OGC Testbed-13

3D Tiles and I3S Interoperability and Performance ER

Table of Contents

Publication Date: 2018-03-05

Approval Date: 2018-03-02

Posted Date: 2018-02-01

Reference number of this document: OGC 17-046

Reference URL for this document: http://www.opengis.net/doc/PER/t13-NG002

Category: Public Engineering Report

Editor: Volker Coors

Title: OGC Testbed-13: 3D Tiles and I3S Interoperability and Performance ER

OGC Engineering Report

COPYRIGHT

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. However, the discussions in this document could very well lead to the definition of an OGC Standard.

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

This OGC Testbed 13 Engineering Report (ER) documents the overall architecture developed in the "Interoperability of 3D Tiles and I3S using a 3D Portrayal Service and performance study of 3D tiling algorithms" activity. The report also summarizes a proof-of-concept of the use of 3D Tiles and I3S as data delivery formats for the OGC 3D Portrayal Service interface standard. The report captures the results from the interoperability tests performed as part of the *3D Tiles and I3S* testbed work package. Specifically, this OGC Testbed activity focused on the following tasks:

- CityGML [http://www.opengeospatial.org/standards/citygml] files converted into Cesium 3D Tiles [https://github.com/AnalyticalGraphicsInc/3d-tiles.git] using Analytical Graphics (AGI's) 3D Tiling Pipeline, and Cesium as the rendering client;
- An OGC CDB [http://www.opengeospatial.org/standards/cdb] data store converted into 3D Tiles using Compusult's Streaming engine, Cesium and Ecere's GNOSIS as rendering client;
- CityGML data store GeoRocket, 3DPS with 3D Tiles as data delivery format, and Cesium as rendering client;
- CityGML converted into I3S, 3DPS with I3S [http://www.opengeospatial.org/standards/i3s] as data delivery format, and Cesium as rendering client;
- CityGML converted into I3S using ArcGIS and FME, 3DPS with I3S as data delivery format, and rendering in ArcGIS client;
- CityGML with application domain extension stored in GeoRocket, converted to 3D Tiles, and Cesium as the rendering client;
- 3D Tiles (generated by all streaming engines visualized) from Ecere's GNOSIS rendering client;
- CDB visualized directly from Ecere's GNOSIS rendering client; and
- I3S visualized from Ecere's GNOSIS rendering client.

1.1. Requirements

An overview of the tests is provided in chapter 4.

Some experiments included the use of commercial software, specifically:

- 3D Tiling Pipeline developed by AGI to convert CityGML into 3D Tiles (available to the testbed for a six-month term for demonstration and evaluation).
- GeoToolbox is a software product developed by Fraunhofer IGD consisting of a set of tools to handle, modify and transform Geodata. The tools are available as command-line tools but also include a Java API. One tool, which was used in the experiment, converts CityGML to the 3D-Tiles structure allowing the user to choose between several options. For example what kind of hierarchical data structure is used, if textures shall be included and how aggressive the textures are simplified inside the 3D-Tiles structure.
- virtualcityPUBLISHER is a commercial software product developed by virtualcitySYSTEMS GmbH for publishing CityGML data sets online using state of the art 3D browser technologies. The software package is usually installed on a hosted or corporate server and provides a comprehensive backend user interface for managing projects, users, data sources, and for

configuring 3D online maps. In this Testbed the embedded command line tool for extracting and processing CityGML data was used in order to generate visualization layers that can be streamed online.

- FME to convert CityGML into I3S

1.2. Key Findings and Prior-After Comparison

There was little prior work to test the interoperability of streaming capabilities relating to the OGC I3S Community Standard [http://www.opengeospatial.org/pressroom/pressreleases/2639] ("I3S") and the 3D Tiles specification proposed as a new work item for an OGC Community Standard [http://www.opengeospatial.org/pressroom/pressreleases/2466] ("3D Tiles"). The interoperability & performance analysis conducted in the testbed's *3DTiles and i3s: Interoperability & Performance* work package provided a demonstration of 3D data streaming capabilities supporting CDB and CityGML data streamed according to the I3S and 3D Tiles specifications. The work also validated the 3DPS 1.0 interface specification using 3D Tiles and I3S as the content delivery formats. As a proof of interoperability, rendering of I3S in Cesium client has been achieved. Based in the results of the experiments, some change requests for the 3DPS version 1.1 have been identified.

1.3. What does this ER mean for the Working Group and OGC in general

From the 3DPS SWG's perspective, the main interest is to validate the 3DPS 1.0 interface specification using 3D Tiles and I3S as the content delivery formats.

For OGC in general, this work examines the interoperability and performance characteristics of streaming capabilities relating to the 3D Tiles and I3S specifications. More specifically, it provided a prototype demonstration to test and validate the interoperability of the OGC 3D Portrayal Service standard using the 3D Tiles and I3S data delivery formats in an urban-centric scenario based on CityGML and CDB data stores.

In addition, open data test data sets in CityGML for testing streaming of 3D city models have been defined. It is recommended to use these data sets in future experiments as well to achieve comparable results.

1.4. Document contributor contact points

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

Table 1. Contacts

Name	Organization
Volker Coors	Fraunhofer IGD / HFT Stuttgart
Ralf Gutbell	Fraunhofer IGD
Athanasios Koukofikis	HFT Stuttgart
Zakiuddin Shehzan Mohammed	Analytical Graphics, Inc.
Dave O'Mahony	Compusult

Name	Organization
Jérôme Jacovella-St-Louis	Ecere
Arne Schilling	virtualcitySYSTEMS GmbH
Keith Ryden	Esri Software Development

1.5. Future Work

The version of GeoRocket that was employed here used the 3D Tiles spatial hierarchy in order to quickly rearrange the data and therefore deliver only the requested subset of the original hierarchy. On the higher levels of the 3D Tiles hierarchy, the delivered elements intersected the queried spatial bounds, so that more data/buildings were visible than was queried, and the 3D Tiles styling feature was used to avoid rendering those intersecting objects. But in the future, it is expected that the returned geometries will be modified to only deliver the requested content. Related to this issue are the extension of the region query in 3DPS towards a polygon with holes. Such region query would enable mixed 3D scenes from different data sources such as a 3D city model with some parts of the model replaced by BIM models served by a BIM server. This is a very interesting use case in urban planning to show future urban developments.

Currently, the tiling strategies are part of external tools developed by different participants. The data delivery formats act as general container of the tiles. A general flexible for the generation of tiles would be very helpful as this is one core elements of streaming 3D and maybe 4D data in future.

The CDB standard provides guidance on storing many other datasets beyond those used here. It is expected that future work will develop representations of the other datasets in suitable 3D Tiles formats. For example, CDB geotypical models could be represented in 3D Tiles "instanced 3D model" (I3DM) format. Also underground structures such as utility networks are of high interest.

Another important topic for future work is a mixed rendering of 3D scenes and image based rendering. With image based rendering, not only bandwidth can be saved, but it has an advantage on data security as well, as the vector data stays on the server side. The 3D Portrayal Service supports image based rendering as well with the conformance class view. However, a 3DPS that supports both scene graph delivery and image based rendering has not been implemented yet.

In general, the link between 3DPS and WFS should be explored in more detail to access attribute data from a 3D scene. Also the integration of sensor measurements and simulation results into a 3D scene is very relevant for the future cities.

1.6. Foreword

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.

Chapter 2. References

The following normative documents are referenced in this document.

NOTE: Only normative standards are referenced here, e.g. OGC, ISO or other SDO standards. All other references are listed in the bibliography. Example:

- OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC 15-001r4, OGC® 3D Portrayal Service 1.0 Standard [http://docs.opengeospatial.org/is/15-001r4/15-001r4.html]
- OGC 17-014r5, OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification [http://docs.opengeospatial.org/cs/17-014r5/17-014r5.html]
- OGC 15-113, OGC® Volume 1 OGC CDB Core Standard Model and Physical Data Store Structure [https://portal.opengeospatial.org/files/?artifact_id=72712]
- OGC 12-019, OGC® City Geography Markup Language (CityGML) Encoding Standard [https://portal.opengeospatial.org/files/?artifact_id=47842]
- World Geodetic System 1984 (WGS 84) [http://earth-info.nga.mil/GandG/wgs84/index.html]
- glTF (GL Transmission Format) - A Khronos Group open standard for runtime 3D asset delivery [https://www.khronos.org/gltf]
- ISO/IEC IS 19775 standard X3D [http://www.web3d.org/standards]

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

3.1. Portrayal

presentation of information to humans. NOTE: This term is defined by [ISO 19117].

3.2. Scene, 3D scene

geometry and texture data that is to be portrayed. The term "3D scene" refers to a digital representation of geographic data that is mainly composed from 3D graphics data (mainly geometry and texture data), which is also often referred to as 3D display elements.

3.3. Scene Graph

a scene graph is a collection of nodes in a tree like data structure. Each node can have many children and has only one parent. A node refers to a set of 3D display elements.

3.4. Root node

The root node of a scene graph is the node without a parent node.

3.5. Tile

A tile is a collection of 3D display elements of a spatial bounding volume along with it’s metadata.

3.6. Tileset

A Tileset is a Scene Graph consisting of tiles.

3.7. Abbreviated terms

NOTE: The abbreviated terms clause gives a list of the abbreviated terms and the symbols necessary for understanding this document. All symbols should be listed in alphabetical order.Some more frequently used abbreviated terms are provided below as examples.

- 3DPS 3D Portrayal Service
- ADE Application Domain Extension
- AGI Analytical Graphics Inc.
- B3DM 3D Tiles Batched 3D Model
- CMPT 3D Tiles Composite Tile

- CPU Central processing unit
- GPU Graphics processing unit
- I3DM 3D Tiles Instanced 3D Model
- I3S 3D Scene Layer
- HDFS Hadoop Distributed File System
- HLOD hierarchical Level of Detail
- LOD Level of Detail
- NYC DoITT New York City Department of Information Technology & Telecommunications
- PNTS 3D Tiles Points
- S3TC S3 Texture Compression
- SLPK Scene Layer Package

Chapter 4. Overview

The goal of the Testbed 13 activity documented in this report was to test and validate the interoperability of the OGC 3D Portrayal Service standard [http://www.opengeospatial.org/standards/3dp] using the 3D Tiles [https://cesium.com/blog/2016/09/06/3d-tiles-and-the-ogc/] and I3S [http://www.opengeospatial.org/standards/i3s] data delivery formats. The focus was on using 3D Portrayal Service implementation instances to generate web-based visualizations with a workflow that used CityGML and CDB data sets as data sources and 3D Tiles and I3S as data delivery formats. The goal was to provide a prototype demonstration of 3D Tiles and I3S visualizations using an urban centric scenario based on CityGML and CDB data.

4.1. The Visualization pipeline

The visualization pipeline provides the key structure in any visualization system. In [REF_Moreland] Moreland gives a survey of visualization pipelines used in several applications. In this report, the concept of the visualization pipeline for the interactive portrayal of geospatial data was based on [REF_OGC_98-061]. It consists of three main components: Filter, Map, and Render. The Filtering step (sometimes called selecting) selects data from larger geospatial data set that should be displayed on the screen. The Mapping step maps features to display elements such as triangles and material definitions for an illumination model. This mapping can be done based on a style guide or based on individual appearance definitions per feature (see CityGML Appearance module as an example). During the Rendering step, the display elements are rendering into a screen buffer (or into an image) to be displayed on the screen.

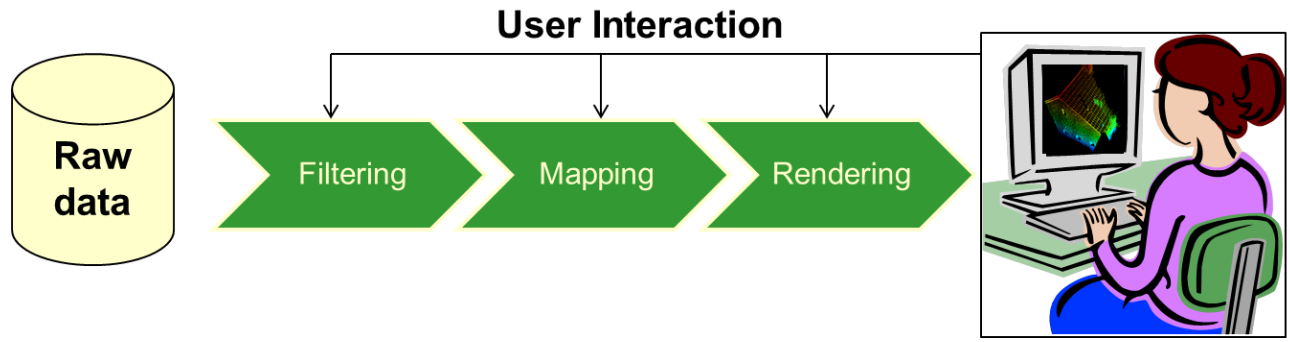

Figure 1. The Visualization Pipeline

User interaction is possible in each step, depending on the system architecture and the purpose of the system. A simple map visualization system may allow interaction in the rendering (zoom, pan, change position of the camera); a visual data analytics system needs to enable user interaction on Mapping and Filtering as well.

The experiments described by this report focused on the use of OGC standards in the Filtering and Mapping step to enable a distributed visualization pipeline. The Rendering is the core of Computer Graphics, libraries such as DirectX, OpenGL, and WebGL are used in this step. It has its own conceptual model, the so-called rendering pipeline. The rendering pipeline has changed tremendously during the last decade, from transformation and illumination models on CPUs to vertex and pixel shaders on GPUs.

The visualization pipeline can be distributed between client and server as follows [REF_OGC_09-104r1].

- Filtering on the server, Mapping and Rendering on the client: This approach is very common in 2D, where a Web Feature Service (WFS) is used to select data. The selected features are transferred to the client - be it a desktop client or a web browser. The Mapping is done on the client side based on a style and the selected features will be displayed on the screen in a map. In 3D, this approach works fine for small models, but does not scale for larger models. One reason is that the Rendering pipeline is fast if the display elements are ordered by material rather than by features.
- Filtering and Mapping on the server, Rendering on the client: The data is stored in a spatial database or file based on the server. The selected features will be mapped to display elements and will be stored in an intermediate file set (often called a scene graph) that is optimized for data streaming and visualization. It may use tiling strategies and binary data formats to reduce the amount of data transferred to the client. The client takes the intermediate data to render it on the screen. This approach is supported by the conformance class Scene of the 3D Portrayal Service (3DPS). In this report, this distribution of the visualization pipeline was analyzed using CityGML and CDB as file-based data storage, and GeoRocket as a spatial database. The results of the Mapping step are stored in both I3S and 3D Tiles.
- Filtering, Mapping and Rendering on the server: The entire visualization pipeline is executed on the server. The Rendering step writes an image that is transferred to the client. This approach is usually called server side rendering. The 3DPS conformance class view supports this approach. However, it was out of the scope of this testbed.

In the work described in this report, the second case of the distribution of the visualization pipeline (filtering and mapping on the server, rendering on the client), was investigated in more detail. A special focus was placed on the use of CDB and CityGML for the geospatial data sources, the I3S and 3D Tiles specifications for data delivery from server to (web-based) client, and the OGC 3D Portrayal Service Standard as a query interface.

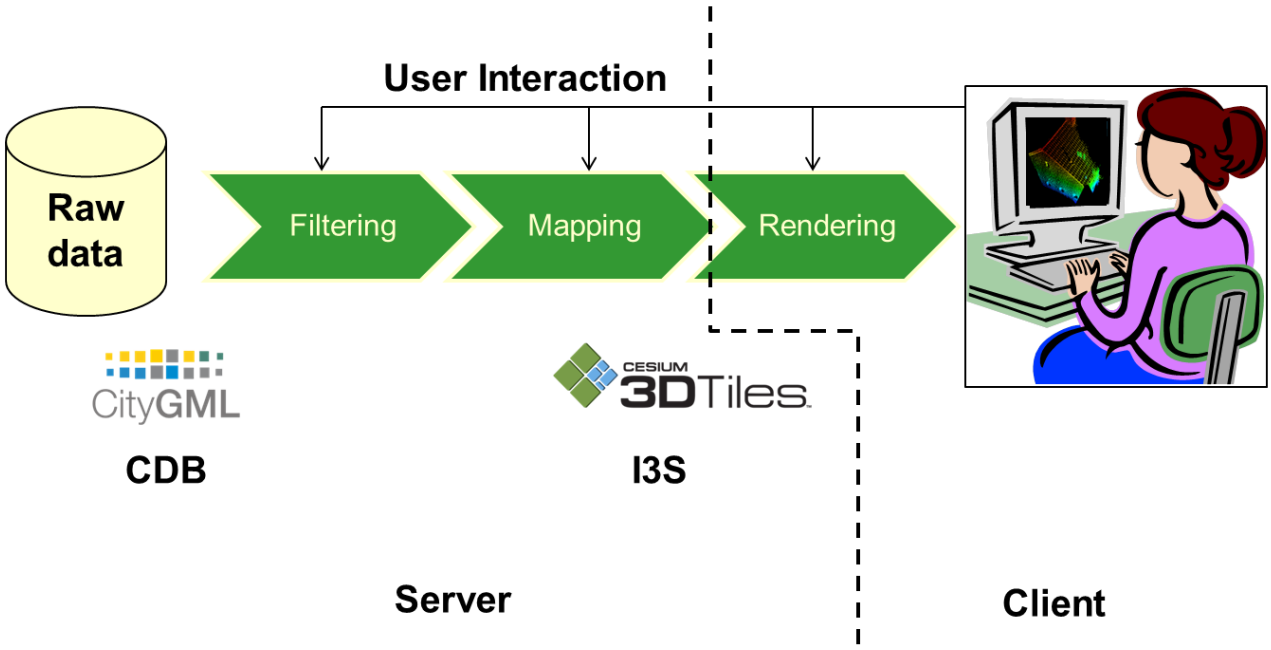

Figure 2. Distributed Visualization Pipeline and the role of OGC standards used in this experiment

NOTE

3D Portrayal Service

The 3D Portrayal Service [http://www.opengeospatial.org/standards/3dp] (3DPS) is an OGC service implementation standard targeting the delivery of 3D visualizations in an interoperable fashion. When client and service(s) involved share a common set of capabilities, it becomes possible to view and analyze 3D geo-information from diverse sources in a combined manner. Major use cases include navigation in the represented scene, retrieving feature information, and analyzing detailed information like simulation results or other 3D spatial information provided using the service instances.

The conformance class ***scene*** of the 3DPS standard supports client side rendering and content delivery using a scene graph. A specific data delivery format is not defined by the standard, but is negotiated by the client and server using the 3DPS ***getCapability*** operator and defined mime types. For images, the data format itself was not examined; instead, suitable formats (such as jpeg, png, etc.) were selected. The same goal was pursued in 3D. I3S and 3D Tiles were suitable data delivery formats for the 3DPS conformance class scene.

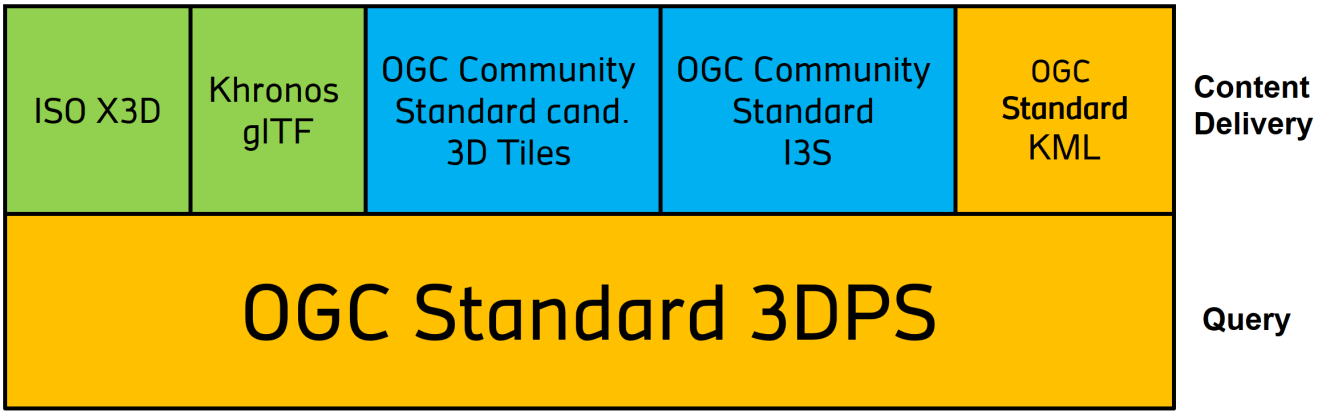

Figure 3. 3D Portrayal Service and scene graph content delivery. In this report, I3S and 3D Tiles will be used as content delivery

A first example from an experiment done in advance of the Testbed 13 [REF_Koukofikis] illustrates the importance of a good mapping strategy. In the experiment, a CityGML data set was mapped to a X3D scene graph using FME. The X3D scene was displayed in a web browser using the JavaScript library X3DOM as a 3D viewer. Two different mapping strategies were compared:

- order by feature (X3D-BF): mapping each building in the CityGML model to a node in the X3D scene graph
- order by material (X3D-BM): mapping all geometry with the same material to one single node in the X3D scene graph

Table 2. Table The test data set

data set	No of Entities	No of Surfaces	Type	file size CityGML [MB]	No of Triangles in X3D	file size X3D-BF [MB]	file size X3D-BM [MB]
s1	1000	12413	Building	16	33930	4	3.5
s2	10000	114939	Building	146	280676	34	30

The average frames per second (FPS) was measured in a web application using CPU Intel Core i5-3210M Processor, Operating System Windows 8.1 Professional (64bit), GPU NVIDIA GeForce GT 620M, RAM 8 GB, Web X3D viewer X3DOM 1.7.1, Web Browser Chrome 48 (64bit), Web Server Apache 2.4.17.

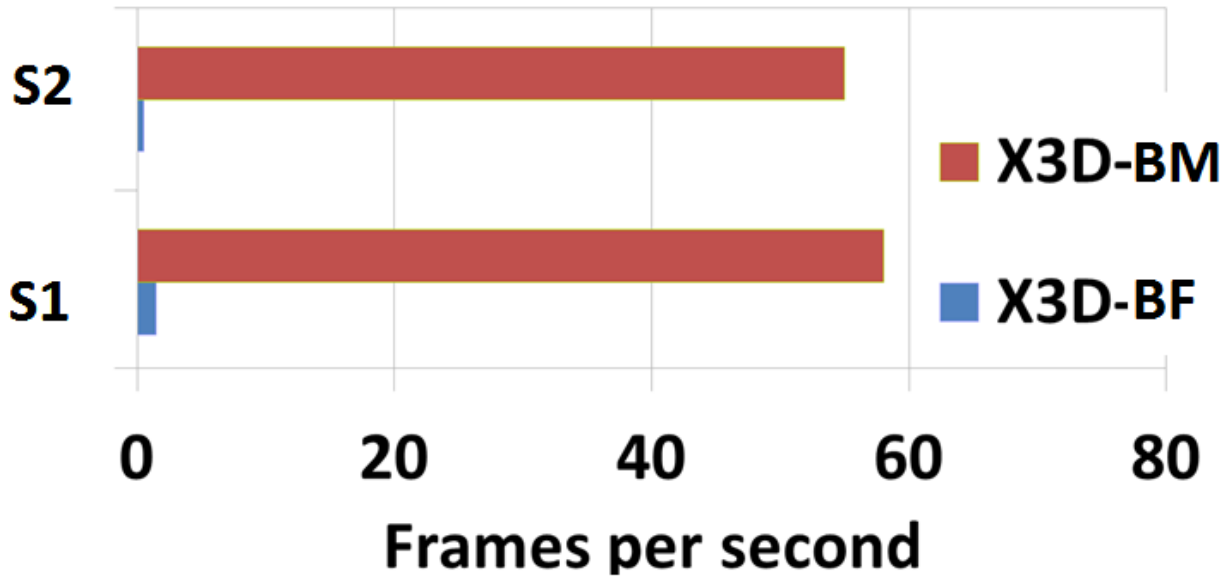

Figure 4. Rendering the same geometry ordered by feature and by material in X3D

This example clearly showed the impact on the mapping on the rendering performance on the client. In addition, the total time from the user query to the first image on the screen has to be taken into account for a distributed rendering pipeline. For larger scenes, tiling had a significant impact on load time.

4.2. Summary of Experiments

To implement the visualization pipeline, several experiments using CityGML and CDB data stores were conducted. AGI created the necessary processing algorithms to convert CityGML into 3D Tiles within its 3D Tiles Processing Tools; Esri created the necessary processing algorithms to convert CityGML into I3S within ArcGIS and by using FME; Fraunhofer and virtualcitySYSTEMS created the necessary processing algorithms to convert CityGML with and without application domain extension into 3D Tiles within GeoRocket. Further, Esri and HFT Stuttgart investigated the ability to convert CityGML into I3S and display it in Cesium to prove interoperability of I3S and Cesium using the 3DPS.

The processing algorithms took into consideration high- versus low-geometrically complex features, textured features, and the geographic distribution of features. 3D clients performed query operations using the 3DPS to return appropriate tiles. Data was streamed according to the 3D Tiles and I3S. In addition, runtime visualization strategies were developed and profiled.

The table below summarizes the experiments conducted.

Experiments 2 and 4 investigated source data management using GeoRocket. For these experiments, a 3DPS instance was implemented on top of GeoRocket. Experiments 1 and 3

investigated source data processing using AGI's 3D Tiling Pipeline and Esri's ArcGIS.

A similar workflow was developed by Compusult for data maintained in a CDB-structured data store (experiment 5), and Ecere implemented both 3D Tiles and I3S visualization (Experiments 6 and 8) as well as implementing direct visualization of CDB in their client (Experiment 7).

Input data included textured terrain and textured 3D buildings, and were provided as CityGML and CDB encodings.

Table 3. Experiments

Experiment	Data storage	Data Delivery	Query	Client	done by
Experiment 1	CityGML	3D Tiles	URI	Cesium	Analytical Graphics Inc. (AGI)
Experiment 2	CityGML & GeoRocket database	(a) 3DTiles, (b) I3S	3DPS	Cesium	Fraunhofer IGD & AGI & Esri & HFT Stuttgart
Experiment 3	CityGML	I3S	(a) URI, (b) 3DPS	ArcGIS client	Esri & HFT Stuttgart
Experiment 4	CityGML & GeoRocket database	3DTiles	URI	Cesium	VCS & Fraunhofer IGD
Experiment 5	CDB	3DTiles	URI	Cesium	Compusult
Experiment 6	CDB	3DTiles	URI	GNOSIS	Ecere
Experiment 7	CDB	—	URI	GNOSIS	Ecere

Chapter 5. Experiments

To complete the above-defined experiments, the following data sets were used. Note that CityGML data can be made available within an instance of a GeoRocket server and queried online. In addition, the 3D Tiles and I3S data sets of the selected models that were converted in Experiments 1, 2, 3, and 4 are available (including the relevant metadata).

5.1. CityGML Source data

5.1.1. CityGML New York City DoITT Data set

The New York City Department of Information Technology & Telecommunications (DoITT) data set [http://www1.nyc.gov/site/doitt/initiatives/3d-building.page] contains 1.1 million buildings modeled as CityGML in Levels of Detail 1 and 2 (LOD1 and 2), though this data does not include textures.

According to the DoITT web site, “Using the Open Geospatial Consortium’s CityGML specification as the basis, the NYC 3-D Building Massing Model was developed to a hybrid specification combining elements from Level of Detail (LOD) 1 and 2. Highlights of the model include the differentiation of building components including roof, facades and ground plane.”

For purposes of the specific work package tests, the focus was on Midtown and Lower Manhattan (south of Central Park).

If more feature types are needed in the future, the New York City road network data that is part of a CityGML data set created by TU Munich [https://www.gis.bgu.tum.de/en/projects/new-york-city-3d/] could be used.

5.1.2. CityGML model of Berlin

A CityGML-based Berlin 3D City Model [http://www.businesslocationcenter.de/en/downloadportal] was also utilized since the DOITT buildings did not have textures.

According to the Berlin 3D City Model web site, "Around 500,000 buildings in an urban area of 890 km² have been photographed from the sky and the roofs surveyed with lasers to create the 3D City Model of Berlin. The 3D City Model of Berlin is neither a commercial model nor is it based on commercially available 3D models. The 3D City Model of Berlin has been developed by Land Berlin’s Senate Administration for Urban Development (Senatsverwaltung für Stadtentwicklung), Senate Administration for Economics, Energy and Public Enterprises (Senatsverwaltung für Wirtschaft, Energie und Betriebe) and Berlin Partner für Wirtschaft und Technologie GMBH."

Batch downloads were available, and spatial subsets of arbitrary size of the model could be downloaded as ZIP archives containing CityGML and image textures. Each building was available as a CityGML LOD2 model (no LOD1, no LOD3) with semantic surfaces: RoofSurface, WallSurface, GroundSurface. A limited number of attributes such as roofType were included.

5.2. Experiment 1 - CityGML to 3D Tiles to Cesium

The overall Experiment 1 goal was to convert CityGML data into 3D Tiles, which were then visualized using a client (Cesium) that could consume 3D Tiles output.

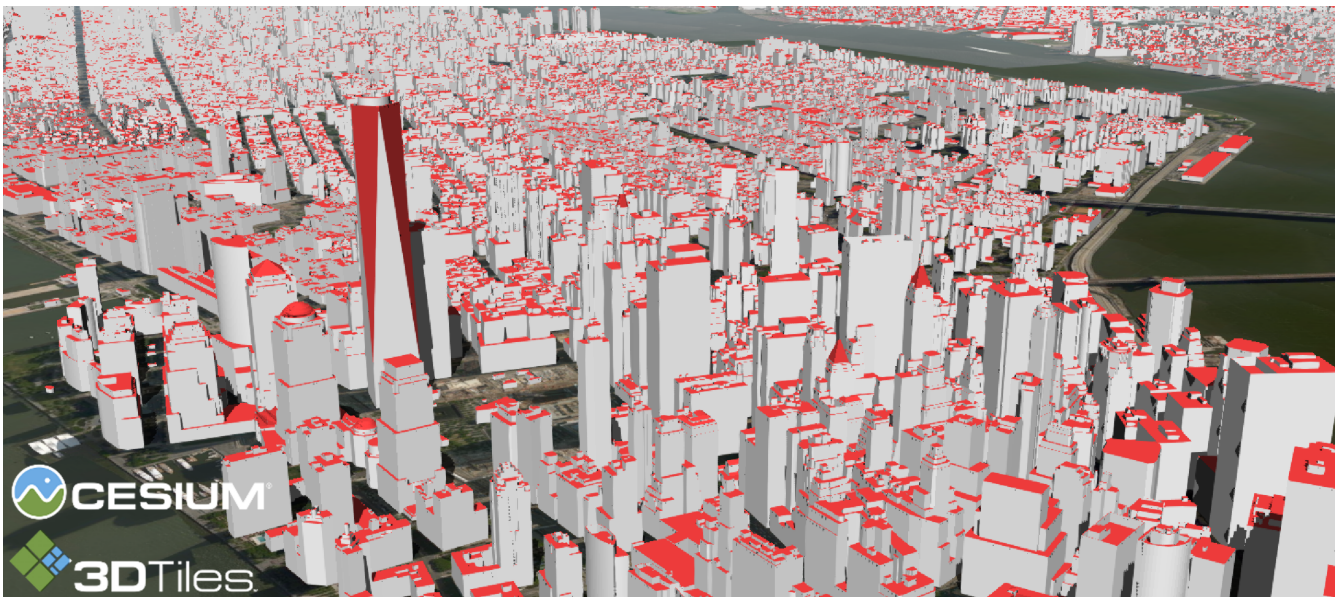

Figure 5. Cesium and 3D Tiles - 1.1 million CityGML buildings in New York City

5.2.1. Subcomponents used in Experiment 1

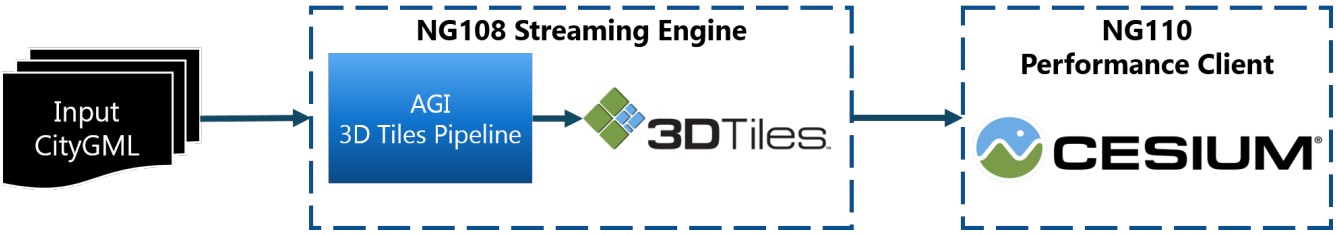

Figure 6. Experiment 1 - Convert CityGML into 3D Tiles using AGI's processing tool into Cesium

Cesium Client

Cesium [https://github.com/AnalyticalGraphicsInc/cesium.git] is an open-source JavaScript library for creating 3D globes and 2D maps in a web browser without a plugin. It uses WebGL for hardware-accelerated graphics, and is cross-platform, cross-browser, and tuned for dynamic-data visualization. Cesium provides complete support for the 3D Tiles specification as well as support for a broad range of open standards and formats.

Cesium is licensed under Apache 2.0 and is free for both non-commercial and commercial applications. It is downloaded more than 10,000 times every month and powers apps that reach millions of users.

3D Tiles Streaming

3D Tiles [https://github.com/AnalyticalGraphicsInc/3d-tiles.git] is a specification for streaming massive heterogeneous 3D geospatial datasets. In 3D Tiles, a tileset is a set of tiles organized in a spatial data structure, the tree. Each tile has a bounding volume completely enclosing its contents. The tree has spatial coherence; the content for child tiles are completely inside the parent's bounding volume.

To allow flexibility, the tree can be any spatial data structure with spatial coherence, including k-d trees, quadtrees, octrees, and grids.

The primary purpose of 3D Tiles is to improve streaming and rendering performance of heterogeneous 3D datasets. 3D Tiles uses a Hierarchical Level of Detail (HLOD) so only visible tiles are streamed - and only the most important tiles for a given 3D view are streamed. Tile payloads can be binary and context-aware compressed, e.g., using oct-encoding (see http://jcgt.org/published/0003/02/01/).

Instead of relying on 2D constructs such as zoom levels, 3D Tiles are based on geometric error for LOD selection and a tunable pixel error. This allows performance and visual-quality tuning and multiple "zoom levels" in the same 3D view.

In 3D Tiles, bounding volumes are 3D, not 2D cartographic extents. In 2D, the tiling scheme is often based on the Web Mercator projection. Web Mercator is not ideal for 3D because the poles project to infinity and because one of the testbed sponsors recommended against using this projection. A more detailed description of the issues surrounding Web Mercator can be found at http://earth-info.nga.mil/GandG/wgs84/web_mercator/index.html . In contrast, in 3D Tiles the tiling scheme is adaptable, in all three dimensions, depending on the models in the dataset and their distribution.

Figure 7. Representation of 3D Tiles as a sum of its components

A tile is defined as a connection of 3D display elements of a spatial bounding volume along with it's metadata. 3D Tiles support the following tile types (further details can be found at the links for each type):

- Batched 3D Models (b3dm) [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/TileFormats/Batched3DModel/README.md]: Used for 3D Buildings, textured terrain and surfaces, BIM and CAD models etc.
- Instanced 3D Models (i3dm) [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/TileFormats/Instanced3DModel/README.md]: Used for objects that are repeated such as trees, lamps, furniture etc.

- Point Cloud (pnts) [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/TileFormats/PointCloud/README.md]: Used for point cloud data.
- Vector Data (vctr) [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/TileFormats/VectorData/README.md]: Used for polygons, polylines and placemarks (in progress).
- Composite (cmpt) [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/TileFormats/Composite/README.md]: Combination of heterogeneous tile formats.

For converting CityGML to 3D Tiles, only Batched and Instanced 3D Models are required and Composite tiles are used to improve performance. Batched 3D Model are used for buildings, surfaces etc and Instanced 3D Model are used for trees, lamps, furniture etc.

The metadata for the models is stored per-tile in a Batch Table. A Batch Table [https://github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/TileFormats/BatchTable] contains per-feature application-specific metadata in a tile. These properties may be queried at runtime for declarative styling and application-specific use cases such as populating a UI or issuing a REST API request. Some example Batch Table properties are building heights, cartographic coordinates, and database primary keys.

A Feature Table [https://github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/TileFormats/FeatureTable] describes position and appearance properties for each feature in a tile. The Batch Table, on the other hand, contains per-feature application-specific metadata not necessarily used for rendering.

3D Tiles Styling [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/Styling/README.md] provides a concise declarative styling of tileset features. A style defines expressions to evaluate a feature's color (RGB and translucency) and show properties, often based on the feature's properties stored in the tile's batch table.

Styles are defined with JSON and expressions written in a small subset of JavaScript augmented for styling. Additionally, the styling language provides a set of built-in functions to support common math operations. For example:

```
{
    "show" : "${Area} > 0",
    "color" : {
        "conditions" : [
            ["${Height} < 60", "color('#13293D')"],
            ["${Height} < 120", "color('#1B98E0')"],
            ["true", "color('#E8F1F2', 0.5)"]
        ]
    }
}
```

Example: Creating a color ramp based on building height.

See the samples section below for styling applied to CityGML datasets.

A presentation on 3D Tiles in action at FOSS4G 2017 [https://cesium.com/presentations/files/3DTilesInAction.pdf] illustrates how the various capabilities and features of 3D Tiles can be used.

3D Tiles uses glTF, a Khronos Group open standard, as its rendering payload.

GL Transmission Format (glTF)

glTF [https://github.com/KhronosGroup/glTF] ™ (GL Transmission Format) is a royalty-free specification for the efficient transmission and loading of 3D scenes and models by applications developed by the Khronos Group [https://www.khronos.org/]. glTF minimizes both the size of 3D assets and the runtime processing needed to unpack and use those assets. glTF defines an extensible, common publishing format for 3D content tools and services that streamlines authoring workflows and enables interoperable use of content across the industry.

Please refer to the glTF 2.0 specification [https://github.com/KhronosGroup/glTF/blob/master/specification/2.0/README.md] for additional background information.

CESIUM_RTC Extension

Massive world graphics applications have position vertex attributes with precision requirements that result in jittering artifacts when naively rendered with 32-bit floating-point values. The `CESIUM_RTC` [https://github.com/KhronosGroup/glTF/tree/master/extensions/Vendor/CESIUM_RTC] glTF extension introduces the metadata required to implement the Relative To Center (RTC) high-precision rendering technique described by [Ohlarik08 [http://help.agi.com/AGIComponents/html/BlogPrecisionsPrecisions.htm]].

In this technique, each position is defined relative to an origin (the center) such that 32-bit floating-point precision is adequate to describe the distance between each position and the center. These relative positions are stored in the glTF vertex data. At runtime, the positions are transformed with a modified model-view matrix that makes the center relative to the eye. This avoids 32-bit subtraction of large translation components on the GPU.

5.2.2. Streaming Engine: CityGML To 3D Tiles

The 3D Tiling Pipeline developed for this requirement are proprietary and commercial software developed by Analytical Graphics Inc (AGI). AGI's data processing tools convert large city-scale geolocated datasets in data exchange formats, CityGML in this experiment, into 3D Tiles for streaming of data over the internet and rendering in 3D Tiles clients such as Cesium.

The software tools developed were made available for a limited period for demonstration and evaluation purposes, along with the Cesium client.

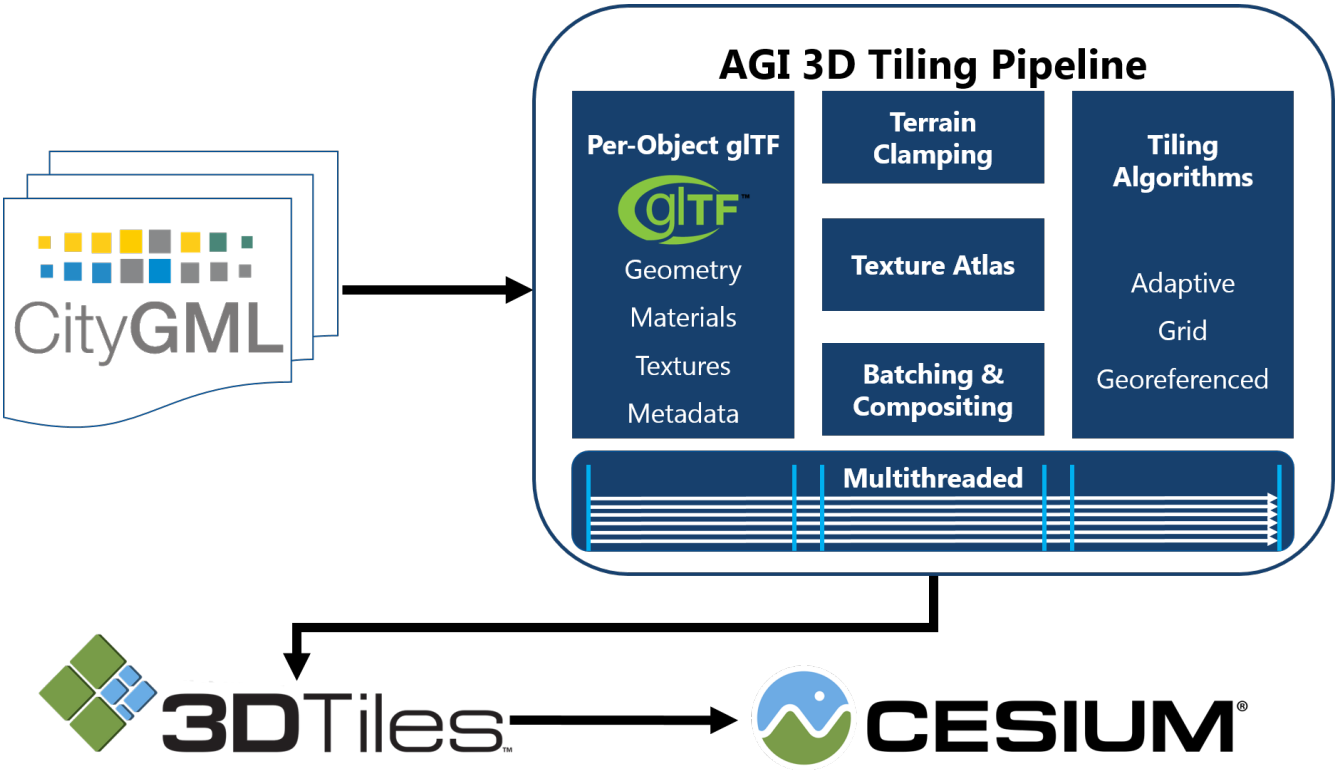

Figure 8. AGI 3D tiling pipeline

The 3D Tiling Pipeline is conceptually divided into 3 parts:

- CityGML to Per-Object glTF
- Tiling Algorithms
- Per-Object glTF to 3D Tiles

5.2.3. CityGML to Per-Object glTF

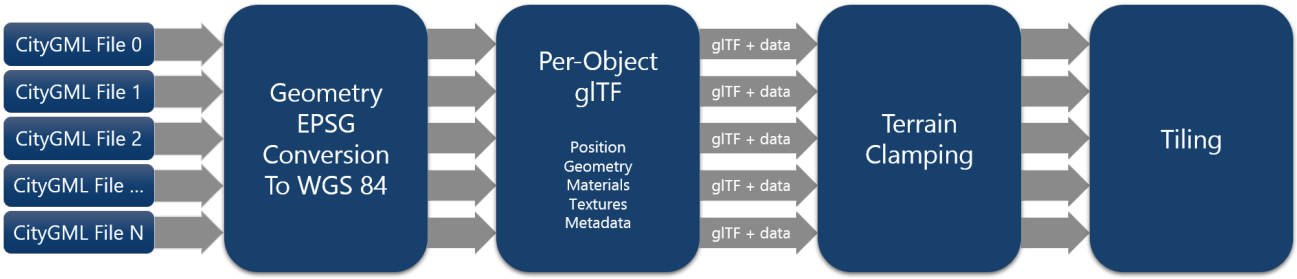

Figure 9. Conceptual diagram of CityGML to Per-Object glTF

The first step in converting CityGML to 3D Tiles was to convert CityGML objects such as buildings, trees, surfaces etc. into glTF and per-object metadata. This glTF and metadata combination was well-suited for conversion to 3D Tiles.

CityGML to glTF and EPSG Conversions

Each CityGML object's geometry, attributes, materials and textures are converted into individual glTF models and are stored along with their metadata and global positions. CityGML files commonly use a local EPSG Coordinate system, whereas, 3D Tiles relies on WGS84. During this initial conversion, the geometry was converted from local reference systems to WGS84. Floating-

point precision was maintained throughout the conversion and tiling process using CESIUM_RTC.

Handling CityGML Implicit Objects

CityGML objects can also be of implicit type, where a single geometry has many locations. Trees and lamps are examples of implicit models. For these type of models, a single glTF model was used for the geometry along with multiple reference information leveraging CityGML's structure for implicit models which have a transformation matrix and a reference point. This approach reduced file size as well as the number of disk operations.

NOTE	In 3D Tiles, implicit objects are known as instanced models. These terms may be used interchangeably based on context.

Clamp to Terrain

One of the aspects of generating building data for cities is to place the buildings on digital terrain models. Most CityGML datasets either do not map to any terrain, or have pre-computed heights for a single terrain model.

The 3D Tiling Pipeline it is possible to place buildings on any terrain data. As each object is an independent model, the terrain elevation at the location of the object could be queried and applied as an offset to the models, thus presenting a realistic visualization of the CityGML data on any terrain.

The models are then further processed to add a skirt to enclose any gaps between the building and terrain due to elevation changes.

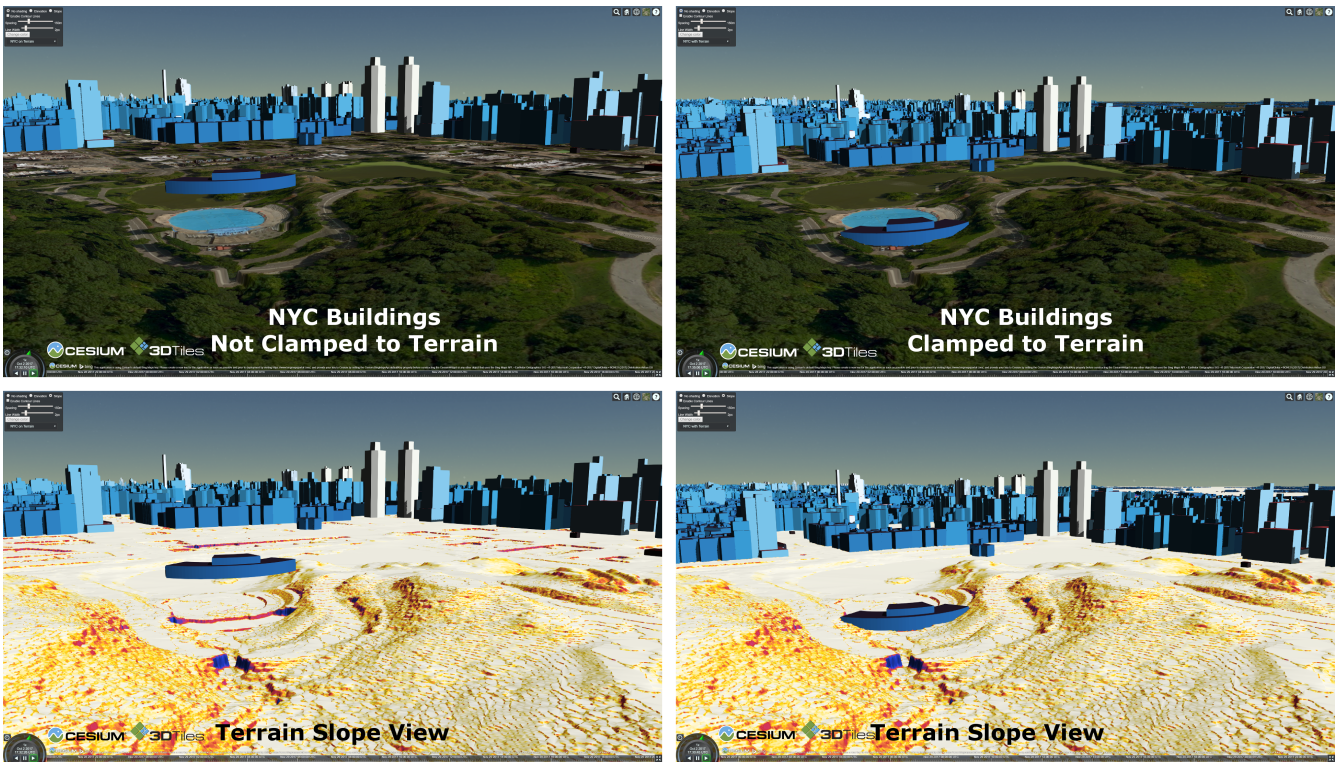

Figure 10. New York Buildings on Terrain - Left: Not Clamped. Right: Clamped to Terrain.

Querying Information

Another advantage of this process is that it enabled extracting subsets of CityGML data using spatial queries, names, metadata properties etc., for the tiling process.

Other Performance and Portability Considerations

This phase is multithreaded and has minimal disk IO to enhance performance. CityGML files can be loaded asynchronously as well, significantly improving performance. They are also platform independent and can be run on Windows and Linux as well as virtualization software.

5.2.4. Tiling Algorithms

There are various tiling algorithms implemented that can process the per-object glTF data including adaptive quadtree, uniform and non-uniform quadtree, and grid or tile map service layout.

The ultimate goal for a good tiling algorithm is to produce tilesets that have good batching and can run at 60 frames per second in the browser even on mid-to-low end GPUs.

Adaptive Quadtree

The Adaptive Quadtree tiling produces tilesets that take into account the number of buildings, the area, density and quality of the geometry of the models and so on to produce tilesets with optimized visual and streaming performance.

Adaptive Tiling tilesets have intermediate tiles with buildings. This is particularly significant as the tiling scheme makes these higher level tiles visible from further away, and as the camera gets closer, more tiles within the view are loaded. This gives the versatility of having multiple levels of detail in a single tileset. It also results in a good visual transition between dense tiles at lower levels and sparse tiles at the higher levels.

The Adaptive Quadtree algorithm also terminates subdivision early if it finds a tile to be optimized. The parameters for Adaptive Quadtree can be tweaked to generate the desired level for performance and visual quality.

Tilesets generated using Adaptive Quadtree can also be tweaked at runtime using the `maximum screen space error` setting. A lower Screen Space Error (SSE) results in more tiles being displayed while a higher SSE displays tiles that are closer with a suitable geometric error. Thus allowing various levels of performance and visual depth using the same tileset.

The Adaptive Quadtree algorithm is capable of generating tilesets that meet the goal of running at 60fps without any user defined parameters.

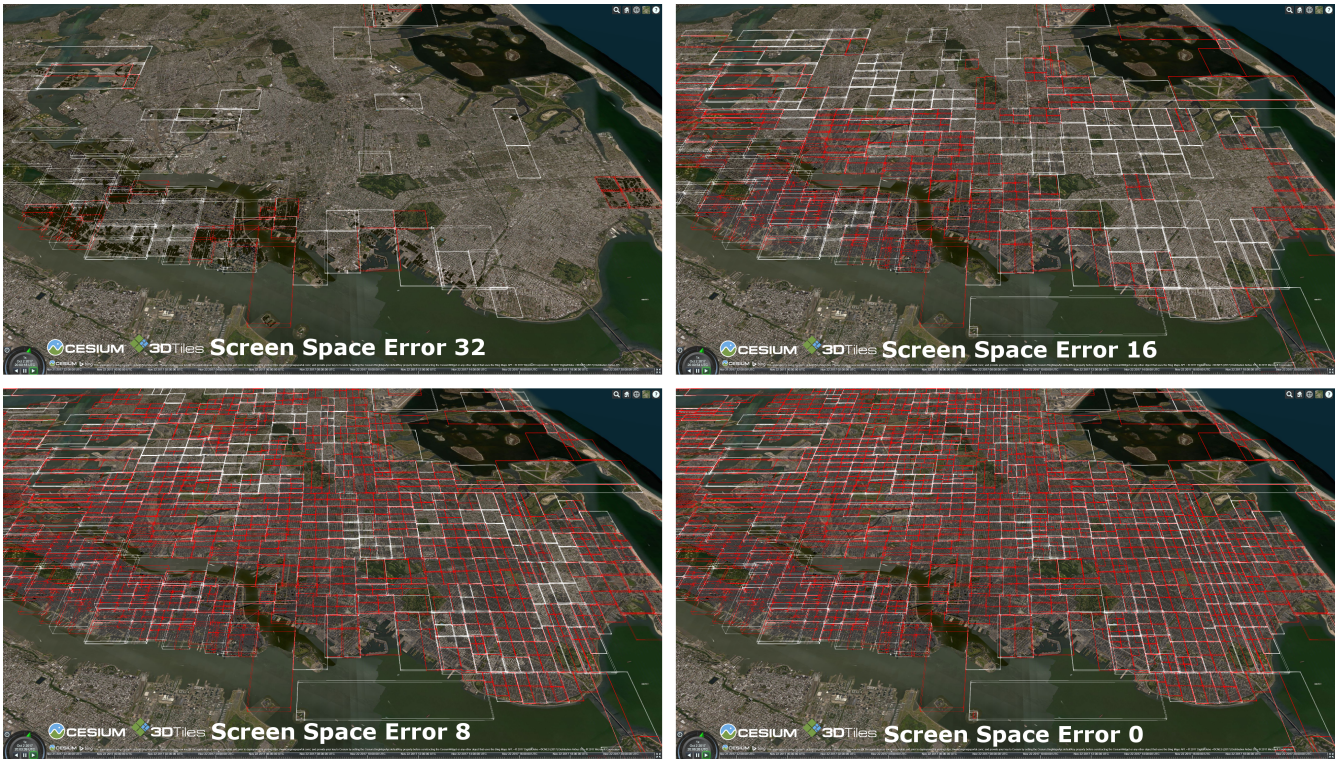

Figure 11. Bounding boxes for New York using Adaptive Quadtree Tiling. White boxes represent non-leaf tiles with content. Red boxes represent leaf tiles. Top Left: Scene at SSE 32. Top Right: Scene at SSE 16. Non-Bottom Left : Scene at SSE 8. Bottom Right : Scene at SSE 0.

Grid Quadtree

The Grid Quadtree algorithm can produce both uniform and non-uniform quadtree tilesets. Grid Quadtree tiling produce tilesets that meet the performance criteria at optimized levels. For example, the New York dataset runs at 60fps for tilesets with levels 4-6.

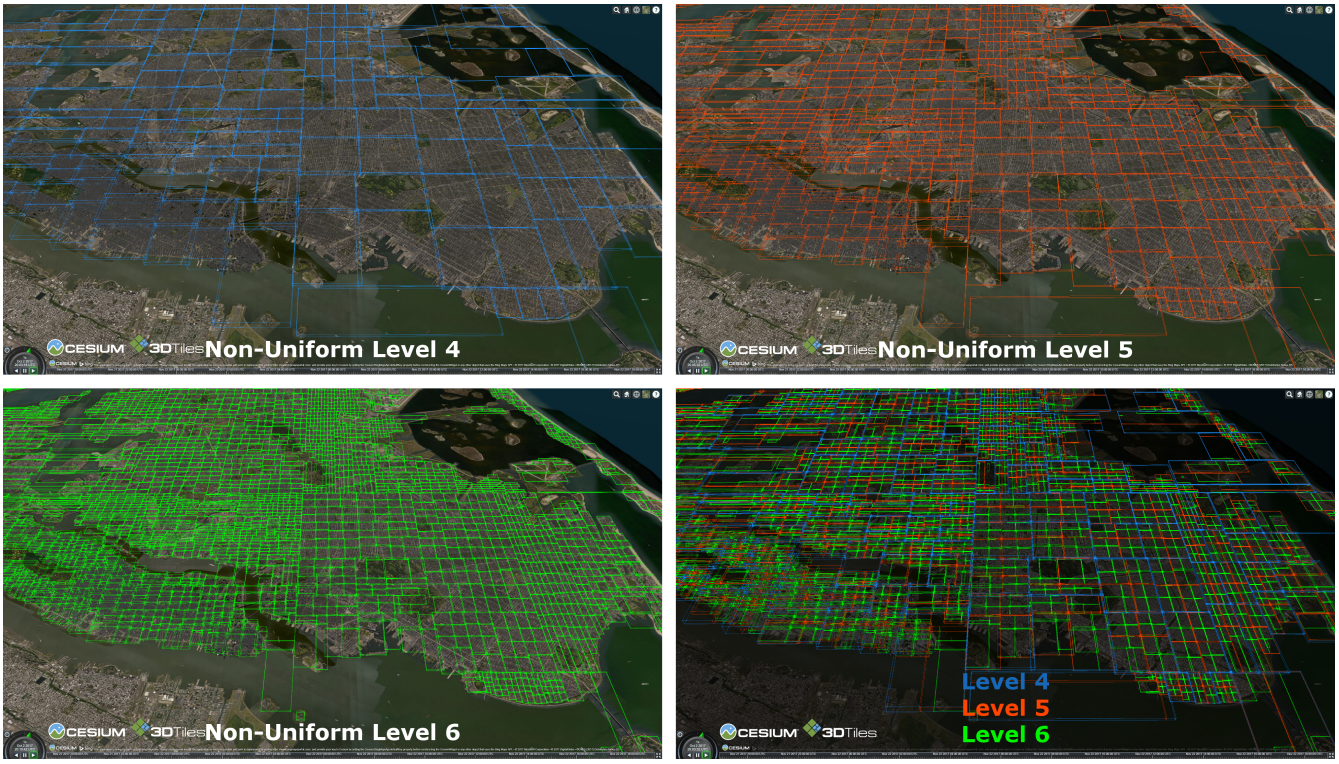

Figure 12. Bounding boxes for New York using Non-Uniform Grid Tiling. Top Left: Non-Uniform Level 4. Top Right: Non-Uniform Level 5. Non-Bottom Left : Non-Uniform Level 6. Bottom Right : Composited view of Non-Uniform Levels 4, 5, 6 for comparison.

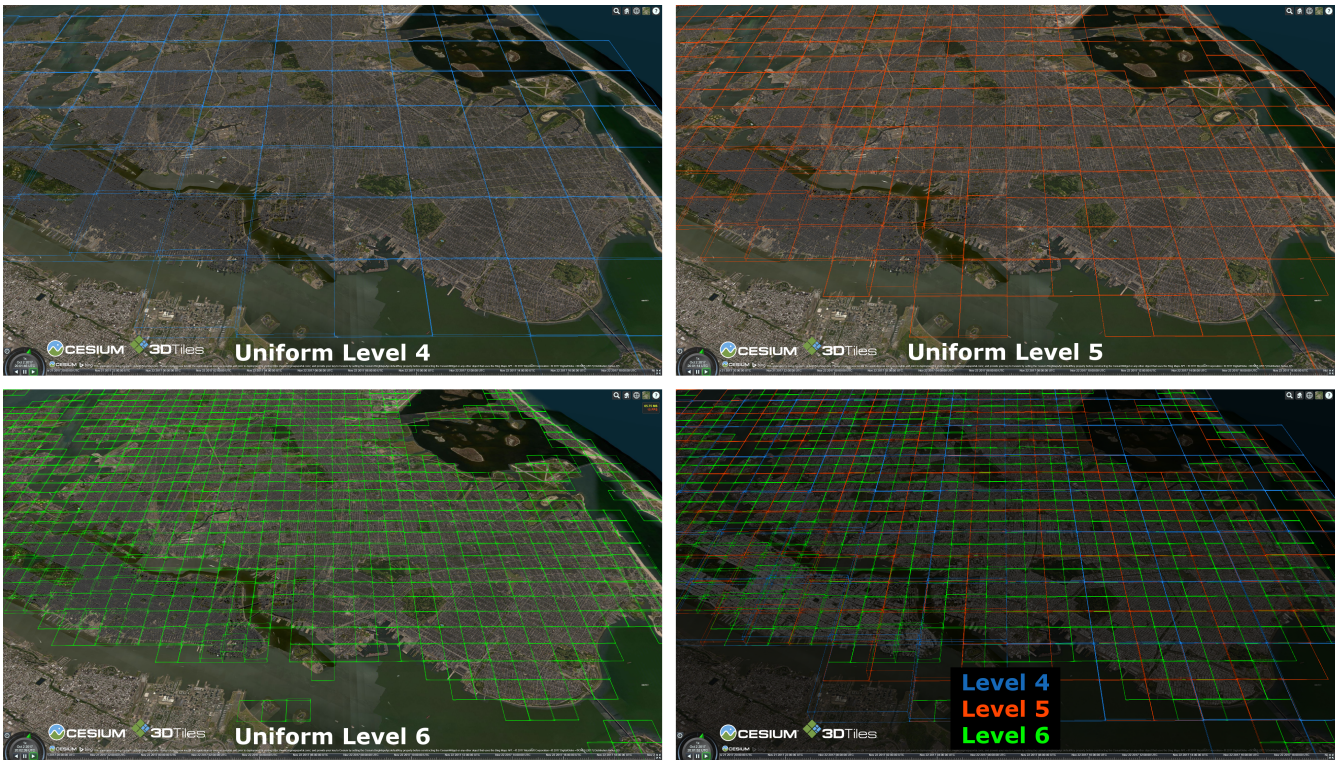

Figure 13. Bounding boxes for New York using Uniform Grid Tiling. Top Left: Uniform Level 4. Top Right: Uniform Level 5. Bottom Left : Uniform Level 6. Bottom Right : Composited view of Uniform Levels 4, 5, 6 for comparison.

Georeferenced Grid Quadtree

The Grid Quadtree tiling produce tilesets that have uniform quadtree subdivision based on the tile map service (TMS) layout. The tiling starts with the root tile extent being the same as level 0 in the

TMS layout. From there, the tile is subdivided into a uniform quadtree similar to TMS.

Most city-scale datasets can be converted into optimized tilesets between levels 12-16 depending on the extent of the data.

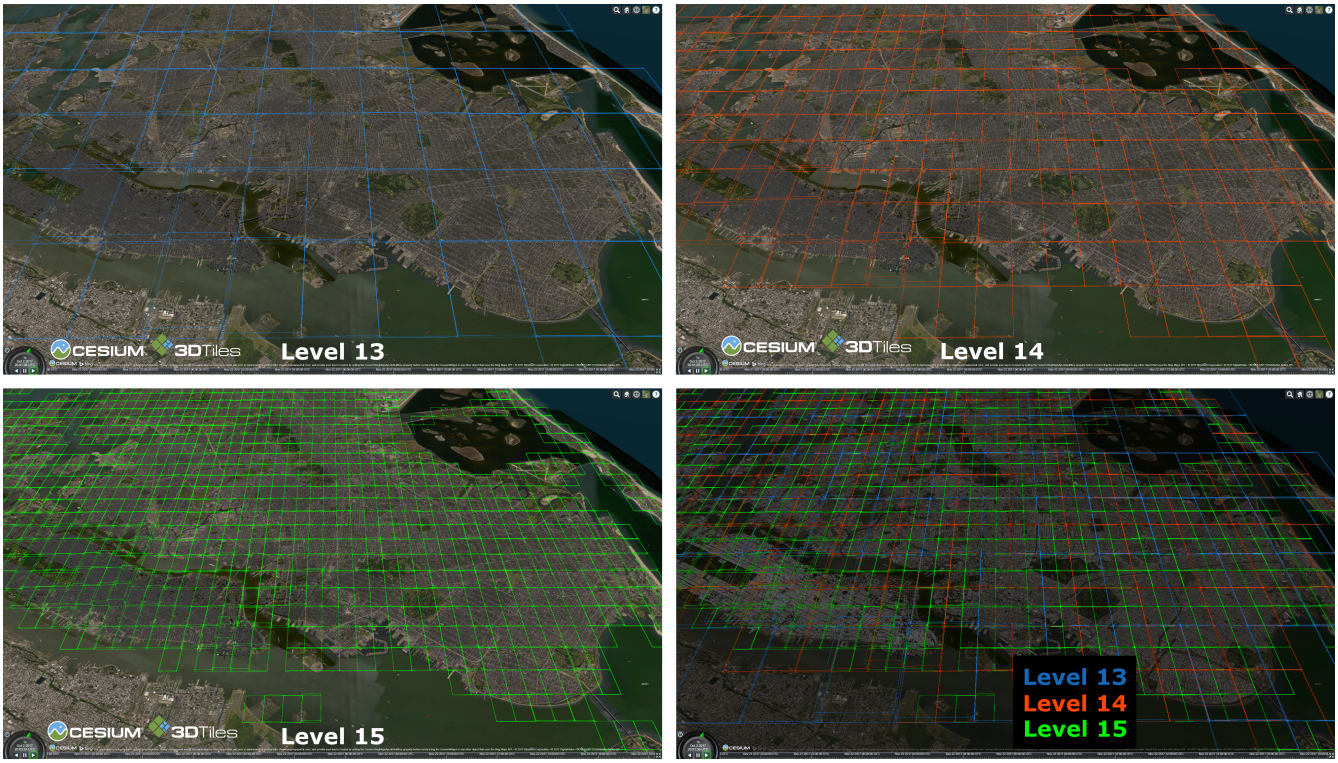

Figure 14. Bounding boxes for New York using Georeferenced Grid Tiling. Top Left: Grid Level 13. Top Right: Grid Level 14. Bottom Left : Grid Level 15. Bottom Right : Composited view of Grid Levels 13, 14, 15 for comparison.

5.2.5. Per-Object glTF to 3D Tiles and Improving Runtime Performance

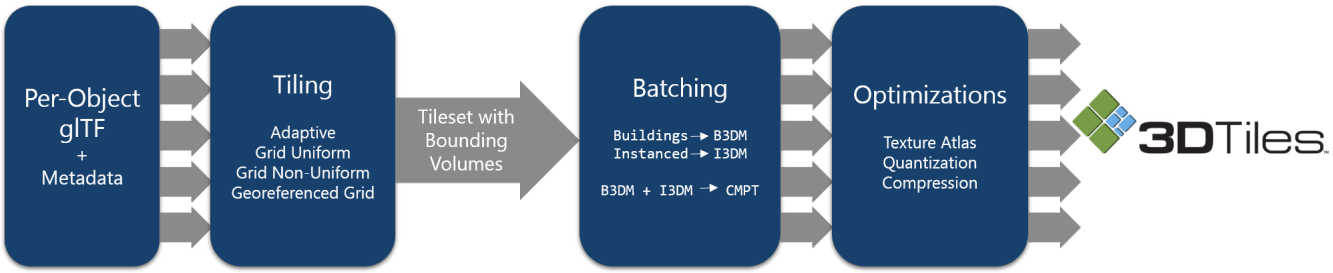

Figure 15. Conceptual diagram of Per-Object glTF to 3D Tiles

Once the CityGML data has been converted to per-object geometry, position and metadata, the data is processed through tiling algorithms that create optimal 3D Tiles datasets. It also allows us to add features such as terrain clamping, terrain skirts and texture atlas that makes it possible to provide the most accurate view of geospatial data.

For improving runtime performance the goal is to balance:

- the size of the streaming data;
- number of HTTP requests for the data;
- number of draw calls;

- memory usage on client.

Tiling algorithms used are discussed in the tiling algorithms section below.

Batched Tiles for Buildings

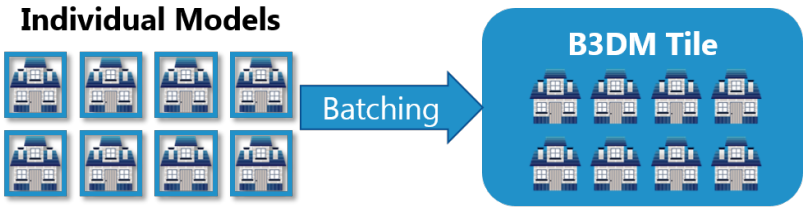

Figure 16. Batched Tiles for Buildings

Batched 3D Model Tiles were used for buildings and other batched models in each tile. In B3DM tiles, each tile could contain numerous objects and their metadata. If each building's glTF were to be stored independently per tile, this would significantly affect runtime performance and make it exponentially slower. Combining buildings into a single model significantly improves runtime performance by reducing the rendering overhead.

Instanced Tiles for Instanced Models

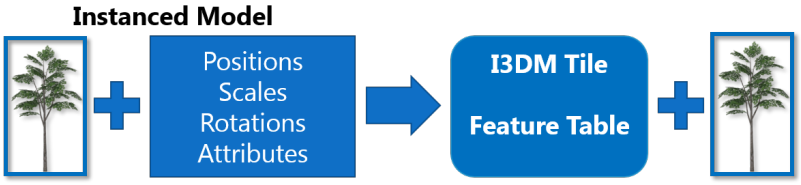

Figure 17. Instanced Tiles for Buildings

Instanced models are objects which have the same base model, but are placed at many positions. These models may also have their own scale and rotation properties per instance. The most efficient way to handle such models is to separate the model from the tile. This way, many individual tiles across the tileset can use the same model. This improves streaming as it eliminates querying and transferring duplicate models, as well as runtime performance by using the same model for rendering which also reduces memory consumption.

For instanced models, 3D Tiles allowed having the rendering payload separate from the Instanced 3D Model tile. A single glTF model was used for all instances of an instanced model across the entire tileset. Because of caching, the browser would not request the instanced glTF model for each tile, thus enhancing streaming performance and runtime memory usage.

Composite Tiles to combine Batched and Instanced Models

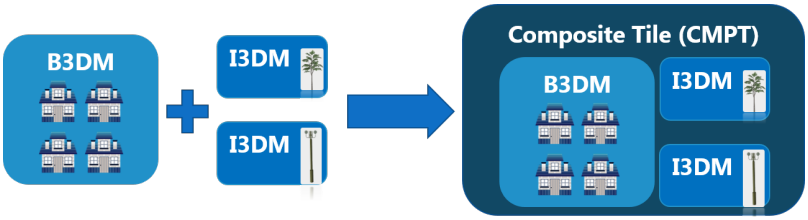

Figure 18. Composite Tiles to combine Batched and Instanced Models

Within the spatial bounding volume of each tile, there may be buildings and different types of instanced models. Although all the buildings are combined into a single B3DM tile, each instanced model required its own I3DM Tile. This resulted in streaming multiple tiles for each spatial bounding volume.

To eliminate this drawback, all B3DM and I3DM tiles of a given spatial bounding volume are combined into a single Composite (CMPT) tile. This resulted in fewer web requests per tile (i.e., 1 request per tile), significantly enhancing streaming performance by reducing the number of requests.

Preserving Metadata

Once the models had been combined into B3DM and I3DM tiles, the metadata of each model was preserved using the 3D Tiles Batch and Feature tables. This allowed picking, styling and querying as if each building were it's own model.

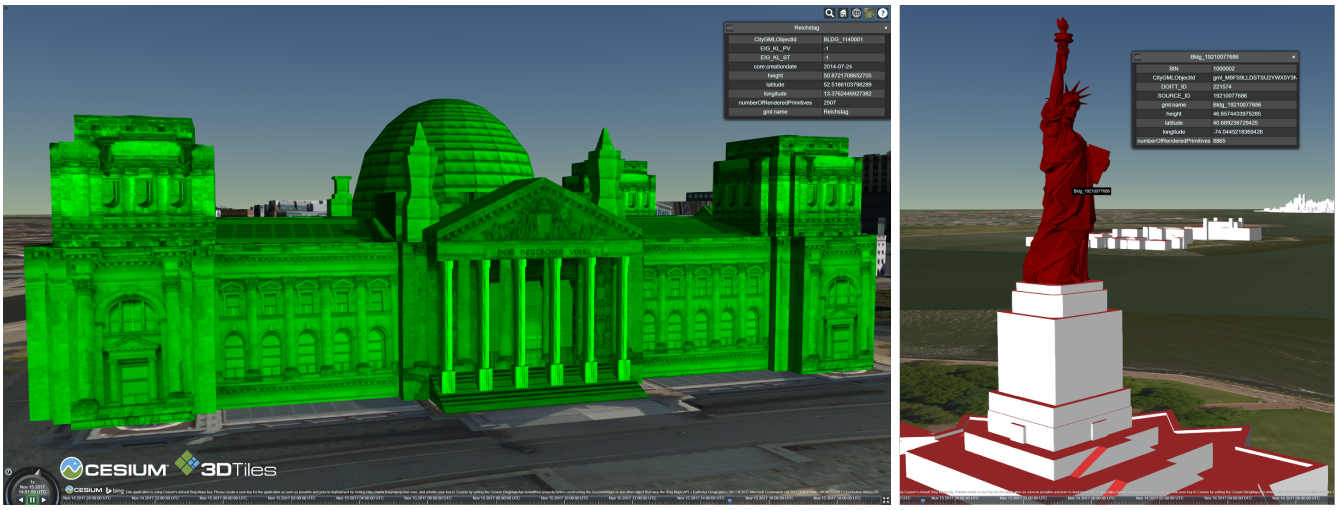

Figure 19. CityGML Model of the Reichstag (left); Statue of Liberty (right) with Properties displayed

Performance Optimizations

Texture Atlas

For textured models, such as the Berlin data set, a texture atlas was created to enhance the rendering performance. Rendering a textured glTF model requires at least one draw call per texture. If textures had been retained as-is from the source data in the glTF model, the number of draw calls would have been equal to the number of textures in the glTF, which would have been unsustainable for real-time rendering.

Texture atlases were used to solve this problem. A texture atlas is an image containing a collection of smaller images, usually packed together. Once the texture atlas is created, texture coordinates associated with each texture in the atlas are modified to reference the texture in the atlas.

When a texture atlas was used, batching was possible across geometry that used different textures. This significantly increased the rendering performance. The runtime performance benefits of using a texture atlas for city-scale datasets cannot be understated.

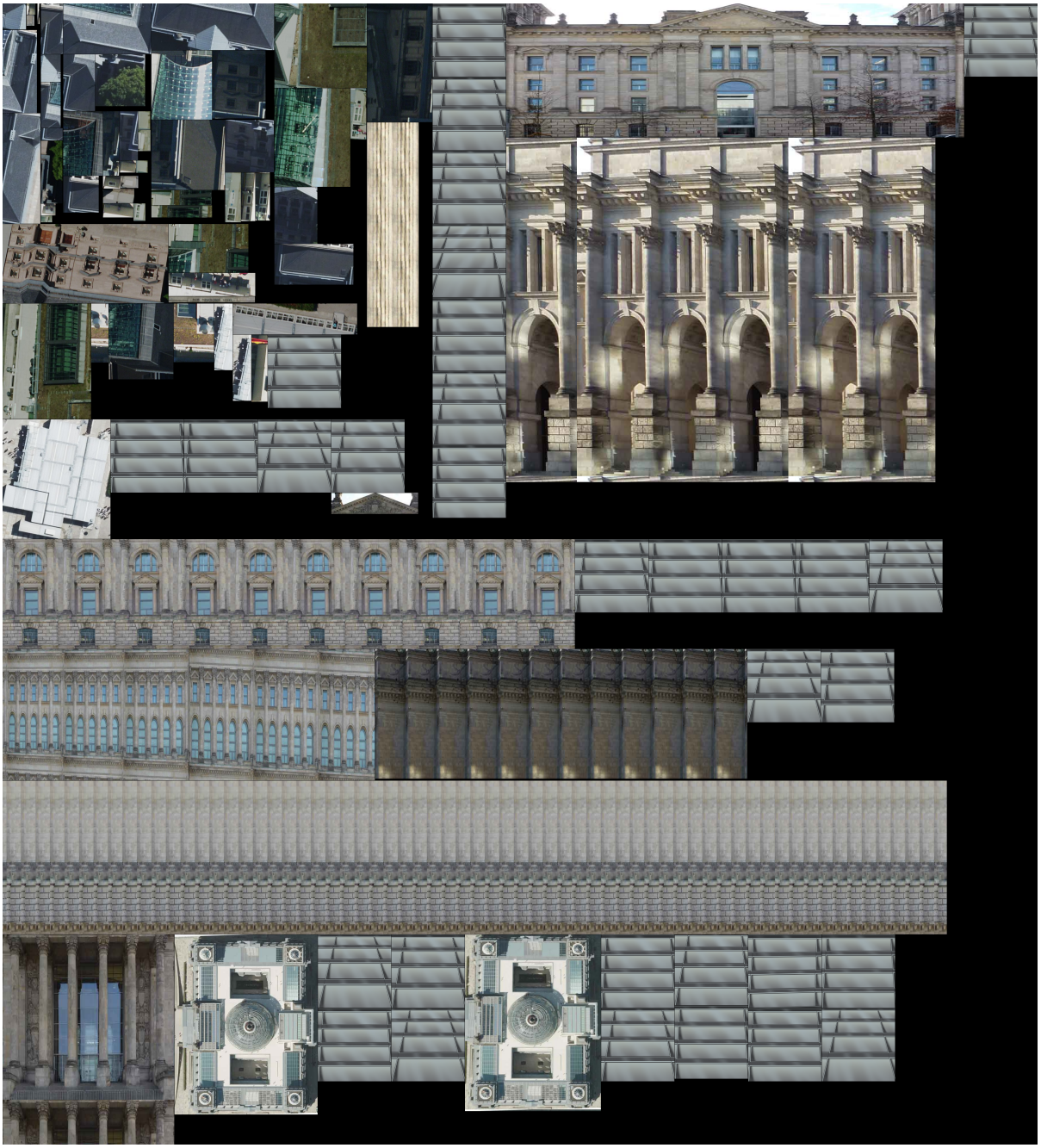

Figure 20. Texture Atlases for tile containing the Reichstag Building

Another challenge of textured models is that if the textures are of high quality, the memory available on the GPU becomes a significant cost. Textures in the Berlin dataset can easily run into hundreds of megabytes just for a few buildings.

To solve this problem, multiple levels of each tile were created, each with a different level of scaling applied to the textures. The tiles at a higher level were scaled down more than the leaf tiles, which were not scaled. Scaling down the textures made the GPU memory usage much more efficient while maintaining visual quality.

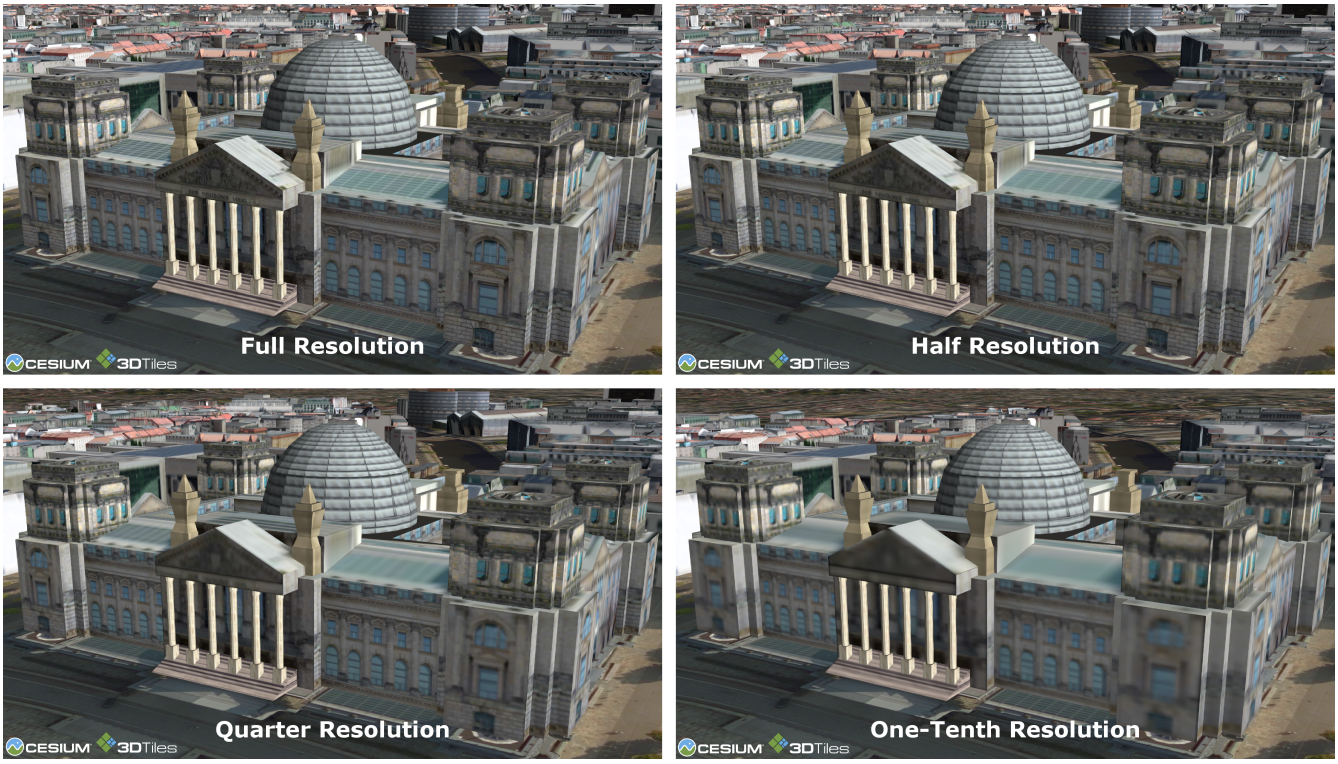

Figure 21. Texture Atlas for Reichstag Model

Quantization and Oct-Encoded Normals

Quantization and Oct-Encoded Normals allow compressing mesh data into a much smaller space. The decompression is done by a simple matrix multiplication in parallel in the vertex shader on the GPU, which results in little-to-no performance overhead. Quantized 3D model files means smaller files, faster downloads, and less GPU memory usage with no performance degradation. Using quantization, the vertex data can be stored in half the amount of space.

Quantization uses `WEB3D_quantized_attributes` extension based on Mesh Geometry Compression for Mobile Graphics, by Jongseok Lee, et al. [http://cg.postech.ac.kr/papers/mesh_comp_mobile_conference.pdf]

Oct-encoded normals were also used. Normals are three-component unit vectors. This representation compresses a three-component unit-length vector to a two component vector where each component is stored in 8-bits each. Thus normals compressed using oct-encoding use 16 bits rather than 96-bits.

Combining these 2 techniques, the size of a non-gzipped New York 3D Tiles dataset was reduced from 3.01GB to 1.43GB; a reduction of 53%, and for gzipped from 780MB to 483MB. There was no noticeable difference in processing for offline conversion.

Compression

Gzip compression was used to compress the tiles. This reduced the size of the tileset by approximately 75%.

Visual Quality

Using Cesium's Sun based lighting, visual improvements such as specular highlights, reflections and shadows were produced. As Cesium is built for time dynamic usage, the Sun's position was

astronomically accurate, thus providing a realistic visualization of the data.

Other Performance and Portability Considerations

The AGI 3D Tiling Pipeline produces 3D Tilesets in a manner intended to have efficient streaming and rendering performance at runtime. The tools were multithreaded to take full advantage of modern multithreaded CPUs and have minimal disk I/O, thus improving offline conversion performance.

The 3D Tiling Pipeline is cross platform, supported on both Windows and Linux. It is possible to run the tools on virtualization software such as Docker.

5.2.6. Results

Sample Images

New York City DoITT CityGML Data Set

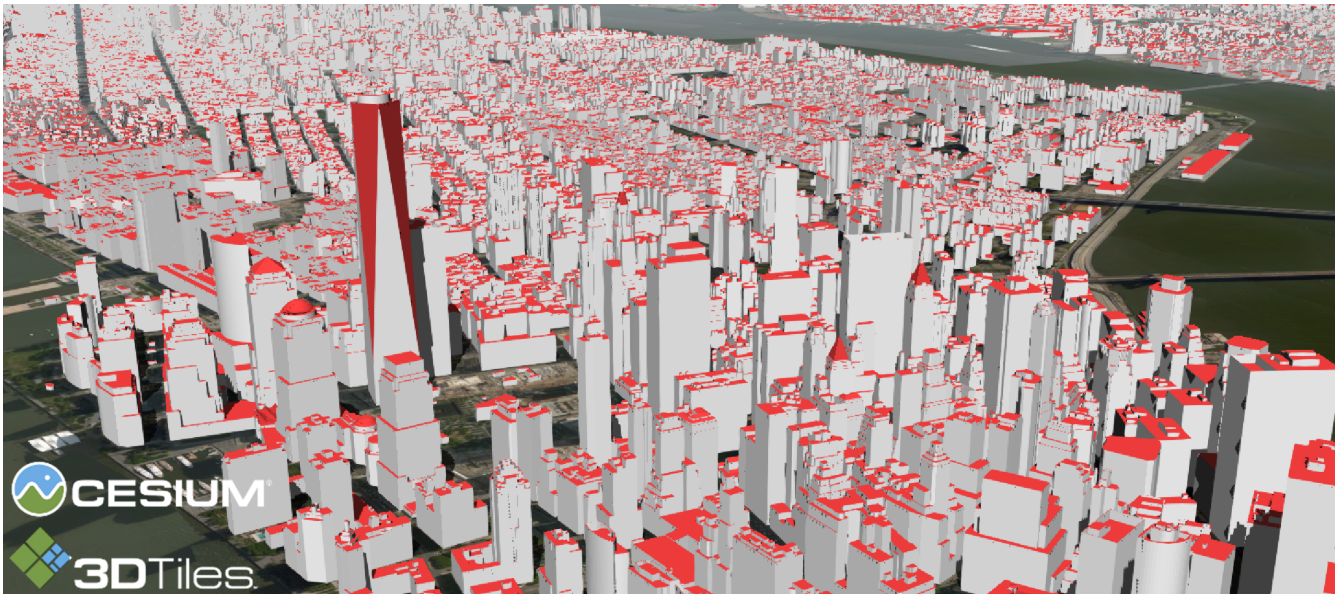

Figure 22. A view of Manhattan from the New York City DoITT CityGML Dataset in Cesium

Figure 23. A birds-eye view of New York City and it's boroughs

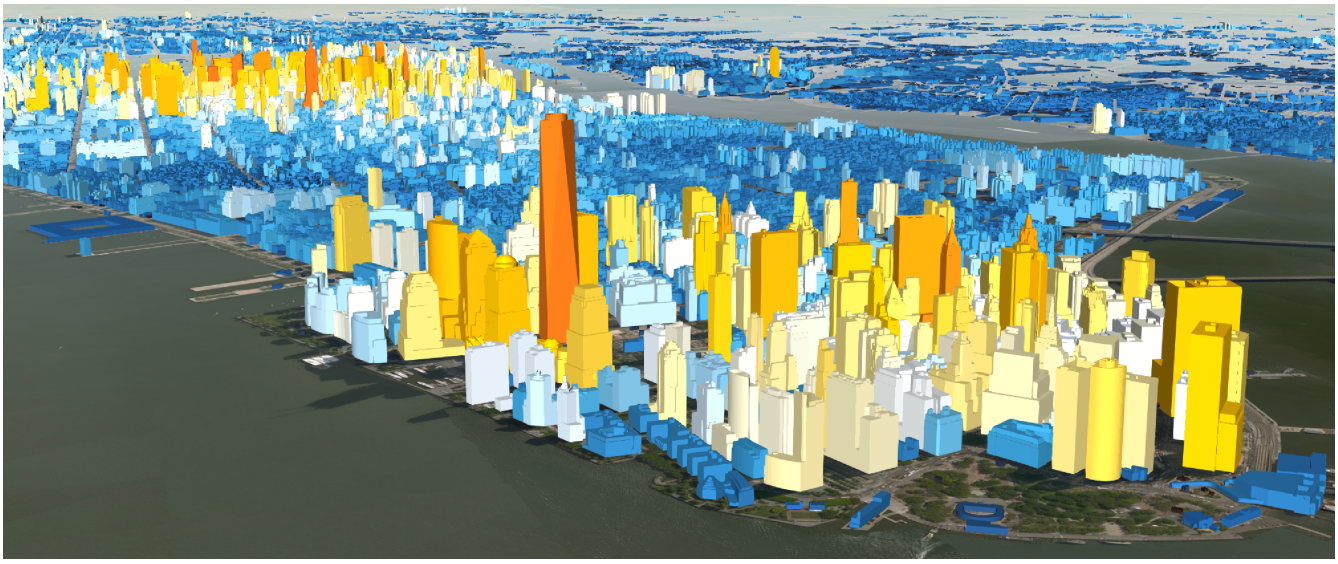

Figure 24. Manhattan buildings colored based on height using 3D Tiles Styling

CityGML Model of Berlin

Figure 25. Textured CityGML Model of the Reichstag (left); Menschen Museum and Fernsehturm (TV Tower) (right)

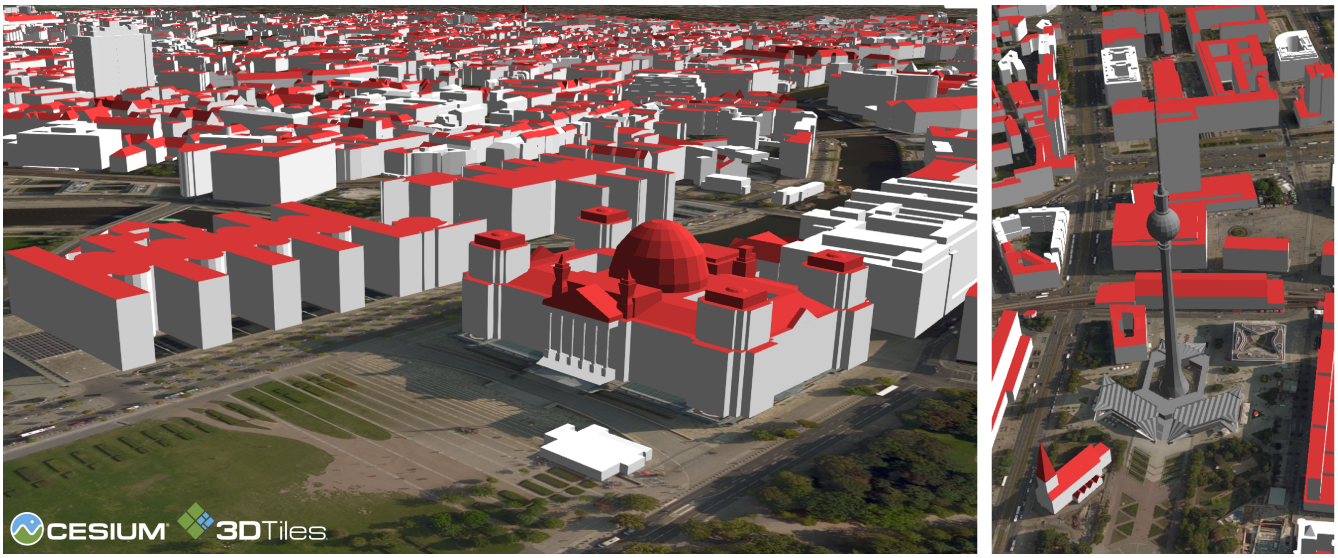

Figure 26. Untextured CityGML Model of the Reichstag (left); Menschen Museum and Fernsehturm (TV Tower) (right)

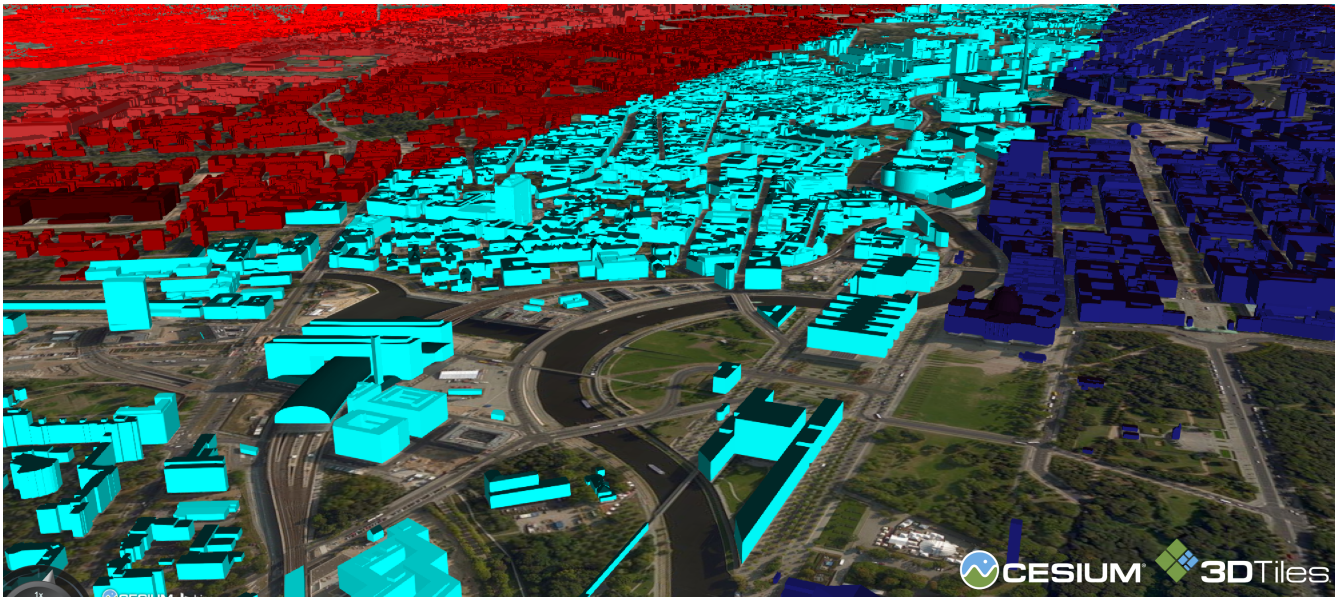

Figure 27. Berlin buildings colored based on latitude using 3D Tiles Styling

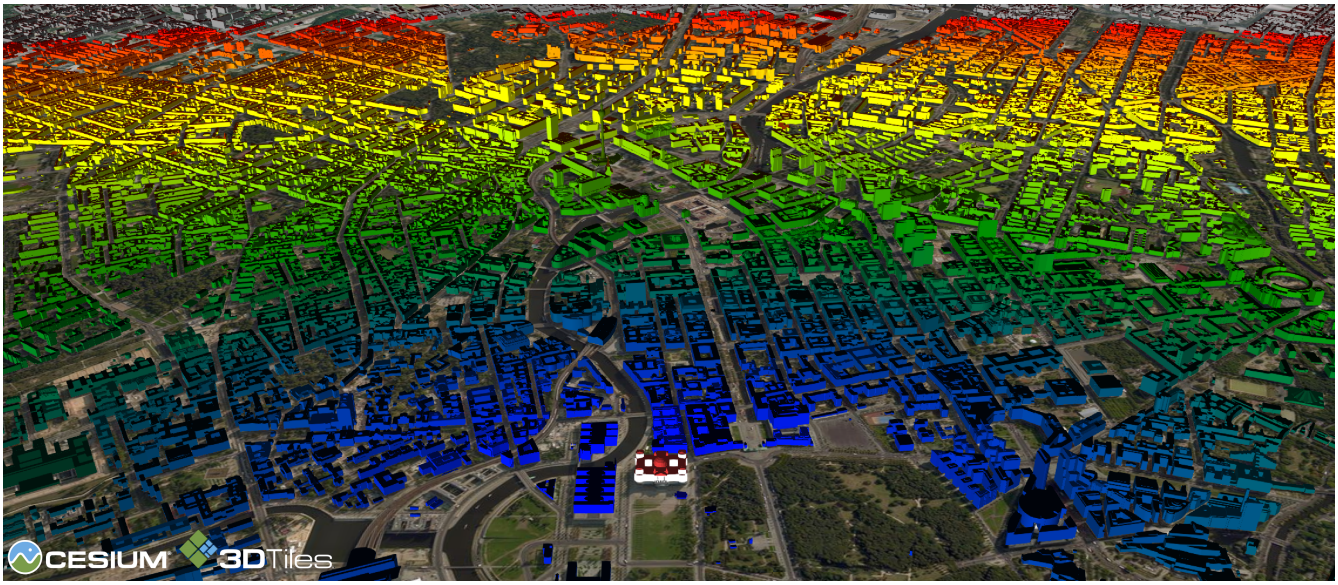

Figure 28. Berling colored using distance from the Reichstag (bottom center) using 3D Tiles Styling

Tiling Performance

New York City DoITT CityGML Data Set (1.08 Million Buildings)

The following results are for an end-to-end pipeline with CityGML as source and 3D Tiles as the output. The processing includes loading CityGML files, EPSG conversions, tiling, processing attribute metadata, batching, quantization and compression.

The performance was benchmarked on an Intel Core i7 4980HQ CPU @ 2.8 GHz.

Table 4. CityGML to 3D Tiles Conversion Performance using 3D Tiling Pipeline

Tiling Algorithm	Level	Time to produce 3D Tiles from CityGML	Number of Tiles	Total Size of Tileset (MB)	Average Size of Tiles (KB)
Adaptive Quadtree	N/A	34m 22s	2,610	499	316

Tiling Algorithm	Level	Time to produce 3D Tiles from CityGML	Number of Tiles	Total Size of Tileset (MB)	Average Size of Tiles (KB)
Georeferenced Grid Quadtree	14	42m 48s	466	483	1743
Georeferenced Grid Quadtree	15	33m 42s	1,624	492	504
Grid Non Uniform Quadtree	5	33m 37s	1,022	488	798
Grid Non Uniform Quadtree	6	29m 49s	4,076	506	204
Grid Uniform Quadtree	5	40m 33s	468	483	1743
Grid Uniform Quadtree	6	31m 33s	1629	492	503

The Adaptive Quadtree algorithm has a good balance of average tile size, number of tiles and the time needed to process it. It also provides the best visual experience as it contains buildings throughout the tree rather than at a single level.

The Georeferenced Grid Quadtree can be useful when working with other data, such as imagery or terrain, that follow the same subdivision algorithm. It provides a one-to-one mapping with the tiles of other datasets as it depends on the starting extent rather than the extent of the buildings.

The Grid Quadtree, which offers both uniform and non-uniform subdivision, provides a good middle ground between Adaptive and Georeferenced grid. It takes into account the extent of the dataset and produces tiles at a single level. It is useful when the desired output needs to be non-uniformly subdivided but maintain a single level of tiles.

5.3. Experiment 2: CityGML on GeoRocket Database to Cesium via 3DPS Query

In Experiment 2 the open source data store GeoRocket (https://georocket.io/) was used for the CityGML data store. GeoRocket is intended to be a high-performance data store for geospatial files. It can store 3D city models (e.g. CityGML), GML files or any other geospatial data sets provided in the XML or GeoJSON formats. A summary of GeoRocket features is included here for reader convenience:

- Data storage with multiple back-ends such as Amazon S3, MongoDB, distributed file systems (e.g. HDFS or Ceph), or use the local hard drive.
- Schema-agnostic and format-preserving, supporting a range of data schemas and not altering stored data. Anything imported into GeoRocket can be retrieved later without changes.
- Search features based on the popular Open-Source framework Elasticsearch. GeoRocket supports spatial queries and searches for attributes, layers and tags.
- GeoRocket is designed for the Cloud. It is intended to be reactive and able to handle large files and a large number of parallel requests.

The CityGML scenario data was stored in GeoRocket and was exported as CityGML using spatial queries. The diagram below shows the most important elements of GeoRocket.

Of the supported data stores the local hard drive option was chosen. The CityGML dataset was initially imported via the GeoRocket HTTP API. Clients could then query via the same HTTP API spatial regions and receive a CityGML file with spatially corresponding content.

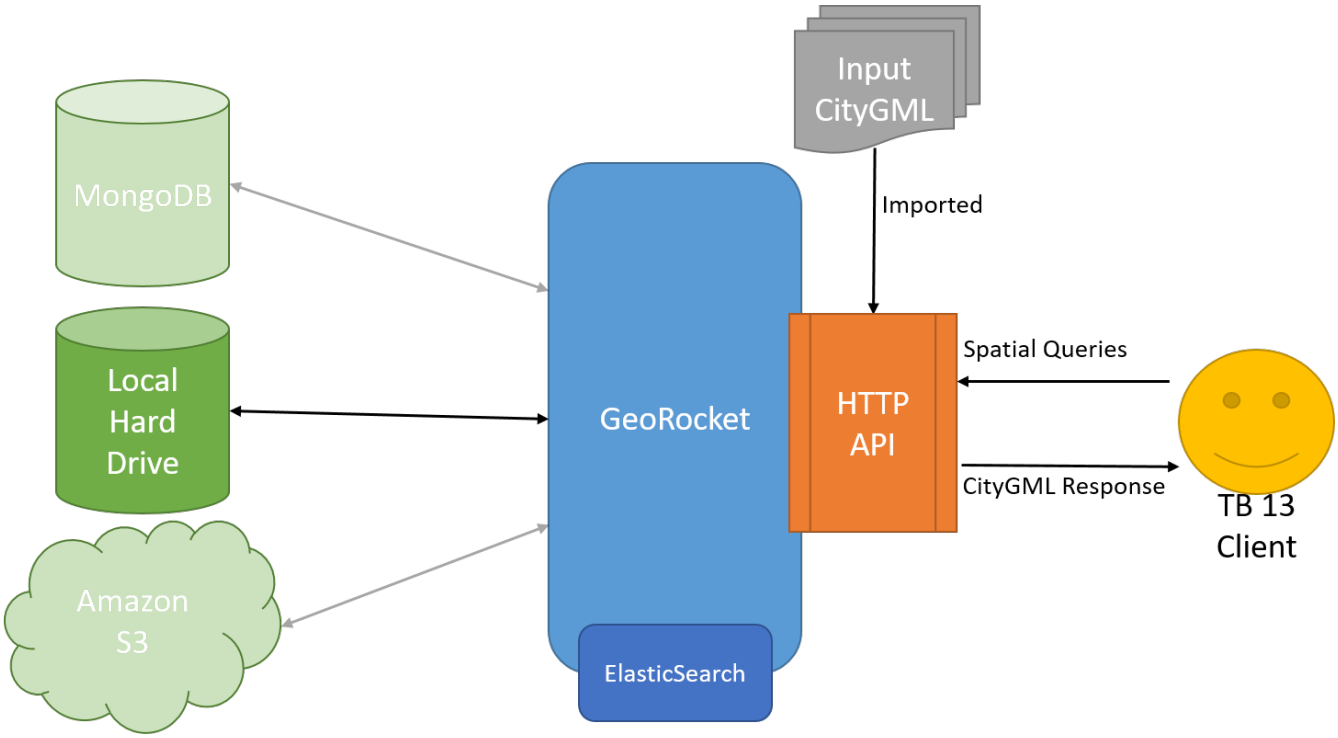

Figure 29. Experiment 2 - This diagram sketches the most important elements of GeoRocket from the supported data stores to the API which the client can use.

The elements of GeoRocket utilized in this experiment included:

- The GeoRocket open source database, open data CityGML models.
- Documentation on how to set up GeoRocket and import the scenario data.
- An instance of a GeoRocket server including the scenario data, that can be queried online within an evaluation period of 6 months that begins after the final demonstration.
- An extension of GeoRocket providing conversion from CityGML to 3D Tiles.
- An investigation of whether it is possible and how much effort it is to export I3S from GeoRocket.
- A software component on top of GeoRocket providing the exported 3D Tiles datasets through a 3DPS conforming service. This component uses GeoRocket's software stack for fast content delivery.

5.3.1. Experiment 2a: CityGML to GeoRocket to 3D Tiles via 3DPS visualized in Cesium Client

This experiment evaluated the complete flow of data from its originating CityGML format to a web-enabled visualization with Cesium via OGC's 3D Portrayal Service (3DPS). This data flow included:

- The conversion from the CityGML data format served by GeoRocket, to 3D Tiles dataset.
- The import of the 3D Tiles dataset to the 3DPS Framework
- The Cesium client which queried:
  - the 3DPS Framework for 3D geometries
  - a separate Attribute Server, provided by the HFT Stuttgart, serving further information via the getFeatureById interface

The pictogram below shows a visual representation of that data flow:

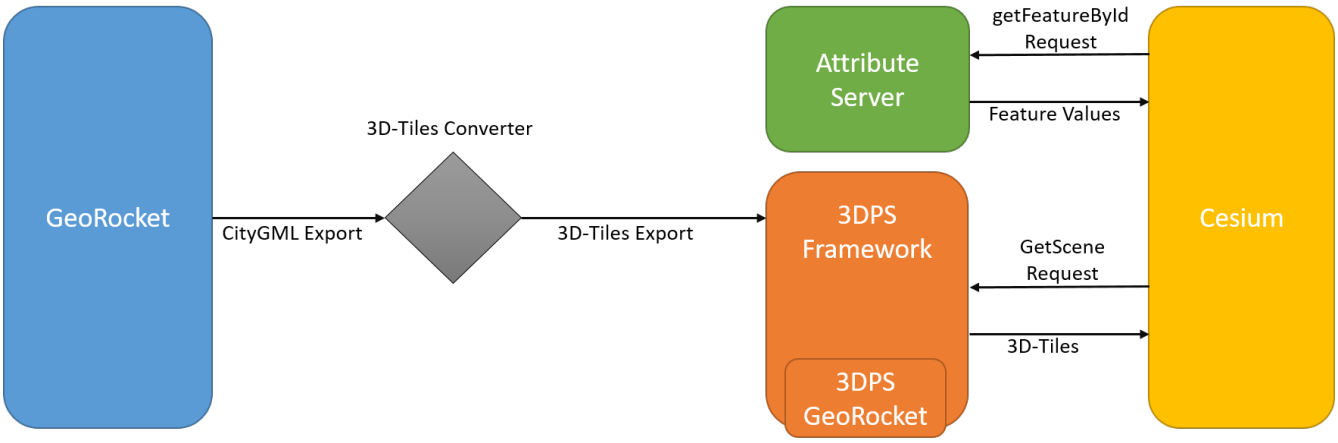

Figure 30. Experiment 2a - The dataflow from GeoRocket to the visualized 3D Tiles, which are requested via the 3DPS queries.

To achieve the best possible spatial query performance using the 3DPS framework, the complete dataset was initially retrieved from GeoRocket and converted into the 3D Tiles format.

The resulting 3D Tiles dataset was then imported into the 3DPS framework, which was able to satisfy the spatial queries. This saved time because it was not required that CityGML content be transformed to 3D Tiles on every spatial query.

On a current high-performance computer with a Core i7 CPU and 32 GB memory, conversion of the New York City dataset (~29.5 GB of data) took about 18 minutes. The flow through the 3DPS Framework took approximately 14 seconds to rearrange the 3D Tiles dataset and deliver it to the client.

Task	Time
Convert New York DoITT to 3D Tiles (29.5 GB)	~18 minutes
Rearrange 3D Tiles dataset of New York DoITT	~14 seconds

Figure 31. Time spent to transform New York's CityGML dataset to 3D Tiles in contrast to rearrange an existing 3D Tiles dataset, to satisfy a spatial query.

A dataset of southern Manhattan with about 325 MB took 13 seconds to convert, while the delivery via the 3DPS framework required a half-second.

Task	Time
Convert Manhattan to 3D Tiles (325 MB)	~13 seconds
Rearrange 3D Tiles dataset of Manhattan	~0.5 seconds

Figure 32. Time spent to transform a CityGML dataset of southern Manhattan to 3D Tiles in contrast to rearrange an existing 3D Tiles dataset, to satisfy a spatial query.

To achieve this performance the 3DPS framework internally used a modified version of GeoRocket. This version used the 3D Tiles spatial hierarchy in order to quickly rearrange the data and therefore delivered only the requested subset of the original hierarchy. One drawback was that on higher levels of the 3D Tiles hierarchy the delivered elements intersected the queried spatial bounds, so that more data/buildings were shown than were queried.

In this experiment, the 3D Tiles styling feature was used to avoid rendering those intersecting objects. But it is expected in the future that the returned geometries will be modified to only deliver the requested content.

Demo

A public demo is available at: tb13.igd.fraunhofer.de:8080/Apps/Sandcastle/index.html?src=3DPS_r.html&label=Showcases [http://tb13.igd.fraunhofer.de:8080/Apps/Sandcastle/index.html?src=3DPS_r.html&label=Showcases]

The 3DPS server offered two datasets:

- Whole of New York
- A subset of Manhattan, which was used to simulate the heat demand of each building which can be visualized in the demo (see following screenshots)
- The textured dataset of Berlin

The demonstration was initially presented at the OGC Testbed 13 S3D Performance work package meeting on Monday, September 4th 2017.

The following sequence diagram shows data flows between the elements depicted in the overview diagram of experiment 2a.

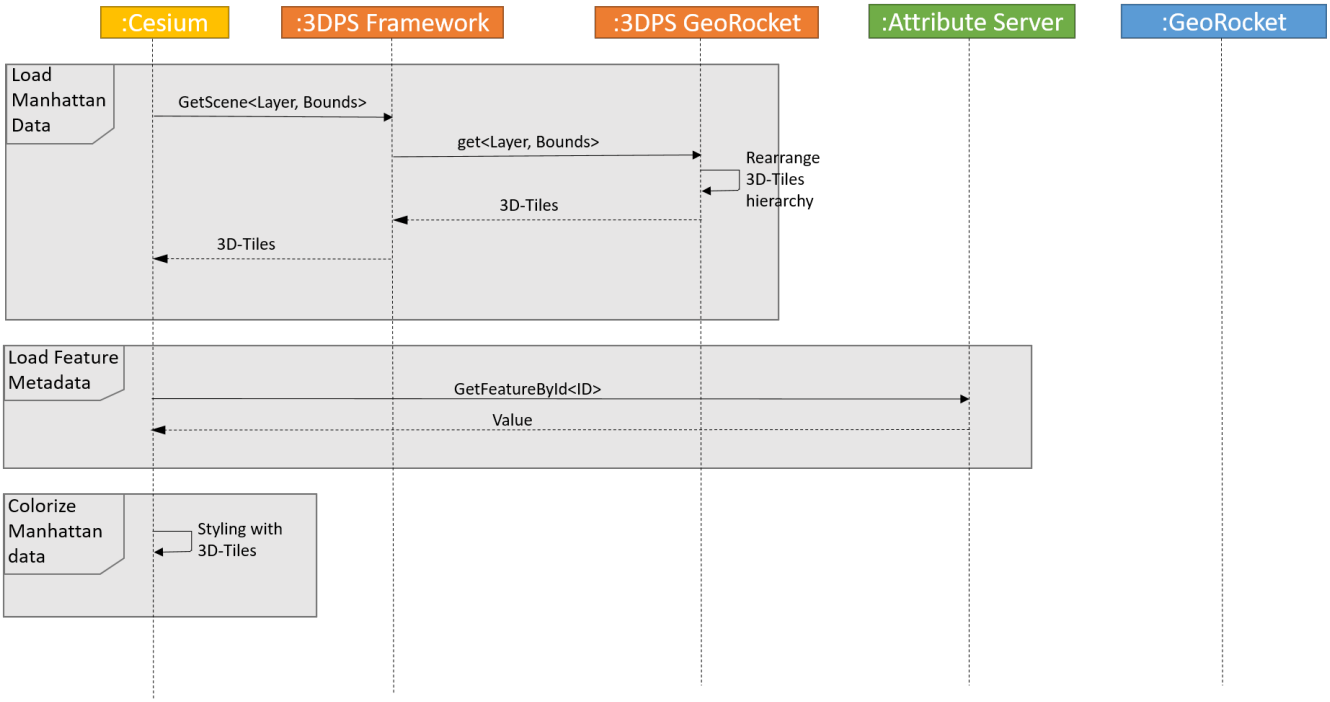

Figure 33. Experiment 2a - The dataflow from GeoRocket to the visualized 3D Tiles, which are requested via the 3DPS queries.

Several demo screenshots follow:

Figure 34. Dataset of New York served via 3DPS.

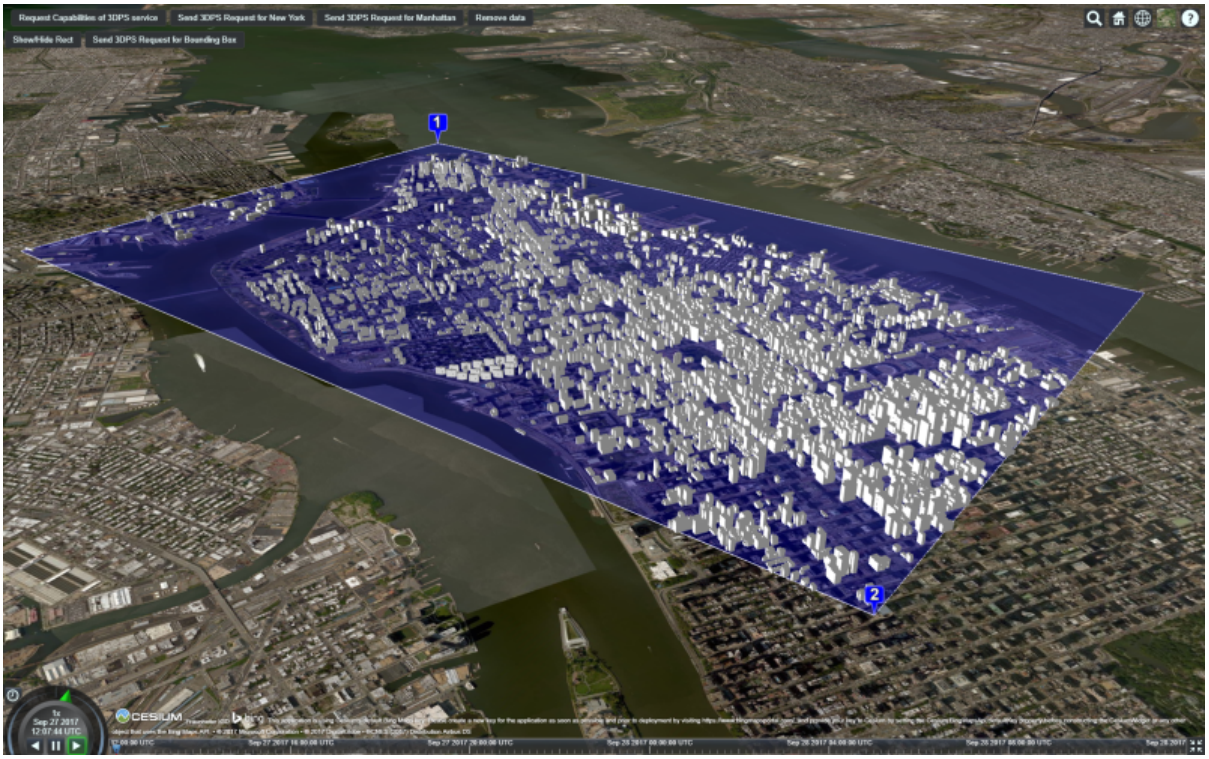

Figure 35. A subset of New York's dataset selected by a spatial query via 3DPS.

Figure 36. Manhattan dataset with simulated heat demand, provided by HFT Stuttgart, and color coded in to the building's appearance.

5.3.2. Experiment 2b: CityGML to GeoRocket to I3S via 3DPS visualized in Cesium Client

This experiment investigated if and how a workflow similar to that used in experiment 2a could be established using the Esri I3S format.

Since the 3DPS framework uses a modified GeoRocket version, which can handle 3D Tiles, the data

flow of this experiment differed from that used in experiment 2a. The following figure highlights the differences with red arrows.

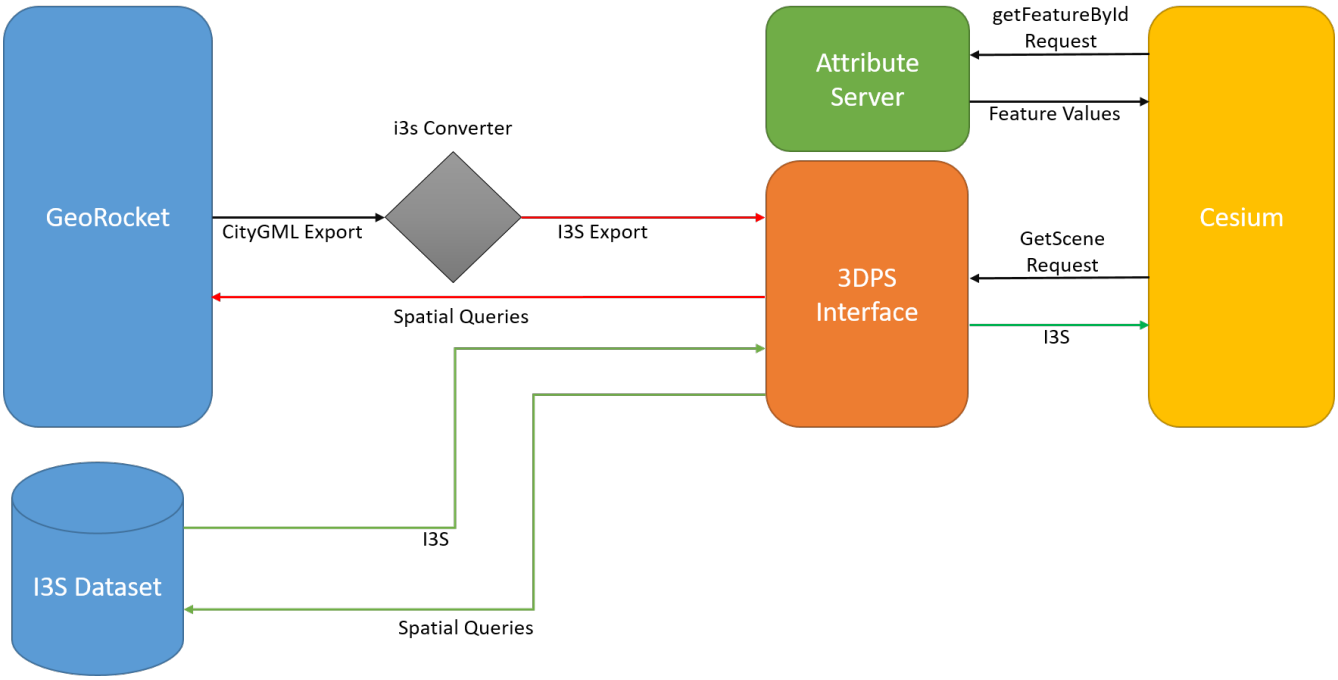

Figure 37. Experiment 2b - This image shows two dataflows: 1: The supposed dataflow (in red) from GeoRocket which exports CityGml, to converted I3S (not implemented), which are then visualized in the Cesium client via 3DPS queries. 2: This dataflow connects (in green) the Cesium client via a 3DPS interface with an initially converted I3S dataset.

The figure points out that the spatial queries would be forwarded to GeoRocket for every spatial 3DPS GetScene request to retrieve the spatially bounded CityGML data, which was then converted to the I3S data format.

In order to do this, a tool to convert the CityGML dataset to I3S would have been needed. The team was not aware of any standalone tool that would have been available in time for the experiment, so it was decided not to implement an I3S converter.

The focus of the experiment was shifted to the rendering of I3S in the Cesium client using the 3DPS. As a result, the user could query a scene via 3DPS by specifying a spatial region (rather than a specific resource via a URI. The result can be delivered either using I3S or 3D Tiles as a data delivery format, depending on which data is available for the specified region. The Cesium client can render both the I3S as well as the 3D Tiles content. This interoperability between geospatial data web consumers and 3DPS regarding the request of hierarchical 3D data storages was proven by a prototype implementation.

The main goal was to investigate if the portrayal service could abstract the access of 3D Tiles or I3S in a dedicated web client/consumer which is designed to be compatible with specific formats (e.g. Cesium, 3D Tiles). Cesium is a web globe API designed to support 3D Tiles. ESRI's ArcGIS API for JavaScript is designed to consume I3S. The first approach attempted to render I3S data in Cesium employing a request scheme similar to 3DPS's ***getScene*** request.

Figure 38. Rendering I3S in Cesium overview

On the client side, a request was sent to the server when the camera changed event was triggered in Cesium. On the server side, which acts as a broker, the implemented API was responsible for handling any requests from the client. The core action of the broker service was to apply the I3S node selection criteria, then generate a response (including the nodes' description) to be rendered in Cesium.

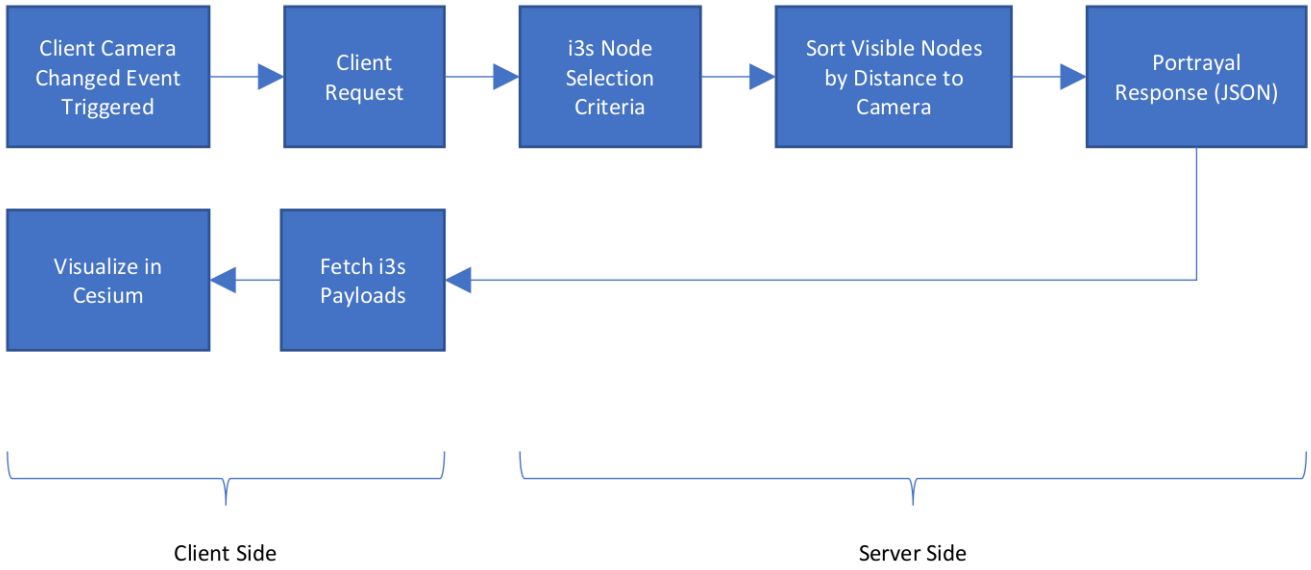

Figure 39. Communication Synopsis

The I3S node selection is a repetitive process which accesses the root node of the I3S layer and continues by traversing descendant nodes of the tree structure. The node selection criteria are:

- node’s minimum bounding sphere (mbs) visibility,
- node’s minimum bounding sphere screen size in pixels,
- existence of a node’s children.

The following pseudo code describes the traversal of an I3S node in the broker service:

```
if node's mbs in not visible in client's viewport
    break traversal of the node and it's children
else if node's screen size >= node’s maxScreenThreshold and node has children
    traverse node's children
else
    add node in the response for rendering
```

I3S supports discrete LODs, where different LoDs are bound to the different levels of the nodes'

hierarchy. Leaf nodes usually contain the original representation with the highest detail, whereas lower node levels contain simplified geometry of the same features using edge collapse algorithms.

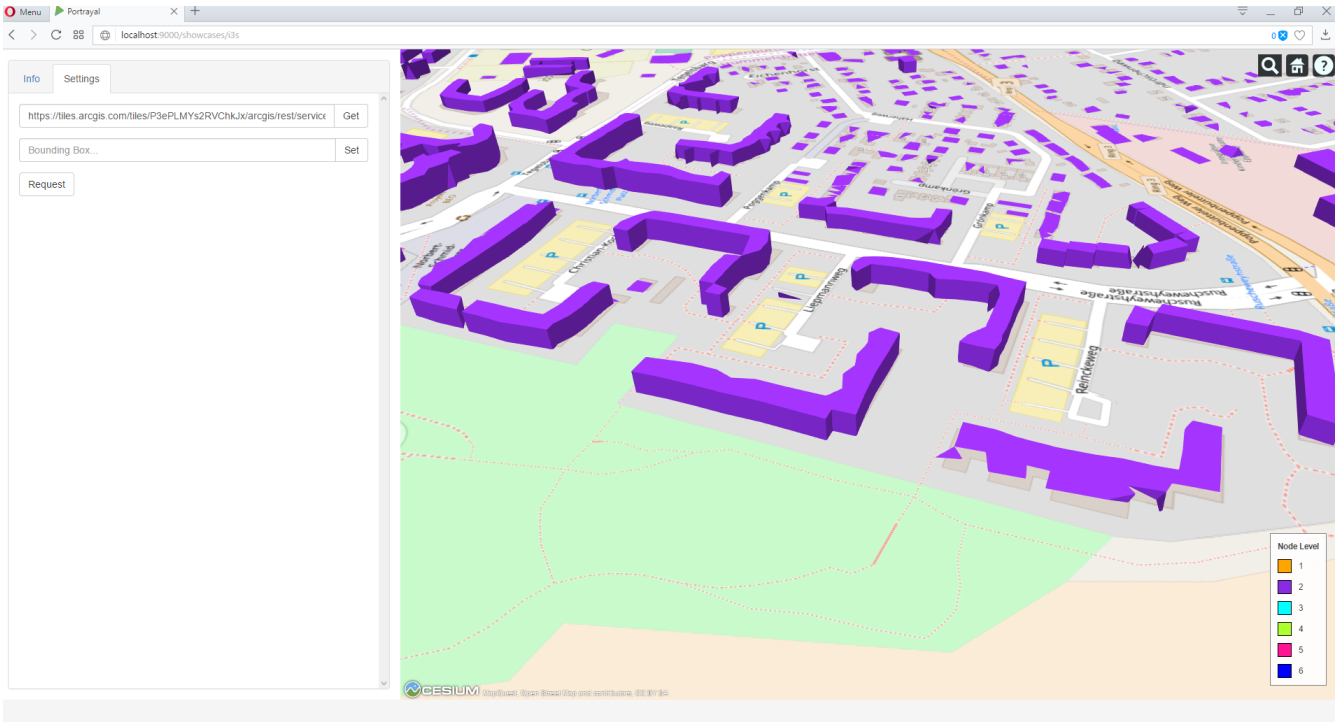

Figure 40. Decimated geometry of I3S low level nodes rendered in Cesium

When the I3S node traversal is completed, a response is generated which contains the essential information to render a node in Cesium. For rendering optimization, the response nodes are sorted in ascending order by distance to camera (see the figure below).


```
[
  + {…},
  - {
        id: "5-1-0-0-0",
        level: 6,
        lng: 9.994545253174223,
        lat: 53.568095227256954,
        radius: 455.4209899902344,
        url: "https://tiles.arcgis.com/tiles/P3ePLMYs2RVChkJx/arcgis/rest/services/Buildings_Hamburg/SceneServer/layers/0/nodes/5-1-0-0-0",
        distanceToCamera: 140.98541019527966,
        time: "1504872740063"
    },
  - {
        id: "5-0-0-2-0",
        level: 6,
        lng: 9.986759068536088,
        lat: 53.57170940098039,
        radius: 563.823974609375,
        url: "https://tiles.arcgis.com/tiles/P3ePLMYs2RVChkJx/arcgis/rest/services/Buildings_Hamburg/SceneServer/layers/0/nodes/5-0-0-2-0",
        distanceToCamera: 162.51702725550808,
        time: "1504872740063"
    },
  - {
        id: "5-1-0-0-1",
        level: 6,
        lng: 9.997446293904332,
        lat: 53.565953434999045,
        radius: 546.78515625,
        url: "https://tiles.arcgis.com/tiles/P3ePLMYs2RVChkJx/arcgis/rest/services/Buildings_Hamburg/SceneServer/layers/0/nodes/5-1-0-0-1",
        distanceToCamera: 214.9592984261286,
        time: "1504872740063"
    },
  + {…},
  + {…},
```


Figure 41. Broker service response to Cesium

The client is online and can be accessed at http://81.169.187.7:9000/showcases/i3s. Instructions of how to use the client can be found at https://www.youtube.com/watch?v=SithZWHPgNQ.

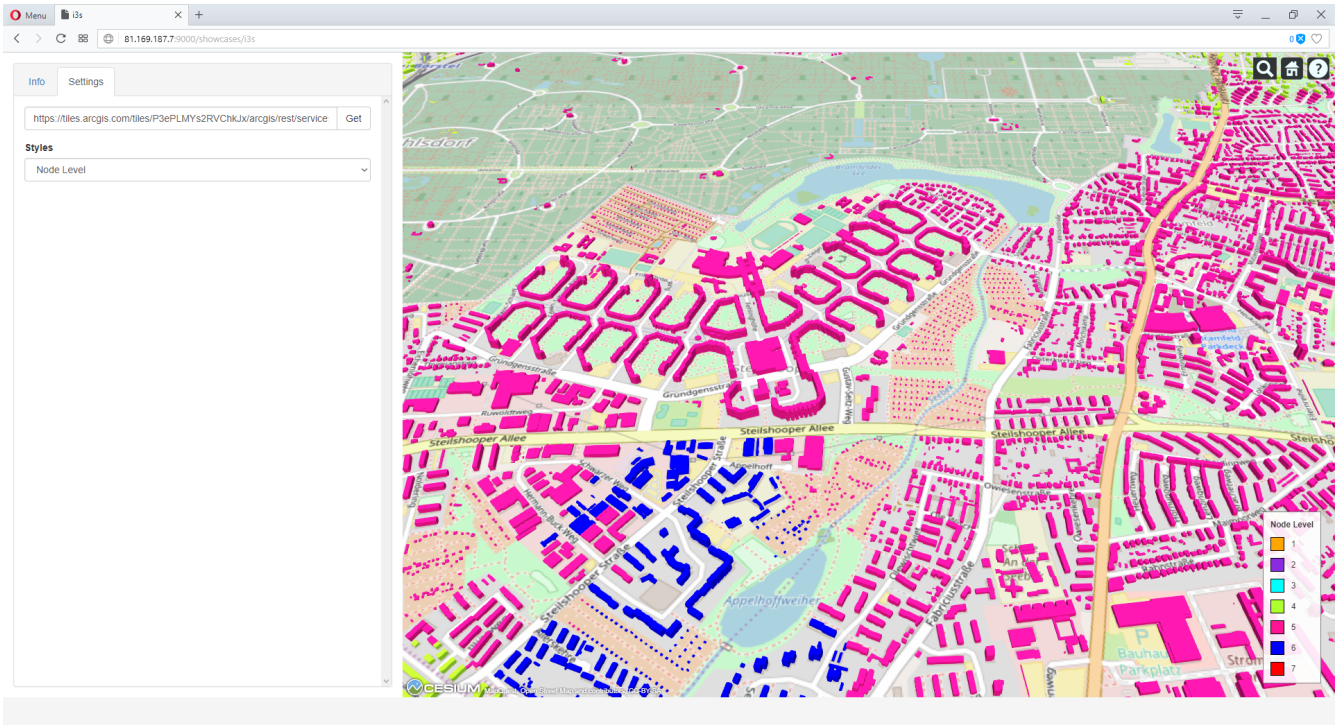

Figure 42. Client to render I3S in Cesium

In this experiment, the 3DPS was used as a broker to simplify data access. It basically translated user queries to a URI of the relevant 3D data set in I3S. Of course, 3D Tiles data sets can be handled the same way, but the focus was on rendering I3S in Cesium. The cost of using the 3DPS as a broker intermediary is O(log n) if a spatial index is used on server side for searching the relevant data set.

5.4. Experiment 3: CityGML to I3S to ArcGIS

5.4.1. Experiment 3a: CityGML to I3S visualized in ArcGIS Client

The OGC I3S Community Standard

Indexed 3D Scene Layers, often abbreviated as I3S, originated from investigations into technologies for rapidly streaming and distributing large volumes of 3D content across enterprise systems that may consist of server components, cloud hosted components, and a variety of client software from desktop to web and mobile applications. In August of 2017, I3S was approved [http://www.opengeospatial.org/pressroom/pressreleases/2639] as an OGC Community Standard [http://www.opengeospatial.org/standards/i3s]. An abbreviated description of I3S is provided below for reader convenience.

A single I3S data set, referred to as a Scene Layer, is a container for arbitrarily large amounts of heterogeneously distributed 3D geographic data. I3S Scene Layers are designed to provide clients access to data. Clients have the ability to then visualize the data for the layer independently according to their needs. Data here refers to vertex geometry, texture as well as any associated attributes. An I3S Layer is characterized by a combination of layer type and profile that fully describes the behavior of the layer and the manner in which it is realized within the specification.

The I3S standard defines the following layer types:

- 3D Objects (e.g., Building Exteriors from geospatial data and 3D models),

- Integrated Mesh (e.g., an integrated surface representing the skin of the earth including vegetation, buildings and roads from satellite, aerial or drone imagery via dense matching photogrammetry), and
- Points (e.g. hospitals or Schools, trees, street furniture, signs, etc. from GIS data).

Layers are described using two properties: type and profile. The type of a layer describes the type of geospatial data stored within it drawing from terms including 3D Objects and Points. The profile for a layer includes additional detail on the specific I3S implementation for the layer that is exposed to clients. Each layer has a canonical profile, but in certain cases multiple layers that represent semantically different types of information can make use of the same underlying profile. In other cases the same layer type can support multiple profiles optimized for different use cases.

I3S organizes information using a hierarchical, node-based spatial index structure in which each node's payload may contain features with associated geometry, textures and attributes. In an Indexed 3D Scene layer, the spatial extent of the data is split into regions, called nodes, with roughly equal amounts of data, and organized into a hierarchical and navigable data structure - the index - that allows the client to quickly discover which data it actually needs and the server to quickly locate the data requested by any client. Node creation is capacity driven - the smaller the node capacity is, typically the smaller the spatial extent of each node will be.

I3S is agnostic with respect to the model used to index objects/features in 3D space. Both regular partitions of space (e.g. Quadtrees and Octrees) as well as density dependent partitioning of space (e.g. R-Trees) are supported. The specific partitioning scheme is hidden from clients who navigate the nodes in the tree exposed as web resources. The partitioning results in a hierarchical subdivision of 3D space into regions represented by nodes, organized in a bounding volume tree hierarchy (BVH). Each node has an address and nodes may be thought of as equivalent to tiles.

LOD is a concept intrinsic to the I3S standard, covering several use cases, including, splitting up very heavy features such as detailed building or very large features (coastlines, rivers, infrastructure), thinning/clustering for optimized visualization as well as support for representing externally authored multiple LODs. Scene Layers may include levels of detail that apply to the layer as whole and serve to generalize or summarize information for the layer, similar to image pyramids and also similar to raster and vector tiling schemes. A node in the I3S scene layer tree could be considered the analog of a tile in a raster or vector tiling scheme. Scene layers support levels of detail in a manner that preserves the identity of the individual features that are retained within any level of detail.

Geometric objects in all Scene Layer types make use of the same fundamental set of primitive geometry types:

- points,
- lines, and
- triangles.

Geometries use binary storage and consumption representation, controlled by Array Buffer View geometry property declarations. I3s provides full control over those properties, such as per-vertex layout of components (e.g. position, normal and texture coordinates), in order to ensure the same pattern for face and vertex elements across the Scene Layer.

Textures are used to provide visual representation of the surface of an object, and are stored as a binary resource associated with a node. The texture resource for a node contains the images that are used as textures for the features stored in the node. The mesh-pyramids profile supports encoding the same texture resource in multiple formats, catering for bandwidth, memory consumption and optimal performance consideration on different platforms. As a result, the I3S standard supports most commonly used image formats such as JPEG/PNG as well as rendering optimized compressed texture formats such as S3TC . In all cases, the standard provides flexibility by allowing authoring applications to provide additional texture formats via the textureEncoding declarations that use MIME types. For example, most existing I3S services provide *image/vnd-ms.dds* (for S3TC compressed texture) in addition to the default *image/jpeg* encoding.

Attribute access in I3S is supported using two different access paths:

1. From optional paired services that expose query-able and updatable RESTful endpoints that enable direct access to dynamic source data, including attributes. The query in this case uses the unique feature-ID key – which is always maintained within each node and is also available as part of the descriptor for any segmented geometry.
2. From fully cached attribute information, in binary form, within an I3S store. Locally Cached Attributes use a binary storage representation based on Array Buffers - the attribute values are stored as a geometry aligned, per field (column), key-value pair arrays. I3S clients can choose to use either or both of these modes even if the attributes are fully cached within I3S store.

The CityGML to I3S conversion process

The CityGML data provided by the New York City Department of Information Technology & Telecommunications (NYC DoITT) consisted of 20 discrete CityGML files for different sections of the city. These files varied in size from 210MB to 1.4GB.

Each file type was processed individually using a Safe FME workbench to convert the CityGML formatted data into discrete file Geodatabase features. These features consisted of “BuildingShell” single multipatch feature of each individual building by building ID and “BuildingShellPart” which outputted the CityGML LOD2 discrete features (roof, walls, ground) for each individual building by building ID. The 20 individual area outputs were merged to a single file Geodatabase. Building attributes from the city’s building footprint were added to the building features.

The resultant file Geodatabase was then processed through the I3S geoprocessing tool to create an I3S scene layer package (SLPK). The resultant SLPK is then either uploaded or published directly to ArcGIS Online.

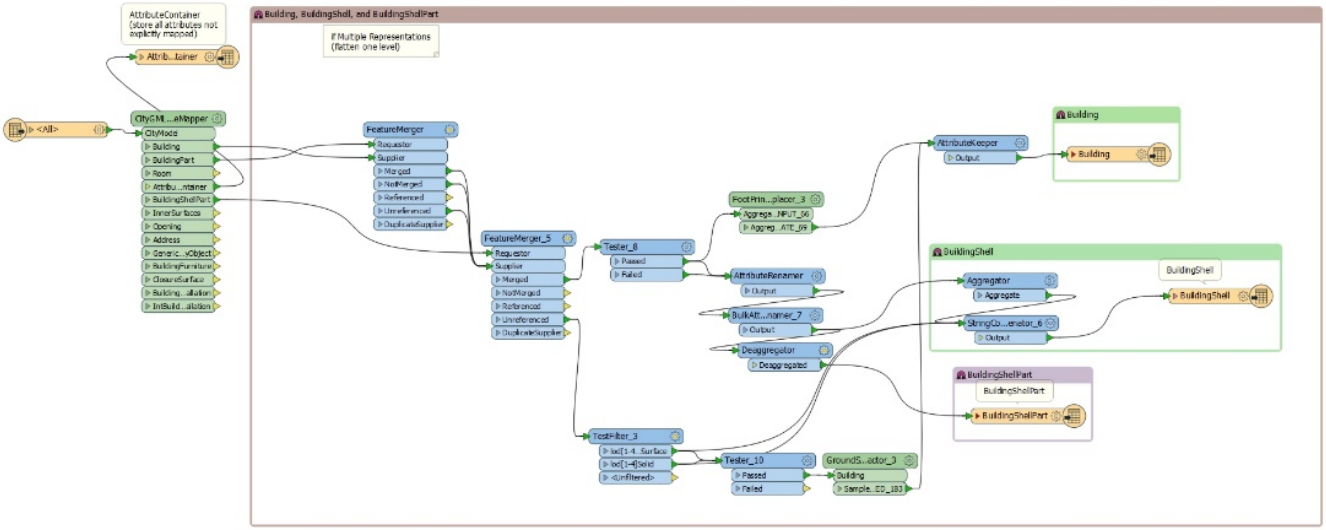

Figure 43. FME workbench to convert CityGML to I3S

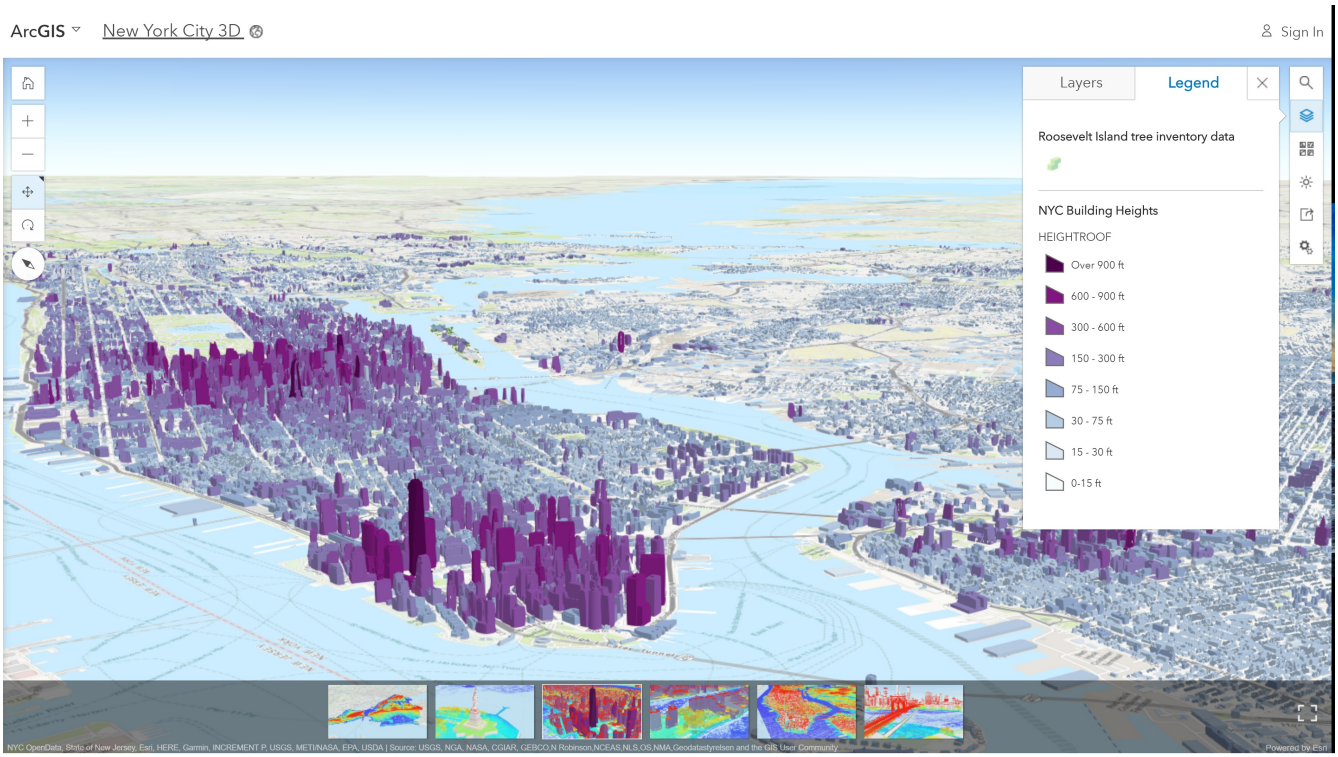

Figure 44. CityGML model of New York City rendered in Esri ArcGIS client using I3S

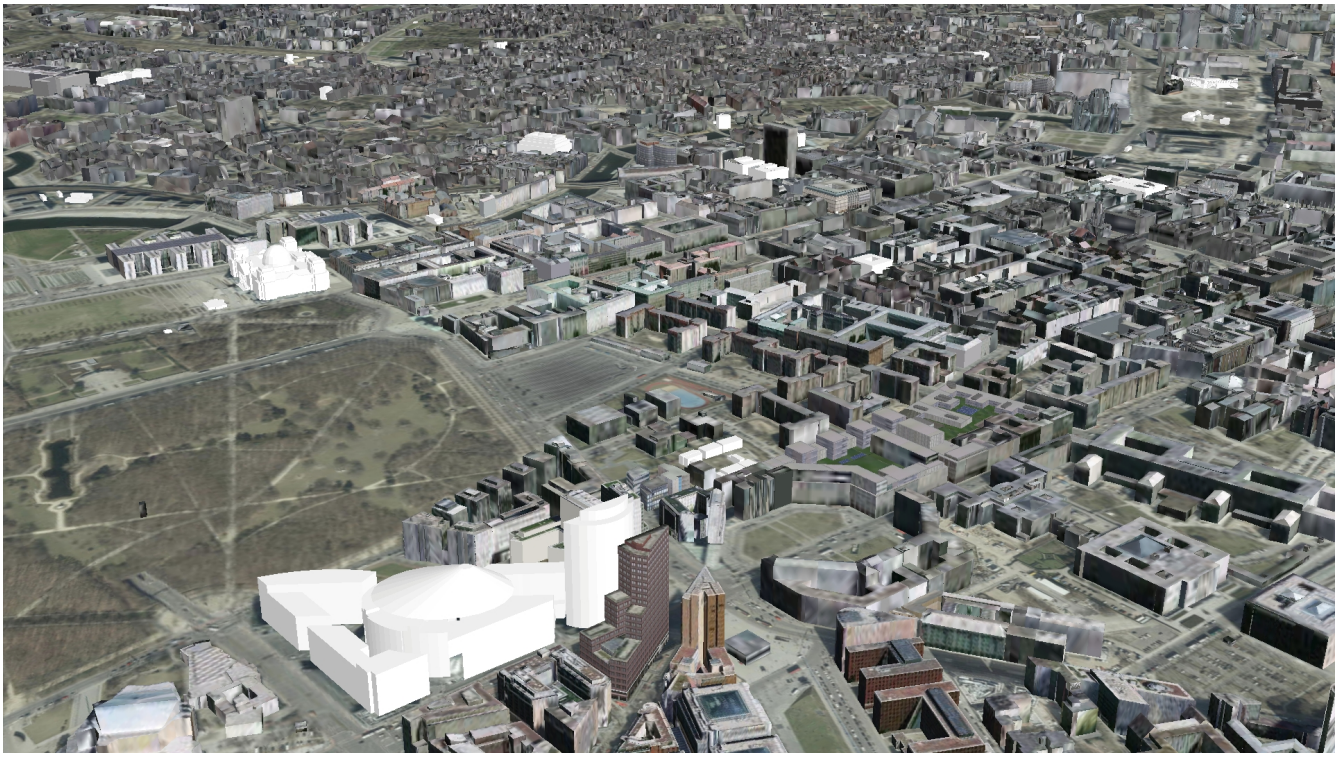

Figure 45. Fully textured CityGML model of Berlin rendered in Esri ArcGIS client using I3S

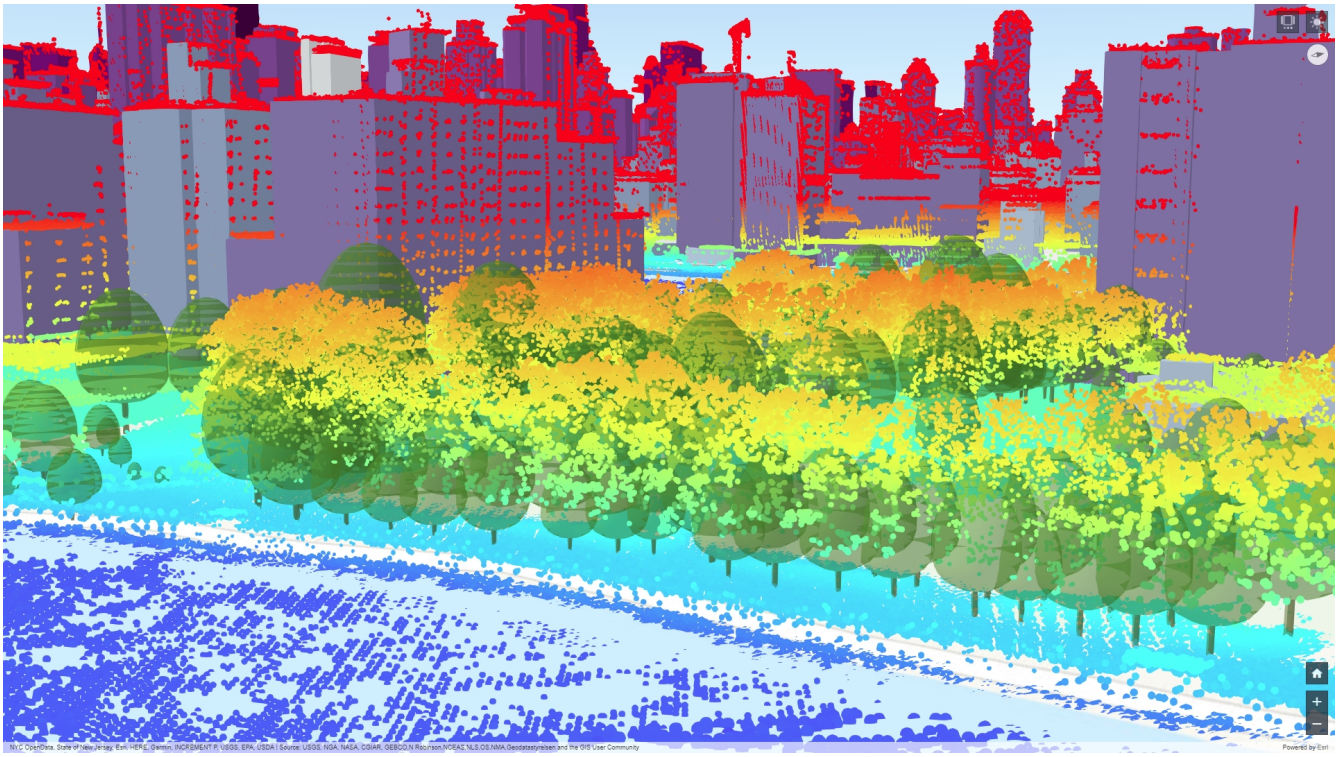

Figure 46. CityGML with Point Cloud of New York City rendered in Esri ArcGIS client using I3S

5.4.2. Experiment 3b: CityGML to I3S via 3DPS visualized in non-I3S Client - I3S interoperability using 3DPS

Geospatial content created in I3S was validated to be easily consumable through OGC's 3D Portrayal Service Standard (3DPS). The work involved serving I3S content, stored in an SLPK format, through a 3DPS enabled server and being able to consume it in a client application that is not native to I3S, to showcase interoperability. This experiment successfully demonstrated that 3DPS can serve as a broker between disparate implementations – that organize and optimize geospatial content, and,

client application, that may have limited direct support for such formats. In effect, 3DPS was demonstrated to benefit and facilitate client application consumption of well authored and organized content, such as I3S, with a low level of effort.

Because 3DPS does not define or endorse a particular content transmission format, but only specifies how geospatial 3D content is described, selected, and delivered, I3S content was able to be mapped and delivered through a 3DPS-enabled server. Such an implementation was successfully consumed by a prototype Cesium application, highlighting the applicability of 3DPS traits where it does not prescribe the content organization nor its representation format, but only provides a general framework for delivering 3D content.

Additional details of this work are provided in section 2b).

5.5. Experiment 4: CityGML ADE on GeoRocket Database to virtualcityPUBLISHER to Cesium

This experiment involved using a commercially available Streaming Engine called virtualcityPUBLISHER [http://www.virtualcitysystems.de/en/products/virtualcitypublisher], which was provided by virtualcitySYSTEMS GmbH. This server component provides data processing functionalities and access to spatial datasets in order to support streaming and visualization of large georeferenced 3D landscape and city models. A conversion pipeline included in virtualcityPUBLISHER was extended in order to meet the requirements of this experiment.

5.5.1. Requirements

The requirements assumed for this experiment were as follows:

- Support of 3D Tiles as means of communicating and transferring metadata and geometries to 3D clients, e.g. virtual globe frameworks based on HTML5 and WebGL;
- Support of GeoRocket as a data store for CityGML encoded 3D city models;
- Support of CityGML Application Domain Extensions (ADE) for encoding domain specific data schemas. Information from ADE enriched CityGML data sets must be processed correctly and made available to clients through 3D Tiles encodings. This includes customized feature types, properties, and complex attribute information.

5.5.2. Implementation Design

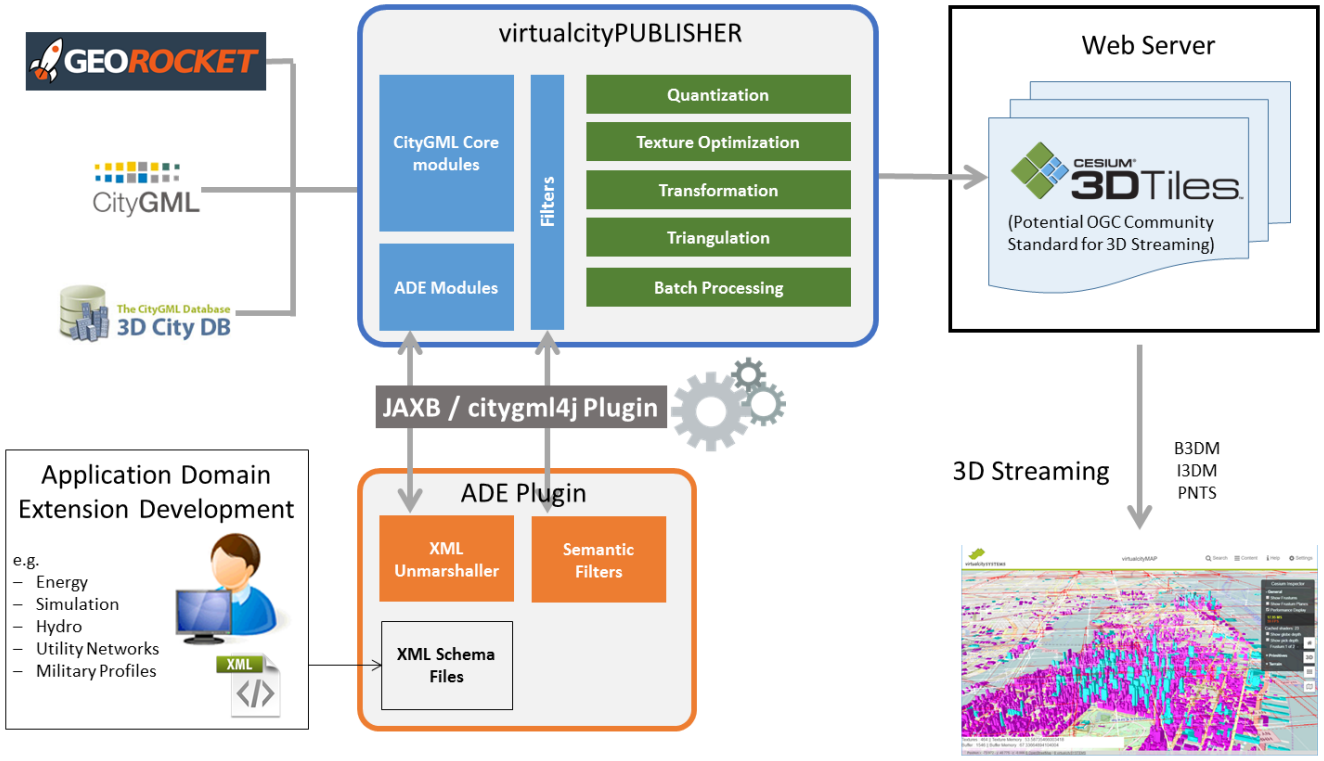

Figure 47. Component design of Streaming Engine "virtualcityPUBLISHER" with support for domain specific CityGML ADEs.

The virtualcityPUBLISHER component is a commercial authoring and data processing tool for making CityGML content available as ready-to-use 3D maps including functionalities for visualization, data extraction, and analysis. The 3D client uses Cesium as the rendering engine.

The included processing module takes CityGML as input, which can be managed in data stores such as 3DCityDB and GeoRocket. CityGML files can be used directly as well. Filters extract the information to be made available in a 3D map. Available filter settings include:

- LoD (CityGML can store multiple LOD geometries for the same object),
- Appearance (once again CityGML can store multiple visual appearances such as photo textures and infrared imagery in the same file),
- Feature types,
- Semantic surface types.

The following 3D model processing steps reduced the complexity of CityGML and made the content available for more efficient 3D rendering.

- Merging of geometries and batch assignment to vertex attributes. This step is important for achieving real-time rendering in 3D clients at high frame rates. Batching geometries by appearance reduces the number of required draw calls dramatically. The information on the model structure is moved to the B3DM wrapper so that individual objects can still be identified and highlighted.
- Triangulation (3D polygons are not supported by OpenGL).
- Transformation from native CityGML CRS (mostly map projections) to local Cartesian model coordinates. The CESIUM_RTC extension is used for increasing the coordinate accuracy, which

eliminated jitter. Model coordinates are stored as single precision coordinates, thus saving network bandwidth.

- Quantization of vertex coordinates for reducing file sizes.
- Adjusting texture image resolutions. Often the resolution of model textures is very high and varies significantly over the the entire data set. This step adjusts the texture resolutions so that pixels appear on the screen at a homogeneous geometrical size. This size is measured in texture pixels per meter;
- Texture atlas generation. This step greatly reduced the number of involved texture images and facilitates batching geometries by appearance.
- Tiling / creation of tile hierarchies for enabling streaming and dynamic refinement of models. Depending on the configuration, the entire data set can be uniformly subdivided into regular grid or multiple levels can be generated, each with different settings regarding generalization, level of detail and filter settings. A refinement strategy allows the showing of larger buildings first at higher levels, and smaller buildings with higher precision when the viewpoint comes closer.
- Encoding of properties and attributes. The batch stable structure as specified by B3DM is used for storing all object attributes.

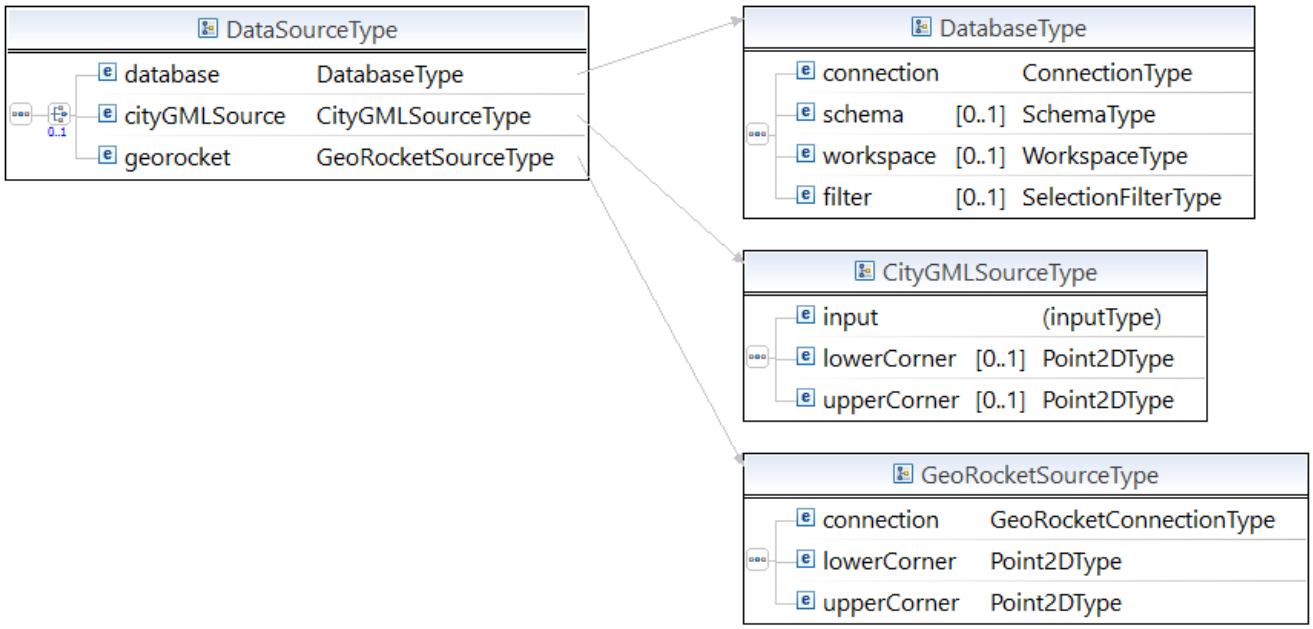

Figure 48. Part of XML schema design, which is used for configuring data sources of 3D city models in virtualcityPUBLISHER.

Data input can be configured as part of an XML configuration document. The XML complexType DataSourceType contains one of DatabaseType, CityGMLSourceType or GeoRocketSourceType. DataSourceType contains credentials for accessing an Oracle or PostgreSQL database, which contains a 3DCityDB profile.

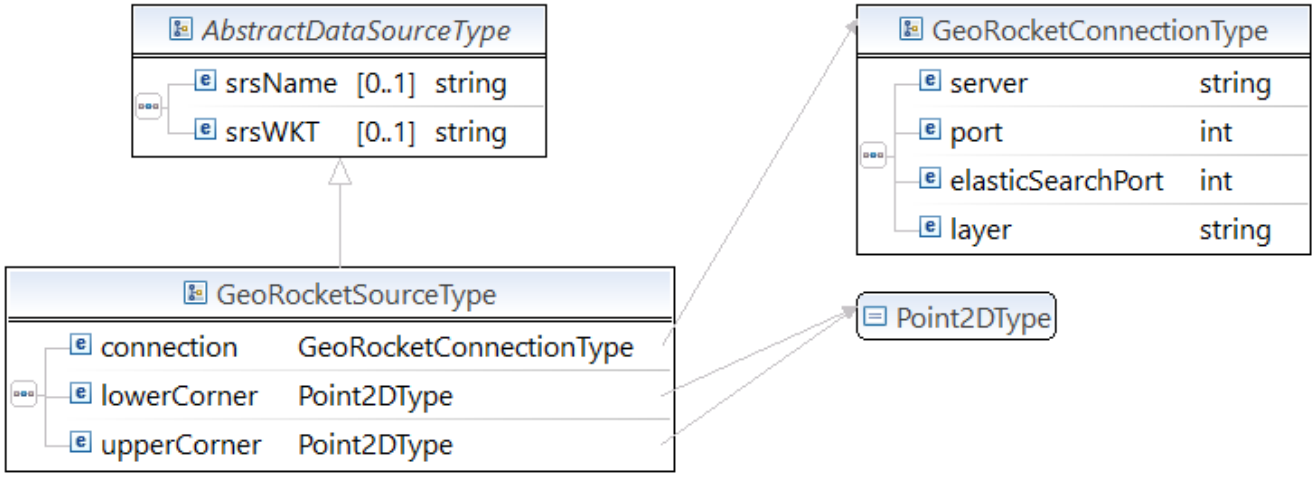

Figure 49. Part of XML schema design, which is used for configuring GeoRocket-specific connection parameters in virtualcityPUBLISHER.

The GeoRocketSourceType contains connection parameters for using the HTTP REST interface of GeoRocket. The data set referred to by the layer parameter must be restricted to a configured Bounding Box defined by the parameters lowerCorner and upperCorner, both defined in the local CRS of the layer. CRS information can be provided as an EPSG code or a full Well Known Text (WKT) description.

5.5.3. 3D Streaming

Results are stored as set of 3D Tiles JSON meta files, B3DM, I3DM, CMPT and PNTS files on a HTTP server. The actual network streaming and decoding of content is done by Cesium and is therefore not part of the virtualcityPUBLISHER Streaming server, which does not have a dedicated interface for 3D streaming.

As mentioned in a previous section, the 3D Tiles specification provides a general framework for spatial partitioning large data sets into tiles. Tiles are defined as 3D boxes containing 3D data encoded in one of the formats that have been developed around glTF. Hierarchical configurations are advisable in order to implement multiple LODs and to reduce the workload of the graphics pipeline. The spatial alignment and hierarchical nesting of these tiles is not specified and can follow concepts that take into account the spatial configuration of data sets such as clustering in densely populated areas.

However, in this implementation, the basic spatial alignment of the output is based on a global grid system with a rectangular subdivision of the globe in WGS84 coordinates. Individual hierarchical levels in this configuration can be configured with different settings regarding level of generalization, filtering, and appearance properties. For instance, the smallest tiles can be configured using the highest available CityGML LOD 3, all available CityGML feature types, and a high texture resolution, whereas tiles in the next higher level can be configured using a lower CityGML LOD 2, only building feature types, a lower texture resolution, and generalized geometries.

5.5.4. Processing of NYC data set

The CityGML data set from DoITT was used for demonstrating the 3D streaming capabilities of virtualcityPUBLISHER. The number of buildings was 1.5 million, and the overall size of CityGML files was 12.8 GB. These files were used as input for the offline conversion to 3D Tiles using the

following parameters:

Table 5. CityGML to 3D Tiles conversion settings in virtualcityPUBLISHER.

Tile Level	10	11	12	13	14	15
Tile Size [m] (approx.)	9783.9	4892.0	2446.0	1223.0	611.5	305.7
LOD	2	2	2	2	2	2
Size Filter	320	160	80	40	20	0
Generalization Tolerance [m]	320	160	80	40	20	0

Tiles were aligned to a global grid system, which started with a 2x4 subdivision of WGS84 space (90 x 90 degrees) at level 0 and a further 4x4 subdivision at each level. Six different levels were processed, each with different size filters and generalization tolerances. The size filter blocked objects smaller than the specified value from being processed whereas the generalization tolerance reduced the number of vertices and triangles by snapping together points that were within the specified tolerance value.

The resulting 3D Tiles file set comprised 14092 B3DM files with a total size of 3.02 GB. On an Intel i5-3470 CPU @ 3,2 Ghz, processing took 43m:46s.

Table 6. Breakdown of tile levels.

Tile Level	10	11	12	13	14	15
Number of B3DM Files	17	64	218	784	2862	10147
B3DM Data Size [MB]	0.1	1.2	13.4	88	665	2270

The resulting 3D Tiles layer was streamed and viewed on a mid-range graphics card (Radeon RX 460) at a continuous frame rate of 40-50 fps.

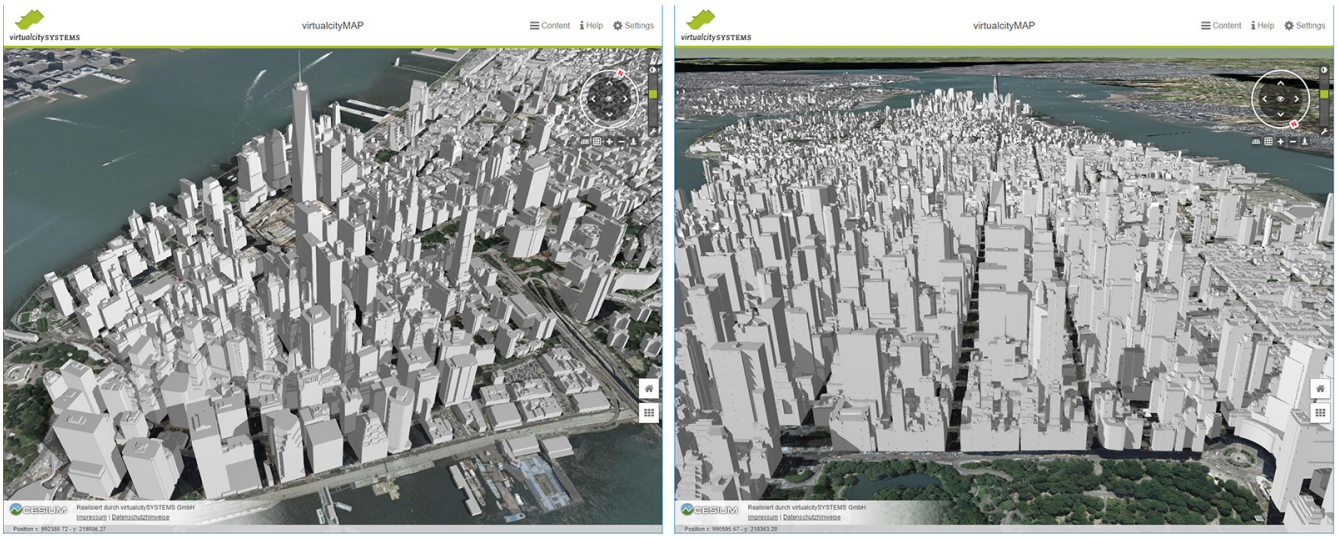

Figure 50. DoITT New York City model rendered in Cesium viewer. Objects in the background are loaded on higher tile levels and are geometrically simplified.

ADE Support

Application Domain Extensions (ADEs) for CityGML are by definition not part of the core specification. ADEs enable the implementation of domain specific feature types that are not supported by core CityGML. This extension feature makes CityGML very flexible. But on the other hand, it requires additional logic for processing workflows that consume ADE content. This logic provides information on the semantics of ADE elements.

In virtualcityPUBLISHER, support of ADEs is enabled by a plugin mechanism for the processing module. The full XML schema of the ADE must be available. This schema may define whatever is possible by the means of the ADE extension mechanism and may include custom feature types, additional attributes and complex documents attached to specific objects, parts, or sites. Plugins are provided as Java libraries (*.jar files), which must follow a specific architecture, i.e. classes must implement a service interface. The plugin represents all available XML elements from the ADE and is tasked with extracting information and filtering CityGML/ADE content of the registered ADE data types. Marshaling of XML schemas is done by JAXB. The feature type hierarchy is represented as data-type extension to citygml4j [https://github.com/citygml4j/citygml4j], an open source Java API for CityGML.

5.6. Experiment 5: CDB Performance

The OGC CDB standard [http://www.opengeospatial.org/standards/cdb] defines a storage structure, and associated consistent file naming conventions, for the storage of a synthetic environment suitable for realistic computer simulations. The CDB Performance experiment consisted of implementing an approach for generating 3D Tiles encoded tilesets from a CDB structured data store. More specifically, it defined an approach for generating terrain and models tilesets from content accessible in such a data store. The following subsections provide additional details.

5.6.1. CDB Overview

A CDB structured data store is physically arranged on disk into the following top level directory structures:

1. **Metadata** contains a set of XML metadata and controlled vocabulary files that are global to the data store.
2. **GTModel** contains geotypical models, generic models that are defined once in the CDB and are intended to be rendered in multiple places throughout the data store (contrast with geospecific or GS models). They aren't intended to represent specific objects, but simply a typical representation of an object type such as a tree.
3. **MModel** contains Moving Models, which don't have a fixed location and are intended to be dynamically placed and moved throughout a simulation. An example is an automobile or aircraft.
4. **Tiles** contains tiled datasets.
5. **Navigation** contains global navigation datasets.

This experiment made use of **Metadata** and **Tiles** directories from a CDB data store. The **Metadata** folder contained various information used in the generation of both the 3D Tiles Terrain tileset and

the 3D Tiles Models tileset. The **Tiles** directory content was used to generate both Terrain and Models tilesets.

The vast majority of content was in the tiled datasets. The experiment made use of the tiled datasets listed below.

1. Elevation
2. Imagery
3. GSFeature
4. GSModelGeometry
5. GSModelTexture
6. GSModelDescriptor

The CDB standard provides guidance on storing many other datasets beyond those used for this experiment. Future work will develop representations of the other datasets in suitable 3D Tiles formats. For example, CDB geotypical models would be represented in 3D Tiles "instanced 3D model" (I3DM) format.

CDB partitions the world into 1-degree by 1-degree cells. Within each cell, the tiled datasets are organized into a LOD hierarchy. The experiment's tileset-generation approach used the LOD hierarchy defined by the CDB as the starting point of the tileset hierarchy. One general finding was that tilesets generated strictly using the CDB LOD hierarchy resulted in tiles that were too large for rendering on the web using WebGL and too large for rendering on Android mobile devices. The CDB LoD hierarchy was further refined to reduce individual tile sizes.

5.6.2. Terrain Generation

The 3D Tiles terrain tileset was generated using the CDB Elevation and Imagery datasets. The Elevation dataset consisted of elevation data and supporting bathymetry data. Both datasets were encoded as TIFF files. The pixels in the TIFF file represented a regular grid of elevation samples. The physical distance between each pixel sample is defined by the CDB LOD. The implementation in this experiment generated vertices, normals and texture coordinates simply by iterating through the pixels in the TIFF file. The texture coordinates map to the imagery pixels found in the corresponding LOD from the Imagery dataset. The individual tiles were generated as 3D Tiles B3DM format. The CDB Imagery dataset was encoded using JPEG 2000 rasters, which were converted to JPEG for use as 3D Tiles textures.

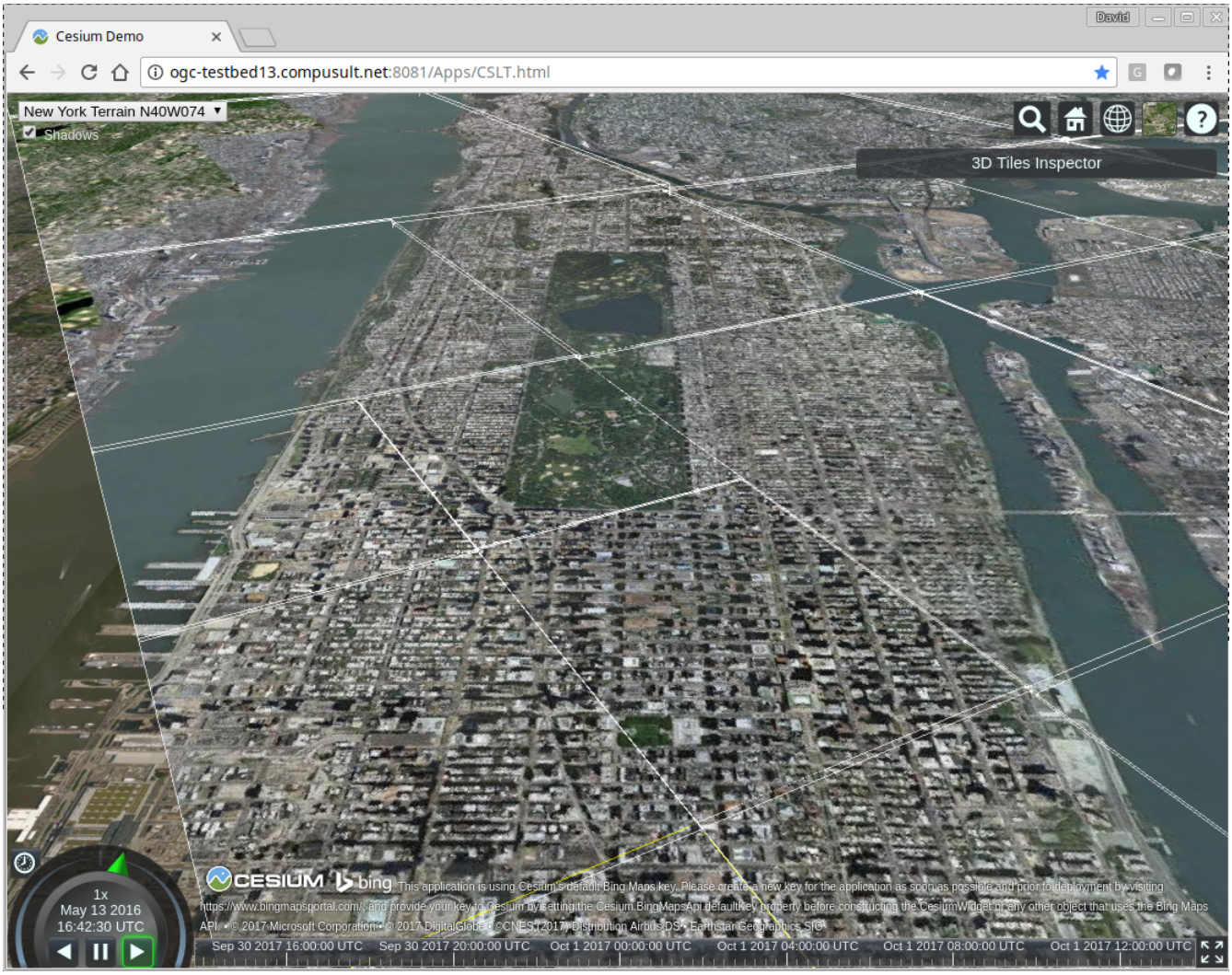

Figure 51. Outline of 3D Tiles terrain boundaries from CDB data over New York City.

Performance Optimizations

To improve the performance of 3D Tiles clients when rendering the generated tilesets, several optimizations were performed to reduce tile sizes. When viewing the terrain at a scale where the imagery was not pixelated, the corresponding elevation LOD resulted in an unnecessarily dense terrain mesh. As an optimization, the imagery at LOD *N* was matched with elevation data from LOD *N-2*. The terrain at LOD *N-2* provided a reasonable amount of visual fidelity to show with the imagery at LOD *N*. The CDB TIFF files were 1024 by 1024 pixels in size. Terrain meshes generated from rasters of this size were found to be too large for acceptable performance on the Web and on mobile devices. Each 1024x1024 was further subdivided into 256x256 rasters prior to generating the 3D Tiles tile. This resulted in tiles with approximately 8000 triangles.

In order to achieve sufficient precision to eliminate jittering artifacts when rendering, the Cesium RTC extension was used to generate vertices that were relative to local coordinates.

Visual Optimizations

To eliminate visual seams between adjacent tiles a "skirt" was generated along the edges of each tile. The vertices of the skirt were generated using the same coordinates as the existing edge vertices with a slightly lower elevation.

Geometric Error

The 3D Tiles specification uses the concept of geometric error to help clients determine a suitable level of detail to display based on the current view. One of the findings was that the specification could be improved by providing examples of how to choose or calculate this geometric error. Ultimately "trial-and-error" was employed, using the Cesium client to find a geometric error that worked the best with the implementation.

5.6.3. Models Generation

The 3D Tiles models tileset was generated using the **GS*** datasets within the CDB data store. CDB stores 3D models using the OpenFlight [http://www.presagis.com/products_services/standards/openflight/more/openflight_specifications/] format. The implementation in this experiment generated batched-3D-models (B3DM) formatted tiles from the OpenFlight models.

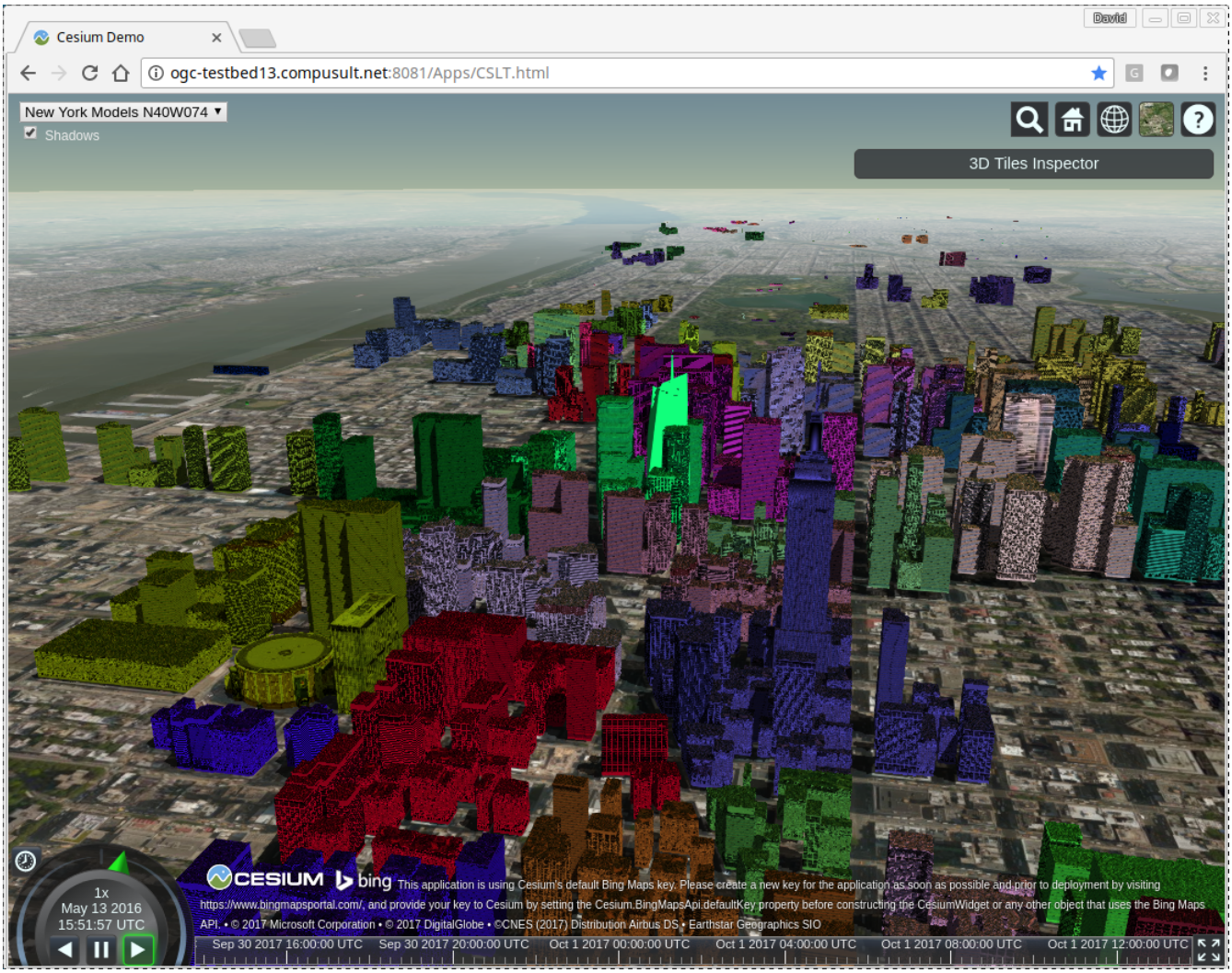

Figure 52. Rendering of 3D Tiles generated from CDB data over New York City (the models are colorized for clarity).

Model locations were determined from Shapefiles [http://desktop.arcgis.com/en/arcmap/10.3/manage-data/shapefiles/what-is-a-shapefile.htm] in the **GSFeature** dataset. The model geometry, textures and supporting information were found in the **GS_ModelGeometry**, **GT_ModelTexture** and **GS_ModelDescriptor** datasets. The filesystem locations of the model artifacts were determined from Shapefile dBase table (.DBF) [http://desktop.arcgis.com/en/arcmap/10.3/manage-data/shapefiles/shapefile-file-extensions.htm] attributes and the CDB file and directory naming schemes.

The geospatial model location was determined from the Shapefiles in the **GSFeatures** directory and the elevation found in the **Elevation** dataset.

CDB Metadata

The CDB specification is still evolving with respect to metadata, and the sample CDB datasets were somewhat lacking. CDB models are assigned feature codes, but these codes were deemed too broad to maximize their value. For example, the vast majority of buildings encountered are simply categorized as FACC **AL015**. Part of the CDB naming scheme is based on FACC codes; however, for rendering purposes the "meaning" of the FACC codes is not significant.

A possible improvement to the CDB metadata support could be to define an additional model DBF attribute that references an arbitrary metadata document such as a NAS document (see the companion "OGC Testbed-13: NAS Profiling ER", OGC Document # 17-020). The FACC codes and naming scheme could remain as-is simply for the purpose of defining the CDB directory structure and naming scheme.

Performance Optimization

Depending on the number and density of the models found in the native CDB tiles, the resulting 3D Tiles encoding could be too large for rendering on the web or on mobile devices. To mitigate this performance issue, the native CDB tiles were iteratively subdivided to reduce the number of the models in any one tile.

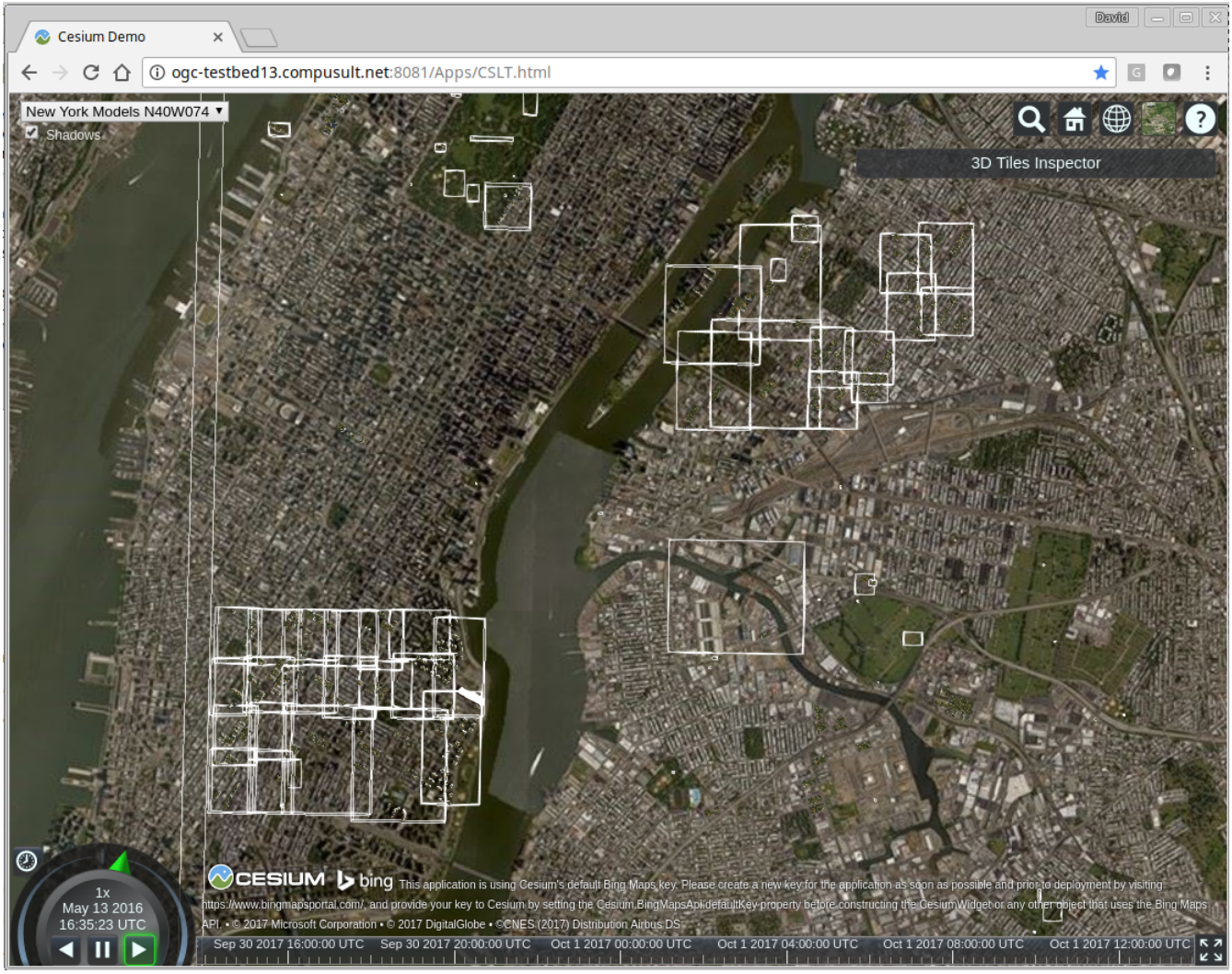

Figure 53. Outline of 3D Tiles boundaries from CDB data over New York City. Clusters of small tiles appear where the density of models is higher.

5.7. Experiment 6: 3D Tiles Visualization in GNOSIS

5.7.1. Overview

The following general approach was taken in this experiment:

- Support for 3D Tiles and glTF 1 was implemented within the GNOSIS 3D visualization engine.
- Since all the streaming engines that GNOSIS connected to in this experiment implemented glTF version 1 (vs. the now-available version 2 [https://www.khronos.org/news/press/khronos-releases-gltf-2.0-specification]), this is what was implemented in GNOSIS.
- Typically, whether in a 2D or 3D view, GNOSIS pre-determines a fixed set of tiles to be displayed. This list can include a mix of different level of details in a scene based on a geospatial quad-trees.
- For 3D Tiles, a geometric error determines instead whether a tile should be further refined. Because this approach is considerably different from simply requesting the required tiles for a given 3D view without having to first look at parent tiles, a custom tile fetching and caching mechanism had to be implemented. The capability to request appropriate children 3D Tiles based on a geometric error being too large was also integrated.

glTF implementation results

Some sample models that were visualized while implementing a GNOSIS glTF parser and renderer appear below.

Figure 54. Sample gITF models

5.7.2. Datasets visualized

The following subsections show visualizations of datasets.

New York Flight Safety CDB dataset converted to 3D Tiles

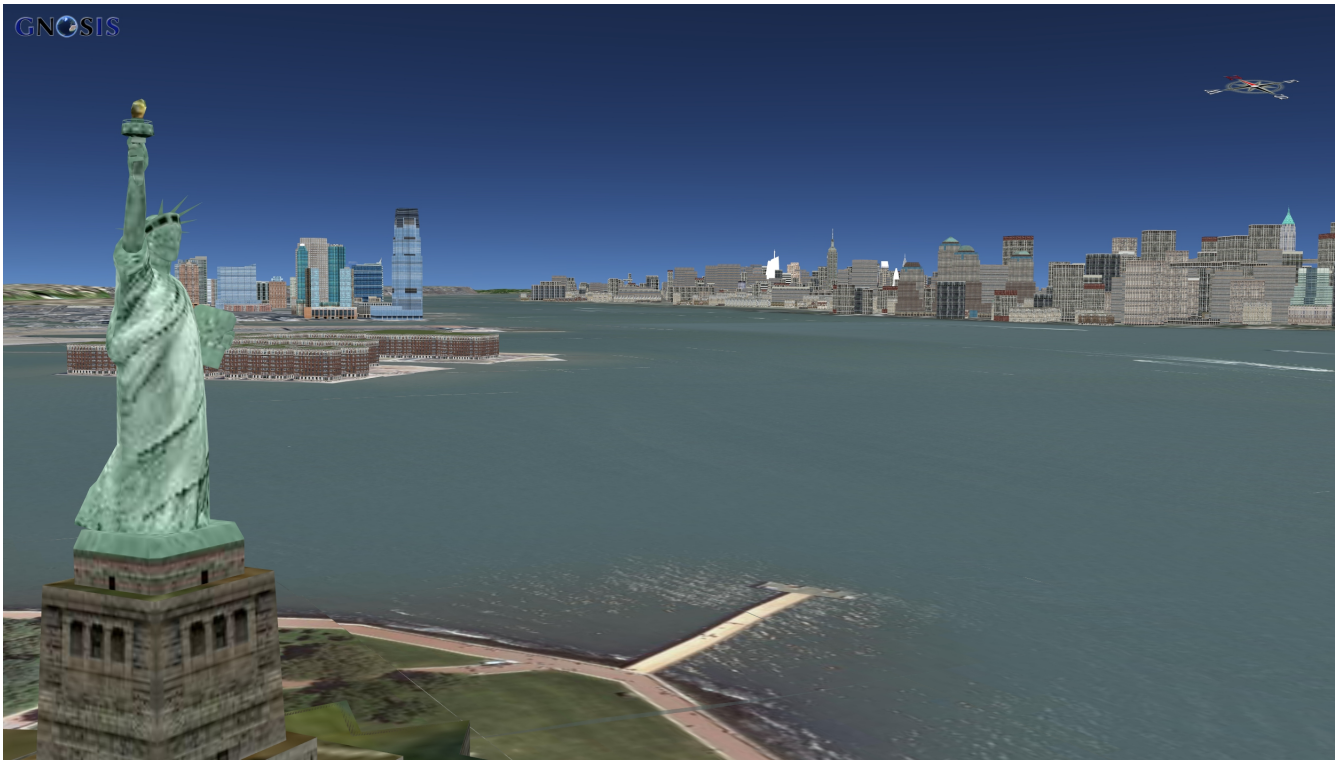

Figure 55. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

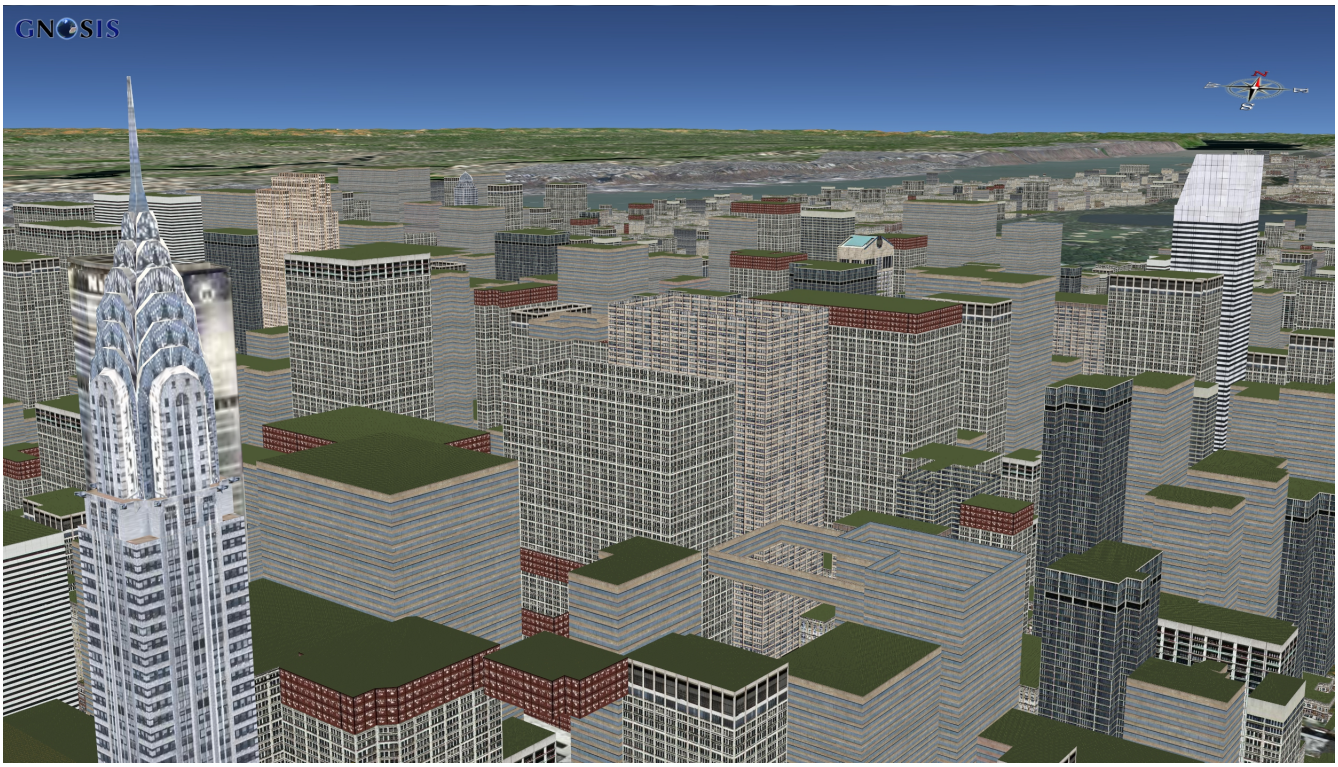

Figure 56. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

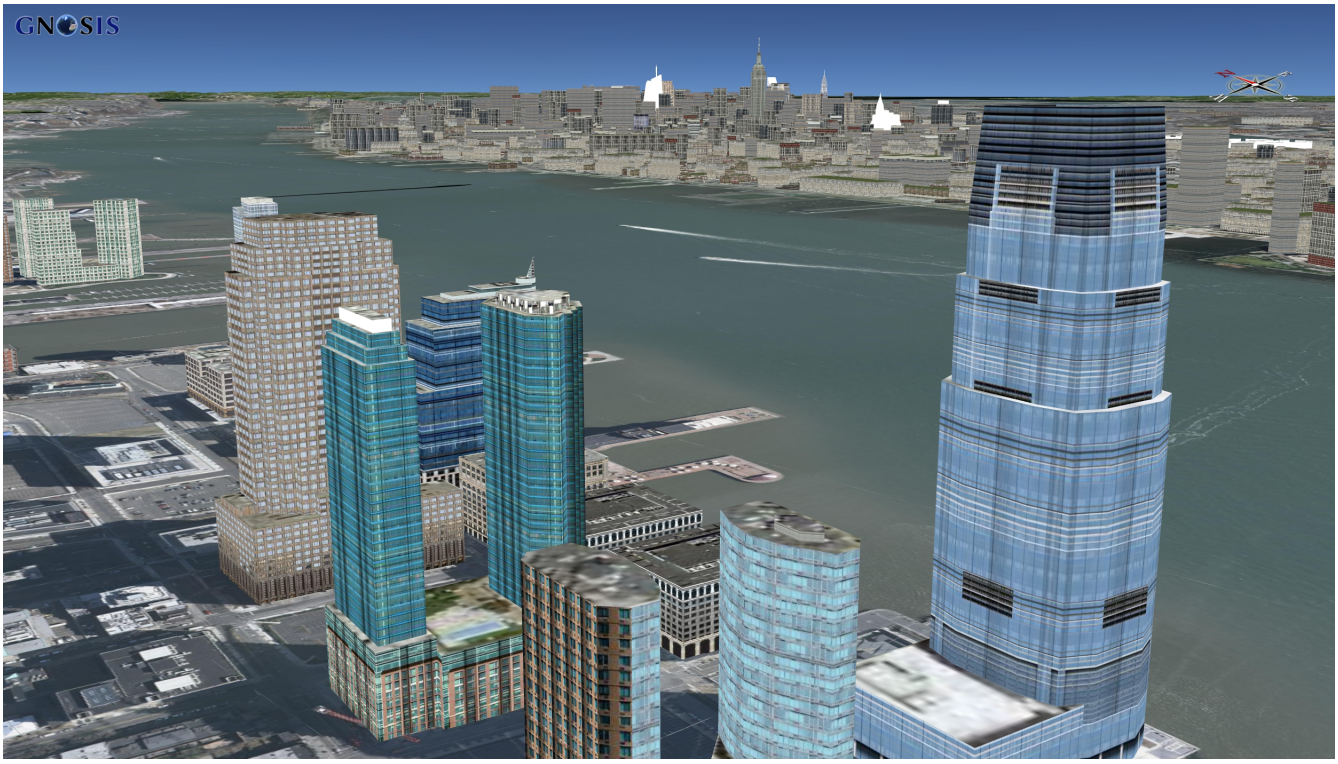

Figure 57. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

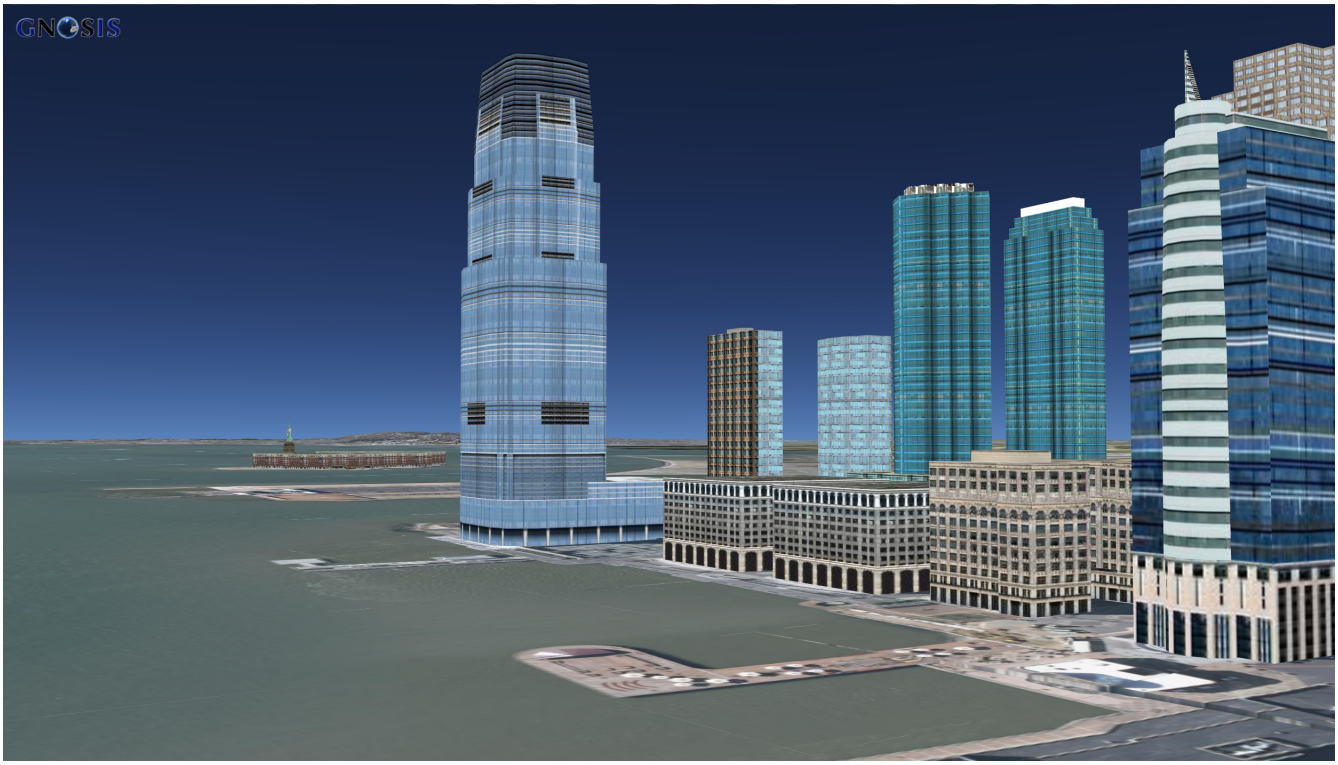

Figure 58. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

Figure 59. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

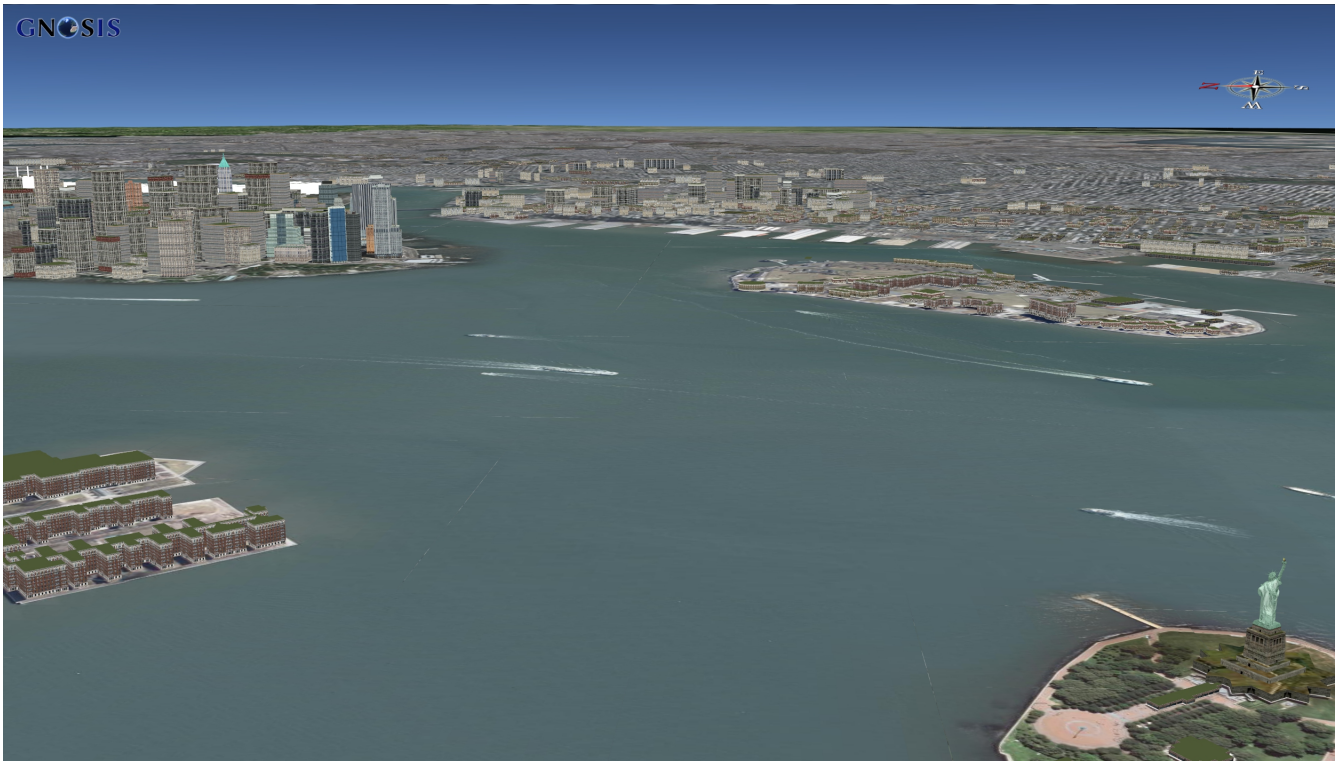

Figure 60. Visualization of Flight Safety CDB converted to 3D Tiles over Manhattan.

Other datasets

Attempts were made at visualizing other 3D Tiles datasets, including:

- CityGML dataset converted to 3D Tiles via virtualcitySYSTEMS tool
- CityGML dataset converted to 3D Tiles via AGI tool

NOTE

Because 3D Tiles and glTF have some variability in how geometry and other aspects can be defined (e.g. a large number of glTF options and settings), additional work will be required to achieve compatibility with the particularities of the additional datasets that were to be visualized. For example, the orientation of the 3D axes can be specified in different manners; geometry can be specified with or without indices (allowing to efficiently re-using vertex data; the VCS tool produced non-indexed geometry. Support for non-indexed geometry was added.). Despite the 3D Tiles showing properly for the output of Compusult's CDB conversion tool; in the last experiments the other data sets were not showing up oriented or positioned properly.

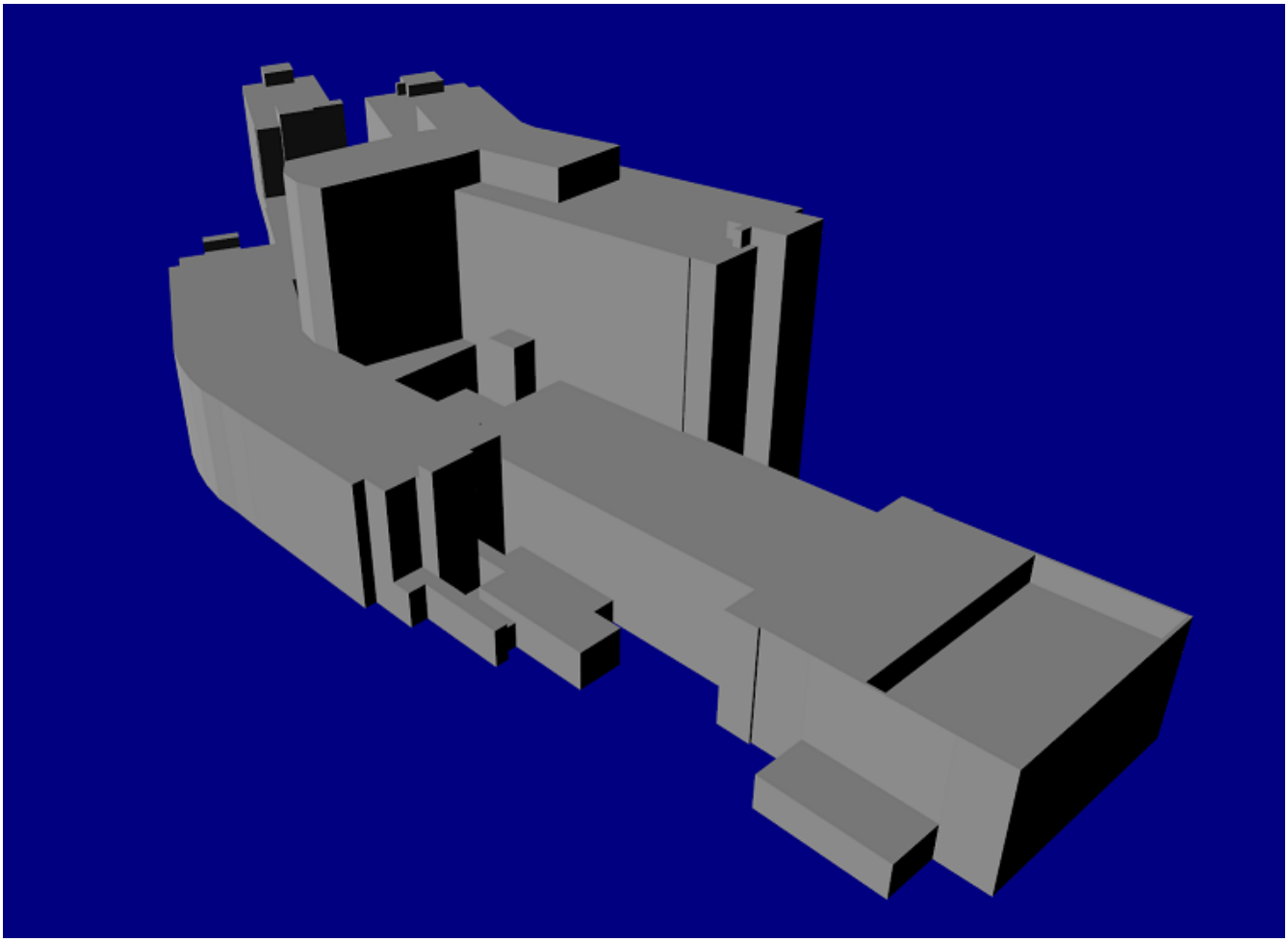

Figure 61. 3D Buildings from 3D Tiles converted from CityGML by virtualcitySYSTEMS.

5.7.3. Results and findings

The results and findings are as follows:

- The approach of embedding both geometry and imagery (textures) within the model (Batched

3D Model — B3DM) suffers from the fact that the imagery is often of significantly higher resolution than the terrain elevation data. As such the terrain data must be repeated over and over again for higher resolution imagery tiles. It also does not make it possible to drape different layers on top of the same elevation data (e.g. vector layers or multiple imagery sets). A representation specific to terrain would be beneficial in addressing this. However a standard way to represent the terrain and imagery separately was not available.

- Disk caching had to be implemented in order to achieve reasonable performance, as the quantity of data to be transferred for the B3DM tiles was quite significant. One reason that might explain this is that unlike CDB, the different 3D Tiles produced from the CDB did not share the same texture or model data, and as such a lot of information is duplicated. Producing tiles referencing external textures would offer a way to mitigate this problem.
- Requesting tiles from Compusult's service had to be done in multiple threads as requesting them one at a time would take too long.
- The proper way to handle replacement and refinement of 3D Tiles might not yet be achieved. For example, how to decide what to display while waiting for additional children tiles while 1 or more is already ready. More research is required to better understand how this should be done and the implementation should be adjusted accordingly.
- Despite using the binary format, parsing JSON is still more costly than a simple binary format. The JSON parser used by the implementation might benefit from further optimization, but nothing will match a simple binary format in terms of loading performance.
- Some shaded borders were noticed around the 3D Tiles output from the Compusult's tool. It is not clear whether this an issue in the GNOSIS visualization or the conversion to 3D Tiles.
- The proper selection of tiles based on geometric error seemed to require different settings for models than for terrain. Despite this, the Statue of Liberty did not display at full detail even up close. The implementation of the geometric error might need correction, and/or it might be an issue with the CDB conversion tool.
- More optimization work is required to really achieve optimal performance.

NOTE

Support for I3S has also been initiated within GNOSIS. An I3S dataset from New York City was used for this experimentation. However there was not enough time to reach a working demonstration, and work to support I3S will be on-going. I3S is very similar to the 3D Tiles approach, in that it does not dictate a tiling scheme and also relies on a geometric error to determine when geometry is 'good enough'. The client must still go through the low resolution nodes before getting to the desired tiles. A client supporting both I3S and 3D Tiles can therefore re-use a significant portion of code.

5.8. Experiment 7: CDB to GNOSIS

5.8.1. Overview

The following general approach was taken in this experiment:

- Support for CDB was implemented within the GNOSIS visualization engine.

- The implementation focused on visualizing the OpenFlight models, imagery as well as the 3D terrain model.
- The positional information for geospecific models was retrieved from Shapefiles.
- Vector features were also supported, although support styling and materials was left for future work.
- The terrain elevation and imagery were pre-processed to the GNOSIS tiling scheme and representation so as to improve performance.
- Also pre-processing the shapefiles would gain a performance advantages, but this approach was not used due to time constraints.

5.8.2. Visualization of Camp Pendleton Sample database

Sample visualizations of the Camp Pendleton database are shown below.

Figure 62. 3D Terrain and imagery from Camp Pendleton sample CDB database.

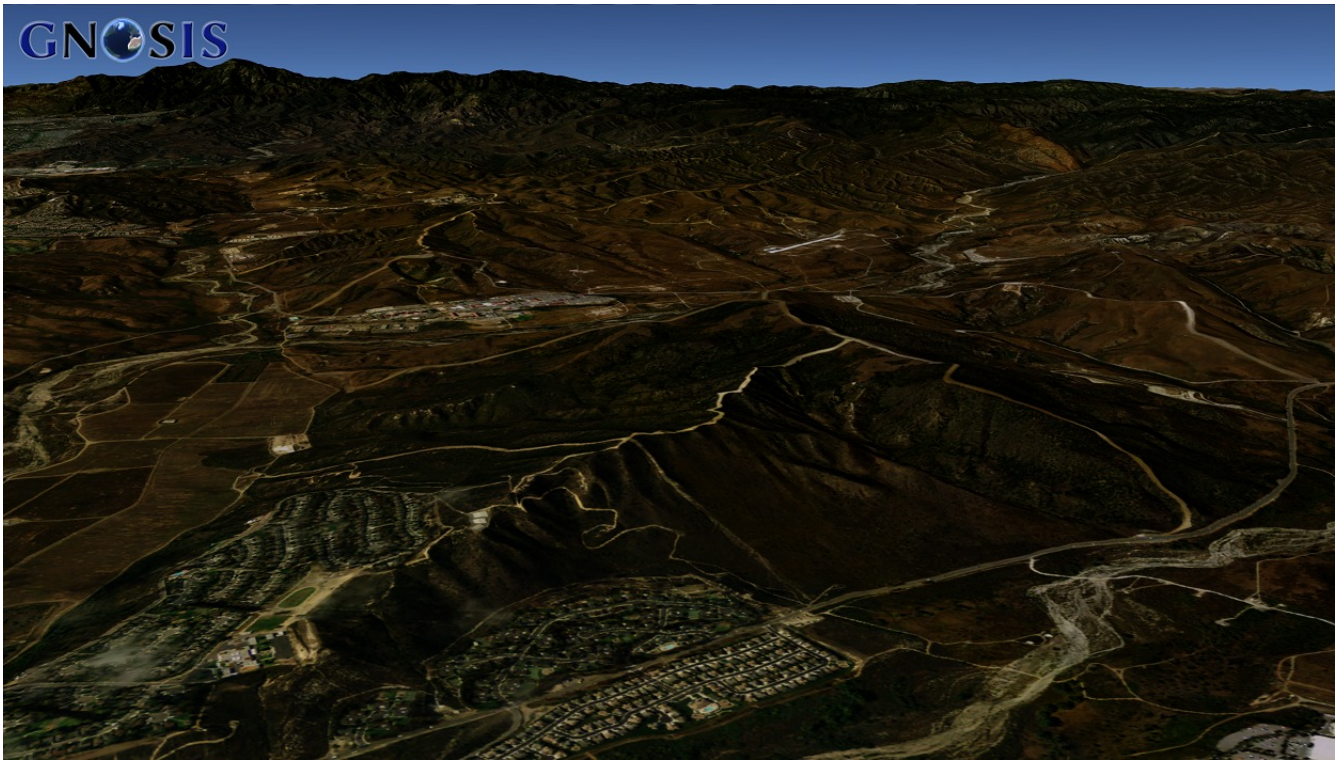

Figure 63. 3D Terrain and imagery from Camp Pendleton sample CDB database.

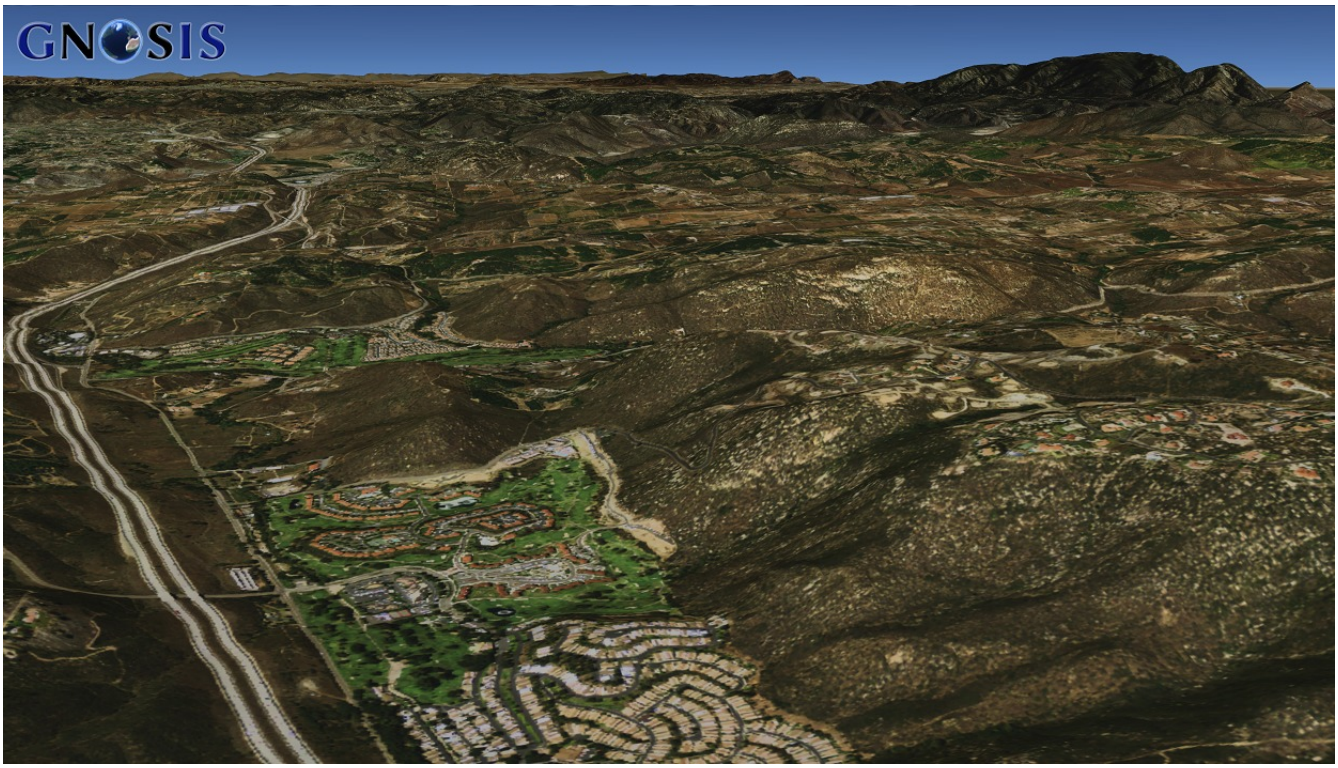

Figure 64. 3D Terrain and imagery from Camp Pendleton sample CDB database.

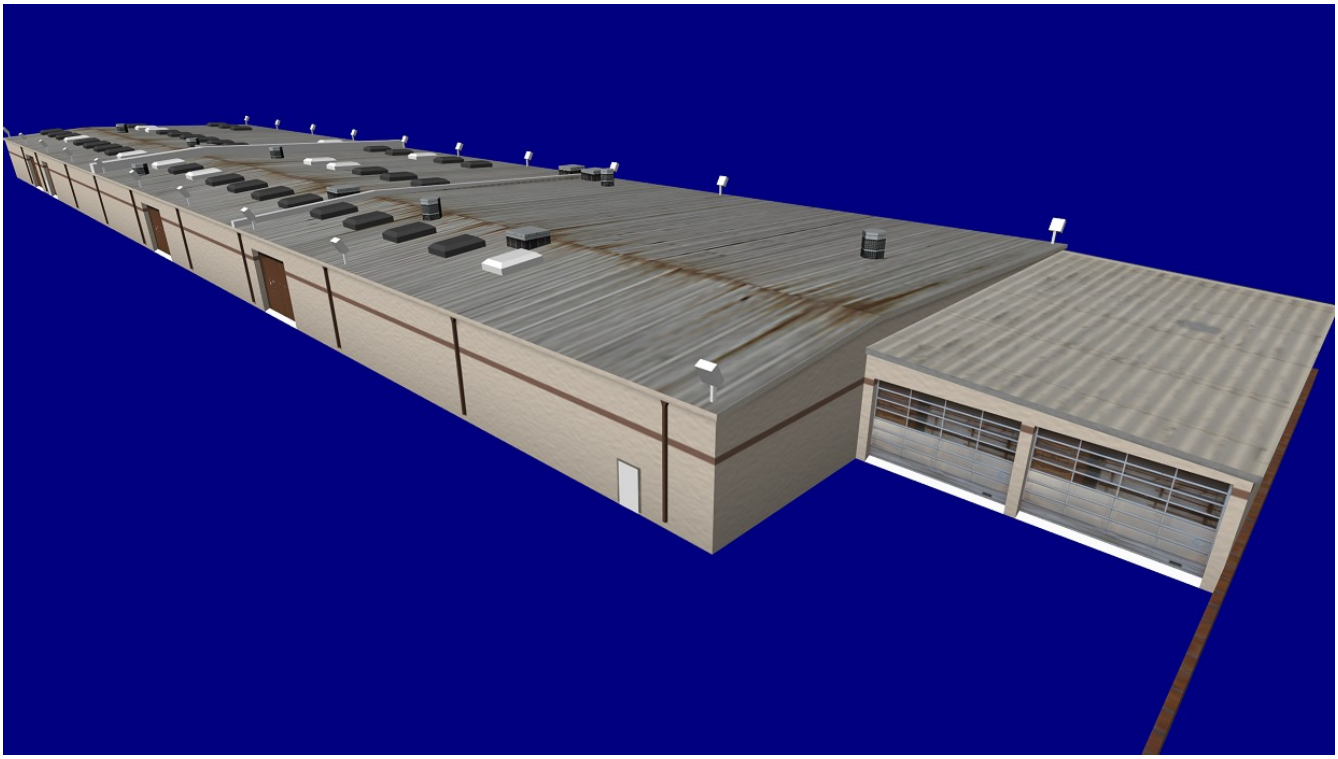

Figure 65. 3D Terrain and imagery from Camp Pendleton sample CDB database.

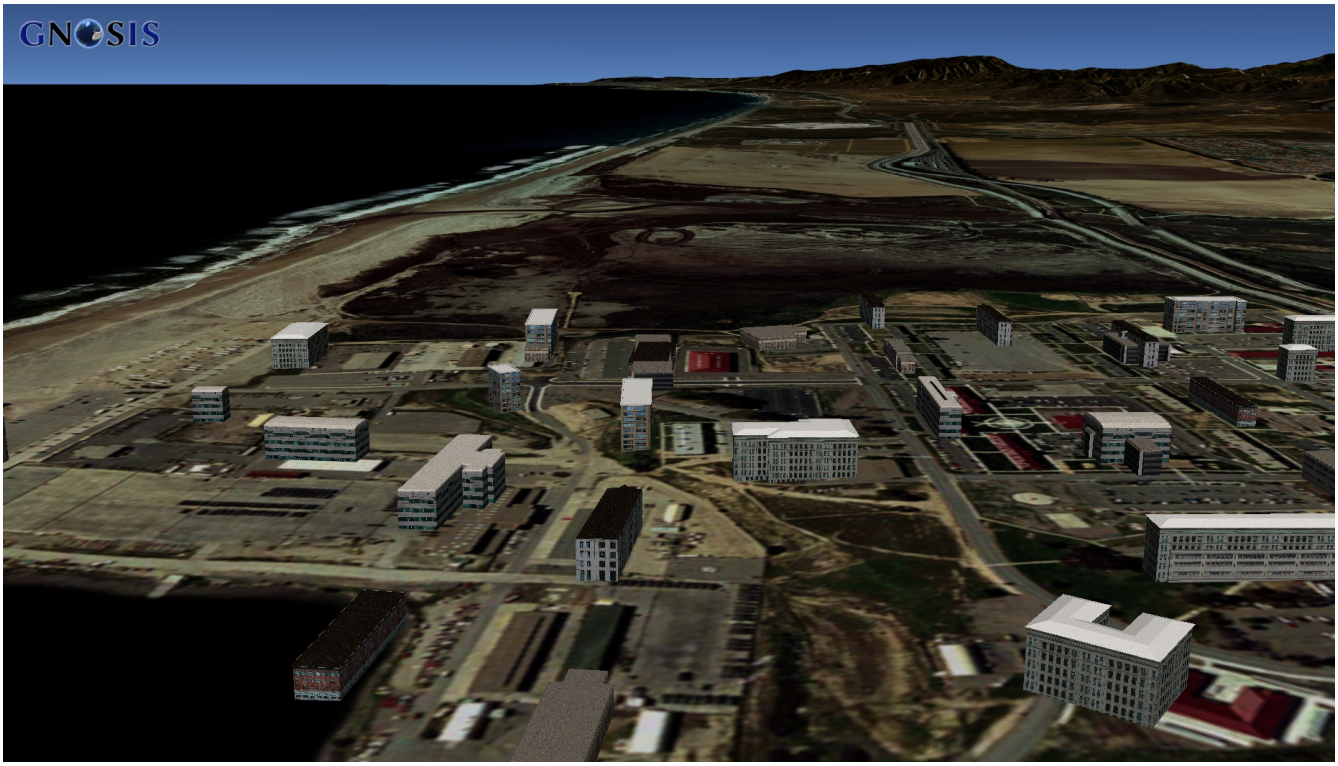

Figure 66. 3D Terrain and imagery from Camp Pendleton sample CDB database.

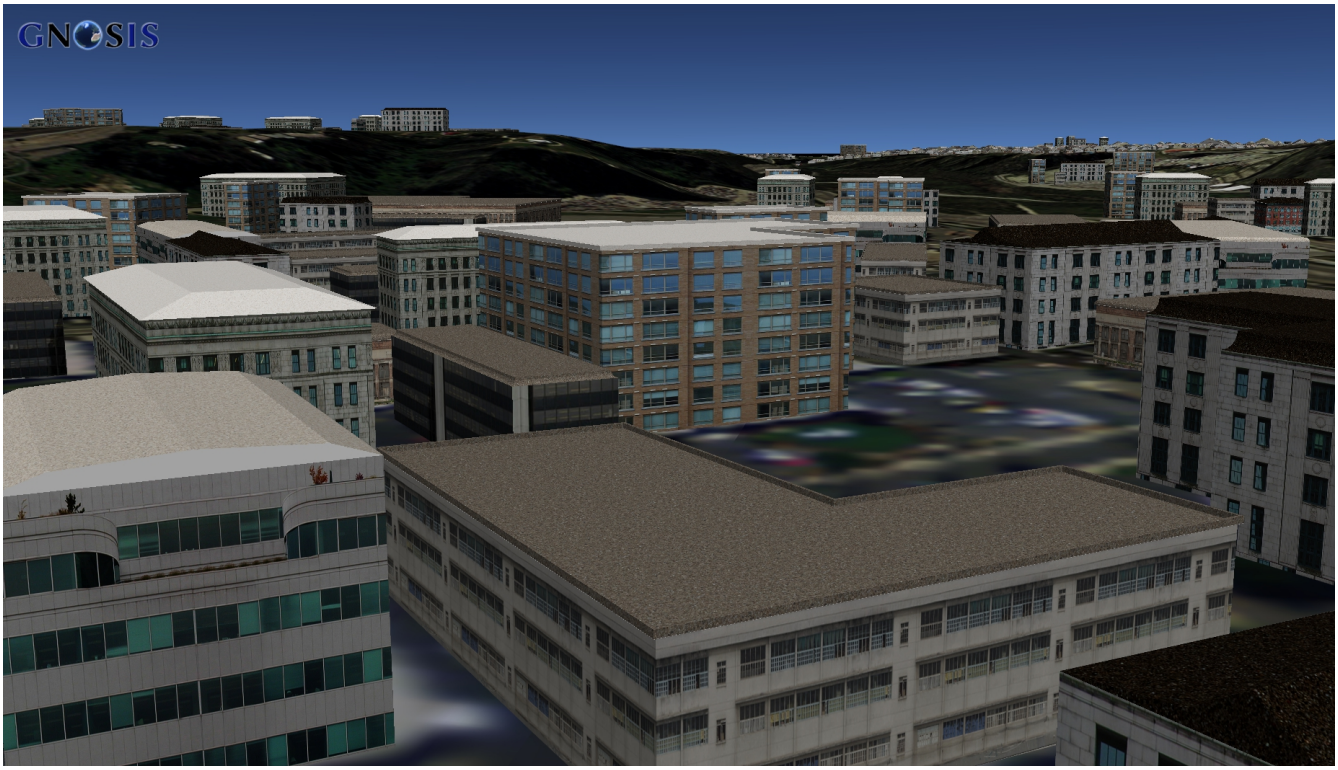

Figure 67. 3D Terrain and imagery from Camp Pendleton sample CDB database.

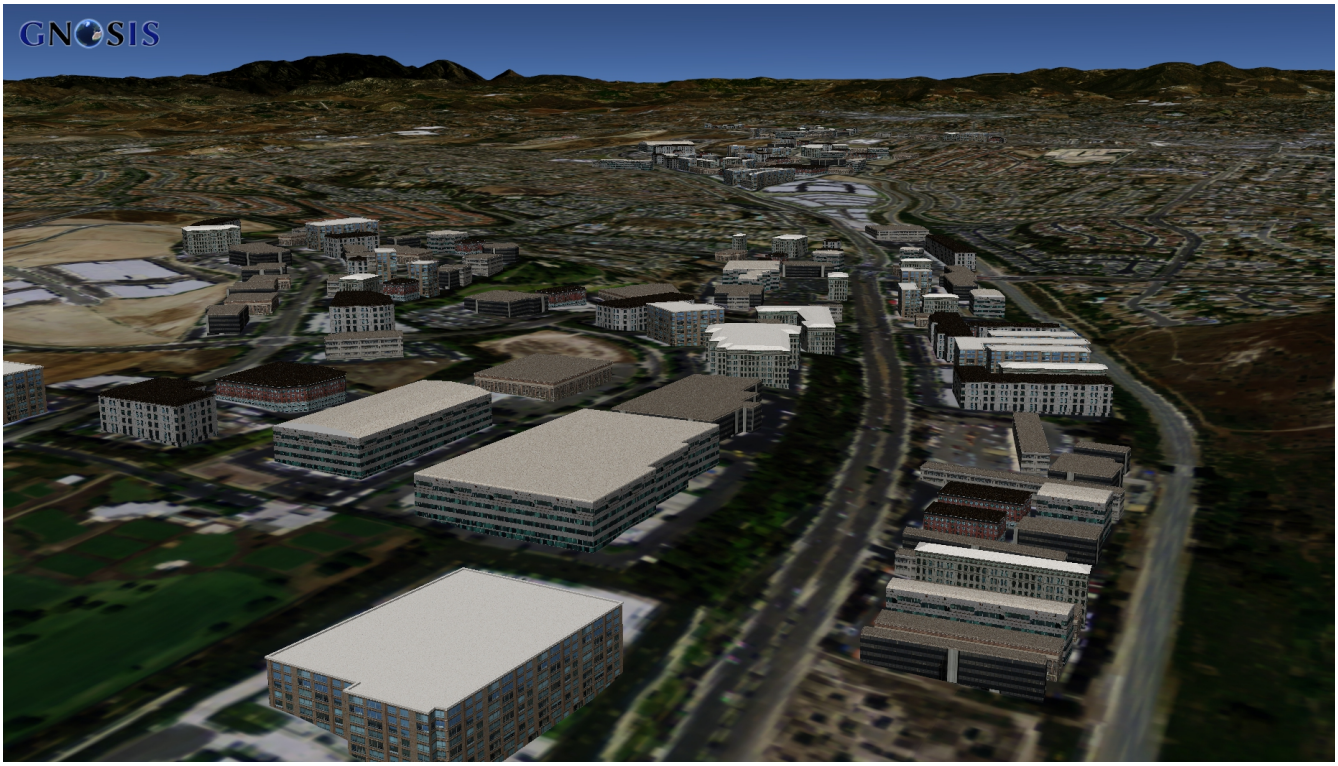

Figure 68. 3D Terrain and imagery from Camp Pendleton sample CDB database.

5.8.3. Visualization of Flight Safety CDB over Manhattan

Sample visualizations of the Flight Safety CDB database are shown below.

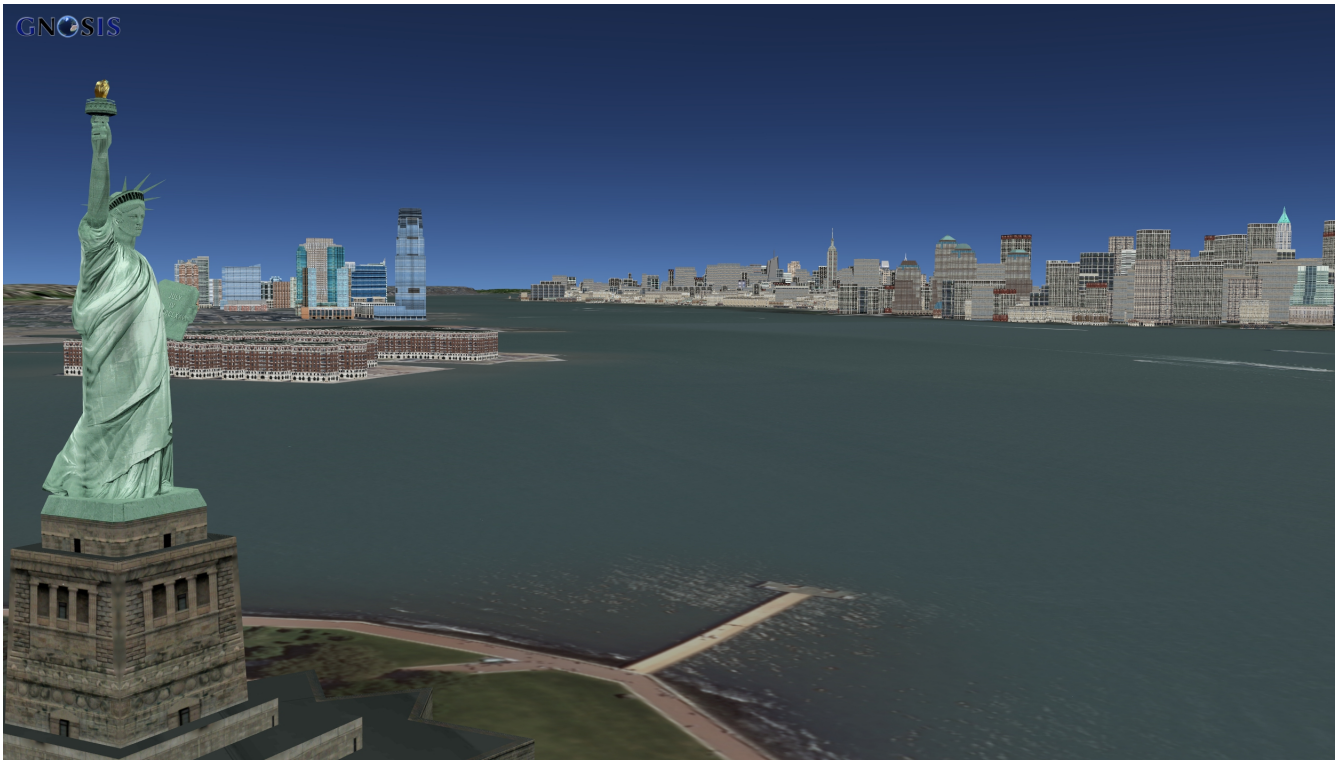

Figure 69. Visualization of Flight Safety CDB over Manhattan.

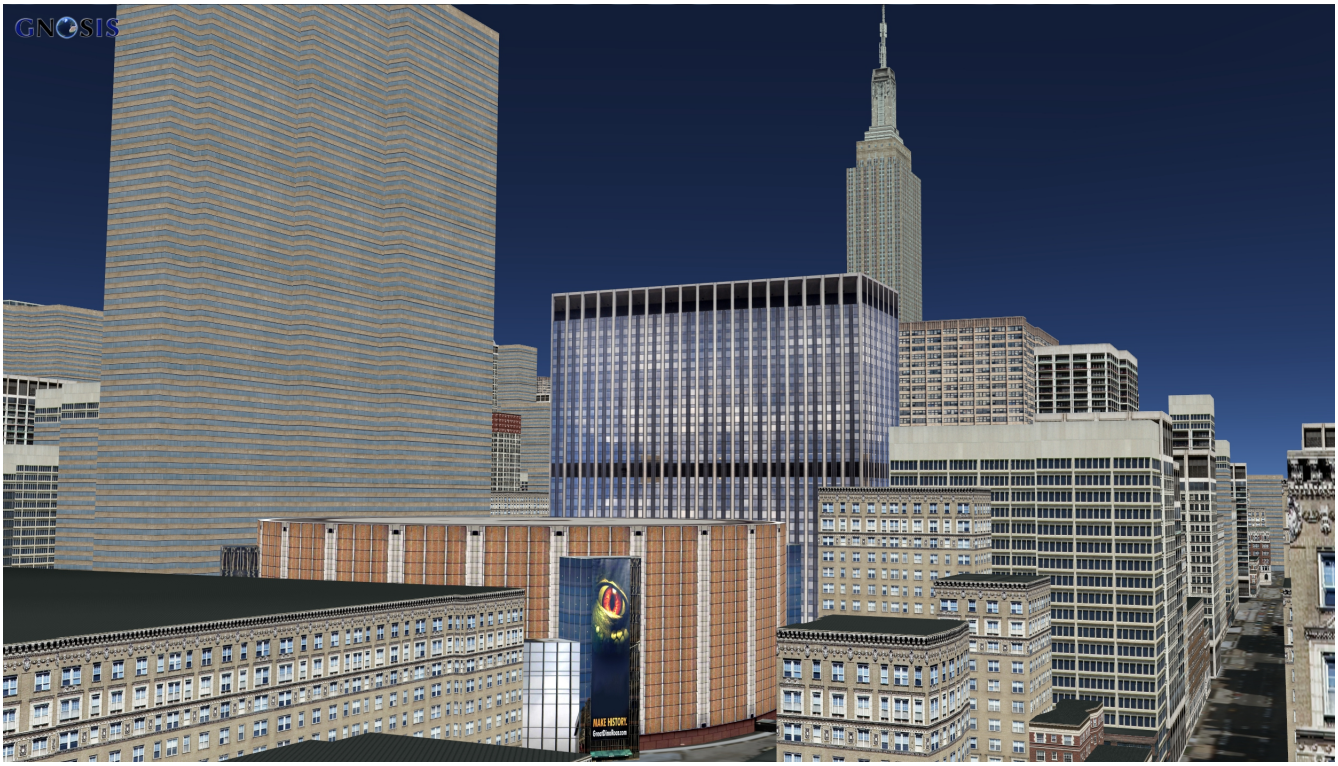

Figure 70. Visualization of Flight Safety CDB over Manhattan.

5.8.4. Results and findings

The results and findings are as follows:

- More work will be required to implement important functionality, e.g. support for geotypical models.
- Support for re-using aerial imagery as textures remains to be implemented (a black roof

textures was used as a placeholder for the direct CDB visualization, whereas the CDB to 3D Tiles conversion tool used the default "world_geo" placeholder texture resulting in greenish roofs).

- Some Z-fighting occurred with polygons of the Empire State building using a flood light texture, probably due to some features specifying a custom rendering technique not yet implemented.
- The definition of colors for faces and vertices as well as normals had to be ignored to achieve proper visualization results. This will need to be investigated as to whether it is an issue with the data sets or with its interpretation.
- CDB offers a very rich set of features as a result of its use for simulating in 3D environments for a long time.
- Despite this, it was possible to implement support for a subset of the functionality to display terrain, imagery and 3D models in the short time frame of the testbed. This would have been made easier had the specifications for the key features that were implemented been summarized in a more condensed form.
- The 'CDB Primer' presentation offered by Presagis helped in this regard, but some key information still had to discovered in the large volumes of specifications.
- A simple CDB subset could be described in just a few pages of documentation which would greatly help new implementations.
- CDB can already represent terrain, imagery, models as well as vector layer types, and does so leveraging already well supported GIS data formats (GeoTIFF, JPEG2000, Shapefiles).
- Being fully tiled and with resources such as textures and models shared between tiles, CDB itself (without requiring conversion to another format) could easily be used as a streaming format, and would in fact be an excellent candidate for doing so.
- Having elevation specified as a layer separate from the imagery enables visualizing the elevation data in various manners such as run-time shading with custom parameters, and applying elevation color ramps.
- CDB terrain easily integrates with other elevation data sources, avoiding any seams or overlapping at the border of tiles or data sources. This works very well with the GNOSIS terrain engine, performing on the fly optimization of the 3D terrain mesh.
- Like CDB, GNOSIS is built around a fixed tiling grid format. The advantage of this approach, contrasting with the approach used by both I3S and 3DTiles, is that it allows one to know beforehand exactly which tiles are required for a given view for all layers. This avoids multiple round-trips to the server first requesting lower resolution tiles before deciding whether the tiles need to be refined based on a geometric error, and repeating that process multiple times. Although the advantage of 3D Tiles and I3S's flexibility in the tiling layout is that it can accommodate varying data density, the philosophy of the GNOSIS approach is that the engine should still be capable of performing well under maximum density properly generalized according to the fixed grid, which accommodates mixing different levels of detail in a single three-dimensional scene.
- More optimization work is required to really achieve optimal performance.

5.9. Terrain and 3D Models Support in GNOSIS Map Tiles

As a result of working with these three different 3D formats (CDB, 3D Tiles and I3S), the natural course for the GNOSIS visualization tools is to integrate the possibility of importing any of those formats within its own open format, the GNOSIS map tiles and data store (described in Annexes B and D of the DS001 - Vector Tiles Engineering Report). That format already supports describing vector data, imagery as well elevation data for 3D terrain. Support for models as well as point clouds would be added. The 3D contents could also be streamed through a proposed Unified Map Service. Support for KML-referenced COLLADA models is already available in GNOSIS, and the same standard representation could extend to those KML/COLLADA models as well. The approach to store 3D models would adopt a uniform tiling scheme, take advantage of textures and models re-use when possible, yet offer the ability to batch geometry by materials and make use of texture atlases.

An open simple 3D model format (not requiring any kind of parsing, using quantized coordinates) would also be proposed, which would be significantly easier to implement support for than OpenFlight, I3S or glTF. It would define a basic simple set of 3D data in a rigid standard so as to facilitate interoperability, but support for additional extensible capabilities accommodating more realistic rendering will also be considered. However, models in other formats could also be directly embedded within GNOSIS map tiles layers (e.g. glTF, OpenFlight, COLLADA).

The representation of terrain elevation data using a binary triangle tree so as to save storage space where there is less elevation variance was investigated. It was found however that such a representation would need to adopt a number of mechanisms to avoid duplication, and doing so would render it complicated. Such a representation would also only save storage for the lower resolution level or when discarding changes in elevation not visible in the 3D terrain model. Because the elevation model could be used for purposes other than visualization as 3D terrain, it might not be ideal. Instead, the quantized gridded coverage format, which can optionally be compressed using a lossless compression algorithm such as the PNG representation, is recommended for storing terrain and distributing elevation tiles.

Secondary elevation layers could also be considered to represent concave surfaces, as CDB allows.

Appendix A: Revision History

Table 7. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
June 30, 2017	V. Coors	.1	all	IER
September 30, 2017	V. Coors	.2	all	initial version of DER
November 7, 2017	S. Serich	.3	all	comprehensive technical edit
January 23, 2018	V. Coors	.4	all	final technical edit

Appendix B: Bibliography

[1] Reed, C. (Editor): Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure, Version 1.0, http://www.opengeospatial.org/standards/cdb

[2] Saeedi, S. (Editor): Volume 11: OGC CDB Core Standard Conceptual Model, Version 1.0, http://www.opengeospatial.org/standards/cdb

[3] NN: OGC CDB Core Standard, Feature Data Dictionary, Version 1.0, http://schemas.opengis.net/cdb/1.0/Metadata/Feature_Data_Dictionary.xml

[4] NN: OGC CDB Core Standard, CDB Attributes, Version 1.0, http://schemas.opengis.net/cdb/1.0/Metadata/CDB_Attributes.xml

[5] Doyle, A.: Essential Model of Interactive Portrayal, OpenGIS project document 98-061, 25.11.1998

[6] Schilling, A, and Kolbe, T. (Editors): Draft for OpenGIS Web 3D Service Implementation Standard, v 0.4.0, OGC document 09-104r1, 18.11.2009, http://www.w3ds.org/lib/exe/fetch.php?media=09-104r1_web_3d_service-0.4.0.pdf

[7] Moreland, K.: "A Survey of Visualization Pipelines", IEEE Transactions on Visualization & Computer Graphics, vol. 19, no. 3, pp. 367-378, March 2013, doi:10.1109/TVCG.2012.133

[8] Koukofikis, A., and Coors, V.: Optimized conversion from CityGML to X3D using FME, Kartographische Nachrichten (Journal of Cartography and Geographic Information), Technical Report, 5/2016, 66. Jahrgang, Kirschbaum, pp. 268-271