (IGD)
Matthias Uden	GIScience at the University of Heidelberg
Marie-Lise Vautier	Institute Geographic Nationale (IGN), France

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2011-10-12	0.1.0	Arne Schilling	All	First version from template
2012-05-31	0.3.0	Benjamin Hagedorn	Many	Draft for the Exeter TC and 3DIM motion as Public Engineering Report. Input from all Partners.
2012-08-22	0.4.0	Carl Reed		Final edits to align with OGC Public Engineering Report
2012-08-24	0.5.0	Benjamin Hagedorn	Many	Final pending edits and corrections

1.4 Forward

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 06-121r3, *OpenGIS® Web Services Common Standard*

NOTE This OWS Common Specification contains a list of normative references that are also applicable to this Implementation Specification.

[1] OpenStreetMap Wiki. The PBF format. http://wiki.openstreetmap.org/wiki/PBF_Format, accessed on 09/12/11

[2] Goodchild, M. (2007). Citizens as sensors: the world of volunteered geography. GeoJournal 69: 211-221.

[3] Hagedorn, B. (Ed.) (2010). Web View Service Discussion Paper, Version 0.3.0, OGC 09-166r2. Open Geospatial Consortium Inc., 2010.

[4] OpenStreetMap Statistics. http://wiki.openstreetmap.org/wiki/Statistics, accessed on 25/11/11.

[5] TIGER data import in the OSM Wiki. http://wiki.openstreetmap.org/wiki/TIGER, accessed on 25/11/11.

[6] Zielstra, D. and A. Zipf (2010). A Comparative Study of Proprietary Geodata and Volunteered Geographic Information for Germany. AGILE 2010. The 13th AGILE International Conference on Geographic Information Science. Guimaraes, Portugal.

[7] Haklay, M. and C. Ellul (2011). Completeness in volunteered geographical information - the evolution of OpenStreetMap coverage in England (2008-2009). In revision for the Journal of Spatial Information Science.

[8] Mooney, P., P. Corcoran and A.C. Winstanley (2010). Towards Quality Metrics for OpenStreetMap. 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. San Jose, CA.

[9] OpenStreetMap Taginfo. http://taginfo.openstreetmap.org, accessed on 30/11/11.

[10] Schilling, A. and Kolbe, T. H. (Eds.) (2009). Draft for OpenGIS Web 3D Service Implementation Standard, Version 0.4.0, OGC 09-104r1. Open Geospatial Consortium.

[11] Statistisches Bundesamt. GENESIS Online Datenbank, accessed on 25/11/11.

[12] Over, M., A. Schilling, et al. (2010). "Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany." Computers, Environments and Urban Systems 34(6): 496-507.

[13] OpenStreetMap-3D project homepage. http://www.osm-3d.org/home.en.htm, accessed on 25/11/11.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] and in OpenGIS® Abstract Specification shall apply. In addition, the following terms and definitions apply.

3.1
portrayal
presentation of information to humans [ISO 19117]

4 Conventions

4.1 Abbreviated terms

API	Application Program Interface
AR	Augmented Reality
CityGML	City Geography Markup Language
CRS	Coordinate Reference System

DTM	Digital Terrain Model
HTTP	Hypertext Transfer Protocol
IE	Interoperability Experiment
KML	Keyhole Markup Language
MR	Mixed Reality
LOD	Level of Detail
OpenLS	OpenGIS Open Location Services
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
W3DS	Web 3D Service
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service
WVS	Web View Service
XML	Extensible Markup Language
X3D	Extensible 3D

4.2 Example URLs

The sample URLs given in this document are only meant for demonstration. No server instances are available at these URLs.

5 3D Portrayal overview

The 3D Portrayal IE addresses the testing and demonstration of different mechanisms for the portrayal, delivery, and exploitation of 3D geo data based on open standards-based formats and services. The IE intended to identify, test, and further develop technologies and workflows that may be the foundation of spatial data infrastructures with a requirement for rapid visualization of very large and complex 3D geo data. Main focus of the IE was on the proposed Web 3D Service (W3DS) [10] and Web View Service (WVS) [3] interfaces, which represent two different approaches to service-based 3D portrayal:

- The Web 3D Service (W3DS) follows a 3D graphics-based approach: 3D data is hosted and managed at a W3DS server; a W3DS client is requesting 3D graphics data (including geometry and texture data, e.g., in X3D or KML/COLLADA format) and renders this data at the client side.

- The Web View Service (WVS) follows an image-based approach: A WVS client request finally rendered images of a 3D view on a 3D scene from a WVS server.

The IE included digital landscape models, city models, and interior models, e.g., in the CityGML data format, as well as different client configurations. The IE intended to clarify the specifics of 3D portrayal services and provide best practices and guidelines for their implementation, integration, and usage. Additionally, the activities and findings of the IE participants were intended to directly support and influence current standardization efforts in service-based 3D portrayal.

Figure 1 shows the general data flow of a service oriented architecture focusing on 3D geo data. The components of this multi-tier architecture can be categorized into metadata management, data access, data processing, image-based portrayal, graphics-based 3D portrayal, and clients. Not all of the components depicted here have been implemented or used in 3DPIE. A detailed description of which services were used how they were linked together follows in the remaining sections.

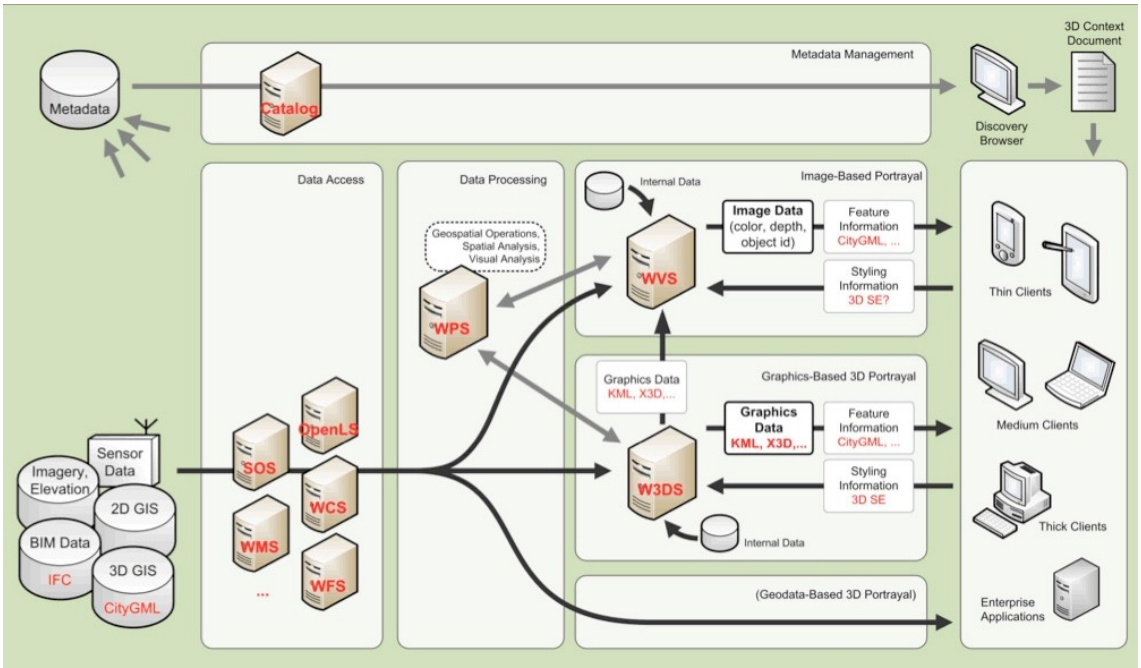

Figure 1: Approaches and context of service-based 3D portrayal.

6 Data models, formats, services, and standards

The following subsections list and describe some of the data encodings and service components that are most relevant for this experiment.

6.1 Service components

6.1.1 Web 3D Service (W3DS)

A Web 3D Service (W3DS) is a portrayal service for three-dimensional geodata such as landscape models, city models, textured building models, vegetation objects, and street furniture. Geodata is delivered as scenes that are comprised of display elements, optimized for efficient real time rendering at high frame rates. 3D Scenes can be

interactively displayed and explored by internet browsers with 3D plug-ins, or loaded into virtual globe applications.

6.1.2 Web View Service (WVS)

The Web View Service (WVS) is a portrayal service for three-dimensional geodata such as landscape models, city models, vegetation models, or transportation infrastructure models. A WVS server mainly provides 2D image representing a 3D view on a scene constructed from 3D geodata that is integrated and visualized by the WVS server. In addition to these color images of a 3D scene, a WVS server can advertise and deliver complementary image layers that include geometrical or thematic information: e.g., depth layers, surface normal data, or object id information.

6.1.3 Tile Caches

Map tiling represents a technology that can help to reduce work load and round trip time for service-based map provision. So, e.g., the WMS Tiling Client Recommendation[1] (WMS-C for short) suggests constraining an OGC WMS to strictly define fixed tiles for its map delivery with fixed extent and resolutions. It then becomes possible to render and cache those "standardized" tiles a priori or on-the-fly. Such cached map imagery leads to reduced loading times at the client side.

TileCache[2] is an implementation of a WMS-C compliant server, which has been developed by MetaCartaLabs and is available under BSD license. It provides a dedicated caching and rendering backend and allows users to set up an own local disk-based cache of any WMS server and use it in any WMS-C supporting client. Its usage speeds up client WMS applications by factors of 10-100.

6.1.4 Coordinate Transformation Service

The Coordinate Transformation Service (CTS) used for 3DPIE is a specific service set up and hosted by the University of Heidelberg. It is based on the proj.4 Cartographic Projections Library[3] and can be used to transform coordinates from and to various Coordinate Reference Systems (CRS) in a service-oriented way.

6.1.5 Google Elevation Service

The Google Elevation Service[4] is a specific Web Service which is part of the Google Maps API and provides height information for arbitrary locations all over the planet, both specified in the WGS84 reference system. For a given coordinate pair, it returns the ellipsoidal height value above (or below) the reference ellipsoid. Supported output formats are JSON and XML. While this service can generally be used for free, the

---

[1] http://wiki.osgeo.org/wiki/WMS_Tiling_Client_Recommendation

[2] http://tilecache.org/

[3] http://trac.osgeo.org/proj/

[4] http://code.google.com/intl/en/apis/maps/documentation/elevation/

number of requests is currently limited to a maximum of 25,000 requests per day. Up to 512 locations can be passed to the service within one single request. Apart from individual point locations, also coordinates of a path can be used and processed.

For instance, the following URL requests an XML document which contains the ellipsoidal height of a square in Central Heidelberg, which is approx. 122 (meter above WGS84 ellipsoid):

http://maps.googleapis.com/maps/api/elevation/xml?locations=49.409605,8.693183&sensor=false

In 3DPIE, the height information retrieved from the Google Elevation Service was used to align building positions to the Google Earth terrain model.

6.2 Data encodings and bindings

A variety of data formats have been standardized for describing, exchanging, and visualizing 3D (geographic) data. In the following subsections, the 3D data models and formats that have been in focus of 3DPIE are described more detailed.

6.2.1 CityGML

The City Geography Markup Language (CityGML) is an open data model and XML-based format for the representation and exchange of virtual 3D city models. It is based on the Geography Markup Language version 3.1.1 (GML3). CityGML represents four different aspects of virtual 3D city models: semantics, geometry, topology, and appearance: CityGML not only represents the shape and graphical appearance of city models but specifically addresses the object semantics and the representation of the thematic properties, taxonomies and aggregations. CityGML objects can be represented in up to five different, well-defined levels-of-detail, LOD0 to LOD4 with increasing accuracy and structural complexity.

Most of the geographic data used in 3DPIE has been provided in the CityGML format, including data from Paris, Berlin, and Mainz.

6.2.2 KML/COLLADA

The Keyhole Markup Language (KML), developed by Google, is an XML-based 3D graphics format used to visualize geographic data in 3D virtual globes and 2D web browser or mobile mapping applications. Google Earth includes KML viewing and editing capabilities. KML 2.2 became an OGC Standard in 2008.

COLLADA is an XML-based exchange format for 2D and 3D graphics data, maintained by the Khronos Group. The format supports object texturing and animation but has only a limited support for semantics information. KML files can reference COLLADA-encoded 3D objects in their own coordinate systems; for example, Google Earth uses COLLADA for the description of 3D objects.

6.2.3 X3D

X3D is the successor of VRML and is standardized as ISO/IEC 19775 (architecture and abstract capabilities), 19776 (encodings), and 19777 (API). It features four increasingly complex baseline profiles and several additional profiles, which define the features that a compliant X3D system must support. Most notably, the "Full" profile supports a "Geospatial" component which provides, e.g., for high-precision geo-referencing, appropriate geometric primitives and dynamic level-of-detail. Besides this, the "Full" profile encompasses all of the 33 standard components. However, as only a small subset of X3D is needed for the purpose of geodata portrayal, the "Geospatial" component is also available in some implementations which do not otherwise conform to the "Full" profile.

Furthermore, X3D defines a document object model called SAI (scene access interface), events and scripting, sound, numerous sensors, programmable shaders, and more. It is therefore suitable to build interactive virtual environments.

6.2.4 X3DOM

X3DOM is an adaption of the X3D standard to (X)HTML, ensuring declarative 3D can be used inside standards-compliant browser. It aims to support a large browser base and decent X3D feature coverage, while working towards a common declarative 3D standard in the Declarative 3D community WG at the W3C. The reference implementation is maintained by Fraunhofer IGD and available under the open-source license MIT.

6.2.5 HTML5/WebGL

HTML 5, the next revision of the Hypertext Markup Language (HTML), will allow a web browser to become a development platform. A primary goal for HTML 5 is to ensure interoperability among browsers so that web applications and documents behave the same way no matter which HTML 5-compliant browser is used to access them. HTML 5 offers numerous improvements over HTML 4 such as WebSockets, embedded parsing and accelerated 3D visualization.

HTML5 is used in 3DPIE, e.g., to provide a UI for accessing the W3DS and portraying over X3DOM. Technically, the XML-based XHTML would also be a possible encoding.

WebGL is a standard for programming in 3D with the browser as platform. The final specification of the standard was published in 2010 and is defined by the Khronos Group, a consortium which is also in charge of Open GL, Open CL and OpenGL ES (embedded systems). WebGL provide a context into HTML5 canvas that is 3D Computer Graphics capable without plug-in.

6.2.6 OpenStreetMap data format

OpenStreetMap uses the XML format to define and convey models. There are only a few simple data primitives in OSM, namely *nodes*, *ways* and *relations*, each can have an arbitrary number of properties (tags) as key-value pairs.

A *node* is the most basic element of the OSM schema. Apart from the id, nodes also use geographic latitude and longitude to uniquely define their spatial position. An altitude value is also welcome for extending the map to the third dimension, but not compulsory yet. Nodes can be either a standalone point representing something like points of interest (POI) or a connected one acting as part of a way. Standalone nodes must have at least one tag while those forming part of a way need not.

A *way* is an ordered interconnection of 2 to 2,000 nodes that describe a linear feature such as a street, footpath, river, area etc. Attributes such as ids, nodes and tags are normally used to define a way. There is a distinctive kind of way in which the first and last nodes are identical: *closed way* or *area*. Areas do not exist as another data primitive; they are simply closed ways that are tagged to represent an area (for example: area=yes, landuse=*, building=yes), thus not all closed ways are areas.

A *relation* is a group of zero or more primitives that are geographically related or with associated roles. The attributes of id, tags and members (a list of primitives with associated role attributes) are normally required to describe a relation. Relations can be used to specify the relationships between objects, for instance, several segment ways may belong to an identical building. With the use of relations, information can be applied to the whole relation rather than repeated on each node/way, for instance, multiple ways that are grouped to a relation tagged with highway. If the limit speed changes, the tag needs to be changed only in one place instead of every way.

Besides the original XML encoding, there exists the PBF Format ("Protocolbuffer Binary Format"), which was designed to support future extensibility and flexibility and is intended to replace the XML format. Data size is about 50% smaller, write access is 5x faster and read access is 6x faster compared to XML [1]. A lot of software used in the OSM project already supports PBF in addition to the original XML format, plus there are several tools to convert OSM data from PBF to XML format and vice versa.

7 Experiment design and architecture

7.1 Data sets

Several partners contributed real world data sets for 3DPIE, which could be integrated into different 3D portrayal servers and/or could be accessed from different 3D clients.

7.1.1 Paris city model

IGN provided a textured 3D dataset of Paris from BATI-3D®. BATI-3D is a 3D city models database from IGN, which can be used for visualization purposes. The database contains 3D buildings with real roof shapes; buildings are textured with the best images of the true orthophotos. The city model is associated with a high resolution DTM. The 3D data is available in several formats: 3DS, CityGML, KMZ, and DirectX. BATI3D is currently developed on a per-contract basis, and licensed accordingly. Data has been produced, e.g., for the following cities: Paris, Montbéliard, Annemasse, Marseille (partial), Lyon (partial), Nantes (partial), Dijon (partial).

The Paris data set provided for 3DPIE contained a terrain model and buildings models in the CityGML 1.0 format, including semantics and appearance data; building models are modeled in LOD2; textures are provided as texture atlases. The data set covered a total area of about 100 km2, provided in 446 tiles (500*500 m2 each), and had a total size of about 62 GB zipped data. Access to this dataset required signature of a Non-Disclosure Agreement.

Integration of the Paris data set into the 3D portrayal servers revealed several issues that IGN was able to correct. Those were an incorrect structuring of CityGML 'interior' elements, a lack of CRS information, and an incorrect MIME type value.

The following issues were identified but could not be solved within the timeline of the project:

- Invalid polygon elements (less than 3 points): This is a defect of the production chain that will be corrected in the future.

- Missing ground surfaces, which make the product improper for analysis purposes: The product is currently for visualization purposes only, other uses will be investigated in the future.

- Chopping buildings at tile borders, using the same id for each part: This design decision taken by IGN turned out to be an issue in the generation of the COLLADA models of these buildings in the 3DCityDB tool, as the building 'id' no longer was a unique building identifier across tiles. Another option would be to encode complete buildings in one tile only (e.g. using the centroid of the building to determine which tile it belongs to).

Additionally, texture atlases used as texturing mechanism in the Paris dataset were not supported by all the tools used for 3DPIE: For example, the 3DCityDB at their current stage of development did not support texture atlases; individual texture images had to be generated using an external tool. The CityServer3D actually was capable to transparently handle and load the texture atlases.

7.1.2 Berlin city model

Before the start of this IE, IGG had already administered a complete 3D city model of Berlin which was commissioned by The Berlin Senate and Berlin Partner GmbH and was developed within a pilot project funded by the European Union in the period from November 2003 to December 2005. The data is managed in a 3D City Database (3DCityDB) and is accessible only to IGG staff. However, this running 3DCityDB instance was used for 3DPIE as the basis for the CityGML to KML/COLLADA portrayal process and is also dynamically queried by IGG's W3DS when requested to serve any area of the city of Berlin.

Also, two small subsets of this Berlin city model, which are also publicly available, have been used for 3DPIE. They include building models in LOD2 with appearance data (textures). The data has also been hosted in a 3DCityDB at IGG.

7.1.3 Mainz city model

The Mainz city model was created on behalf of the city of Mainz (located in Southwest Germany). It is based on airborne first- and last echo laser scan, which was corrected using fixed reference points on the ground. Cadastral data was used to help determine building and rooftop geometries. The elevation model was stored in PCI Geomatics RAW format. From this data an LOD1 model was generated automatically and finally converted to files in standard formats, amongst them CityGML. The Mainz data set is available in the Gauss-Krüger Zone 3 CRS (EPSG:31467). It encompasses approximately 138,000 Buildings and covers around 285 km2 (including several suburbs).

Also, more detailed planning data was available for the Mainz city model. This includes building data of the Münsterplatz in Mainz with quite detailed textures and also partial sewage ducts together with manholes. These data sets were provided as separate layers through a CityServer3D instance. The W3DS, as an integral component of CityServer3D, reflects this choice by offering several layers.

7.1.4 Blacksburg city model

The Blacksburg city model is a product of the 3D Blacksburg Collaborative – an open community effort launched by the Center for Geospatial Information Technology (CGIT) and the Visual Computing Group at Virginia Tech (VT). The project targets the development of a spatial data infrastructure for the campus and the town that enables interactive 3D visualization in a variety of real-world applications and display venues. With both private and public models, the 3D Blacksburg project addresses domains such as community resilience and emergency management, town planning, social networking, crowd sourcing, and economic development. Publishing to X3D is a key piece of the project strategy in terms of IP and data control as well as providing the same assets that can be shared in open source multi-user systems like DeepMatrix and also experienced on a mobile device or a fully immersive VR room.

3D Blacksburg is published from an ESRI ARC-GIS backend to a set of file directories for KML/COLLADA and X3D clients. For X3D, this includes the latest public terrain and imagery data sets from the town and the state. There are a little over 3,500 LOD1 buildings in the database, 69 at LOD2 and 23 at LOD3; most features are referenced in VA State Plane coordinates. Additionally, layers for the regional watershed and stream center-lines have been published to the X3D platform. The 3DBlacksburg.org public distribution is Creative Commons with Attribution and demonstrates how geospatially-referenced 3D can be valuable to diverse stakeholders. Rather than as a versioned archive, W3DS and WVS promise to deliver these resources in a more dynamic and service-oriented way.

7.1.5 OpenStreetMap data

The data from the OpenStreetMap[5] (OSM) project is fundamentally different from the other data sets described above. Firstly, it is not a local 3D city model but contains world-

[5] http://www.openstreetmap.org

wide 2D geometries along with a number of attributes (tags), which only partly imply 3D information (such as building:height). A description on how this data can be transformed into 3D geometries and hence be used for 3D portrayal is given in Section 8.6. Secondly, the data has not been acquired by some reliable, high-quality public administration authority but rather has been created by voluntary mappers all over the world. This crowd-sourced approach is also referred to as "Volunteered Geographic Information" (VGI) [2]. Promising and popular as VGI is, it raises questions about the completeness and quality of OSM data. Recent work on these issues is reviewed briefly in the following.

As of November 2011, about 250 GB OSM data has been collected by more than 500,000 users all over the world, who have created approximately 1.2 billion nodes and over 110 million ways [4]. The main data source for the volunteers are individually measured .gpx tracks or the digitization of aerial imagery which has been released to the public domain for that purpose e.g. from Bing or Yahoo. However, huge parts of the data have not been captured by the users directly, but result from public domain data imports. For instance, the TIGER data of the US Census Bureau has been entirely imported into OSM and makes up a big part of the available map data in the US [5]. The completeness of OSM is generally very heterogeneous. It depends on several factors like, e.g., the population density, the widespread access to broadband internet, and the popularity of the OSM project. It can be observed that most mappers are active in Central Europe and hence the most complete maps can be found there. Also, there is a huge discrepancy between urban and rural areas. For example, the OSM data collected for Germany and the UK is richer than authoritative data in urban areas, however, it is less complete in rural areas (see Zielstra and Zipf [6] and Haklay and Ellul [7]). At the bottom line and without going into too much detail, it can be said that the most complete and highest quality OSM data is currently available for metropolitan areas of big cities in Central Europe.

In recent months, the mapping of building ground plans in OSM has become more and more popular. As of November 2011, there is a total number of about 46 million buildings (and counting) mapped world-wide, of which 5 million are located in Germany [9]. This is nearly 30% of the entire German building stock [11]. Together with the detailed street and path network and other mapped features apparent in OSM, it becomes possible to use this data for creating 3D city or landscape models. 3D-related tags that can be used for this purpose include the building height (currently over 620,000 tags), the number of levels (ca. 440,000 tags) or the roof type (ca. 60,000 tags). This 2D to 3D transformation is being done within the OSM-3D project (cf. [12], [13], Section 7.2.4 and Section 8.6).

7.2 Software

Based on various software systems, servers and client implementations, several approaches for setting up a 3D portrayal pipelines have been tested and demonstrated. These different software systems and how they were used for the 3DPIE are described in the following.

7.2.1 CityServer3D

The CityServer3D[6] is a client-server system for the storage, visualization, and processing of spatial data. Geo-information from different sources is integrated into an object-relational database and optionally placed on the web to be accessed from different clients. Spatial data is made available through various standard-based interfaces. Interoperability of CityServer3D with existing spatial data standards and infrastructures supports an effective management of 3D city models. In addition to 2D and 3D geometries in different levels of detail (LoD) CityServer3D can be used to store and process technical and metadata.

The CityServer3D includes different tools to support the sustainable management of three-dimensional city models. An integrated rule-based system allows application-specific access and consistent data management. Configurable development rules serve to automatically keep distributed data sources consistent in terms of content. Different methods integrated into the system support the integration and harmonization of heterogeneous spatial data. In this way a high quality of the data stock can be reached and ensured in the long run.

The rule-based system also allows the context-sensitive selection and visualization of 3D models. In this model, the data is made available to different users in the form they need to perform their daily work. The handling of a city model by different authors is governed by access rights. The built-in versioning allows to trace all processing steps.
The CityServer3D also includes different methods for a rule-based analysis and enhancement of the data quality of spatial data. 3D city models can e.g. be checked for completeness and consistency of content. Faults in geometries are reliably spotted and often eliminated.

For the storage of spatial data the CityServer3D uses an object-relational or schema. Due to the multilayer architecture the CityServer3D is independent of the actual database management system. The modular structure allows for high extensibility and maintainability. As the integration of new, application-oriented software modules is very easy the connection to existing system landscapes and business processes is possible without artificial difficulties.

CityServer3D is implemented in Java using the OSGi component model. It can use MySQL, Oracle, PostGIS/PostgreSQL and other Databases as a storage back-end, while offering standard services such as WMS, W3DS and proprietary services to manage the city model itself. CityServer3D runs on Windows and Linux.

As part of the experiment, two CityServer3D instances were provided on a 4-Processor Intel XEON-based machine. The two instances served geodata from two local MySQL databases loaded with the Paris and Mainz datasets.

[6] http://www.cityserver3d.de/en/the-product/technologies/

7.2.2 3DCityDB

The 3D City Database[7] (3DCityDB), developed by IGG, is a free 3D geo database to store, represent, and manage semantically rich all-purpose 3D city models based on CityGML and on top of a standard spatial relational database. Its database model contains semantically rich, hierarchically structured, multi-scale urban objects facilitating complex GIS modeling and analysis tasks, far beyond visualization.

CityGML is the format of choice at IGG for city model storage, not for visualization in this IE though. For visualization purposes KML/COLLADA (portraying the CityGML contents) was chosen instead. Stored data can be exported in different 3D visualization formats and styles from 3DCityDB through the 3DCityDB Import/Export tool, a Java-based front-end that allows for high-performance importing and exporting spatial data. Supported data formats are CityGML, KML, and COLLADA.

In this IE, two 3DCityDB instances containing the city models of Paris and Berlin were provided and served as data repositories and source for the preprocessing step of converting CityGML data to KML/COLLADA (through the 3DCityDB Import/Export). They also provided the running W3DS serving the Paris and Berlin data with necessary input for discerning which objects are comprised in the requested bounding box and must be delivered to the client.

A complete Berlin city model was already available in a 3DCityDB at IGG (not accessible to other participants), which was immediately ready for use. A test dataset containing a small part of the Berlin city model was made available to all participants in CityGML format.

The Paris data set provided by IGN in CityGML format was imported directly into a 3DCityDB (by means of the 3DCityDB Import/Export tool) hosted at IGG.

7.2.3 IGG Web 3D Service

For 3DPIE, IGG developed a Web 3D Service based on the "Draft for Candidate OpenGIS® Web 3D Service Interface Standard, Version 0.4.0." and ran two W3DS servers for central areas of the cities of Berlin and Paris. In its first version, the IGG W3DS delivers exclusively buildings objects; other object types like street furniture, vegetation, terrain models will be included in a later version.

While mostly compliant, the IGG W3DS differs in two main characteristics from the specification: Firstly, the supported output format is KML/KMZ instead of X3D. Secondly, the IGG W3DS allows a consumer to apply more than one style to a layer. Currently, the IGG W3DS implements the W3DS GetCapabilities and GetScene operations; also, both HTTP GET and POST methods with KVP and XML encoding are supported.

[7] http://www.3dcitydb.net/

The IGG W3DS is implemented in Java making use of Servlets and runs on Tomcat 7.0.16 on top of a Linux server. The implementation is open source but has not been made widely available to the public due to the lack of documentation.

The W3DS servers running at IGG for both Berlin and Paris data share the same code base. They only differ in their URLs, the cached data they serve, and the 3DCityDB they are connected to.

7.2.4 OSM-3D Web 3D Service

The OSM-3D[8] W3DS has been developed by the University of Heidelberg and has been used in several research projects dealing with Spatial Data Infrastructures and 3D graphics. The main focus was to provide a component for delivering virtual city models within a distributed geospatial information system, which comprises services for web maps, geocoding, routing, sensor data, catalog, geo-processing, and others.

Advanced features include handling of multiple LODs for buildings, custom SLD styling, and automatic tiling of large models. The W3DS implements the interface versions 0.3.0, 0.4.0, and 0.4.1 interface versions.

In addition to the default X3D format, it also supports VRML and COLLADA. Importers exist for Shapefiles, CityGML, VRML, and OpenStreetMap data. Processed 3D data is stored. The implementation is based on open source products Apache Tomcat, Java Topology Suite, Xj3D, Aristoteles, GeoTools, PostGreSQL, and PostGIS for spatial indexing.

7.2.5 HPI 3D Server and Web View Service

The HPI Web View Service is developed as reference implementation for the Web View Service interface specification. Accordingly, it follows an image-based approach to deliver 3D content to various types of clients. The HPI Web View Service is part of the HPI 3D Server, which is an ongoing project to create a flexible service-based 3D portrayal infrastructure and focuses on the storage, management, and delivery of large scale, textured 3D geovirtual environments, especially, on digital 3D city models. In the scope of the 3DPIE, the CityGML datasets of Paris (provided by IGN) and the publicly available Berlin dataset were integrated as a basis for image generation and delivery.

7.2.6 XNavigator

The Java-based XNavigator software is a feature-rich W3DS client, which can be used to view and interactively explore 3D city and landscape models using data from distributed data sources. The 3D content in X3D format is fetched automatically from a standards-based W3DS server. This includes automatic tiling and view-dependent generalization of the Digital Elevation Model (DEM) as well as the objects on it (e.g. buildings or trees). The different layers that are available in the W3DS such as buildings, labels or POIs can be added to or removed from the created 3D scene on the fly. The client offers a wide

[8] http://www.osm-3d.org/home.en.html

range of rendering properties, so that the look of the scene can be changed by the user. Also, it supports styling via 3D-SLD [22].

Apart from the W3DS rendering, several other OGC-compliant services are supported. For example, external data from a Web Feature Service (WFS) can be included as well as sophisticated analysis functions defined in a Web Processing Service (WPS) (e.g., [23]). Also, many parts of the OpenGIS Open Location Services (OpenLS) standard are supported, such as geocoding and a POI search based on the Directory Service. Moreover, routes from A to B can be calculated with the Route Service and displayed in 3D. The 3D Emergency Route Service uses a combination of a WPS describing an emergency scenario and the Route Service, avoiding critical areas [24]. Both integrated in one scene leads to a useful visualization. With this standards-based client-server architecture, many different data sources can be (re-)used and a high level of interoperability is guaranteed. The XNavigator is open source and can be either downloaded or launched via Java WebStart[9]. It has been developed by the GIScience group at the University of Heidelberg.

7.2.7 InstantReality Player

The instantreality-framework is a high-performance Mixed-Reality (MR) system, which combines various components to provide a single and consistent interface for AR/VR developers. Those components have been developed at the Fraunhofer IGD and ZGDV[10] in close cooperation with industrial partners.

The key component in 3DPIE is the instant reality player, which implements an X3D system with a large coverage of the X3D standard components. Development also aims at improving and extending the X3D standard.

However in 3DPIE, only a fraction of the capabilities are used, since service-delivered scenes are of a static nature and many X3D components target dynamic scenes.

7.2.8 BSContact Geo

The BS Contact Geo 3D engine from Bitmanagement Software GmbH enables the visualization of geographic information in third-party hardware and software products. BS Contact Geo is based on BS Contact and its capabilities. The software is expanded and is focusing on the special requirements of the GIS community using ISO standard compliant formats like CityGML and Geo-nodes from X3D and VRML standards of the Web3D consortium.

7.2.9 HPI 3D WVS Clients

As proof of concept, the HPI has developed several clients that exploit the HPI 3D Web View Service. These clients are based on different technologies, support different client platforms and configurations, and provide different functionalities, e.g., they provide

[9] http://www.osm-3d.org/Start.en.htm

[10] Zentrum für Graphische Datenverarbeitung, Darmstadt, Germany

different degrees of interactivity. All these clients consume the Web View Service interface provided by the HPI 3D Server. Clients include, e.g., the HPI 3D Web Display Client and the HPI Mobile Client.

7.2.10 Google Earth

3DPIE also tested how to integrate service-based 3D portrayal capabilities with Google Earth, which is Google's virtual globe application that is widely used for a variety of applications and use cases. In its base version, Google Earth is a free software that allows for the interactive exploration of geospatial data including, e.g., a 3D terrain model, satellite and aerial images, 3D city models, or vegetation models. Also, it provides additional geodata of various categories; this includes, e.g., photographs, videos, borders and labels, environmental phenomena, etc. This underlying base data is managed and hosted by Google itself; however, Google Earth can be extended by user-specific spatially referenced data, e.g., 2D and 3D models (KML/ COLLADA), traces, or images. Google Earth also provides functionalities for accessing and integrating map data from WMS servers. In 3DPIE, we tested this feature for accessing a W3DS. Additionally, we made use of NetworkLinks, a KML feature, to access remote 3D portrayal services.

7.3 Experiment setup

7.3.1 Experiment phases

The experiments are grouped into three work items, which represent three different phases of the IE and which are aligned along the 3D portrayal pipeline: data integration, service integration, and service delivery.

7.3.1.1 Data integration

Data integration phase includes experiments that test and demonstrate the integration of 3D geodata in standard 3D formats into 3D databases, servers, and services.

7.3.1.2 Service integration

Service integration phase tests the interoperability of multiple W3DS and WVS servers, as well as the exploitation of these services from various clients.

7.3.1.3 Service delivery

Service delivery phase mainly tests the applicability of W3DS and WVS approaches for web-based and mobile 3D portrayal of geospatial 3D environments. For this, various web clients and mobile clients are accessing the W3DS and WVS servers established in the phases before.

7.3.2 Experiments overview

The overall 3DPIE included several experiments within the three experiment phases (Figure 2). These experiments are characterized in the following sections.

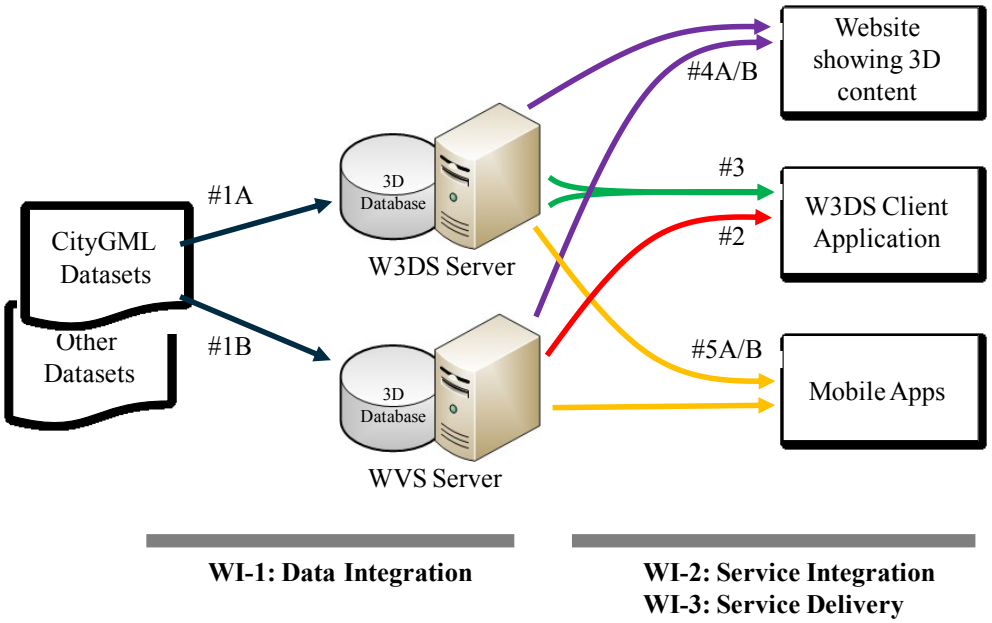

Figure 2: Data flows of all experiments in the 3D Portrayal Interoperability Experiment.

7.3.2.1 Import raw data into W3DS and WVS servers (Experiment 1)

CityGML is a potential input format for 3D streaming and visualization services. However, also other formats from BIM and GIS fields may be addressed. This input data needs to be converted and optimized to be served as 3D graphics data through a W3DS or to be rendered efficiently through a WVS; e.g., performance can be increased by batching geometry meshes or combining textures. This experiment will include these conversion and optimization processes. COLLADA, X3D, and 3DS will be considered as storage and exchange formats. Results will be stored in a database or as files, depending on service implementation.

1A) Integration of 3D geodata into a W3DS server

1B) Integration of 3D geodata into a WVS server

7.3.2.2 Linking WVS and W3DS (Experiment 2)

This experiment addresses the interoperability between W3DS and WVS on the interface and application level by integrating WVS images in a W3DS client: Perspective images will be displayed on a virtual canvas in the 3D scene of a W3DS client. A benefit of this is that more complex rendering techniques can be applied or other information layers can be made visible, that are too heavy for client-side real time 3D rendering (e.g. streamlines) or that cannot be transferred to a client due to legal issues.

7.3.2.3 Integration of multiple W3DS in a 3D client (Experiment 3)

In this experiment data is integrated seamlessly on the object/model level. This experiment will test basic W3DS interface capabilities that are necessary to merge multiple data sets, e.g., format and CRS conversions.

7.3.2.4 W3DS/WVS for browser-based portrayal (Experiment 4)

This experiment will test the ease of including 3D geospatial content in web pages as multimedia component, e.g., based on Java applets, dynamic images, or WebGL viewers. Also technologies for direct HTML integration may be tested. This could be the basis for further mashup experiments. Many client components are already available and provided by participants, but they need to be tested against other server implementations. W3DS and WVS are addressed:

4A) Browser-based 3D portrayal through W3DS

4B) Browser-based 3D portrayal through WVS

7.3.2.5 W3DS/WVS for mobile portrayal (Experiment 5)

This experiment will test the applicability of the 3D portrayal approach for the visualization of and interaction with large 3D worlds on thin clients such as smart phones and tablets:

5A) Mobile 3D graphics from W3DS

5B) Mobile perspective images from WVS

7.4 Use Cases

Interoperable 3D portrayal capabilities represent a valuable building block for various applications and systems for many application areas. One major use case we had in mind when designing the 3DPIE experiments is urban planning (Figure 3). Interoperable 3D portrayal capabilities can support the various stakeholders, e.g., city planners, architects, construction engineers, but also the public to participate in urban planning processes. For this, 3D portrayal can provide means to analyze, visualize, communicate, assess, and decide about actual and planned situations. For example, a land surveying office or an external 3D content provider could set up a 3D portrayal server that integrates and provides access to geospatial data that is relevant for planning purposes, including terrain models, thematic maps, building models and even subsurface data such as utility network models. City planners could easily access and use this data for planning purposes. Also, architects could be provided with 3D context buildings through a 3D portrayal service as a basis for their design work. Further, W3DS/WVS-based (web and mobile) clients and viewers could be used to communicate several designs to the public and, e.g., to request for comments or to set up polls on those.

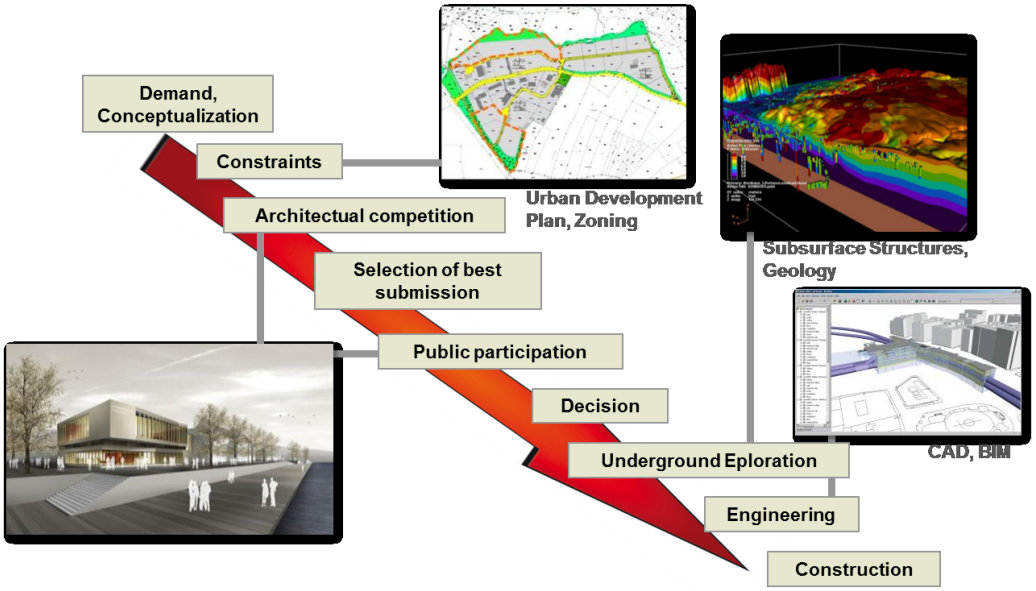

Figure 3: Examples of using 3D portrayal capabilities within the urban planning process.

8 Experiment activities

8.1 Importing Paris data into IGG Web 3D Service

8.1.1 General approach

As KML is the data format of choice at IGG for visualization, all 3D data provided by the IGG W3DS is served in KML format. Depending on the style requested by a client, the W3DS serves either pure KML (footprint, extruded, geometry styles) or KML with embedded COLLADA models (COLLADA style). The provided KML/COLLADA data focuses on coordinates, surfaces and textures and is as such only a subset of all information available from the original CityGML data: KML mainly emphasizes on data presentation while CityGML is more semantically oriented.

Only geometries and textures of building objects served by the W3DS are included in the KML representation. Additional display options like "balloons" or building "highlighting on mouse over" were not considered for this experiment, even though they were technically feasible with 3DCityDB Import/Export tool means.

The IGG W3DS is designed to serve to a consumer a master KML file that references pre-cached KML/COLLADA data. To determine which objects to include into a W3DS GetScene response, the W3DS makes use of the underlying 3DCityDB. The overall architecture of the IGG W3DS and the 3D portrayal pipeline are illustrated in Figure 4. It consists of three main building blocks:

1. A cache of previously generated KML and COLLADA files for all buildings in all styles contained in the bounding box for which the Web 3D Service is set up. These KML and COLLADA files are generated in one swipe by means of the 3DCityDB Import/Export tool. This tool connects to the 3DCityDB and with the

corresponding settings generates a separate file for each building and style in the area of interest. All these files (around 60,000 for Berlin, 14,000 for Paris) must be put into the WebContent folder of the Servlet for cached access.

2. An active connection to the 3DCityDB the KML and COLLADA files were generated from. This connection will be used for spatial queries on the DB (with the bounding box data from the request) in order to determine which buildings must be served. The DB must be the same since the cached files in the WebContent folder are all named after the gml:id (and style) of the building they represent.

3. A thin business logic layer binding the previous two blocks together. The business logic will check the incoming request parameters and if they are valid, generate with the help of the connection to the 3DCityDB a master KML file containing pointers to the relevant cached building files for the requested bounding box. The cached building files will then be subsequently requested by the client and automatically served on demand by Tomcat. No further logic interaction is needed. This architecture ensures fast response times at the cost of higher network traffic, which is an acceptable trade-off in this environment.

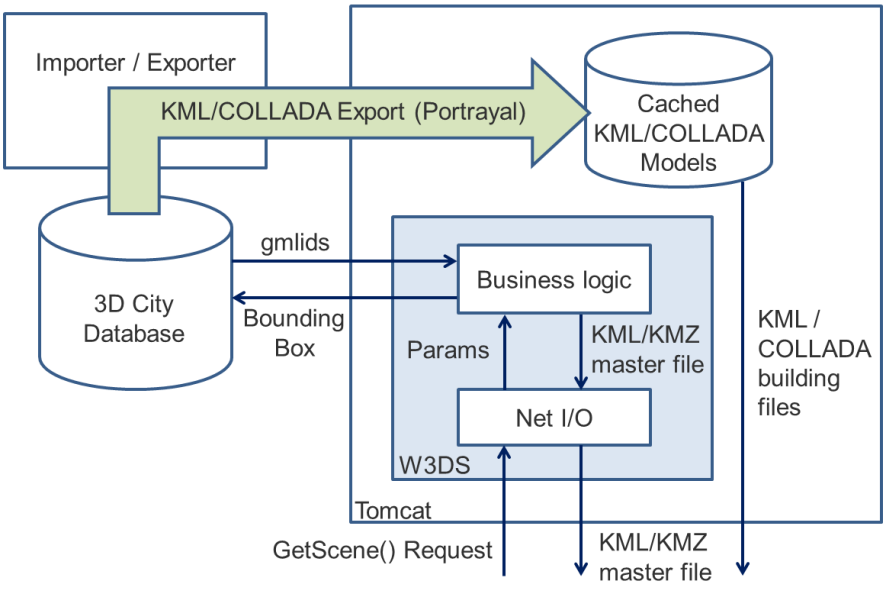

Figure 4: Scheme of the GIScience W3DS.

Differing from the W3DS specification the output format supported by the IGG W3DS is not X3D but KML/KMZ. Also, differently from the W3DS specification it is possible to specify more than one style for the same layer. To do so, the layer name must be present as many times as the layer's style names in the requests. Both lists must have the same length. It is even possible to define visibility limits for switching from one style to another; the visibility limits must simply be added at the end of the style name in the format:

_<min_visibility_limit>px[<max_visibility_limit>px]

The IGG W3DS set up for 3DPIE serves only a single layer called “citygml:Building” in four different styles named “footprint”, “extruded”, “geometry” and “collada”.

Responded 3D scenes are tiled, but not in a conventional way. The returned master KML file (containing pointers to the relevant cached building files) is itself untiled. Each building that the master file points to is contained in a region of its own (see <Region> tag in the KML specification). The region with all features it contains, in this case just the building, becomes visible when it reaches <min_visibility_limit> pixel size specified in the request, it becomes invisible again after reaching <max_visibility_limit>. If no visibility limits are specified and only one style is defined in the request, the regions (that is, the buildings) are always visible (<min_visibility_limit> = 0, <max_visibility_limit> = -1). If no visibility limits are specified and more than one style is defined in the request, predefined visibility limits apply (0px20px for footprint, 20px40px for extruded, 40px60px for geometry, 60px-1px for COLLADA).

Examples of GetScene requests are:

http://IGG.3DPIE.OGC.org/W3DS/Berlin?SERVICE=W3DS&REQUEST=GetScene&VERSION=0.4.0&CRS=EPSG:4326&FORMAT=model/kml&BoundingBox=13.40154,52.533,13.406,52.53615&LAYERS=citygml:Building&STYLES=collada

http://IGG.3DPIE.OGC.org/W3DS/Berlin?SERVICE=W3DS&REQUEST=GetScene&VERSION=0.4.0&CRS=EPSG:4326&FORMAT=model/kml&BoundingBox=13.40154,52.533,13.406,52.53615&LAYERS=citygml:Building,citygml:Building&STYLES=geometry_10px45px,collada_45px

Through the course of 3DPIE it turned out that an export of X3D directly from a 3DCityDB could be a valuable building block of a standards-based 3D portrayal pipeline. This could be used, e.g., to load specific data sets, e.g., buildings in a planning area, into X3D-enabled clients. This might be useful, e.g., for cases, where a full W3DS (which of course could also deliver X3D) is not required. Also, such capabilities could help to implement and set up a W3DS that does provide data in X3D format such as shown for the case of IGG’s W3DS in this experiment.

8.1.2 Paris data import and W3DS configuration

The Paris data provided by IGN in a series of CityGML files covered an area of approximately 137 km2 of the Paris city center. It was provided in a tiled form: The whole bounding box (643174.3, 6857518.932, 657513.248, 6867080.967) was divided into 434 tiles that summed up to 50 GB.

This complete dataset was imported into a 3DCityDB hosted at IGG by means of the 3DCityDB Importer/Exporter Tool. This tool is provided along with the 3DCityDB and allows a user to easily load CityGML data into the database.

Then, from the filled 3DCityDB a relevant subset of this data was exported directly into the KML/COLLADA format using again the Importer/Exporter Tool. It achieves data

export (from 3DCityDB) and format conversion (into KML/COLLADA) in one single step. The Importer/Exporter Tool allows a user to precisely control the export and conversion process. This includes, e.g., the region and object types to export, the level of detail to use, color parameters and texture settings, KML-specifics to apply, and usage of the Google elevation service to position objects correctly on Google's terrain model.

Also, settings were chosen so that for each building a subfolder named after the building's gml:id was generated. Each subfolder would contain four different files for the same building, one for each style. The file name results from the combination of the building's gml:id and the style's name.

As a result of this naming strategy each Building is accessible under the URI http://<W3DS_address>/<folder_for_building_layer>/<building_gmlid>/<building_gmlid_style>.kmz. These KML/COLLADA files serve as the service's cache; they are generated only once. Later, these URIs will be dynamically embedded in the master file that are created and returned as service response by the W3DS. The transparent structure of the links makes debugging and possible future enhancements of the W3DS a straightforward task.

After this generation of KML/COLLADA files per building, the whole subfolder collection for buildings was copied to the WebContent folder of the IGG W3DS. The WebContent folder is, as the name implies, the default root folder all cached contents for a servlet are delivered from. The business logic layer of the IGG W3DS is written accordingly to pick the right files from the right subfolder in WebContent when the corresponding gml:id of a building was found inside the Bounding Box of the W3DS GetScene request. The business logic is the same for all IGG Web 3D Service instances. Only the service's connection data must be adapted to connect to the right database and be able to check a request's bounding box against the correct 3D city model data.

8.2 Importing Berlin data into IGG Web 3D Service

Already before the beginning of 3DPIE, a complete Berlin city model was available in a 3DCityDB at IGG (not accessible to other participants) and immediately ready for use and test. The activities required for importing the data into the IGG W3DS for Berlin were identical to those for the IGG W3DS for Paris: portrayal of the 3DCityDB contents to KML/COLLADA using the Importer/Exporter Tool, copying the resulting collection of subfolders into the WebContent folder of the servlet implementation and specifying the connection data for the Berlin 3D city model.

8.3 Importing Mainz data into CityServer3D

The city of Mainz uses CityServer3D for the management of their 3D city models. The data to import was provided as CityGML files along with PNG images as textures for LOD2 data like Münsterplatz and sewage duct planning data. Additionally, a digital terrain model is available. The data was generated via laser scanning of the terrain and imported in PCI Geomatics Raw data format.

As CityServer3D directly supports the formats used for 3DPIE, the import did work as expected and no problems occurred.

With the help of the CityServer3D WebAdmin, a web client, the Mainz data set was imported into the CityServer3D.

CityServer3D was one of the first implementations of W3DS since the 0.3.0 draft was published, and has been updated to conform better to W3DS 0.4.0 for 3DPIE. As the W3DS implementation is a first-class citizen in CityServer3D, at import timethere is no decision about styles or data formats to use in the experiments.

The "GetScenario" operation, a proprietary W3DS extension for delivering tailored portrayals is CityServer3D's substitute for styles. It is conceptually unfit for standardization (which might change given some progress in production rules standardization), but has been used in 3DPIE to deliver tiled terrain data in Mainz. It is mostly compatible to the GetScene operation and can, potentially, be substituted for GetScene.

The CityServer3D contains on-demand request and texture caches which are employed for 3DPIE. Therefore, some requests may take substantially longer than others. Pre-generated or pre-filled cache content was not provided, except as a side-effect of internal testing.

8.4 Importing Paris data into CityServer3D

The Paris data was supplied as zipped CityGML files along with TIF images as textures. The main issue was the large amount and size of image data, which made up the major part of the data set. Especially for the use in a web browser the sheer size of the TIF images leads to massive performance degradations in terms of resource usage and bandwidth. Therefore we decided to convert the images to the JPEG format. This conversion was done with the standard tool ImageMagick. All references within the CityGML files had to be adapted accordingly. This was done with a custom Unix Shell script. This reduced the size of the data from 61.9 GB to 8.2 GB, which was easier to handle.

8.5 Importing Paris data into HPI Web View Service

The goal of this experiment is to integrate a large CityGML dataset into an existing Web View Service instance. Generally, this requires converting the 3D data provided in CityGML format into an optimized internal representation that can be rendered efficiently while at the same preserving their association to the origination feature information.

8.5.1 Workflow

The target of the integration process is to generate compact and efficiently accessible data representations from the given input data. The Paris dataset includes a textured 3D building models as well as a digital terrain model textured with orthophotos. The processing of the terrain model was performed analogous to the processing of other types of city objects.

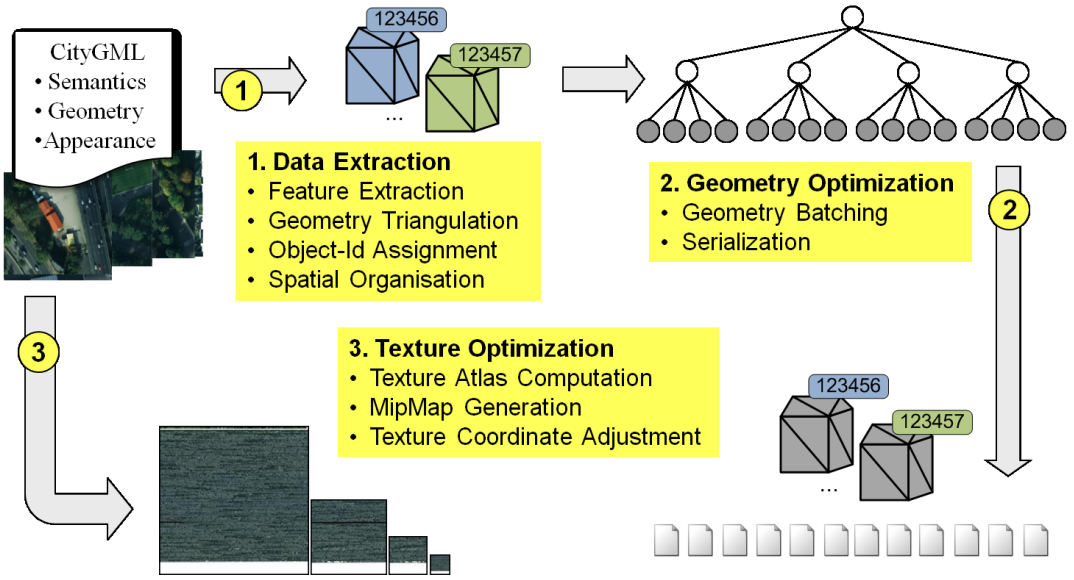

Figure 5: Integration of massive CityGML data into HPI 3D Server includes three stages: data extraction, geometry optimization, and texture optimization.

Data integration is a preprocessing step, which basically consists of three stages: data extraction, geometry optimization, and texture optimization (Figure 5), which include, e.g., the following operations:

1. Extract feature data:
    - Extract all features from the CityGML documents and create internal feature representations
    - Transform geo-coordinates into computer graphical coordinate system
    - Triangulate feature geometries
    - Assign object ids to feature geometries
2. Optimize geometries
    - Organize geometries in an optimized out-of-core spatial data structure, e.g., as tiles of a regular quad-tree
    - Batch geometries, i.e., merge geometries, to further reduce data footprint
    - Serialize the quad-tree tiles, which include batched geometries
3. Optimize textures
    - Reorder and compress textures
    - Generate optimized mip map levels
    - Adjust texture coordinates to specific texturing techniques

Input for the WVS data integration process were a) the original data set provided by IGN, which included texture atlases, and b) a second CityGML data sets without texture atlases, which was exported from IGG's 3DCityDB that was hosting the Paris buildings.

8.5.2 Results

The complete Paris city model was successfully preprocessed and loaded into the HPI Web View Service. It turned out that the techniques to process and optimize building data worked out also for large terrain data.

Due to specifics of the Paris data model regarding multiple occurrence of object ids in various CityGML files, the resulting internal representation contains few inconsistencies for buildings cut at data tile borders (see next Subsection), which however could be resolved by extending feature extraction mechanisms to work across input files.

To reduce processing time, we exploited the HPI Future SOC Lab, a high performance computing platform, which provides, e.g., server systems with huge main memory (up to 2 TB) and multiple CPUs (up to 128 logical cores). To reduce I/O costs, a 160 GB RAM disk for processing the Paris data was used. Also, the CityGML input files were processed by 8 threads in parallel. So, it took, e.g., less than 6 hours to import the Paris data set without texture atlases (418 CityGML tiles with JPEG-compressed textures and a total data size of 12 GB unzipped). The size of the resulting (internal) data is currently 4.4 GB in total.

8.5.3 Problems and solutions

8.5.3.1.1 Granularity of object ids

Assigning object ids to the extracted and processed geometries is required for associating the internal geometries to additional feature data, e.g., feature attributes. However, it is not fully specified what granularity should be used for assigning such object ids. Therefore, several object ids could be assigned according to different criteria. For example, they could be assigned for each occurring city object (e.g., each sub feature of a Building instance gets assigned an own id), per object class (e.g., same id for all Building objects), or per top level CityObjects (e.g., same id for all sub features of a Building instance). Finally, we chose an assignment of object ids per top level city object, because this fits best to the Paris dataset, which is modeled primarily in CityGML LOD2.

8.5.3.1.2 Terrain data encoded in CityGML

In earlier implementations of the preprocessor, there was no focus on supporting terrain elevation data included in the CityGML data source; rather a separate data source had been used for generating a separate digital terrain representation. However, the approach of merging geometry into batches of triangles (originally designed for single objects such as buildings) proved to be applicable also to the tiled terrain data. Nevertheless, this tiling leads again to a problem of object id assignment: The data loading and import mechanisms might need to recognize the Relief data distributed over several files as parts of one special scene object, i.e., one terrain model.

8.5.3.1.2.1 Strict geographical tiling of the data set

The Paris 3D city model was delivered strictly tiled by geographical space: Feature's whose geometries overlap tile borders were cut into several parts and distributed over these tiles each stored in a separate file. However, all these feature parts carry the same gml:id. Currently, this conflicts with the 3D server's internal data import and object id assignment mechanism, which assumes that one feature (i.e., one gml:id) shows up only once during the overall processing of the input data. Hence, these different parts get assigned different object ids. However, this circumstance does not affect the image rendering and display process; only functionalities that rely on object ids (e.g., object

selection) are affected by this fact. While it is perfectly valid in CityGML to have the same gml:id in several CityGML documents, it could ease the processing of this data if a single feature would be completely included in one document, e.g., by assigning the whole overlapping feature to a single tile and accepting possibly overlapping bounding boxes of the single tiles. However, the HPI 3D Server's import process is planned to be extended to cope with such multiple occurrences of identical gml:ids.

8.6 Importing OpenStreetMap data into OSM-3D Web 3D Service

As already described in the Section 7.2.2, OSM data mainly comprises two-dimensional geometries. These geometries are provided in that way, that several geo-referenced nodes are combined to ways and/or relations. Additional 3D-related information (e.g. the height of a map feature) can be attached as a key-value pair to the corresponding OSM feature.

Thus, processing of OSM data for the usage in the OSM-3D W3DS is a two-step process: Firstly, two-dimensional geometries of the corresponding map features (mainly building footprints) need to be computed. Secondly, all available information (i.e. key-value pairs) and the 2D-geometry can be utilized for generating three-dimensional models.

The generation of two-dimensional geometry is based on database processing algorithms. A regularly updated PostgreSQL database with OSM data is utilized as a data container. Several SQL scripts and PostGIS functions (e.g. ST_Geometry etc.) are then utilized for generating two-dimensional footprints of the different map features (buildings, land usage areas etc.). Thereby, it needs to be considered that a polygon in OSM can be either a simple polygon, which is mapped by one closed way (a closed way is a way whereat the first and last node are equal), or a more complex polygon (e.g., a polygon with holes), which is mapped with the aid of relations. In the latter case, the SQL script has to generate a polygon by deriving the outer polygon(s) and cutting out the inner polygon(s). After a polygon has been created, it is then again stored in a PostgreSQL database. For more information on OSM processing and the generation of 2D geometry, especially for more details on the system setup and SQL commands please refer to [13].

These 2D footprints are the basis for further processing steps. Due to the open key-value pair methodology in OSM, there is (in theory) a lot of 3D-related information available in the OSM database. For the generation of 3D building models, one of the most important characteristics of a building is its height. Within OSM, there are potentially two keys available which contain such information: on the one hand the key height (which directly contains the height of the building) or the OSM key building:levels (the amount of levels), which allows an approximation of the height of a building. Obviously, the former key is advantageous, but nevertheless if not available, the information contained in the latter key is better than no information at all. This height information is used to extrude the footprint geometry of a building and to generate a 3D volumetric body. This results in a block model that corresponds to a CityGML LOD1 building model. For the generation of even more realistic building models, roof geometries are generated on top of the extruded buildings. For that, the OSM keys that contain roof information (currently building:roof, building:roof:shape and building:roof:type) are evaluated. Since building:roof:shape is defined as best practice [14] and also most commonly used, this key is evaluated at first, and only if not supplied, the other keys also considered.

Currently, the building generation process supports the creation of flat roofs, pitched roofs, cross-pitched roofs, hipped roofs, pyramidal roofs and gambrel roofs (cf. [14]). Additional roof types such as mono-pitched roofs etc. will be also be implemented in near future. How far the roof is raised in the air is either explicitly added in OSM with the key building:roof:height or implicitly with the key building:roof:angle (in this case the real height needs to be computed with trigonometric equations).

Regardless of what type of roof geometry needs to be generated, the algorithms always strongly depend on the complexity of the footprint polygon. For simple geometries such as those consisting of five points, the processing is quite straight forward (mainly linear algebra computation), as well as for shifted or rotated geometries. Some building geometries in OSM are slightly concave, i.e. per definition they are concave but comparing the geometry with the oriented bounding box (OBB) of the geometry, all points are very close to the OBB. This either results from imprecise mapping, but also occurs in the case of very small notches. For the creation of roof geometries for such building footprints, the approach in OSM-3D first applies a slight simplification on the building footprint, so that imprecise mappings are neglected. Additionally, it is assumed that geometries, for which every point of the geometry is closer to the OBB than a distinct threshold, do also have a simple roof geometry which is based on the OBB (can be computed with linear algebra equations). Currently, this threshold has been defined as one meter. These roofs are not necessarily equal to the real world roofs, but they can be considered as a quite good approximation of the reality. Finally, for making the building models even more realistic and appealing, the roofs are colored according to the tags building:roof:colour or building:roof:color, as well as the building body can be colored according to building:facade:colour, building:facade:color, building:colour or building:color.

8.7 Displaying KML from W3DS in Google Earth

External resources can be embedded in Google Earth (GE) using so called Network Links. GE does not support OGC services directly and cannot extract meta information such as service provider, supported protocols, supported coordinate reference systems, and available layers from a Network Link. There is neither content, language nor version negotiation. The W3DS logic respectively the description of the service content must be encapsulated in a KML document from which the user can then choose available options, e.g. enabling a specific layer. A Network Link can act as view-dependant spatial anchor triggering the download of further KML resources over the network. W3DS resources can be embedded as GetScene or GetTile request URLs in KML or by directly pointing to W3DS objects using unique IDs.

Several approaches for embedding W3DS content as KML have been evaluated in this IE which are described in the following subsections.

8.7.1 Connect CityServer3D and Google Earth

CityServer3D's approach is to offer a separate operation "GetTileDefinition" for W3DS (see 10.3.2 for details). As currently implemented, it serializes a pre-configured tile set with matching W3DS references in a number of formats. One of these being KML, the

Mainz data set could be visualized using Goolge Earth (Figure 6). As can be seen, the height and location mismatches between Google's globe and Mainz data set are quite pronounced. The exact cause of the location mismatch was not determined, but is probably linked to the CRS conversion process or problems of the same.

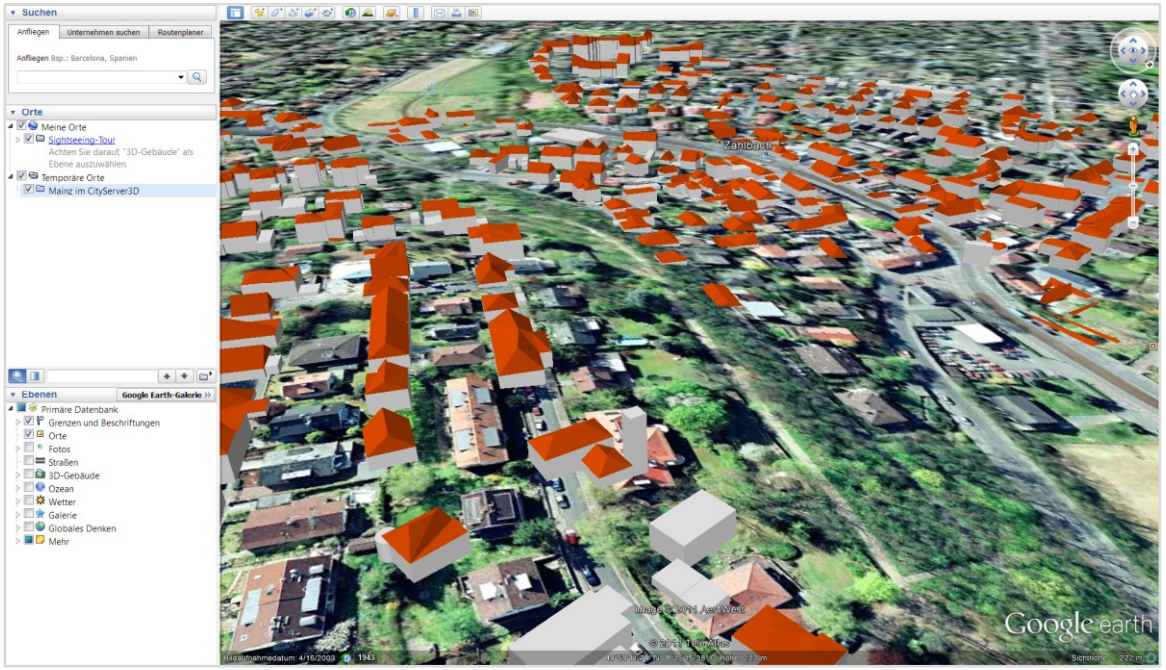

Figure 6: Mainz in Google Earth, made available over GetTileDefinition.

8.7.2 Connect IGG W3DS with Google Earth

The easiest way to display any response results from the IGG Web 3D Services is to add a Network-Link in Google Earth itself specifying a valid W3DS request in the link address field. When activated by selecting the attached checkbox on sidebar to the left the request will be sent to the W3DS and the results will be shown automatically in Google Earth (Figure 7).

Another simple way of displaying KML/COLLADA from the W3DS is making Google Earth the default application for showing kml/kmz files and call the W3DS from a web browser's address field. The download of the kml master file will start immediately and the user will be asked to save or open the downloading file. When open (at the moment or later) the files containing those single buildings included in the scene will be requested by Google Earth when the building's (footprint-) bounding box becomes large enough on screen. The Tomcat instance hosting those cached files will then serve them on demand.

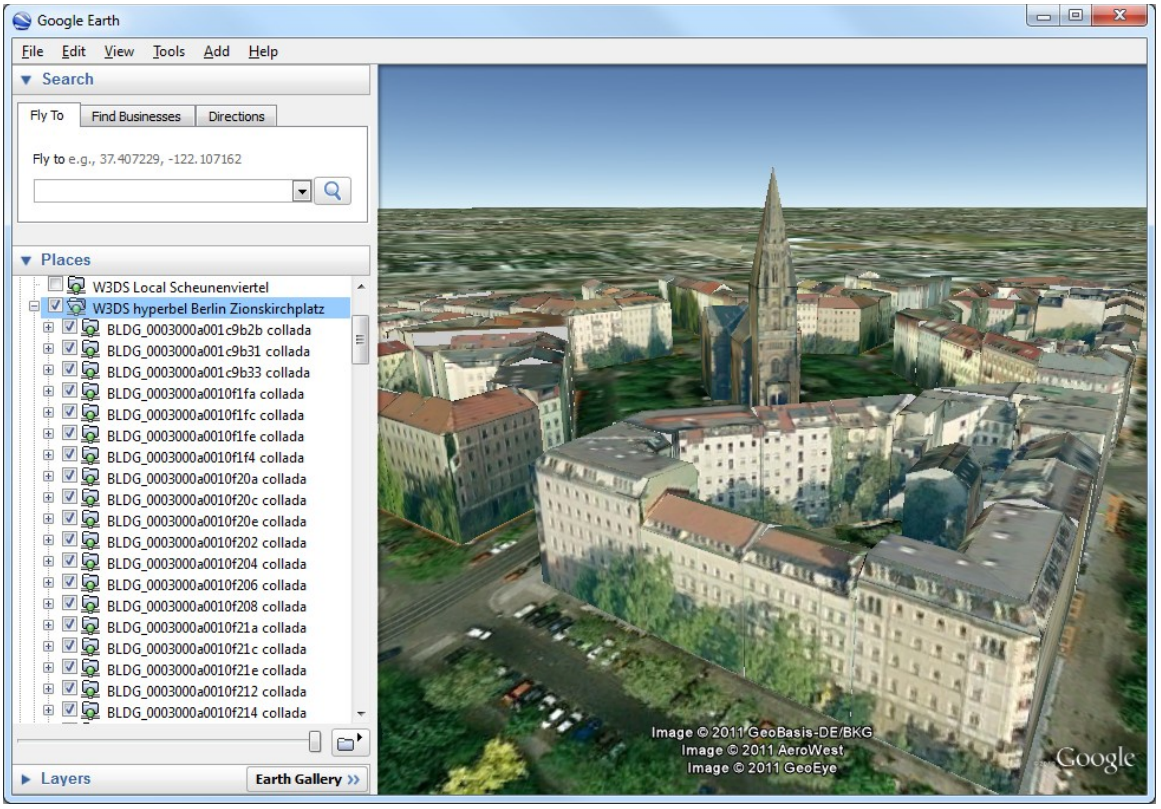

Figure 7: Network-Link defined in Google Earth.

8.7.3 Connect OSM-3D W3DS with Google Earth

W3DS GetScene and GetTile requests are using axis parallel bounding boxes for extracting a spatial subset which can be added to the client's 3D scene. Usually space is divided two-dimensionally into a grid of adjacent rectangles for defining spatial batch selection instead of downloading each object individually. The grid can be used as skeleton that defines the structure of the resulting scene. Each cell of the grid contains a service request which is triggered by a Network Link. Basically, pixel size of the cell controls when a Network Link is activated. Instead of downloading the entire data set, only the cells near the virtual camera are filled with content.

The OSM data set covers the whole planet. Creating a grid in WGS84 with reasonably sized cells of about 39.5 arc seconds (ca. 1,200 meters at the equator) would result in an excessive number of 536,870,912 grid cells. This is far too many to be encoded in KML and processed by normal personal computers. For integrating the OSM-3D the grid was therefore replaced by a dynamically created quadtree. Only a base cell of 360x180 degrees is loaded by GE as KML file. The quadtree is implicitly created by the W3DS logic. Each KML contains four sub-cells each defined by a Network Links containing a W3DS GetScene request. The W3DS does not grant direct access to its data for arbitrary bounding boxes. Only if the bounding box is smaller than a specified value, the actual data is retrieved from the database, encoded in COLLADA and packaged in a KMZ file, which is then loaded by GE. This threshold size value corresponds to the MaxScaleDenominator in SLD, which also denies access to the data at smaller scales,

resulting in a blank image. If the bounding box size is too big, then the W3DS returns another KML file with four sub-cells each defined by a Network Links containing a W3DS GetScene request. GE is able to traverse through the quad tree structure very quickly and starts downloading data when the camera is close to the ground. This integration approach was tested with OSM buildings, which are generally available at major cities (Figure 8).

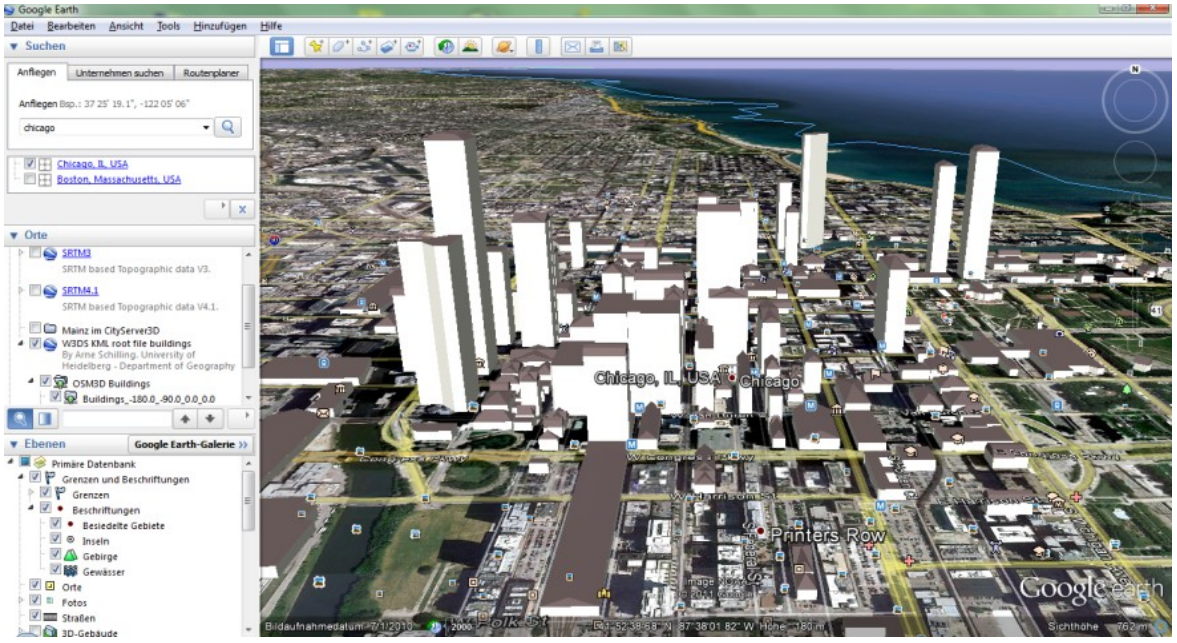

Figure 8: OSM-3D buildings in Chicago integrated in Google Earth as Network Links.

8.8 Accessing X3D

8.8.1 Web-based portrayal through W3DS using Instant Reality Player

The aim of this experiment is to we are visualize data from OpenStreetMap3D server with Instant Reality Player release 2.0. The supported data formats are X3D in its various encodings and VRML. The player directly visualizes the W3DS responses. Manual download to the local file system is not necessary. The Instant Reality Player is used to access the OSM-3D Web3D Service using VRML and X3D format.

8.8.1.1 Test subject: OpenStreetMap3D server

8.8.1.1.1 Test Setup 1: 3D scene in VRML format

Example GET request for retrieving OSM-3D data in VRML format:

http://GIScience.3DPIE.OGC.org/OSM3D/W3DS?SERVICE=W3DS&REQUEST=GetScene&VERSION=0.4.0&CRS=EPSG:900913&FORMAT=model/vrml&BoundingBox=9685,63449,971500,63464&layers=Buildings,DEM&OFFSET=969450,6344450,0

Figure 9: Scene in InstantPlayer.

Result: The scene was displayed correctly (Figure 9).

8.8.1.1.2 Test Setup 2: 3D scene in X3D format

Example GET request for retrieving OSM-3D data in X3D format:

http://GIScience.3DPIE.OGC.org/OSM3D/W3DS?SERVICE=W3DS&REQUEST=GetScene&VERSION=0.4.0&CRS=EPSG:900913&FORMAT=model/x3d&BoundingBox=9685,63449,971500,63464&layers=Buildings,DEM&OFFSET=969450,6344450,0

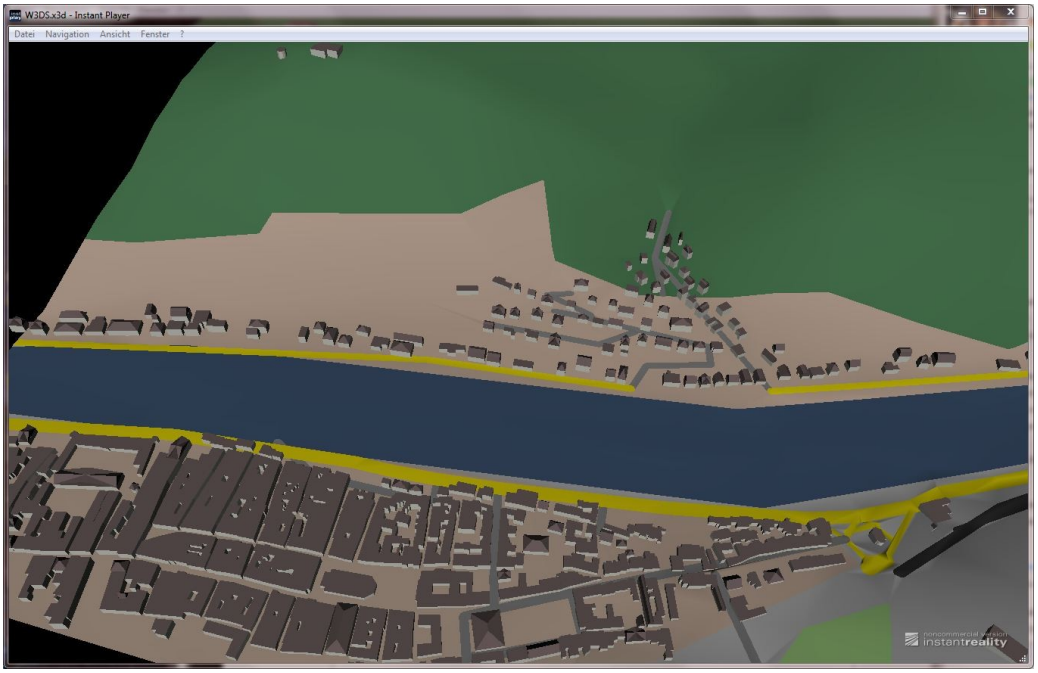

Figure 10: Scene in InstantPlayer.

Result: The scene was displayed correctly (Figure 10).

8.8.1.1.3 Test Setup 3: 3D scene in X3D binary format.

Example GET request for retrieving OSM-3D data in X3D binary format:

http://GIScience.3DPIE.OGC.org/OSM3D/W3DS?SERVICE=W3DS&REQUEST=GetScene&VERSION=0.4.0&CRS=EPSG:900913&FORMAT=model/x3d%2Bbinary&BoundingBox=790053.1172,6572361.3988,790664.6134,6572972.895&layers=Buildings&offset=790053.1172,6572361.3988,0

Result: The binary format is not working; the Instant Player stops loading with an error message. There seems to be an incompatibility in the binary representations expected. Whether the underlying cause is a violation of the binary encoding standard or a result of the known ambiguous specification could not be determined.

8.8.1.2 Test subject: 3D Blacksburg data

The 3D Blacksburg data is directly used from http://www.3d.cgit.vt.edu/index.php/83-explore-3d-blacksburg/downloads/96-download-models after required registration at http://www.3d.cgit.vt.edu/.

8.8.1.2.1 Test: 3D scene in X3D format.

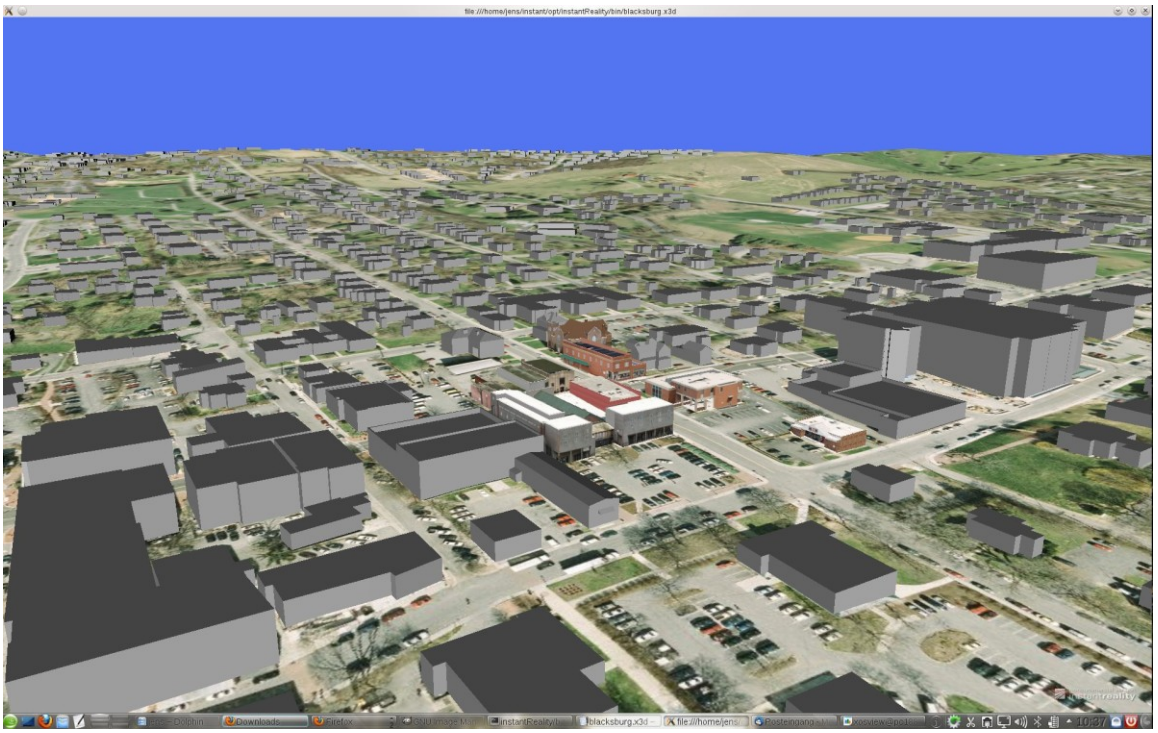

Figure 11: Blacksburg 3D scene in InstantPlayer on Linux.

Result: There are some minor incompatibilities with the viewpoint/coordinate setup. When getting closer to the buildings, the scene was clipped. After adjusting the near plane zratio in the InstantPlayer the scene could be visualized as expected (Figure 11).

8.8.2 Portrayal through W3DS using BS Contact Geo

In this experiment, the interoperability of X3D client software and a W3DS server was tested. In general the tests were successful in rendering 3D graphics data retrieved from a CityServer3D instance in Bitmanagement's BS Contact Geo. The experiment was run with Bitmanagements Client software BS Contact Geo, Version 7.220, Development build 11/2011 accessing IGD's CityServer3D instance hosting the Mainz data.

The client did use the server's operation GetTileDefinition:

http://IGD.3DPIE.OGC.org/Mainz/W3DS?do=GetTileDefinition&template=entrypoint&surface=mainz_stadt

This experiment did identify some inconsistencies regarding the interpretation and implementation of the X3D standard, which could be resolved in the course of the experiment:

4. The field 'height' of the 'GeoElevationGrid' node was not exported correctly. Some of the numbers were replaced with a certain "Unicode" character.

5. In XML coded X3D files some of the MFString attributes containing only one string value were not coded correctly. The double quotes were missing.

6. In VRML coded X3D fields the PROFILE statement was missing.

7. In VRML coded X3D files all top-level nodes were enclosed in a 'Scene' node. This is done only with XML coding.

8. Some MFString values in XML coded X3D files were not yet coded correctly.

9. Non-equal values for the xSpacing and zSpacing fieldsd of the GeoElevationGrid node did lead to distorted geometries and renderings.

8.9 Merging data from multiple Web 3D Services in XNavigator

The motivation for this experiment was to demonstrate the interoperability between multiple W3DS instances. We used the OSM-3D W3DS and the CityServer3D W3DS implementations on server side and the integrated client XNavigator as client. The goal was to merge both data sets in the client by accessing the servers in parallel with only a few configuration settings.

XNavigator is an open source development specifically designed as W3DS client and for visualizing global data sets with the accuracy needed for showing even the smallest details in a global reference system. As W3DS client it supports the W3DS logic and syntax. It supports version and language negotiation as well as automatically parsing the contents section providing information of available layers, styles, CRSs etc. Since the W3DS logic is supported, it must be possible to connect to any service instance. XNavigator was already used as a client for GIScience's W3DS in the OpenStreetMap3D project, which is available as live service.

The CityServer3D, developed by Fraunhofer was used as a second service instance due to its support of X3D. It contains the Mainz data sets, comprising building geometries, sewage ducts, as well as a detailed model of an area around the Münsterplatz. The OSM-3D W3DS by GIScience contains a terrain model, simple building geometries and points of interest.

8.9.1 Workflow

The workflow for integrating multiple W3DS servers in the XNavigator is illustrated by the sequence diagram in Figure 12.

As a first step, the service description (meta data) of CityServer3D was accessed using the GetCapabilities request and processed by the W3DS parser of XNavigator. This proved to be possible without major problems because both, the service and the client connectors, were derived from the same reference XML schema.

The second step involved modification of the client code in order to support several server instances simultaneously. The client is configured at startup using an XML file, which contains all service URLs, camera, light and navigation settings. The configuration

was extended to support multiple W3DS URLs and connection credentials (username/password). No further settings were required.

On startup, connections to all W3DSs are established and the results of GetCapabilities requests parsed. The contents sections contain information on the available layers, which is collected so that a list is provided to the user for enabling or disabling single layers. Each layer is connected to a specific W3DS URL. GetScene requests are always issued in a tiled fashion, i.e. with adjacent bounding boxes. If no tile set is defined for a specific layer, then a default tiling schema is constructed, which can be changed by the user afterwards. Credentials are passed along GetScene requests if required using HTTP Basic Authentication.

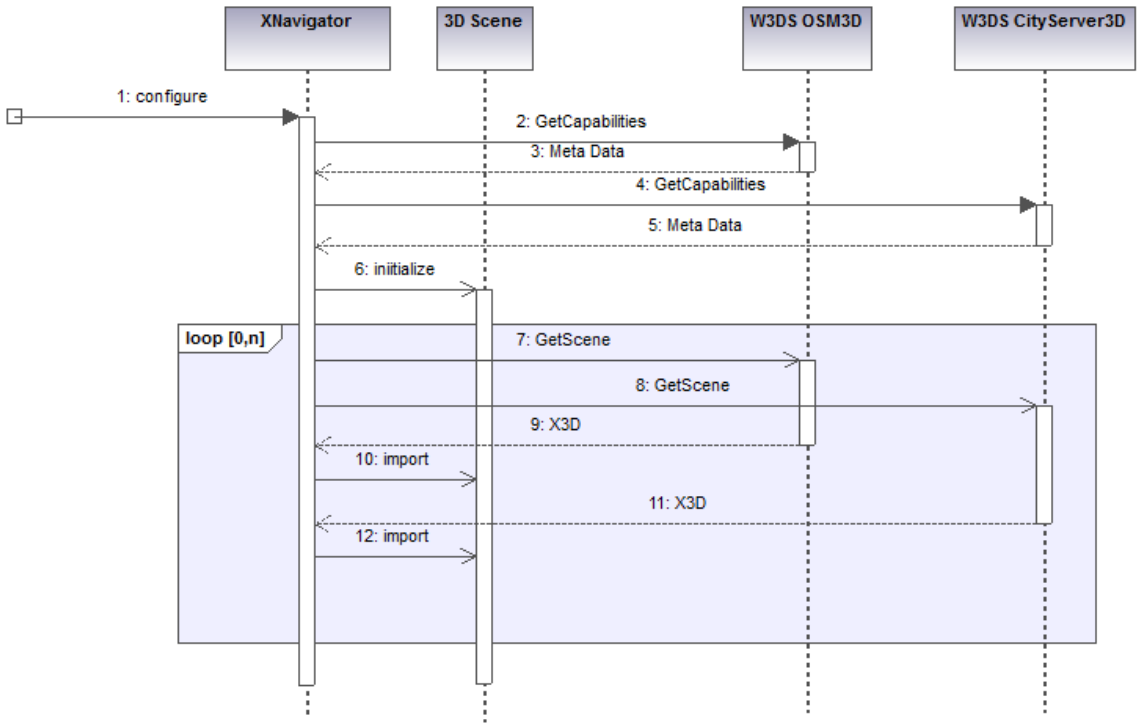

Figure 12: Sequence diagram showing XNavigator startup phase.

8.9.2 Results

An important aspect of service interoperability could be demonstrated by this experiment: that it is possible to use the OGC version, language and content negotiation mechanism for plugging in 3D services into existing applications. As an example, the XNavigator client was used to integrate 3D data from the GIScience W3DS and the CityServer3D W3DS (Figure 13).

8.9.3 Problems and solutions

We faced a few problems that could be overcome by applying special options only to CityServer3D requests.

10. Order of coordinate axes: CityServer3D uses a right hand coordinate system in which the z axis is pointing upwards. XNavigator requires a right hand coordinate system with the y axis pointing upwards. This problem was addressed by introducing an option for flipping y and z axes.

11. Usage of synonymous EPSG codes for Spherical Mercator projection: Unfortunately there are two EPSG codes around for Spherical Mercator, which was used for accessing 3D models (EPSG:900913 and EPSG:3857).

12. Layer names versus Layer identifiers: Partly layer names were used for accessing W3DS layers instead of the unique identifiers.

13. Height values of the different data sets do not match: The terrain model was created from SRTM data, which has an inherent vertical accuracy of 16m. The Mainz model was created by the local municipality having much higher resolution elevation data available. Most buildings are sunk into the ground a few meters.

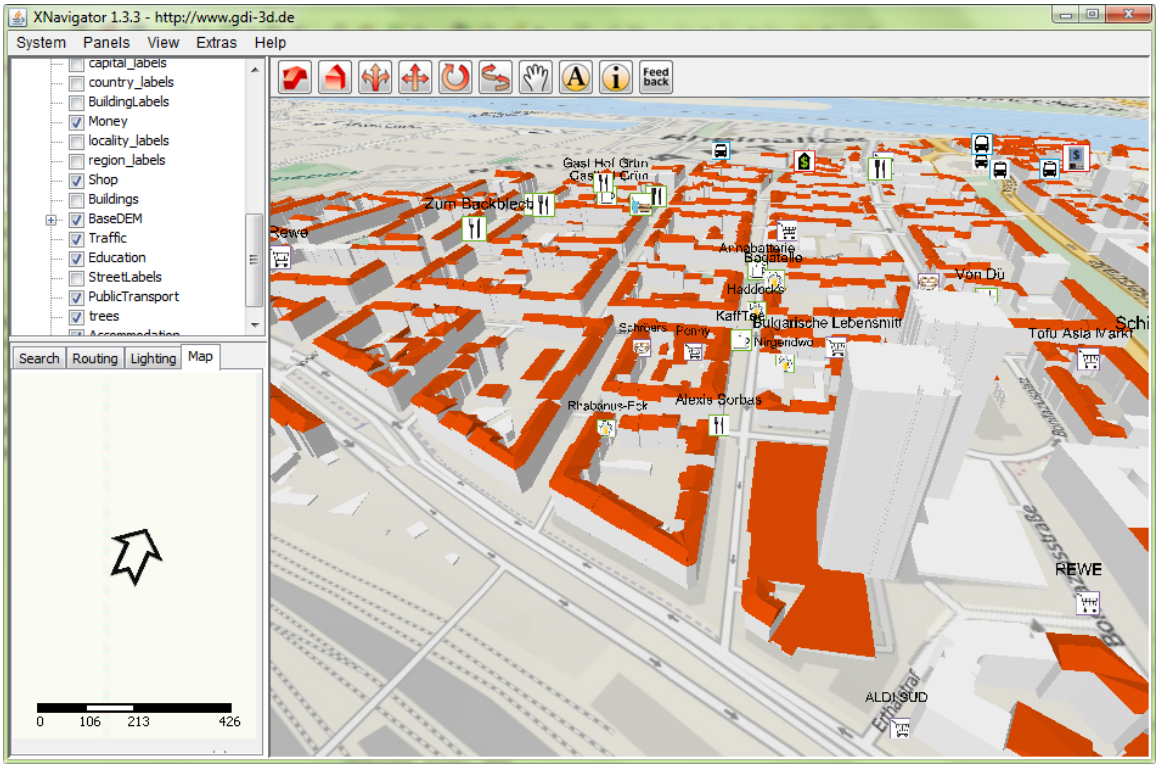

Figure 13: 3D Data from two W3DS instances merged together: 1) terrain model (textured with OpenStreetMap) and POIs from GIScience's server, 2) buildings from CityServer3D. Location: Mainz, Germany.

8.10 Merging 3D models from W3DS and imagery from WVS in XNavigator

The motivation for this experiment was to figure out how image-based services (Web View Service, panorama images, webcams, video feeds, orthographic imagery) and geo-referenced imagery can be integrated into real 3D geo applications: Photographs and web cams may provide a much more accurate impression of how a site looks like. A precondition of merging such imagery with a 3D model is a proper definition of the location and viewing angles at which the photographs were taken. Often, switching to perspective and panoramic images requires an alternative set of behaviors for zooming and panning actions. Displaying imagery can be realized within the 3D application using textured spheres, boxes or canvases, without switching to a 2D rendering mode.

Also, renderings of synthetic 3D worlds can be "outsourced" to a dedicated rendering server which allows free definition of camera location, angles and other properties, in contrast to photographs. This is the purpose of the Web View Service. Performing server side rendering may be necessary, e.g., if

14. the 3D model is too big or too detailed to be rendered by the client at acceptable frame rates,

15. the network bandwidth is too limited to transmit the amount of data required for 3D models at reasonable time,

16. the rendering exploits advanced algorithms usually not available in real time graphics (ray tracing, ray casting, caustics, soft shadows, depth blurring).

8.10.1 Workflow

The XNavigator client was used as frontend for accessing W3DS and WVS. To have a basic 3D model as reference, the OSM-3D W3DS was used. The basic setup is identical to the setup used by the live osm-3d.org service. An additional panel was implemented as plug-in for accessing a WVS. It allows the user to set the URL of the service and a spatial reference system as EPSG code, which is used for describing coordinates in WVS requests. Once the user has navigated to a place where data is provided by the WVS, he can manually trigger GetView requests towards the used WVS. The GetView requests contains a perspective definition, which must be derived from the virtual camera of the 3D viewer (Figure 14). The virtual camera is always defined as a 4x4 transformation matrix in computer graphics, from which the current position and viewing angles (pitch, heading, roll) can be easily extracted. A GetView request requires a point of camera (POC), a point of interest (POI), and an up vector (UP). A POI can be generated using POC, camera orientation, and a default distance in that direction.

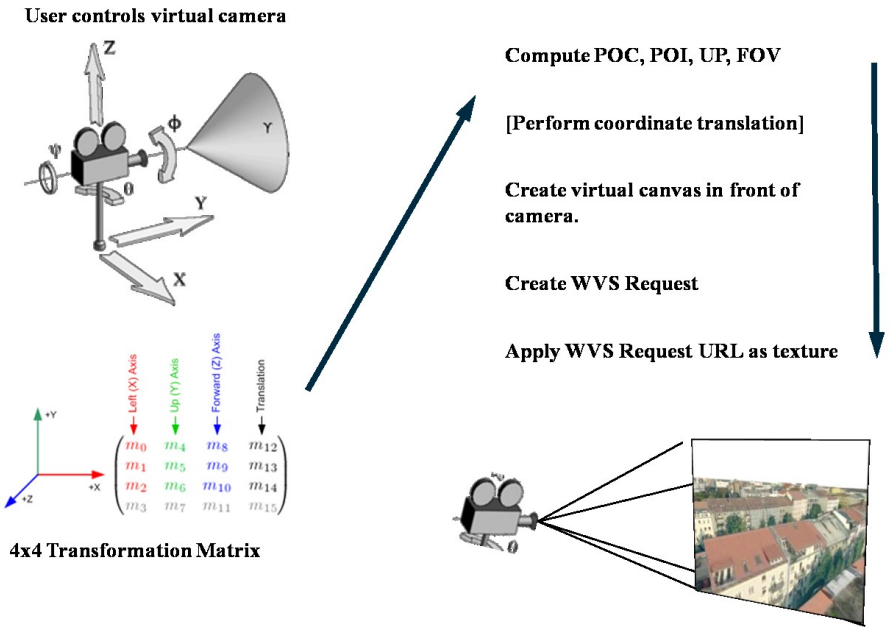

Figure 14: Extraction of GetView perspective parameters from the camera definition in XNavigator and creating a virtual canvas for image display.

Since the reference system used by the viewer may not be supported by the WVS, an optional coordinate transformation step must be implemented. For this purpose an external Coordinate Transform Service (CTS) can be used, supporting the full set of EPSG codes. This takes effect if the EPSG code in the WVS panel does not match the internally used reference system.

The constructed request URL is used as texture on a small rectangular geometry that is placed in front of the camera. The so constructed virtual canvas occludes the 3D model in the background and displays the output of the WVS GetView request.

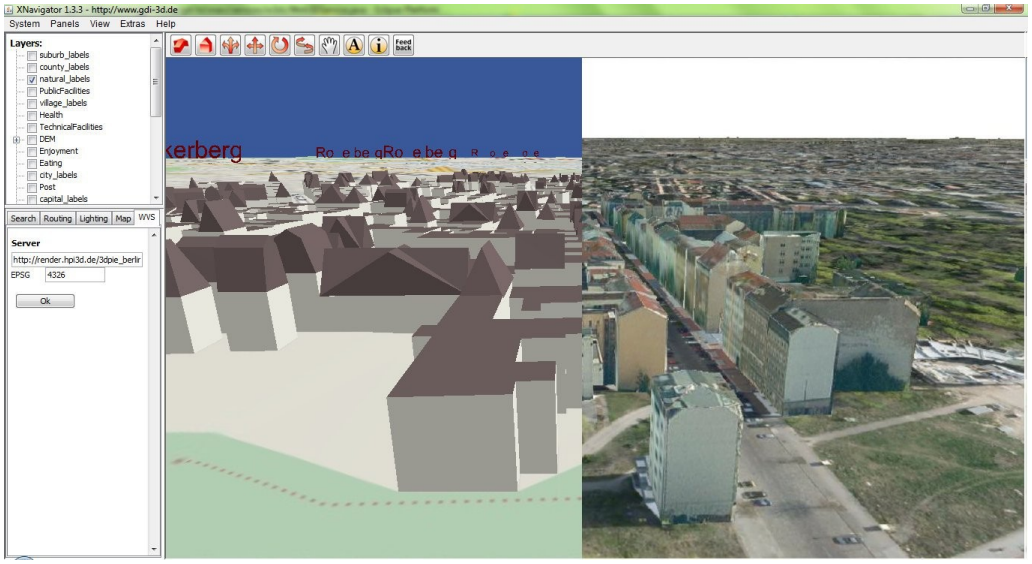

Figure 15: Perspective imagery from a WVS displayed in the 3D viewer XNavigator. The montage shows 3D content from W3DS (left) and WVS (right) for the same camera position.

8.10.2 Results

The setup was tested with both Web View Services provided by HPI. The first one was containing a subset of the Berlin model with 2 areas with textured buildings and a complete textured terrain model. The second one was containing the complete 3D model of Paris as described in section 7.1.1. It was possible to switch between both at runtime (Figure 15). The resulting imagery was almost perfectly matching the underlying OSM-3D model. Only minor perspective deviations could be observed.

8.10.3 Problems and solutions

The underlying 3D model used by the WVS may be defined in a local coordinate system, e.g. UTM. In this case it is impossible to perfectly merge the perspective imagery with a 3D globe model as used by the osm-3d.org project. However the effect is barely visible at the scales used in this experiment. It becomes only visible when zooming out and requesting an overview image.

Also, the problem of different height references becomes visible again as a vertical shift of the displayed buildings and terrain when switching to the WVS-generated image in XNavigator. This is because, on the one hand, the WVS provides images of the Berlin building models and a terrain model that is matching perfectly this building data and, on the other hand, the 3D building models generated and provided by the OSM-3D W3DS are based on a different terrain model.

8.11 Sharing and displaying WVS imagery in web browsers

8.11.1 Sharing static 3D views

Based on its HTTP-GET binding, the WVS allows for encoding requests for an image showing a specific 3D view on a 3D scene by URL. As these rendered 3D views are

compressed and formatted by standard image formats (e.g., JPEG, PNG) they can be easily integrated, e.g., in website or generally in web-based applications. Also these URL encoding 3D views can be easily shared with others, e.g., through sending by email or posting to social media networks.

An example WVS GetView requesting for accessing a 3D view showing the Louvre in Paris is:

http://HPI.3DPIE.OGC.org/Paris/WVS?SERVICE=WVS&VERSION=0.3.0&REQUEST=GetView&CRS=EPSG:4326&PORTRAYALS=WIDTH=640;HEIGHT=640;Projections=Perspective,2.336462,48.860645,70,2.336462,48.861145,35,0,0,1,90,90,1,90000;IMAGELAYERS=COLOR;FORMATS=image/jpeg;QUALITIES=95&LAYERS=terrain,bldgs&STYLES=default,textured

8.11.2 JavaScript-based interactive client

The HPI 3D Web Display Client is a 3D client application that is running in a web browser. It is mainly implemented in JavaScript and thus can be integrated, e.g., into web portals and web applications.

The client is requesting 3D views as images from the underlying Web View Service. These images are displayed to a client user, i.e., no 3D rendering capability is required at the client-side. Also the client provides a user interface that allows a user to interactively navigate through 3D portrayals in a manner that is similar to 2D map clients such as Google Maps. For example, when a user moves the virtual camera foreword, a new image that represents this new 3D view is requested and displayed; through this a step-by-step navigation in the virtual scene is possible.

Additionally, if available at the client-side, the HPI Web Display Client exploits the WebGL-capabilities of a web browser and requests not only a single image but a complete panorama (i.e., six sides of a cube map) from the Web View Service, drapes these on a cube around the virtual camera, and allows the user to look around in the scene in an highly interactive manner. As mainly based on JavaScript technology, the HPI Web Display Client can also be used in web browsers running on a mobile device.

The HPI Web Display Client has been used to access and explore the Paris 3D city model (Figure 16).

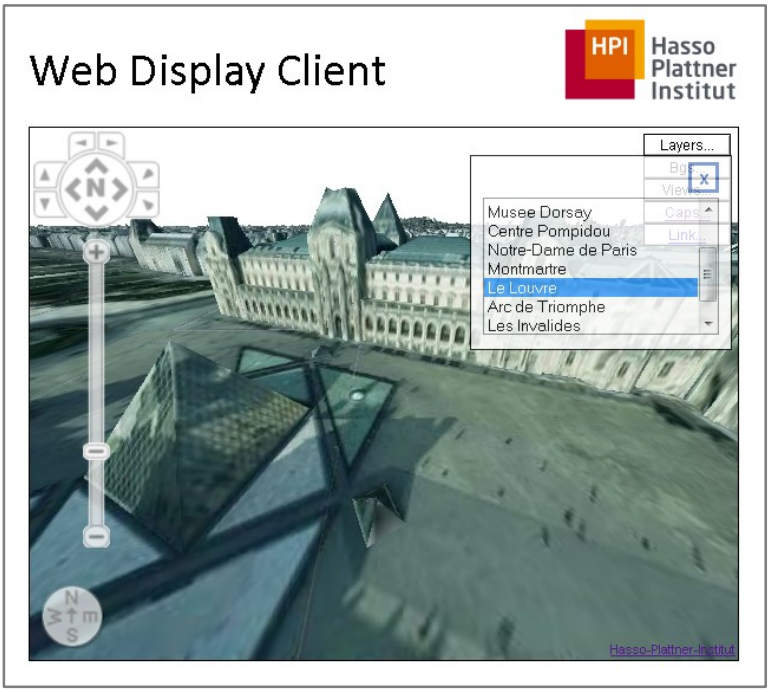

Figure 16: HPI 3D Web Display Client running in a web browser allowing to explore the 3D Paris data set.

8.12 Displaying WVS imagery on mobile clients

The motivation of this experiment was to exploit a WVS server from a 3D client application running on a mobile device for interactive 3D portrayal.

For this experiment, we used the HPI 3D Mobile Client, which provides an interactive user experience based on image-based rendering technologies. Currently it is primarily provided as an App for iOS devices, e.g., iPhone and iPad; however, as implemented in a platform-independent way, it could be ported to other mobile platforms, too. The Mobile Client provides continuous 3D visualization and creates an interactive user experience that is comparable to common desktop 3D clients. This is reached through image-based rendering techniques: The client consumes not only color images from a Web View Service but also depth representations and object id images, from which a 3D scene is reconstructed and rendered in real-time at the client side. All interaction techniques are made through the iPads or iPhones multi-touch display.

8.12.1 Workflow

The client is implemented as an iOS application. Right after starting the App, the client is requesting a specific App server to gather information about the App configuration and the WVS instance to connect to. For a specific camera position, the client then requests images for the six faces of a cube that is surrounding the virtual camera. For each of these faces it requests color images, depth representations, and object id images. To reduce network load and allow for server-side optimizations, all faces and layers are requested by a single WVS request. An example for retrieving multiple WVS image layers of such cube face by one GetView request is:

http://HPI.3DPIE.OGC.org/Berlin/WVS?SERVICE=WVS&VERSION=0.3.0&REQUEST=GetView&CRS=WGS84&BACKGROUND=default&PORTRAYALS=WIDTH=256;HEIGHT=256;Projections=Perspective,13.409292,52.518795,210,13.408311,52.521346,120,0,0,1,90,90,1,10000000;IMAGELAYERS=COLOR,DEPTH,OBJECTID;FORMATS=image/jpeg,image/png,image/png;QUALITIES=80,100,100&LAYERS=terrain,BERLIN&STYLES=orthophoto,default

According to such request the server loads the required data, renders the requested image layers, encodes those, e.g., in standard image formats, and returns those as HTTP multi-part response.

The client is consuming this data from the received data stream and is using it in various ways for visualization purposes. Firstly, it is rendering the cube surrounding the virtual camera and drapes the retrieved images on this cube. This gives the users the illusion of being positioned in a complete 3D environment. The user can explore this environment, e.g., by rotating the virtual camera or zooming in and out. In the case of zooming in, e.g., the client requests new color images showing the actual scene section in more detail. Secondly, the client is reconstructing and rendering a textured 3D mesh from the depth layer provided by the WVS, i.e., it is no more displaying the textured cube. In this mode the client allows a user to move the virtual camera freely, e.g., moving it forward/backwards or sideward.

Based on the object id layer requested from the server, the client is also capable to distinguish different objects and allow a user, e.g., to select and highlight specific objects (e.g., buildings) and to rotate around such objects to inspect it.

Figure 17: iOS App running on the iPad and iPhone providing WVS-based access to and interactive visualization of the 3D Paris city model.

8.12.2 Results

The HPI WVS Mobile Client was used successfully to access high-quality rendered images of the Paris 3D city model including building models and digital terrain model and to display and explore the 3D scene on the iPad and iPhone. The client allows a user to interactively control the virtual camera to explore the 3D city model as well as to select, highlight, and inspect specific buildings (Figure 17).

8.12.3 Problems and solutions

As already describe in Section 8.5.3, the issue of strict geographical tiling of the original data currently results in different internal object ids assigned to different parts of, e.g., the same building or even for (original) terrain tiles. In the client App this gets visible when selecting and highlighting such objects (Figure 18).

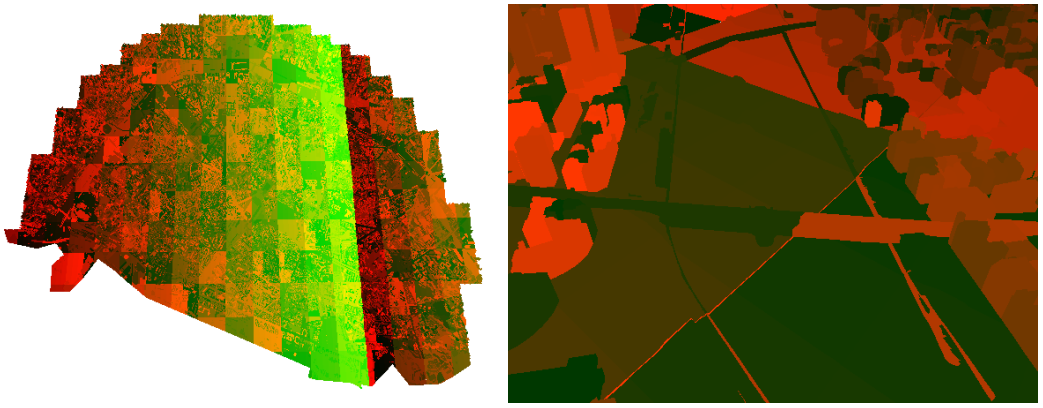

Figure 18: Object id images retrieved from the HPI WVS for the Paris data set. They show that currently different terrain tiles (left) and parts of the same bridge (right) have assigned different object ids.

8.13 Rendering CityGML data in the web browser

8.13.1 Overview of the approach

The LSIS Laboratory (Laboratoire des sciences de l'information et des systèmes), component of the French CNRS (Centre national de la recherche scientifique), has been working on 3D data exchange between client and server. Our attention has only been focused on building representation in CityGML.

The general approach follows the idea that geometry and semantic need to be exchanged together. Generally, this is not taken into account in a purely graphics-based approach for 3D portrayal of urban data. However, this approach aims at keeping feature semantics closely linked to its geometry representation. For this, three kinds of tests have been done, which will be presented in detail in the results section. The first test fetches an entire CityGML file from an OGC standardized stream. The second test includes processing of the original CityGML file to improve the data transfer characteristics

between server and client. In the third test, data exchange is done based on a JSON-stream.

Two clients have been prototyped to study these data exchange approaches. The first one is a thin client based on WebGL. The second one is a thick client based on C++.

In server side, a solution that is based on Java components constructed by the French company Geomatys is proposed. Geomatys also supported the development of WebGL client and Java server.

8.13.2 Tests

8.13.2.1.1 Transferring pure CityGML data

In our first experiment, we have studied the solution used in the OWS-4 testbed, which is particularly based on Snowflake software's WFS server to exchange CityGML data. For this part of the experiment, the thick client based on C++ was used to query the WFS server.

In a first step, *all* the building objects contained in a layer were requested from the snowflakes WFS server and were successfully portrayed from the received CityGML data (Figure 19).

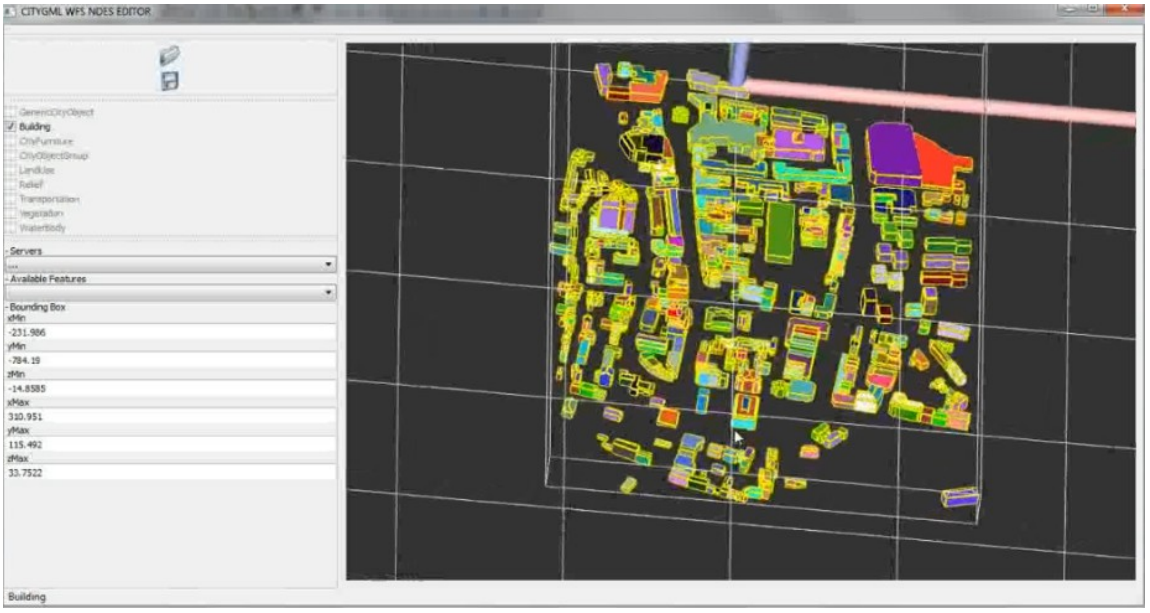

Figure 19: Screenshot of the LSIS CityGML thick Client.

Downloading an entire CityGML file implies an important latency. The GetFeature query can be tuned by using the bounding box parameter (BBOX). It is then possible to decompose the query in several ones. This of course requires managing possible duplicate building objects that could be contained in several bounding boxes.

Even if we use Snowflakes software's strategy, it appears important to cut the CityGML files in smaller ones. In this part, we have proposed to have different decomposition process:

- The first one is a thematic decomposition. In our work, we have taken into account the building layer. Other elements can be treated in a same way.
- A second decomposition is based on a spatial grid which permits to make an indexation of each element. For example, for a square of a grid, it is possible to know which building is entirely or partially contained.

Two other processes have been proposed to reduce latency and improve the portrayal process. We have built a process to simplify the geometry which is possible to use. At the server side, a tessellation operation is proposed. It permits to send only triangulated geometry, which can be rewritten in CityGML format to facilitate the visualization in client side. Another operation is based on the possibility to compute missing LOD. For instance, with a LOD3, it is possible to deduce LOD2 or LOD1. This functionality is not finished yet.

Figure 20 shows the architecture in the server side. This part has been developed in Java by the Geomatys Company. It is based on Mapfaces and Geotoolkit , open source projects led by Geomatys. The CityGML file is parsed using JAXB. After a thematic decomposition and spatial indexing, each element is eventually tessellated decomposed in different LOD.

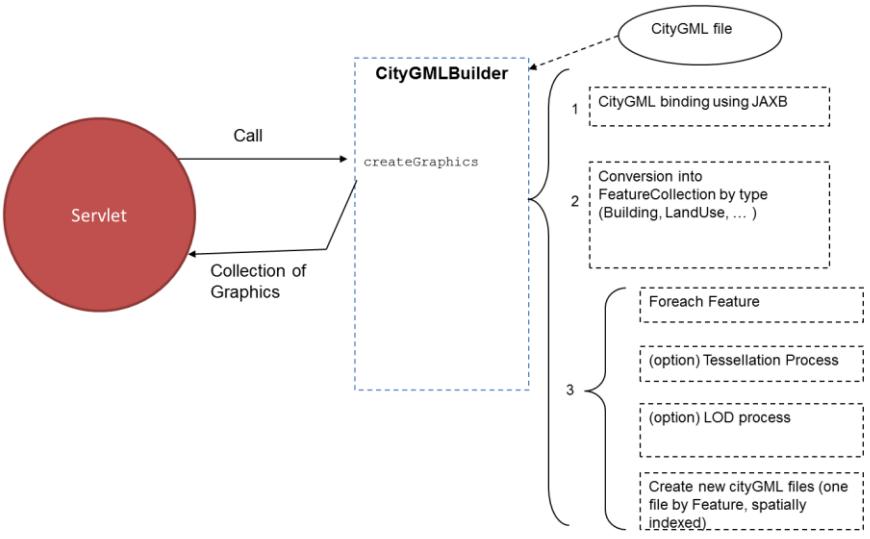

Figure 20: Architecture of the LSIS server.

The communication mechanism between client and server is very close to the WFS interface. The server responds with a CityGML document, which contains only one feature (Figure 21).

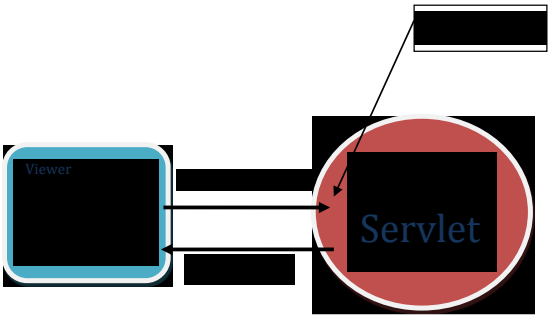

Figure 21: Data Exchange between LSIS client and server.

In a last experiment, we proposed to stream replace the CityGML stream with one using JSON as exchange format. JSON is understood in our client based on the three.js library and begin to appear in geo services. It permits to choose the information that we want to send. First experiments have been done on the geometry part; currently, thematic data and texture data is not taken into account.

In conclusion, in these experiments, we were interested in the study of 3D urban data transfers between a client and a server. Initially, we studied the possibility of transferring a CityGML data stream. The size and complexity of data makes it difficult to use such data stream on the client side, especially in the case of clients with limited computation capacities such as web browsers with WebGL support. In a second step we propose to simplify CityGML data on the server side. We finally made first experiments on a stream based on JSON. Such kind of data is easily usable with WebGL. It is then possible to send geometric and semantic data to a client.

In the future, LSIS would like to continue their work to offer a most successful prototype using WebGL. It would use a JSON data stream. Even if they can have a tiling and LOD process for urban data, it would be interesting to hide some parts behind 2D data by using a process similar to billboarding based, e.g., on images retrieved from a WVS.

8.14 Rendering W3DS data in the web browser

Test data setup: Mainz data set

Server: CityServer3D

Using a CityServer3D web-interface to render W3DS-delivered data in the client browser

The X3DOM 3D City Viewer, as the name implies, is based on X3DOM, a technology also developed at Fraunhofer IGD. It is based on Javascript and WebGL or Stage3D, which together cover a major and rising share of installed browsers.

Short Description of X3DOM 3D City Viewer

The screen in Figure 22 is divided in a 3D display to the left and an OpenStreetMap display to the right. In the map display you can move or zoom with the buttons next to

the upper left corner of the map. You can click with the left mouse button in the map and drag the mouse. This opens up a rectangular area which is to be displayed in the 3D view.

Within the 3D view you can click and drag with the mouse to rotate the 3D view. You can also click right and drag to zoom.

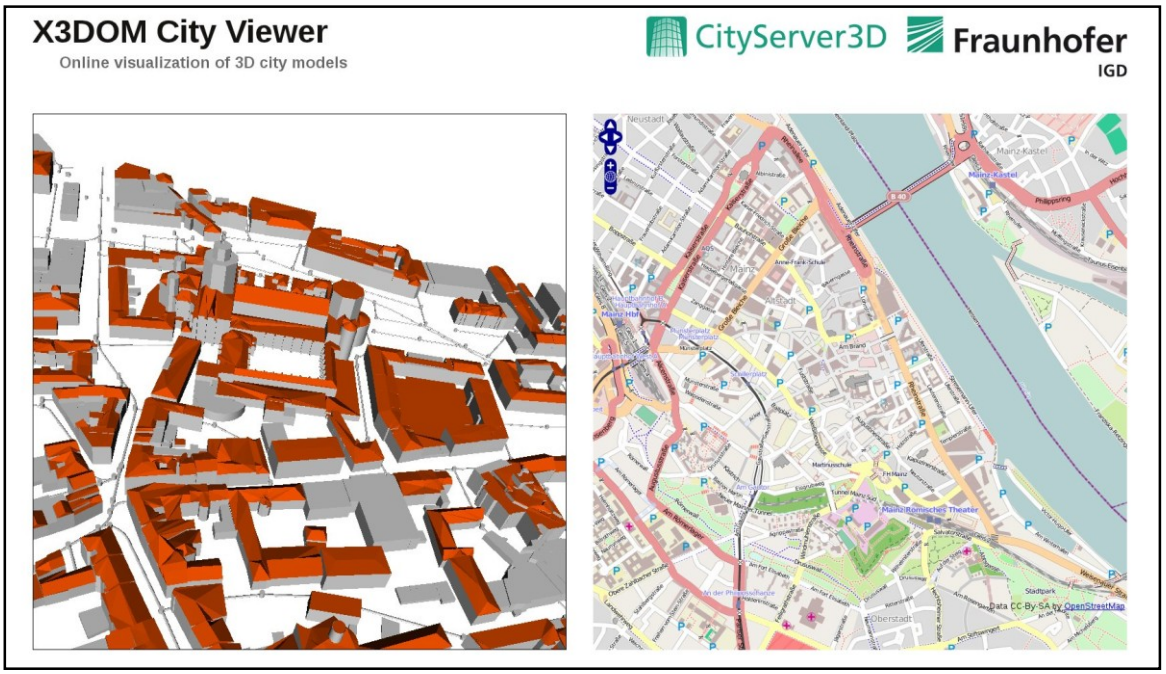

Figure 22: X3DOM City Viewer.

8.15 Rendering W3DS data on mobile devices

Test data: Mainz data set

Server: CityServer3D

X3DOM city viewer over Firefox on Android 2.3

The x3dom city viewer web interface shown above can be used on mobile devices. Currently, the x3dom component cannot actually be used, because touch-based interaction (as is common on mobile devices) is not yet implemented in X3DOM. However the component renders the geometries delivered over W3DS without noticeable additional delay (compared to a desktop system). Figure 23 and Figure 24 show the X3DOM city viewer running on a smartphone and a tablet, both running Android 2.3 oeprating system.

Figure 23: X3DOM City Viewer on Android 2.3 (Samsung Galaxy S II).

Figure 24: X3DOM City Viewer on B&N NOOK color with Firefox and Android 2.3.

8.16 Extended LOD concept for X3D

Motivation

The extensibility of X3D as a client portrayal platform provides a powerful means to implement and test new functionality. In the X3D Immersive Profile, for example, authors can use the Prototype mechanism to encapsulate and extend the scene graph with custom nodes. For this OGC project, we have sought to better align X3D LOD semantics and performance with the CityGML data model. We developed a testbed and method to explore the computational and quantitative impacts of the native ISO LOD node versus one designed with the CityGML application and semantics in mind.

Since model popping and awkward loads are a hindrance to interactive portrayal, we set out to understand the computational performance and perceptual impacts of these LOD techniques in urban and semi-urban environments. Toward this goal, we present our work to develop a new X3D LOD node whose switching semantic considers the tradeoff of speed and accuracy between LOD3 and LOD4 and whose definition considers a simplified, proxy shape. We quantify the effect of model switching with Frames-Per-Second logged over several example worlds and show that this extension is significantly faster (with upstream processing of a proxy shape) than with the native radial-based X3D LOD, even when implemented with an ECMAScript Script within a Proto.

The ISO X3D specification provides a native LOD semantic for radial, distance-based tests and child switching. However, there are many common situations where buildings are not symmetric in all 3 directions (towers, train stations, city blocks). In addition, it is typically in the transition to and from LOD4 that incurs the greatest network and rendering load. Such an LOD extension would be especially valuable to bridge the scale between GIS and BIM models of a building, exteriors and interiors for example.

Prototype

We implemented an X3D PROTO node with a target of CityGML semantics and applications where the different LODs of a building were separate X3D models loaded by an Inline (fetched from a URL). Up until LOD4, the LOD switching is driven by the simple distance calculation. However, the LOD3–LOD4 transition is managed by computing if the user is within a proxy six-sided prism defined by 8 points (Figure 25).

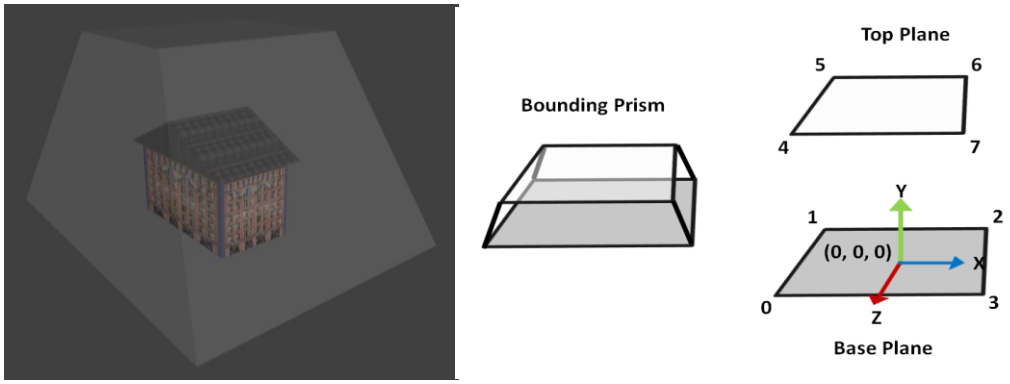

Figure 25: Proposed proxy shape for X3D LOD extension between LOD3 and LOD4.

Thus, the node signature for our 3DPIE LOD Prototype is a simple extension as shown below; the LOD4 test is 'within a six-sided manifold' (a proxy shape) defined by the 8 points of the proxyPrism field:

```
<ProtoInstance name="LODPIE" DEF="BLDG1225D" >
    <fieldValue name="position" value="100 0 100"/>
    <fieldValue name="rotation" value="0"/>
    <fieldValue name="LOD4">  <Inline url="inlines/LOD4.x3d" /></fieldValue>
    <fieldValue name="LOD3">  <Inline url="inlines/LOD3.x3d" /></fieldValue>
    <fieldValue name="LOD2">  <Inline url="inlines/LOD2.x3d" /></fieldValue>
    <fieldValue name="LOD1">  <Inline url="inlines/LOD1.x3d" /></fieldValue>
    <fieldValue name="LOD3_cutoff" value="300"/>
    <fieldValue name="LOD2_cutoff" value="550"/>
    <fieldValue name="LOD1_cutoff" value="650"/>
    <fieldValue name="proxyPrism" value="-20 0 -15 -20 0 15 20 0 15 20 0 -15
                  -12 15 -9 -12 15 9 12 15 9 12 15 -9"/>
</ProtoInstance>
```

Evaluation

Ten (10) test environments were generated and we measured Frames-per-second (FPS) for each LOD type. Each test environment was generated with different waypoint paths through a 10 x 10 city grid. Each grid block contained one building model, which is shown in Figure 26 below; model properties are described in Table 1. For these tests, the camera animation travel speed was held constant over a fixed number of waypoints (10), each of which was a visit into a building, loading the LOD4 model.

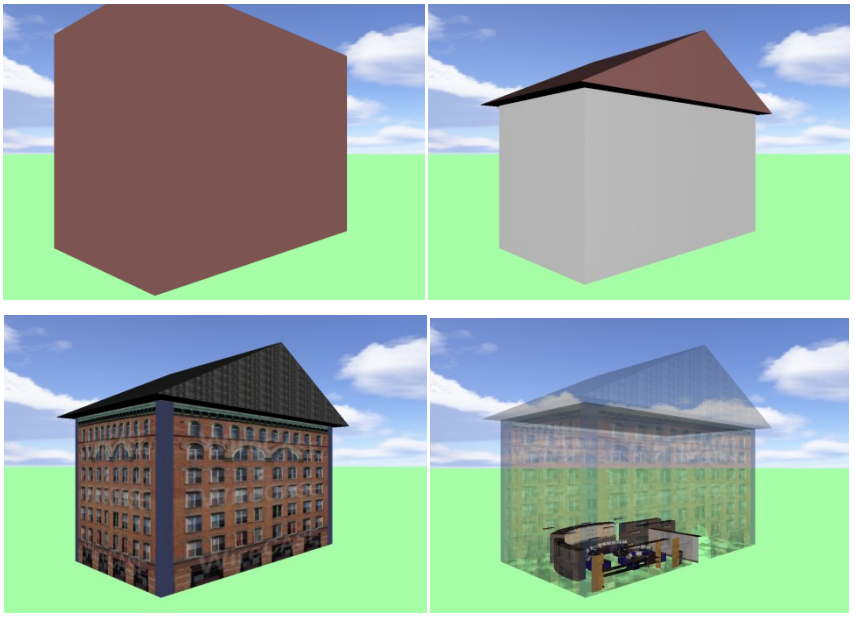

Figure 26: Screenshots of the X3D building model used for LOD1, LOD2, LOD3, and LOD4.

Table 1: X3D model attributes used in the X3D LOD extension experiment.

LOD1		**LOD2**	
Triangles:	12	Triangles:	377
Vertices Transformed:	24	Vertices Transformed:	491
Textures:	1	Textures:	1
Texture Memory (bytes):	7100	Texture Memory (bytes):	7100
LOD3		**LOD4**	
Triangles:	377	Triangles:	52224
Vertices Transformed:	491	Vertices Transformed:	66594
Textures:	3	Textures:	11
Texture Memory (bytes):	303275	Texture Memory (bytes):	761003

The LOD prototype and test environments were implemented and instrumented in X3D and tested with InstantPlayer, we collected FPS with an internal X3D Script node. FPS values were buffered to an ECMAScript array and printed out at the end of the run. The Render Window Size was 1024x768 on an Intel Core i7 2.67GHz with 8 Cores and 12 GB RAM; 3 x NVIDIA GeForce GTX 285 running Windows 7 Enterprise.

Results

With over 50,000 Frames per Second (FPS) recordings over all trials, we compared the average under each condition for each of the 10 test cases through Student's t-Test (Table 2). A t-test shows that the differences between conditions is significant, since n= 10 , df=9, t-Ratio=-6.19079; p < .0001 .

Table 2: Basic statistics for the FPS performance result over all trials.

	Proxy Prism	Radial Distance
Standard Deviation	1.2883727	1.3186482
Average FPS	45.434102	43.751629

These results show that even in a regular grid where building blocks are rectangular, our proxy prism PROTO outperformed the native LOD overall in terms of FPS. If the proxyPrism shape can be computed upstream from the X3D client (i.e. on the server), then building models can be switched more efficiently. For X3D immersive clients, the PROTO method with an interpreted language will be respected, but certainly some performance gains may be made if engines support a native implementation of the node.

There are several exciting avenues of future research, including additional server-side optimization, alternative LOD schemes and leveraging the X3D Binary encoding. We will publish the full results of our X3D LOD extension test in an upcoming paper. Our future work includes a perceptual investigation into these and other techniques for real time LOD switching.

9 Results

9.1 Testing of service-based 3D portrayal approaches

3DPIE participants did successfully test and demonstrate how to set up 3D portrayal pipelines based on open formats and services for various client platforms and devices. Based on the experiments of the IE's three work items, we tested and demonstrated various aspects of service-based, 3D graphics-based and image-based 3D portrayal of complex 3D models. Especially, complex sets of detailed 3D city models (in the CityGML format; including geometry and textures) could be integrated into various W3DS/WVS servers and could be delivered to various clients, desktop clients, web-based clients as well as clients running on mobile devices.

3DPIE experiments did lead to several server instances hosting various 3D data sets and their access and usage by various clients. Especially, through 3DPIE we did establish a number of server-client connections that did not exist and have not been tested before 3DPIE (Figure 27).

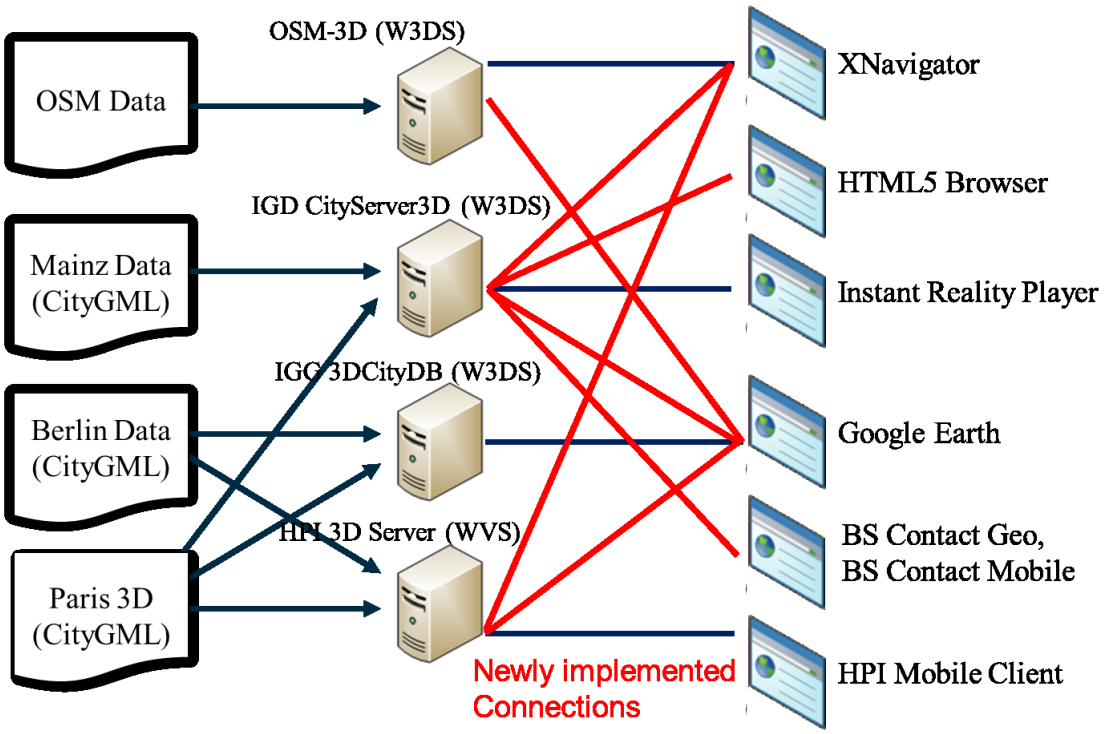

Figure 27: Server-client connections that were newly established and tested by 3DPIE experiments.

9.2 Extending the implementation basis for W3DS

Some of the software systems used for the experiments have been extended or have even been completely new developed. The following subsections name and describe these implementations.

9.2.1 New implementation of IGG W3DS

For this experiment, IGG designed and implemented a completely new W3DS. This W3DS provides cached KML/COLLADA data, which is exported from a 3DCityDB by means of the 3DCityDB importer/exporter tool and cached at the server-side. The architecture and specifics of the IGG W3DS and the workflow to prepare large data sets for this W3DS is described in detail in Section 8.1.

9.2.2 CityServer3D was adapted to latest W3DS specification

The CityServer3D's W3DS support was improved to conform to W3DS 0.4.0 Draft. This includes support for http GET-based GetScene, GetFeatureInfo and GetCapabilites. The 0.3.0-capability was retained.

9.2.3 Improvements of OSM-3D W3DS

In order to perform the Google integration tests, COLLADA and KML/KMZ exporters have been added. In case that another than the default format (VRML) is requested, the stored VRML content is parsed and converted into an internal scene graph representation. This scene graph is then serialized using the X3D or COLLADA exporter. The behavior of switching between COLLADA and KML export based on the bounding box size was also added to the layer configuration.

9.2.4 Extension of the XNavigator client to consume different W3DS

The capability of handling multiple W3DS instances required changes in the tile loading mechanism, implementation of the memory cache, and definition of the configuration file. The modifications also include special handling of CityServer3D requests, identified by the service URL. A preliminary GUI dialog was introduced to manage W3DS connections. All these changes were introduced in the main code of XNavigator as of version 1.4.0.

9.3 Increasing conformance of service implementations

9.3.1 W3DS conformance of CityServer3D

The CityServer3D supports all mandatory W3DS operations, except that SOAP was not considered mandatory as it is not in OGC Web Services Common (WSC) 2.0. The SOAP encoding, the GetLayerInfo and GetTile operations are not supported.

9.3.1.1 Deviations in recommended GetScene parameters

The W3DS GetScene operation defines the following request parameters where the CityServer3D implementation deviates in interpretation or ignores the input (Table 3). Deliberately ignored items indicate conceptual issues seen by Fraunhofer IGD.

Table 3: Implementation of the W3DS GetScene requests parameters in CityServer3D.

Names	Definition	Comment
minHeight MinHeight	Vertical lower limit for boundingBox selection criteria	Ignored (but checked for errors)
maxHeight MaxHeight	Vertical upper limit for boundingBox selection criteria	Ignored (but checked for errors)
spatialSelection SpatialSelection	Indicates method of selecting objects with BoundingBox	Ignored
format Format	Format encoding of the scene	Not 100% compatible to [OGC 06-121r9] clause 10.5 (no support for parameterized mimes)
layers Layers	List of layer to retrieve the data from	Interpreted as layer names, not identifiers.
styles Styles	List of server styles to be applied to the layers	Ignored
lods LODs	List of LODs requested for the layer	Deliberately ignored. In the view of Fraunhofer IGD, the LoD problem is not adequately addressed by these two parameters. In particular, the binding to the layers parameter is questionable.
lodSelection LODSelection	Indicates method for selecting LODs	
time Time	Date and time	Ignored
offset Offset	Offset vector which shall be applied to the scene, i.e. subtracted from the scene	Will be subtracted or used as X3D 3.2 GeoOrigin (depending on format and custom parameters (1)).
background Background	Identifier of the background to be used	Deliberately ignored. (Underspecified)
light Light	Add light source	Deliberately ignored (Underspecified)
viewpoints Viewpoints	Add Viewpoints to choose from	ignored

(1) The X3D 3.2 Geo extension cannot be used without an offset (GeoOrigin in X3D 3.2), so there is an ambiguity resolved by this parameter. X3D 3.3 can only use the Geo extension without offset, so again there is an ambiguity. This is not an issue when not targeting X3D browsers with and without geo extension available. However, we view a simple decision by the requested X3D profile as insufficient for this use case because only the "Full" (most advanced) profile encompasses the geo extension.

9.3.1.2 Additional GetScene parameters in the CityServer3D

Also, Fraunhofer's CityServer3D supports additional parameters for the W3DS GetScene operation (Table 4).

Table 4: Implementation of additional W3DS GetScene request parameters in CityServer3D.

Names	Definition	Comment
x3d.forceUseFaceSet	Boolean: Emit X3D FaceSets even if TringleSets could be emitted	This is used to steer the X3D profile, and could possibly be implemented as a parameterized mime type.
x3d.geoExtension	Boolean: Use the X3D Geo extension.	Used to create offset-subtracted X3D (without Geo extension).
x3d.origin	Like offset, but for X3D Geo extension only	Discontinued, used in 0.3.0 implementation
x3d.optimize	Boolean: true to optimize for faster display	False by default, preserving information

9.3.1.3 Other W3DS conformance aspects

The CityServer3D does various things slightly differently than was anticipated during the course of the project. These minor aspects are listed here for completeness:

- CityServer3D does not have an OGC service dispatcher. As a consequence, different operations of a service are reachable under different service prefixes. There is no technical requirement here, but it is an intuitive assumption for many.

- The CityServer3D W3DS is not as portrayal-oriented as other implementations. By default, it does NOT "optimize" (e.g. recombine them for performance) result geometries. Since such endeavors usually incur loss of information, and CityGML precisely exists to prevent loss of information, we suggest addressing the issue in an upcoming standard in more detail. A possibility could be adding an optional parameter to preserve information if possible, and including this as a capability.

- The layer parameter is not interpreted conformant by CityServer3D. However we think that it should be possible to use human-readable names in query parameters given they resolve unambiguously.

The W3DS specification is supposed to be based on OGC WSC 2.0. We have spotted some misalignments in comparison to the current WSC specification (06-121r9, not the draft reference), which are listed here:

- The GetScene operation is implemented solely as KVP (HTTP GET). This is, strictly speaking, disallowed by the current draft which mandates XML via HTTP POST. We view this specification as being in excess of the customs in WSC 2.0.

- WSC 10.2.3 Bounding box KVP encoding specifies a way to encode N-D bounding boxes. Therefore, the minHeight and maxHeight parameters can be dropped.

- WSC mandates specifications to name HTTP codes corresponding WSC errors/error codes. This is not done in W3DS.

9.3.2 Conformance tests for 3D portrayal services

In order to validate the current W3DS implementations a compliance test tool was developed using JMeter (http://jmeter.apache.org/) scripts. The main aim was to test the supported mandatory and optional requests and parameters and validate the result send by the server. In total, 57 test patterns have been implemented to test possible combinations of query parameters and results based on draft v0.4.0.

As a result of the conformance test, an overview graphics showing the implementation status of mandatory and optional request parameters is created. Detailed reports are provided to the organizations that are implementing the respective server only and will not be published. The main idea of the JMeter compliance test suite is to support the development of the W3DS.

9.4 Increasing conformance of data format implementations

9.4.1 Impact on IGN's CityGML implementation

The large 3D city model of Paris, provided by IGN, represented a major contribution to 3DPIE. This data set was integrated into various portrayal servers and was delivered to various 3D clients through the W3DS and WVS interfaces, i.e., several 3D portrayal pipelines could be established and tested. It was ensured that the original CityGML data and also derived graphics and image data was processed and delivered only by participants that signed a proper NDA. Also, IGN did receive valuable feedback regarding the proper modeling, export, and usage of this specific data set.

9.4.2 X3D conformance in CityServer3D

Numerous shortcomings of the CityServer3D's X3D implementation have been identified and subsequently addressed. They were mostly the result of lax (probably context-free) parsing by the implementation we most often talk to, InstantReality. Among many others, the few notable improvements were:

- Non-uniform terrain grids (with holes) are now exported as zero-height where no data is available. X3D, as a rendering format, has no need to express information irrelevant to rendering.
- A proper X3D header with profile information is now being generated.
- If not known, the X3D default to assume convex faces is now suppressed, correcting some rendering issues.
- X3D xml and binary encoding are supported now.

10 Discussions

10.1 Precision issues in interactive 3D display of geo data

Hardware accelerated graphics using OpenGL or DirectX have a major shortcoming. All vector and matrix calculations on the GPU are done in 32 bit single precision floating

point arithmetic. This is insufficient for representing geo coordinates directly. In case of a globe we have coordinates with a magnitude of approximately $6.4*10^6$ meters. The 23 bit mantissa of IEEE binary32 (float data type) gives us an accuracy of about 0.8 meters ($6.4*10^6 / (2^{23}-1)$). Although it might seem enough to represent simple building geometries, it also affects matrix computations of transformation groups and causes jitter, severely reducing rendering quality. This design was copied by many popular 3D formats, including Wavefront's obj format, Autodesk's 3ds format, and X3D, all specifying vertex data and vectors as floats.

On the other hand, GML uses geo-coordinates directly for describing 3D geometries. In order to visualize GML content correctly, optimizations must be applied, which reduce the magnitude of all coordinates and shift the model towards the origin. Otherwise the above described effects will be visible.

Both server side and interactive 3D portrayal of geospatial data need to consider spatial accuracy when choosing scene graph APIs and delivery formats.

10.2 Serving Large City Models

10.2.1 Suitable encodings for the delivery of large city models

Within the Web3D Consortium, developments led to the Geospatial Component extension for X3D, specifying a couple of node types for defining geographic content, for example GeoLocation, GeoOrigin, GeoCoordinate, and GeoViewpoint. All of these node types encode coordinates as double precision vectors along with a CRS identifier (supported are geodetic, UTM, and earth-fixed-geocentric). Especially the GeoLocation node is very useful for geo-referencing arbitrary models and importing them into 3D GIS. Unfortunately, the Geospatial component is ignored by all 3D editors, converters, and authoring tools so that it cannot be relied on during content creation and editing phases. But it can be used for import and export tasks and for deploying 3D servers. Unfortunately, Sprites are not supported by X3D, which makes is difficult to represent placemarks. The concept of a Sprite is to render a static icon directly on the screen, maintaining the icon's original pixel size without perspective deformations. This is important for ensuring the readability of placemarks.

In KML, coordinates are also specified as double precision vectors. They are mostly used for defining Placemarks. The height value is usually retrieved from the terrain model. A Placemark can be visualized as icon, symbol or text. It is also possible to link to an external COLLADA model by a URL or relative path within a KMZ archive. COLLADA is more powerful than KML since it supports transformation groups, materials, textures, and a wide range of visual effects. The combination of KML and COLLADA allows using standard 3D authoring tools and easy deployment of the models in 3D servers

10.2.2 Challenges of WVS-based 3D portrayal of large city models

A key challenge of the WVS approach is that two consecutive WVS requests could request totally different views on a 3D scene, e.g., by requesting different data layers or requesting disjunctive view areas (Figure 28). Also, service-based rendering requires a WVS server to synthesize and deliver a final, high-quality image for the actual request.

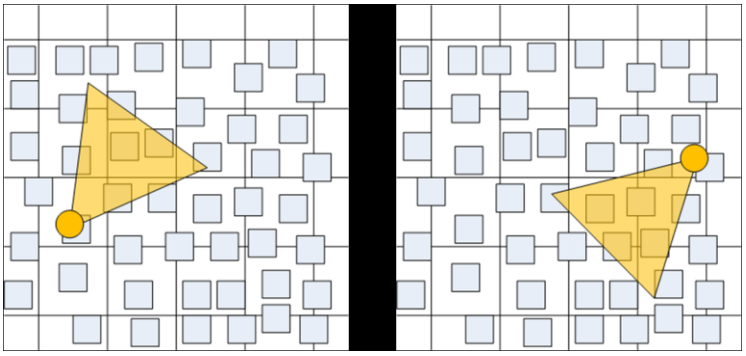

Figure 28: Scene objects covered by the view frustums of two different GetView requests. Two consecutive requests could require totally different data to be loaded to the graphics card and rendered.

As a consequence, a WVS server's rendering subsystem might have a) to load per request totally different data (and dismiss previously loaded data in the case of large city models) and despite of that b) has to render the required view quite immediately, i.e., "with the first frame" to deliver this to the service consumer. In this issue, service-based rendering is much more challenging than client-side rendering (such as implemented by W3DS clients or full desktop 3D applications), which usually take several frames to load, render, and display the complete data in highest quality.

To reduce rendering time and so the actual request/response round trip time, a server-side rendering system (such as a WVS server) needs to consider specific techniques for data management and rendering. These include, for example:

Specialized rendering techniques

- Out-of-core data structures and out-of-core rendering
- Massive texturing techniques (e.g., texture atlases)
- Rendering acceleration techniques, (e.g., impostor/billboard-based representations)

Parallel rendering technologies

- Exploiting several multiple render nodes
- Exploiting several GPUs per render node

Massive hardware to reduce I/O delay

- Using huge amount of RAM to store geometry and texture data
- Using fast secondary storage, e.g., solid state disks (SSD)
- Large GPU graphics memory to store large data sets

Also, server-side preprocessing of complete image sets (e.g., full panoramas, i.e., cube maps as described before) and caching these image sets would help to reduce the time to deliver imagery. Pre-computing and caching 3D panoramas at the server side would lead to a 3D equivalent of the Web Map Tiling Service (which is providing 2D map data). Currently, server-side caching is not considered by the WVS specification. Also, pre-computed 3D panoramas are valid only for a single 3D position of the virtual camera; if required transition panoramas would have to be requested from the portrayal service anyway or would have to be derived at the client side.

Additionally, a specific WVS server could introduce session handling as a higher-level approach to manage rendering resources and to ensure, e.g., that data that is required for a specific user (e.g., for exploring a specific part of the 3D scene) remains in memory.

10.2.3 Managing texture data in large urban data sets

Large urban 3D data sets can include massive texture data; e.g., for all façade surfaces textures could be extracted from oblique imagery and stored along with the surface geometry. Several tools and techniques can help to handle this massive texture data; examples are texture atlases and texture compressions.

A texture atlas packs multiple separate textures into one texture, the texture atlas. The concept of texture atlases can occur in different stages of a 3D portrayal pipeline, e.g.:

- Modeling and storing texture data by the help of texture data.
- Internal representation of loaded texture in a 3D portrayal server for efficient texture management and handling.
- Texture atlases for packing and efficiently transferring texture data to a graphics card for rendering the final portrayal.

According to the tools and technologies used for modeling, processing, loading, managing, and rendering the original 3D geodata, the texture atlases in each of these stages could differ, e.g., in content, size, format, data compression. Thus, texture atlases need to be used carefully and with respect to actual 3D portrayal pipeline.

Texture atlases can be applied for merging the textures of a 3D city model, e.g., many small façade textures. Here, texture atlases mainly help to reduce the number of image files to be handled. For example, the appearance model of the CityGML format allows specifying texture coordinates for textured surfaces. Aligned to this, the Paris data provided by IGN are heavily using texture atlases and are combining all the terrain and façade textures into texture atlases; for each data tile one or more texture atlas are provided.

If texture atlases are used for modeling the 3D geodata, the tools used to import the data into the 3D portrayal services would have to support these combined textures. – For example, the 3DCityDB did not yet support texture atlases; thus, importing the raw Paris data into the database would lead to a plenty of copies of the texture atlas in the database, one for each surface referencing the texture atlas. Thus, a preprocessing step was required

that cut the texture atlases into several sub textures and adapted the CityGML appearance data accordingly to reference the extracted textures.

Image size and image compression are an additional important factor for handling large texture data. According to the type of thematic data and the type of application (e.g., overviews vs. pedestrian views) appropriate (lossless or lossy) data compression algorithms and formats can be applied to the raw image data. Facade photos, e.g., can often be compressed by JPEG compression without losing too much information.

When texture atlases need to be compressed, one should consider if the sub textures should be extracted from the atlas, compressed separately, and repacked into a new atlas. This is because some compression algorithms, e.g., lossy conversion into the JPEG format, could lead to artifacts at the adjacent borders of the packed sub textures, e.g., colors from one sub texture could bleed into the adjacent sub texture. At the end this would lead to artifacts in the finally rendered image of the 3D scene. Generally, image compression algorithms need to be selected carefully.

If a server shall provide data in different "data layers" (i.e., of different feature types), it would be beneficial not to pack textures of objects of these different data layers into one texture atlas. If they were combined in one atlas and if only one of the data layers would be requested by a client, the server would have to handle (and in the case of WVS load and render) texture data that is actually not required for this request.

Additionally, the portrayal servers might have specific schemes for structuring and managing the loaded texture data. Texture atlases play an important role here, too: To use textures for the rendering process, the required textures need to be transferred to the graphics card. Storing multiple smaller textures into one larger is often more efficient to transfer and use this texture. – So, a rendering technique could rely on a texture atlas that is quite different from the one in which the raw data's textures were modeled in.

10.3 Dealing with tiled data

10.3.1 W3DS tiling approach

The usual way to access map data is by defining a rectangular bounding box which is used as spatial selection filter. The bounding box can be of arbitrary size which allows very flexible client configurations. This capability is provided by the W3DS operation GetScene, which contains parameters for selecting layers, styles, time, CRS, LODs, and a rectangular, axis parallel bounding box.

In many cases it is useful to spatially partition all data in a layer into smaller chunks of data which can be prepared prior to be delivered to web clients. This applies mostly for terrain data or other data representing 2.5D surfaces. Like in image pyramids, surface data can be made available as a set of adjacent tiles aligned on a grid or raster. For this kind of data, the preferred way of accessing individual tiles is not by defining a bounding box as spatial selection filter, but by using row and column indices referring to the position within the grid.

The W3DS includes a GetTile operation which facilitates the access to tiled data. A TileSet definition provides information on the spatial alignment of the tiles in a hierarchical grid structure.

The general use case for the GetTile operation is that a smart client is dynamically assembling the displayed scene graph from multiple tiles that are downloaded from a W3DS server (Figure 29). Tiles may be available in multiple sizes and resolutions or accuracies, referring for instance to the triangle density of the data. Tile sizes are related by powers of two and share the same origin. This ensures that tiles of several levels seamlessly fit together in a multi-resolution scene, which is the preferred way to achieve perspective views on complex landscapes.

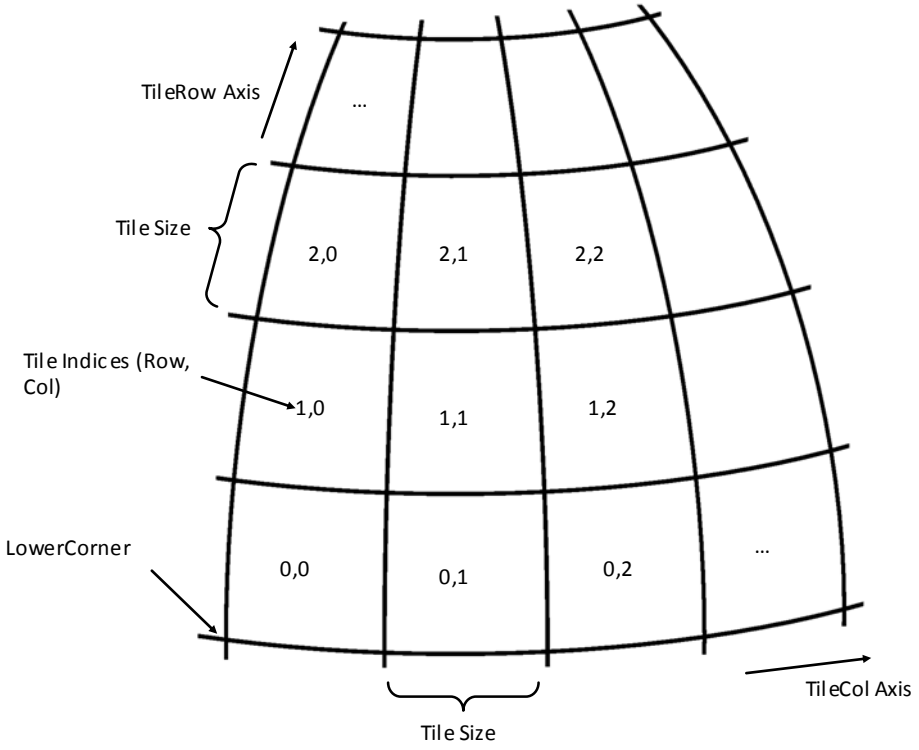

Figure 29: A tile level in the TileSet definition can be described as grid with origin at LowerCorner.

The concept of multiple tile levels is loosely coupled to the concept of Levels of Detail (LODs). However, tile levels refer to the strictly hierarchical organization of spatial subsets of the data of one layer. Dividing a rectangular tile into four quarters defines the next higher level. This higher level contains data of higher accuracy, or triangle count. The relation of the data amount (features, textures, triangles etc.) between tile levels should be approximately 1 to 4, to be consistent with already established tiling schemas and image pyramids.

The spatial partitioning of an existing layer must be described by a TileSet element so that the the GetTile operation can be used correctly. The TileSet element is contained in the Layer definition of the server's meta data.

An example of a planet wide TileSet in WGS84 with 11 levels may look like this:

```
<w3ds:TileSet>
    <ows:Identifier>dem_tileset</ows:Identifier>
    <w3ds:CRS>EPSG:4326</w3ds:CRS>
    <w3ds:TileSizes>180 90 45 22.5 11.25 5.625 2.8125 1.40625 0.703125
    0.3515625 0.17578125</w3ds:TileSizes>
    <w3ds:LowerCorner>-180.0 -90.0</w3ds:LowerCorner>
    <w3ds:NumBaseCols>2</w3ds:NumBaseCols>
    <w3ds:NumBaseRows>1</w3ds:NumBaseRows>
</w3ds:TileSet>
```

Each tile level can be described as a grid with origin at the LowerCorner coordinate, which is the lower left or south east corner of the extent covering all data in the layer (Figure 29). The LowerCorner is defined only once in a TileSet. Each tile level therefore shares the same origin. For example, tile level n is defined by a grid originating at LowerCorner and grid size of value n in the TileSizes list. Tile extents are between the grid lines.

This data structure is comparable to an image pyramid or quad tree. Each tile is divided into four quarters in the next higher level representing a higher degree of accuracy.

10.3.2 GetTileDefinition in CityServer3D

The GetTileDefinition operation in CityServer3D is a simple template-based mechanism. It is intended to serve tiling definitions (or parts thereof) to clients in a range of formats.

In 3DPIE, X3D GeoLOD and KML NetworkLink nodes were generated on the server so an X3D- or KML-compatible client can consume tiled data without actually knowing it communicates with a W3DS server.

The most significant difference to the draft-recommended GetTile is that the numerical tile processing and the translation to text representing the results is being handled on the server side. Thus, the approach is less dependent on client-side implementation details. This is important because there is virtually no way of achieving consistent tiling results across clients, which then hinders cache hit-rates and potentially impairs reliability (depending on the selection strategy, small gaps may amount to missing buildings).

The underlying problems are mostly that IEEE 754 does not fit nicely with decimal plain-text notation, and CPU-specific details in floating point handling. Since both will stay for some time, we suggest GetTile should not be specified to rely as much on the client as it does now.

10.3.3 Tiling for WVS

Providing tiled data allows a client to fetch large amount of data more easily and allows a server to cache the data that could be requested. Thus, one major benefit of tiling mechanisms is to reduce response time. However, for image-based 3D portrayal no tiling scheme has been discussed and specified so far. Key questions to address in such effort would be, e.g., how to discretize the 3D space and how to address these tiles.

10.4 Dealing with height references

The exact elevation of buildings and other structures may not be captured very accurately due to the following reasons:

- The model was created based on photographs.
- The model was created based on ground plans and rule based reconstruction.
- No accurate elevation data was available.

Also, the quality of the terrain data used for the visualization varies significantly. Data available from municipalities is mostly captured using land surveying techniques or LIDAR and has a very good quality. Many research and commercial projects use globally available data sets such as SRTM, ASTER or other open products with very low accuracy.

When trying to merge data from different servers within a Spatial Data Infrastructure, for instance buildings from one source and terrain from another source, vertical inconsistencies will most likely occur. Usually it is preferred to use the terrain as reference and adjust other ground objects relative to it, even if the quality of the terrain is worse. Basically two strategies for reducing vertical inconsistencies can be distinguished:

- Ground objects are delivered in a format that supports a reference point per object and an attribute indicating that the elevation is always relative to the ground. This can be encoded in KML/COLLADA combination using Placemarks. However, this leaves the logic of how to attach the object to the ground to the client. This can be done by continuous picking or by physical gravity models. The computational cost is proportional to the number of objects.
- Ground objects are corrected by the service that delivers the terrain data. A separate operation can be added to the 3D portrayal service that computes elevation values for arbitrary reference points by using interpolation. The functionality is identical to that of the Elevation service described in section 6.1.5. The terrain data must be same as used for the visualization. Since the elevation data is only retrieved from a service for a set of reference points once, it does not need to be loaded by the client in advance.

10.5 Potential changes in W3DS and WVS interface definitions

10.5.1 W3DS GetTile operation

Fraunhofer IGD suggests making the W3DS GetTile less client-dependent, i.e., to release a W3DS client from computing tile boundaries (Section 10.3.2).

10.5.2 W3DS custom extensions

Fraunhofer IGD proposes to specify a reserved (vendor) prefix for custom extensions. This could take the form of specifying that parameters beginning with "x-" or "x-<vendor>-" are never specified as a part of the standard (e.g. the IETF uses such prefixes). Although this should be WSC business, we didn't identify a comparable mechanism there.

10.5.3 Rethinking the concept of data layers

As in the WMS, the LAYER parameter of the W3DS GetScene request and the WVS GetView request specifies which data sets a server shall consider for processing the operation responses. However, while the concept of layers and drawing them one upon the other fits well for the WMS, this does not really fit for the 3D case. In the case of W3DS and WVS we are even more dealing with graphics representation of feature sets. This could be reflected better, by renaming the LAYER parameter, and making this conceptual difference clear in the service specifications.

10.5.4 Styling data layers

Closely linked to the meaning of "layers" (or rather feature representations) is the issue of styling selected 3D data sets. For example, thematic maps (i.e., raster data to use, e.g., as textures) could be modeled in two different ways: First, a server could advertise thematic maps as separate "data layers". Second, a server could provide such textures as a specific style, e.g., of a terrain "layer". While the first way is closer to drawing "layers" on top of each other, the latter is closer to the idea of selecting feature data to portray and also closer to how, e.g., CityGML is modeling the appearance of feature data.

10.5.5 Semantics of data layers

To allow W3DS or WVS clients to request a meaningful representation (3D graphics data or image) of the 3D environment provided by the servers, it would be required to provide some kind of semantics along with the metadata describing the data sets offered by the WVS servers. So a W3DS or WVS client could, e.g., ensure to always include a terrain model in the list of requested data set. Allowing a client to distinguish and request offered data sets according to their semantics (and to their importance for a scene representation) could help a client user to assemble a specific scene representation and so could help to reduce the size of transferred data.

11 Future Work / Next steps

Only selected issues of service-based, standards-based 3D portrayal have been tackled by 3DPIE. Some issues that could be relevant to look at, e.g., in future phases of 3DPIE, are described in the following.

11.1 Navigation in the 3D scene

Navigation represents the core functionality of a 3D client to allow users to explore and experience provided 3D data. Still today 3D clients require users to control the virtual camera mainly directly, e.g., to move it or rotate it by mouse, keyboard, or touch-input. To provide more efficient (e.g., automated and assisting) navigation techniques, additional knowledge about the 3D scene and the portrayed features is required at the client-side. Thus a general question is how to provide W3DS/WVS clients with appropriate information about the scene and the portrayed features and how to integrate this knowledge in a service-based portrayal pipeline. Especially, e.g., thin WVS clients that do not fetch any geometric data from a WVS require additional server-side functionalities for efficient navigation support.

11.2 Feature data access

Geodata Portrayal, i.e., the display of 2D/3D geodata, provides the platform to allow a user to explore and – in the broadest sense – use the data that is portrayed. A key functionality is to access the underlying data of a portrayed 3D scene. A WMS, e.g., specifies a GetFeatureInfo operation that allows a client to data of a feature at a specific pixel position in a portrayal requested via GetMap request. – Also for the W3DS and WVS corresponding capabilities and operations are specified. However, in the course of 3DPIE it turned out, that also alternative ways to request feature data from a W3DS exist and are already implemented. Future work could include to test and evaluate these approaches for fetching feature information from 3D portrayal services.

11.3 Data analysis

Another key functionality of 3D clients is to analyze portrayed 3D scenes, e.g., to measure distances and paths in the scene. While for the case of W3DS-based portrayal, such capabilities could be implemented mainly at the client side, different approaches are required for the image-based 3D portrayal approach of the WVS. Especially for thin WVS clients that do not reconstruct any geometric information (e.g., retrieved from a depth image) additional service operations are required to support analysis functionalities. Future work could include testing the analysis capabilities of the current WVS specification.

11.4 WVS/W3DS standardization

A major next step is to foster the standardization of one or more 3D portrayal services according to OGC's standardization process.

Regarding future W3DS standardization, Fraunhofer's position is to find a small, agreeable core for standardization and spare advanced features for later revisions. This

core could consist of GetCapabilities and a basic set of the current GetScene parameters, defined as a profile or conformance class. The document should build upon the current WSC and align to it where possible. Advanced features could then be added to other profiles/conformance classes and be subject to later standardization rounds. Section 9.3.1.3 lists some aspects that need to be addressed in the course.

Also, W3DS and WVS share general concepts (e.g., spatial selection via bounding box (W3DS) or via view frustum (WVS)). Because of this, 3DPIE participants discussed the possibility to create and specify one general "3D portrayal service" that provides a common core as well as additional modules for W3DS and WVS.

Bibliography

[14] Goetz, M., Lauer, J., Auer, A. (2012): An Algorithm Based Methodology for the Creation of a Regularly Updated Global Online Map Derived From Volunteered Geographic Information , 4th Int. Conf. on Advanced Geographic Information Systems, Applications and Services. GEOProcessing 2012. Valencia, Spain.

[15] OSM. Proposed features/Building attributes. Available from: http://wiki.openstreetmap.org/wiki/Proposed_features/Building_attributes , accessed on 25/11/11

[16] 3D City Database, Weblink (accessed October 2011) http://www.3dcitydb.net/

[17] A. Altmaier, T. H. Kolbe: Applications and Solutions for Interoperable 3D Geo-Visualization. In: D. Fritsch (ed.), Proc. of the Photogrammetric Week 2003 in Stuttgart, Wichmann Publisher, 2003

[18] Kolbe, T. H.; König, G.; Nagel, C.; Stadler, A. (2009): 3D-Geo-Database for CityGML, Version 2.0.1, Documentation, April 24th. http://opportunity.bv.tu-berlin.de/software/attachments/606/3DCityDB-Documentation-v2_0.pdf

[19] Kolbe, Thomas H. (2009): Representing and Exchanging 3D City Models with CityGML. In: Lee, Jiyeong / Zlatanova, Sisi (Ed.): Proc. of the 3rd Int. Workshop on 3D Geo-Information, Seoul, Korea. Lecture Notes in Geoinformation & Cartography, Springer Verlag, 2009.

[20] Schilling, A.; Kolbe, T.H. (2010): Draft for Candidate OpenGIS® Web 3D Service Interface Standard, Version 0.4.0. http://portal.opengeospatial.org/files/?artifact_id=36390

[21] Wilson, T. (2008): OGC® KML, OGC® Standard Version 2.2.0. Open Geospatial Consortium, Doc. No. 07-147r2, April 14th. http://portal.opengeospatial.org/files/?artifact_id=27810

[22] Neubauer, S. and Zipf, A. (2007). Suggestions for Extending the OGC Styled Layer Descriptor (SLD) Specification into 3D - Towards Visualization Rules for 3D City Models. Urban Data Management Symposium (UDMS 2007).

[23] Walenciak, G., B. Stollberg, et al. (2009). Extending Spatial Data Infrastructures 3D by Geoprocessing Functionality. The Int. Conf. on Advanced Geographic Information Systems & Web Services. GEOWS 2009. Cancun, Mexico.

[24] Neis, P., A. Schilling, et al. (2007). Interoperables 3D Routing auf Basis von OpenLS - Ein 3D Emergency Route Service (3DERS) als Aggregation eines Emergency Route Service (ERS) und eines 3D Route Service (3DRS). AGIT 2007. Symposium für angewandte Geoinformatik. Salzburg, Austria.

