INTEROPERABLE VISUALIZATION OF 3D CITY MODELS USING OGC'S STANDARD 3D PORTRAYAL SERVICE

A. Koukofikis [1], V. Coors [1], R. Gutbell [2]

[1] Hochschule für Technik Stuttgart, Faculty Geomatics, Computer Science and Mathematics, Schellingstr. 24, 70174 Stuttgart - (volker.coors, athanasios.koukofikis)@hft-stuttgart.de
[2] Fraunhofer Institute, Fraunhoferstr. 5, 64283 Darmstadt - ralf.gutbell@igd.fraunhofer.de

Commission IV, OGC

KEY WORDS: 3D Portrayal Service, OGC, Interoperability, 3D City Models, Visualization, I3S, 3D Tiles

ABSTRACT:

The demand of serving large 3D spatial data, mainly of urban areas, reflects the need of hierarchical data structures for 3D data. During the last years the OGC community standard I3S (Indexed 3d Scene Layer, ESRI) and 3D Tiles (Analytical Graphics, Inc.) emerged in order to deal with this challenge. Conceptually, hierarchical structures for 3D data operate similarly to web map tiles, differing only in the implementation. Although 3D hierarchical formats can transmit arbitrary sized geospatial data, they are not interoperable with consuming/visualization technologies on the client. A series of prototype implementations focus on rendering of hierarchical organized massive 3D data in various web client technologies employing the 3D Portrayal Service. As a result, the user can query a scene via the 3D Portrayal Service by specifying a spatial region, rather than a specific resource via a URI. The result is delivered either using I3S or 3D Tiles as a data delivery format, depending on which data is available for the specified region. The client APIs are capable of rendering either the I3S or the 3D Tiles content.

1. INTRODUCTION

1.1 Interoperability

The technological pluralism in 3D web visualization constantly increases the need of interoperability. The availability of several 3D delivery formats and client consumers indicates a need for a generic approach for accessing 3D information. This paper defines an attempt to realize the interoperability between geospatial data web consumers (web globes) and OGC's 3D Portrayal Service (Coors et al., 2017) regarding the request of hierarchical 3D data storages. 3D Tiles (3D Tiles Specification, 2018) and I3S (I3S Specification, 2018) are the latest implementations of hierarchical data storages.

1.2 OGC Testbed 13

OGC Testbed 13 focuses on 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. It 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 conducted report captures the results from the interoperability tests performed as part of the 3D Tiles and I3S testbed work package. The present study gives an insight into OGC's Testbed 13 experiments 2 and 3 which relate to 3D Portrayal Service interoperability capabilities.

2. RELATED WORK

Use cases of the 3D Portrayal Service appear to be scarce, since the version 1.0 of the standard was released recently. (Gaillard et al. 2015) utilized the 3D Portrayal Service View conformance class for client side rendering of tiled 3D city models. Building geometry, originating form CityGML, was converted and served as JSON using a GetScene request. (Gutbell et al. 2016) implemented a server-side rendering framework to visualize 3D city models using the 3D Portrayal Service GetView request.
Cesium as visualization component is used in several cases. (Krämer & Gutbell, 2015) produced a surface model visualization using Cesium's Terrain Builder. (Kim et al. 2017) presented a visual extension of Cesium to visualize moving objects in a space-time cube. (Lu, Guerrero, Mitra & Steed, 2016) implemented an online 3D city model interactive editor using a light-weight viewer based on Cesium.

3. CONCEPT & METHODOLGY

OGC Testbed 13 aims at testing the interoperability of streaming capabilities relating to the OGC I3S Community Standard and the 3D Tiles specification, which is a OGC Community Standard candidate. A number of tests were formulated in order to examine the interoperability and performance characteristics of streaming efficiency 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 a CityGML data store.

To implement the visualization pipeline, several experiments using CityGML data stores were conducted. AGI created the necessary processing algorithms to convert CityGML into 3D Tiles within its 3D Tiles processing Tools; ESRI provided the necessary processing algorithms to convert CityGML into I3S within ArcGIS and by using FME (Feature Manipulation Engine, Safe Software); Fraunhofer, an institute for applied science, and the SME virtualcitySYSTEMS created the necessary processing algorithms to convert CityGML with and without application domain extension into 3D Tiles as extension on top of GeoRocket (GeoRocket, 2018). 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 3D Portrayal Service. 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 3D Portrayal Service to return appropriate tiles. Data was streamed according to the 3D Tiles and I3S formats. In addition, runtime visualization strategies were developed and profiled.

Experiments 2 and 4 investigated source data management using GeoRocket. For these experiments, a 3D Portrayal Service instance was implemented on top of GeoRocket (see Figure 4). Experiments 1 and 3 investigated source data processing using AGI’s 3D Tiles Pipeline and ESRI’s ArcGIS.

3.1 Visualization pipeline

The visualization pipeline provides the key structure like any visualization system. (Moreland, 2013) gives a survey of visualization pipelines of several applications. In this report, the concept of the visualization pipeline for the interactive portrayal of geospatial data is based on Doyle & Cuthbert (1998). It consists of three main components:

1. Filter
2. Map and
3. Render (Figure 1)

The Filtering step (sometimes called selecting) extracts data from larger geospatial data set that should be displayed on the screen. The Mapping step maps features to geometrical 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.

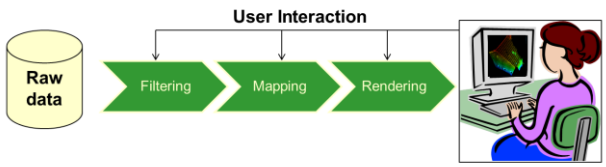

Figure 1. The Visualization Pipeline.

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 (Figure 2). 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 (Schilling & Kolbe, 2009).

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, since the scene graph is significantly smaller in the first case.

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. In this report, this distribution of the visualization pipeline was analyzed using CityGML as file-based data storage, and GeoRocket as a spatial database. The results of the Mapping step are stored in I3S as well as in 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 3D Portrayal Service conformance class view supports this approach. However, it was out of the scope of this testbed.
Two datasets were used in the experiments:

- CityGML New York City DoITT Data set: The data set 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.
- CityGML model of Berlin: The Berlin 3D City Model was also utilized since the DoITT buildings did not have textures. The 3D City Model of Berlin has been developed by Land Berlin’s Senate Administration for Urban Development, Senate Administration for Economics, Energy and Public Enterprises.

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 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.

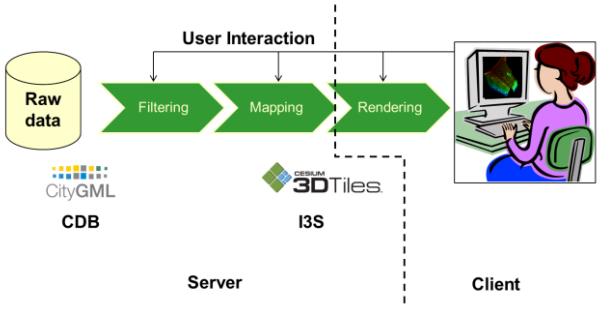

Figure 2. Distributed Visualization Pipeline and the role of OGC standards used in this experiment.

3.2 OGC Testbed 13 Experiment 2 and 3

In OGC’s Testbed 13 experiments 2 and 3 analyzed the capability of 3D Portrayal Service to support interoperability. While experiment 2 focuses on utilizing the open-source renderer Cesium for visualization of 3D Tiles and I3S data. Another key feature of experiment 2 is the use of the open-source datastore GeoRocket to deliver the CityGML datasets.

The following subsections discuss the setups in detail.

3.2.1 Experiment 2: 3D Tiles and I3S in Cesium using the 3D Portrayal Service

In experiment 2 the open source data store GeoRocket 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.

The CityGML scenario data was stored in GeoRocket and was exported as CityGML answering spatial queries. Figure 3 sketches the most important elements of GeoRocket.

Of the available data store options, 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.

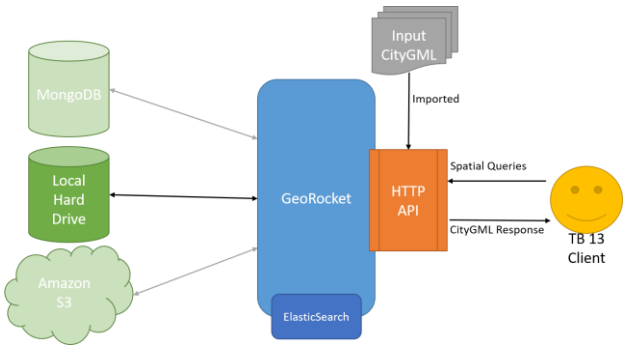

Figure 3. This diagram sketches the most important elements of GeoRocket from the supported data stores to the API, which the client can use.

The delivered CityGML files are then processed to 3D Tiles or I3S. These processes are explained in detail in the following subsections.

3.2.1.1 Experiment 2a: 3D Tiles in Cesium via the 3D Portrayal Service

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. This data flow included:

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

Figure 4 pictures this data flow:

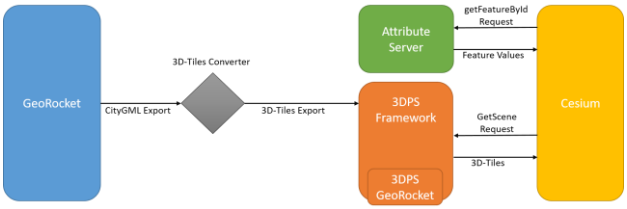

Figure 4. The data flow from GeoRocket to the visualized 3D Tiles, which are requested via the 3D Portrayal Service queries.

The resulting 3D Tiles dataset was then imported into the 3D Portrayal Service framework, which is able to satisfy the spatial queries. This saved time because it was not required that CityGML content be transformed to 3D Tiles for every spatial query. The data flow is pictured in Figure 5 as a sequence diagram.

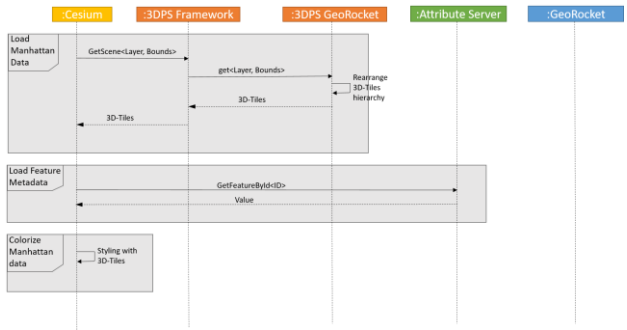

Figure 5. The data flow from GeoRocket to the visualized 3D Tiles, which are requested via the 3D Portrayal Service queries.

3.2.1.2 Experiment 2b: I3S in Cesium via the 3D Portrayal Service

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 3D Portrayal Service 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.

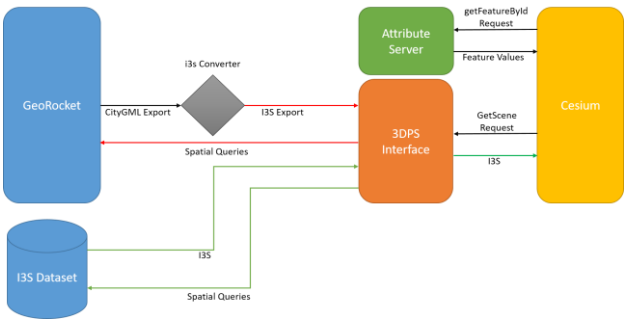

Figure 6. This image shows two dataflows: 1: The supposed dataflow (in red) from GeoRocket which exports CityGml, to

converted I3S (not implemented), which are then visualized in the Cesium client via 3D Portrayal Service queries.

Figure 6 points out that the spatial queries would be forwarded to GeoRocket for every spatial 3D Portrayal Service GetScene request to retrieve the spatially bounded CityGML data, which was then converted to the I3S data format. Since I3S is a recent technology, the availability of conversion tools from CityGML to I3S is limited. Moreover, an implementation of our own converter would not be an effortless task due to the complexity of I3S.

The focus of the experiment was shifted to the rendering of I3S in the Cesium client using the 3D Portrayal Service. As a result, the user could query a scene via 3D Portrayal Service 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 3D Portrayal Service regarding the request of hierarchical 3D data storages was proven by a prototype implementation.

The main goal was to investigate if the 3D 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 3D Portrayal Service’s getScene request, pictured in Figure 7.

Figure 7. 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 8).

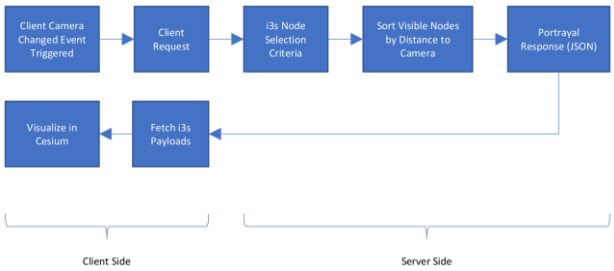

Figure 8. Communication Synopsis.

3.2.2 Experiment 3: CityGML to I3S to ArcGIS

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 (Figure 9). 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.

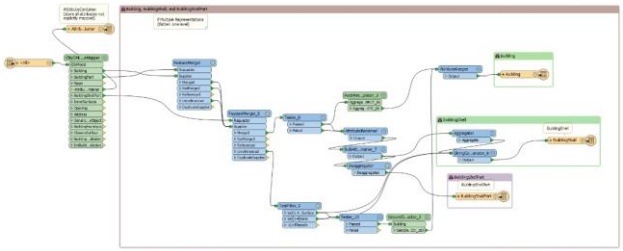

Figure 9. FME workbench to convert CityGML to I3S.

4. EXTENDED 3D PORTRAYAL SERVICE IMPLEMENTATION

An agile software development approach was used in this study to implement both a 3D Portrayal Service server and a web-based client. The 3D Portrayal Service is implemented utilizing the Play Framework in Java programming language. The code base of the client application is modularized using the Dojo Toolkit. The user is able to dynamically define a cartographic bounding box which is spatially intersected by the 3D Portrayal Service available extents in order to identify the requested resource, either 3D Tiles or I3S. Additionally, a cartographic buffer is generated around the requested bounding box to include neighboring I3S nodes (Figure 10).

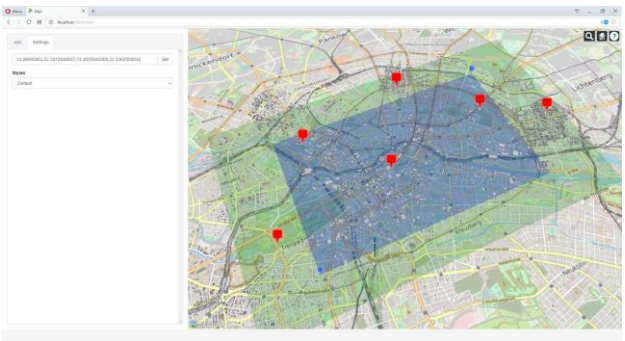

Figure 10. I3S nodes rendered in Cesium. Blue rectangle: requested bounding box. Green rectangle: buffer. Red pins: centers of the minimum bounding spheres.

The 3D Portrayal request is extended with URL parameters which are not defined in the request scheme of the standard. These parameters, originating from Cesium, are used to apply the I3S node selection criteria and to reject outdated responses by the 3D Portrayal Service (Figure 11).

service: 3DPS
acceptversions: 1.0
request: GetScene
boundingbox: 9.7350568228,53.397645455,10.3193171359,53.7369413172
cullingvolume: -0.489520008514,0.87030950429,-0.054143586849,1533027.01880,-0.161886420120,-0.9853188378,-0.054217845856,1530930.62031,-0.62377256088,-0.110165543187,0.77380316966,-1550108.68011,-0.3480433034,-0.061413811752,-0.9354646987,6121145.3019,-0.97181586470,-0.171579355008,-0.161661529262,4571035.6240,0.97181586470,0.171579355008,0.161661529262,495428963.37
camera: 3737336.29540,658537.61433,5109703.3728,-0.97181586470,-0.171579355008,-0.161661529262
frustum: 1,0.57735026918,0.35572226262
drawingbuffer: 573,930
time: 1516234129317

Figure 11. Extended 3D Portrayal request with additional parameters highlighted in blue.

5. EVALUATION OF EXTENDED 3D PORTRAYAL SERVICE

To assess the performance of the extended 3D Portrayal Service, request/response measures were determined and compared with an equivalent ArcGIS JavaScript API application, which served the same I3S Scene layer. To ensure that the same I3S tree space would be traversed, both applications requested the same scene determined by the camera position/orientation in 3D space and view frustum geometry. The camera was positioned above the city of Berlin at geographic position (WGS84) 13.4050 degrees longitude, 52.5200 degrees latitude and oriented to produce a “bird’s eye” view with 45 degrees tilt, 0 degrees heading and 60 degrees field of view. Additionally, three distinct camera height positions would formulate three request scenarios, i.e., high camera position (3000 meters) where a large city area is visible (Figure 12), medial camera position (1000 meters) where some city blocks are visible (Figure 13) and low camera position (400 meters) where a small city area is visible (Figure 14).

Figure 12. I3S scene of Berlin with camera position at 3000 meters rendered in ArcGIS JavaScript API (left) and extended 3D Portrayal Service/Cesium (right).

Figure 13. I3S scene of Berlin with camera position at 1000 meters rendered in ArcGIS JavaScript API (left) and extended 3D Portrayal Service/Cesium (right).

Figure 14. I3S scene of Berlin with camera position at 400 meters rendered in ArcGIS JavaScript API (left) and extended 3D Portrayal Service/Cesium (right).

To perform a realistic comparison between the ArcGIS JavaScript API and the extended 3D Portrayal Service/Cesium applications a specific subset of the total amount of the http requests/responses was measured. This is due to the extended 3D Portrayal Service/Cesium implementation, which requests only the mesh geometry and the face normals from an I3S node, whereas the ArcGIS JavaScript API requests additionally shared resources, i.e., material definitions and textures. The average time spans recorded for both applications include the handshake/capabilities phase and the I3S node traversal excluding the payload fetching, payload processing and rendering. The average DOM Content Load Time and the average Page Load Time were recorded but not considered in the evaluation process (Table 15).

Client application	Camera height [m]	DOM Content Load [sec]	Page Load Time [sec]	Handshake/I3S traversal [sec]
AJS API	3000	1.40	4.02	4.70
x3DPS	3000	0.99	5.46	6.42 (+36%)
AJS API	1000	1.46	3.99	5.61
x3DPS	1000	1.13	5.37	6.60 (+18%)
AJS API	400	1.33	3.86	4.05
x3DPS	400	1.26	5.45	4.84 (+20%)

Table 15. Performance comparison between ArcGIS JavaScript API (abbreviated to AJS API) and extended 3D Portrayal Service/Cesium (abbreviated to x3DPS).

6. SUMMARY AND FUTURE WORK

This paper presents the experiments performed in OGC’s Testbed 13 including the 3D Portrayal Service Standard. The standard has proven to be a flexible interface for requesting portrayals of massive georeferenced data in form of 3D city models.

The experiments covered the whole data flow of storing raw CityGML data with GeoRocket, a cloud-enabled open source data store for Geodata, to the final visualization based on WebGL renderer.

The experiments proved that the 3D Portrayal Service can be the connecting element in between the user and the renderer, which are still bound to their specific formats. This was shown by extending the popular renderer Cesium and ArcGIS with the 3D Portrayal Service interface. This enables the end user to change between systems using different renderer without having to adopt to the specific APIs of the renderer.

Especially the GetFeatureById functionality of the 3D Portrayal Service allowed easy inclusion of other different data sources, like a simulation of heat demand for New York, enabling the implementation of rich applications based on 3D Portrayal Service.

It is important to understand that 3D Portrayal Service does not define a data format to transmit data between client and server and can’t transform the render format of one renderer to the format of the other render. Our extended evaluation proved that it is possible to embed services in between render client and data server, which can perform this task.

This will be the focus of our future work, the challenge of transforming data in between very specific formats having their own pros and cons.

REFERENCES

3D Tiles Specification, 2018. GitHub. Retrieved 27 March 2018, from https://github.com/AnalyticalGraphicsInc/3d-tiles

Coors, V., Hagedorn, B., Thum, S., Reitz, T., Gutbell, R., 2017. 3D PORTRAYAL SERVICE 1.0. (OGC Document Number 15-001r4)

Doyle, A., Cuthbert, A., 1998. Essential Model of Interactive Portrayal. (OGC Document Number 98-061)

Gaillard, J., Vienne, A., Baume, R., Pedrinis, F., Peytavie, A., Gesquière, G., 2015. Urban data visualisation in a web browser. *Proceedings Of The 20Th International Conference On 3D Web Technology - Web3d '15*. doi: 10.1145/2775292.2775302

GeoRocket, 2018. GitHub. Retrieved 27 March 2018, from https://github.com/georocket/georocket

Gutbell, R., Pandikow, L., Coors, V., Kammeyer, Y., 2016. A framework for server side rendering using OGC's 3D portrayal service. *Proceedings Of The 21St International Conference On Web3d Technology - Web3d '16*. doi: 10.1145/2945292.2945306

I3S Specification, 2018. GitHub. Retrieved 27 March 2018, from https://github.com/Esri/i3s-spec

Kim, K., Kim, D., Jeong, H., Ogawa, H., 2017. Stinuum: A Holistic Visual Analysis of Moving Objects with Open Source Software. *SIGSPATIAL/GIS*. doi: 10.1145/3139958.3140011

Krämer, M., Gutbell, R., 2015. A case study on 3D geospatial applications in the web using state-of-the-art WebGL frameworks. *Proceedings Of The 20Th International Conference On 3D Web Technology - Web3d '15*. doi: 10.1145/2775292.2775303

Lu, Z., Guerrero, P., Mitra, N., Steed, A., 2016. Open3D. *Proceedings Of The 21St International Conference On Web3d Technology - Web3d '16*. doi: 10.1145/2945292.2945302

Moreland, K., 2013. A Survey of Visualization Pipelines. *IEEE Transactions On Visualization And Computer Graphics*, *19*(3), 367-378. doi: 10.1109/tvcg.2012.133

Schilling, A., Kolbe, T., 2009. Web 3D Service 0.4.0. (OGC Document Number 09-104r1)