OGC Testbed-14

CityGML and AR Engineering Report

Table of Contents

1. Summary . . . 4
   1.1. Requirements & Research Motivation . . . 5
      1.1.1. Requirements . . . 5
      1.1.2. Objectives . . . 6
   1.2. Prior-After Comparison . . . 8
      1.2.1. A review of ARML capabilities for 3D visualization and AR . . . 8
      1.2.2. ARML capabilities not yet covered to be considered for future work . . . 9
   1.3. Intended audience . . . 9
   1.4. Recommendations for Future Work . . . 10
   1.5. Document contributor contact points . . . 11
   1.6. Foreword . . . 12
2. References . . . 13
3. Terms and definitions . . . 14
   3.1. Abbreviated terms . . . 14
4. Overview . . . 16
5. Augmented Reality . . . 17
   5.1. Augmenting reality with geospatial data . . . 17
   5.2. Similarities and particularities of styling for 3D views . . . 18
   5.3. A common flexible geographic features styling conceptual model . . . 18
   5.4. Linking to attributes from source data in annotations . . . 19
   5.5. Extended capabilities applying specifically to AR . . . 19
   5.6. Indoor localization challenges: pattern matching, computer vision . . . 19
6. Situating reality in a geospatial context . . . 20
   6.1. The camera . . . 20
   6.2. The GPS (and other means of determining locations) . . . 21
   6.3. Motion sensors . . . 21
      6.3.1. The magnetometer . . . 21
      6.3.2. The gyroscope . . . 22
      6.3.3. The accelerometer . . . 22
      6.3.4. The barometer (as an altimeter) . . . 22
      6.3.5. 3D orientation . . . 22
      6.3.6. 3D position . . . 23
   6.4. Size, weight and power usage considerations . . . 23
7. Augmented Reality Clients . . . 24
   7.1. GNOSIS / Android mobile client by Ecere . . . 24
   7.2. ARKit / iOS mobile client by GIS-FCU . . . 39
      7.2.1. Using ARKit . . . 39
      7.2.2. Rendering 3D Models . . . 40

7.2.3. Client Demonstration . . . 42
8. Client-Server Communication . . . 50
8.1. Services overview . . . 50
8.2. Transmission data formats . . . 51
8.2.1. 3D Tiles & I3S . . . 51
8.2.2. Rationale behind decision to use E3D for this initiative . . . 52
8.2.3. ETC2 Texture Compression . . . 53
8.3. Performance considerations . . . 53
8.3.1. Tiling and Caching . . . 54
8.4. Disconnected environment and intermittent connectivity . . . 56
8.5. Relationship with 3D Portrayal Service . . . 56
9. Geospatial Services for Augmented Reality . . . 57
9.1. Ecere Service . . . 57
9.1.1. GetCapabilities . . . 57
9.1.2. DescribeFeatureType . . . 57
9.1.3. GetFeature . . . 58
9.1.4. GetModel . . . 61
9.1.5. GetTexture . . . 62
9.1.6. WFS3 / Next Generation / Harmonized Map Service (REST API) . . . 63
9.1.7. Unified Map Service . . . 64
9.2. GIS-FCU Service . . . 65
9.2.1. GetFeature . . . 65
9.2.2. GetModel . . . 66
10. Geospatial data . . . 67
10.1. Original data sources (and their formats) used for the experiments . . . 67
10.2. Intermediate CityGML data model (GIS-FCU) . . . 69
10.3. Conversion from CityGML to E3D models data set (GIS-FCU) . . . 73
10.4. Conversion of CDB data set to GNOSIS data store (Ecere) . . . 74
10.5. Importing OpenStreetMap data (Ecere) . . . 76
Appendix A: E3D 3D Model Specifications . . . 77
A.1. Coordinate system . . . 77
A.2. A simple cube example . . . 77
A.3. Adding normals attributes . . . 82
A.4. Compression . . . 84
A.5. Detailed description of all block types . . . 85
A.6. Sample E3D models . . . 91
Appendix B: GNOSIS Map Tiles . . . 95
B.1. GNOSIS Map Tile Header . . . 96
B.1.1. Geospatial data types . . . 96
B.1.2. Flags . . . 98
B.2. Compact tiled vector data representation . . . 100

B.2.1. Compact storage as localized vertices with accuracy proportional to scale . . . 100
B.2.2. Pre-triangulated for high performance GPU rendering and optimal service-to-display processing 100
B.2.3. Enforced topologically correct representation (shared vertex indices) . . . 100
B.2.4. Unique feature identifier . . . 101
B.2.5. Center lines for curved area labels . . . 101
B.3. Binary layout for Points, Point Clouds and Positioned & Oriented 3D Models tiles . . . 101
B.3.1. 3D Models . . . 105
B.4. Binary layout for Lines tiles . . . 106
B.5. Binary layout for Polygons tiles . . . 107
B.5.1. Vertex flags for identifying tile boundaries and artificial edges . . . 111
B.6. Embedded 3D Models . . . 113
B.7. Binary layout for Imagery tiles . . . 113
B.8. Binary layout for gridded Coverage tiles . . . 114
B.9. Paeth filtering . . . 116
Appendix C: Geospatial features styling with support for 3D and AR . . . 119
C.1. Conceptual Styling Model . . . 119
C.1.1. Portrayal Rules . . . 120
C.1.2. Rendering styles (a.k.a. symbolizers) . . . 120
C.1.3. Markers, Labels, and GraphicalElements . . . 121
C.1.4. Expressions . . . 121
C.2. GNOSIS Cascading Map Style Sheet encoding example . . . 125
Appendix D: Revision History . . . 128
Appendix E: Bibliography . . . 129

Publication Date: 2019-03-07

Approval Date: 2019-02-28

Submission Date: 2019-02-06

Reference number of this document: OGC 18-025

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D028

Category: Public Engineering Report

Editor: Jérôme Jacovella-St-Louis

Title: OGC Testbed-14: CityGML and AR Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

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-14 Engineering Report (ER) describes the results of the Augmented Reality (AR) work performed in the Testbed-14 ***CityGML and Augmented Reality*** work package which was part of the Next Generation Services thread.

By integrating information available from urban models within a view of the real world through a mobile device, this testbed activity explored the possibilities offered by AR in a geospatial context. The ER additionally discusses the approach used to bring in these urban models from various data sources. The experiments also covered to some extent Virtual Reality (VR) where any city can be explored freely from a computer display or potentially within a VR headset.

A continuation of these experiments would have looked at a combination of Augmented and Virtual Reality (Mixed Reality). The portrayal of AR and three-dimensional (3D) content through extending a common conceptual model to style classic geospatial features (as explored in the Testbed-14 Portrayal [https://portal.opengeospatial.org/files/77327#Portrayal] work) is also touched upon. The efficient transmission of 3D content is also a subject of this document through the use of a simple 3D transmission format developed during the initiative.

This ER provides many insights that showcase what is now made possible by the combination of AR, VR and integrated urban models.

The testbed work shines light on the benefits of applying a common portrayal approach to AR, bridging the gap between AR applications and traditional Geographic Information Systems and services.

The ER introduces a new, simple approach and conceptual model for transmitting 3D geospatial content which could be the basis to define simple profiles for the I3S [http://www.opengeospatial.org/standards/i3s] and 3D Tiles [http://www.opengeospatial.org/pressroom/pressreleases/2829] community standards. It could also inform enhancements to the 3D Portrayal Service [http://www.opengeospatial.org/standards/3dp] (3DPS) and/or next generation services (e.g., WFS 3.0 [https://rawgit.com/opengeospatial/WFS_FES/master/docs/17-069.html]) for delivering 3D contents in a format agnostic manner.

Finally, the ER covers methods to bring in different types of geospatial content from various sources for integration into AR applications.

During Testbed-14, the participants demonstrated AR experiences with geospatial datasets providing integrated views of urban spaces. Two clients and two services were shown to be interoperable, streaming AR content through a simple 3D transmission format, leveraging either GeoJSON or GNOSIS Map Tiles, as well as E3D 3D model specifications.

The feasibility of extending a classic portrayal conceptual model for AR was also shown. In order to serve them to the clients in the supported transmission formats, geospatial data sets of various types and in various formats were successfully imported for consumption by the services.

1.1. Requirements & Research Motivation

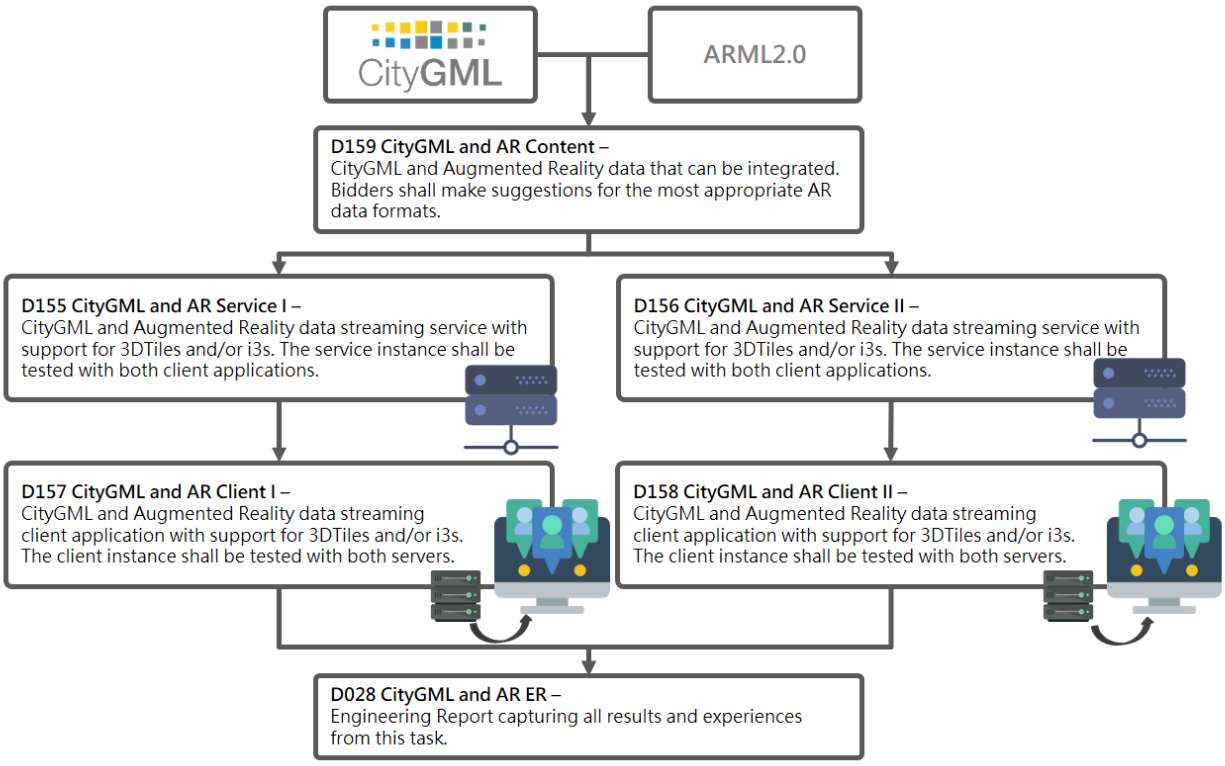

Figure 1. Diagram of CityGML and AR initiative deliverables

1.1.1. Requirements

As stated in the Call for Participation [https://portal.opengeospatial.org/files/77327], the requirements for the Testbed-14 *CityGML and Augmented Reality* work package were:

- To evaluate the possible methods for integrating AR content into CityGML content. The effort shall include an analysis of whether Augmented Reality content should be integrated into the CityGML data or whether the data content (features) should be linked for visualization purposes.
- To demonstrate client-side support for visualization of streamed CityGML with AR content. Bidders shall include suggestions for suitable AR data in their proposals and shall investigate the role and applicability of OGC Augmented Reality Markup Language 2.0 (ARML 2.0) [http://docs.opengeospatial.org/is/12-132r4/12-132r4.html], the Community Standard 3D Tiles [https://github.com/AnalyticalGraphicsInc/3d-tiles/blob/master/README.md], and OGC Community Standard I3S [http://docs.opengeospatial.org/cs/17-014r5/17-014r5.html].

The goal was to provide highly detailed visualization of fully integrated urban spaces, as modeled in CityGML. This would notably be very valuable for urban warfighters or first responders, in the planning and execution of modern missions.

This integrated view brings together:

- Geography
- Topography

- Infrastructure & utilities
- Indoor spaces
- Detailed building models

The stakeholders clarified that the AR content will be a presentation of these elements, either as 3D geometry, classic 2D geospatial features, or annotations sourced from the associated attribute information.

Since the primary objective of this initiative was to highlight the possibilities of AR, the requirement to stream contents specifically as 3D Tiles and/or I3S was relaxed to become optional. This allowed the participants to more quickly focus efforts on the AR aspects.

Furthermore, because abundant amounts of relevant CityGML data were not readily available, and because the AR clients did not deal directly with CityGML-encoded content, the stakeholders decided that the services could additionally import other source data types to be served in the transmission format directly through the best suited mechanism.

1.1.2. Objectives

Two primary objectives corresponding to different scenarios were identified: the first focused on AR whereas the second is better described as VR. The experiments done with the clients and services for these different scenarios are covered in different sections of this report. Additional objectives pertained to the particularities of portraying geospatial features to augment reality, and to efficient transmission of content.

The Augmented Reality Scenario

In the AR scenario, a real view of the surroundings from a mobile device camera is augmented with geospatial data overlays. The real world is the foundation of the user experience, and the integration of camera and sensors (gyroscope, magnetoscope, accelerometer) synchronize the 'augmented' elements with the real world. The 3D geometry data, such as originating from a CityGML package, makes up the AR content. This scenario is tied to device location, and combined with the difficulty of finding relevant data for an area where the participants were located, this posed logistical challenges for the initiative.

In this scenario, a sub-set of the urban model or extra 3D elements were overlaid on top; some potentially with translucency so as not to completely obscure the camera view of the real world.

The stakeholders decided that the focus would be on an outdoor scenario for a number of reasons:

- Outside scenarios benefit from easier geolocalization and synching with the real world on a mobile device from GPS sensors and the cellular signal.
- Outside scenarios are likely to involve traveling a large distance and would benefit greatly from streaming 3D contents as tiles.
- Emergency response has been cited as a potential application of such an outside scenario.

An indoor scenario, however, would have presented a different set of challenges:

- It would have to rely on recognition of the environment (e.g. pattern recognition) from the camera due to the smaller scale of features and limited GPS/cellular connectivity. The challenge is an order of magnitude more difficult, which was not well suited for the scope of the initiative.
- CityGML is often used to represent large integrated environment, and would not have been the best source of data for indoors information. However, some elements such as ADEs providing BIM might still be relevant.
- InDoorGML is another standard which may provide more relevant data for such a scenario.

The Virtual Reality Scenario

Virtual Reality is the second scenario, where an integrated urban view can be freely explored virtually. In this scenario, most of the 3D geometry makes up the base scene (i.e. the 'virtual' reality). The 3D models cover an entire view, with no need for integration of the real-world camera view. This scenario is not dependent on the physical location of the user (e.g., data from Taiwan could be visualized in Washington, DC).

In this scenario, some of the 3D elements could have different styles applied to highlight these features, and the visibility of the different elements could be toggled. Given more time, this could have been demonstrated through a VR headset such as the Oculus Rift [https://www.oculus.com/rift/].

Mixed Reality

Another interesting experiment would be to combine AR and VR (*Mixed Reality*), where a VR headset is used together with stereoscopic cameras such as the Ovrvision Pro [http://ovrvision.com/entop/] or ZED Mini [https://www.stereolabs.com/zed-mini/] by Stereolabs.

A new generation of AR glasses with stereoscopic vision and binocular AR capabilities, such as the ODG-R9 [https://www.osterhoutgroup.com/r-9-smartglasses] (although the availability of these AR glasses is now doubtful given recent development [1]) or the MagicLeap [https://www.magicleap.com/] also offer a glimpse of new Mixed Reality possibilities in the near future. The mixed reality devices market however is still very volatile, as illustrated by the often overwhelming challenges faced by many companies in the space despite large amounts of funding, as well as the limited availability of stereoscopic AR glasses, particularly outside of the US market.

The OGC is currently executing the Mixed Reality to the Edge Concept Development Study [http://www.opengeospatial.org/pressroom/pressreleases/2926]. Findings and recommendations from this study could feed directly into a mixed reality scenario.

Portrayal of AR content

All of these scenarios were to feature annotations in the forms of labels, markers and other visual elements (e.g. simple 2D and/or 3D primitives) tied to certain geographic locations / features. The annotations source was from the non-geometry attributes of the source data (e.g. CityGML), and transmitted to the client through a mechanism defined in the transmission format.

The stakeholders had to decide the best way to describe which additional elements to display (e.g., extra 3D geometry such as utility ADE vs. textual annotations), when to display them, where to overlay them, and how to style them.

How to 'link' this AR content to the base visual elements (either the camera feed, or the rendering of a virtual 3D view of the integrated urban model represented by the CityGML) also had to be determined.

Efficient transmission of AR content

An efficient transmission of AR content between clients and services had to be implemented. This is covered in detail in Chapter 8.

1.2. Prior-After Comparison

1.2.1. A review of ARML capabilities for 3D visualization and AR

The OGC® Augmented Reality Markup Language 2.0 [http://www.opengeospatial.org/standards/arml] (ARML 2.0) standard allows users to describe visual virtual objects in an augmented reality (AR) scene with their appearances and anchors (a broader concept of a location) related to the real world as depicted in visual camera input. ARML 2.0 also defined ECMAScript [https://www.ecma-international.org/publications/standards/Ecma-262.htm] bindings to dynamically modify the AR scene based on user behavior and input. Part of the motivation for ARML 2.0 [https://www.wikitude.com/arml-20-standard-approved/] was to enable several AR Browser vendors, Layar [https://www.layar.com/], Metaio [https://en.wikipedia.org/wiki/Metaio], and Wikitude [https://www.wikitude.com/] to create a common format to enable better interoperability among their browsers at a time when the mainstream mobile browser market was still immature. ARML 2.0 also used the Web IDL Specification [http://www.w3.org/TR/WebIDL/].

There are significant differences in the approach to describe augmented reality contents used in this initiative and that of ARML 2.0, both in assumed use case scenarios and in technical details. For example, the latter contemplated browser-based clients using XML-based COLLADA [https://www.khronos.org/collada/] files for 3D data exchange. By contrast, the approach used in this testbed initiative was intended for native apps running on low-capacity clients operating in Denied, Degraded, Intermittent, or Limited Bandwidth (DDIL) environments as might be found in urban warfighting or emergency-response.

Another major difference is that while ARML combined both geospatial data and styling rules in a single payload, the approach used in this initiative opted for the preferred approach of keeping data and portrayal options separate. By using a styling conceptual model also applicable to typical Geographic Information System (GIS) data sources, this approach helped bridge the gap between Augmented Reality and GIS. The ability to readily integrate the vast amount of readily available geospatial data in AR applications can help minimize application development costs while also maximizing investments in geospatial data collection. Further, much of the data is already encoded or transferred according to OGC standards.

Conversely, the concepts of portraying and labeling features in a 3D view are not specific to Augmented Reality. They also apply to a regular 3D GIS application where views are purely virtual. For this reason, it makes more sense that such capabilities (which were the ones this initiative was mostly concerned with in its usage scenarios) be integrated in a way which applies to both classic and 3D GIS applications, than with a language targeting Augmented Reality specifically.

To compare the capabilities offered by ARML with the work done in this testbed, the latter focused

strictly on Geospatial Augmented Reality. Computer Vision-based augmented reality was not looked into, being better suited for a smaller scale or indoor scenarios. For Geospatial AR, the core concepts in ARML include *Feature*, *Anchor* and *Visual Assets*.

The *Features* from ARML, defined as the real world objects themselves, are closely related to the geospatial data being defined in this tested. At various stages this data was defined in CDB, Shapefiles, CityGML, OpenStreetMap data, GNOSIS Map Tiles, GeoJSON and E3D models formats. As entire cities were covered, the geospatial data were very large. As such, a text-based format (e.g. XML) is not well suited to represent these data sets in an AR environment. Furthermore, it was desirable to only have a small subset of a much larger dataset relevant to a given view. For this reason, partial requests and tiles were used.

ARML also defines the concept of an *Anchor*, which in the geospatial AR can simply be a single geospatial point. For the testbed, the portrayal language could be thought of as what is anchoring visual assets to specific point or feature sets. But the actual geospatial coordinates would be in the data itself. In the case of lines or areas, labeling rules can control exactly where visual assets should appear.

Then ARML has the concept of a *Virtual Asset*, either two-dimensional (2D) or 3D, which is to be drawn into the scene thus "augmenting" the view. In the approach used for this testbed, this is also the case and these virtual assets are defined or referenced using the portrayal language. This approach allows referencing attributes associated with the geospatial data to display textual information, or to control various visual properties such as colors. It could also reference symbology, for example in the form of Portable Network Graphics (PNG) or Scalable Vector Graphics (SVG), or even 3D models to be displayed at the anchored location.

Most of the efforts of the initiative however focused on superimposing virtual buildings 3D geometry at their proper location in the view. In this case, the 3D building's geometry could be thought of as being a *Feature*, *Anchor* and *Virtual Asset* all at once.

1.2.2. ARML capabilities not yet covered to be considered for future work

In spite of the differences between ARML 2.0 and the approach to describe Augmented Reality contents in this initiative, ARML can still be a source of inspiration for important capabilities to consider re-integrating in styling extensions specific to augmented reality. A most obvious example of such capabilities is the concept of a computer vision-based anchor (*Trackable*). This could be considered for future work, tied more with computer vision and potentially better suited for indoor localization challenges.

1.3. Intended audience

This ER provides findings and recommendations which may impact a number of OGC Working Groups that deal with standards for working with 3D geospatial data.

- The 3D Information Management (3DIM) Domain Working Group(DWG) [http://www.opengeospatial.org/projects/groups/3dimdwg] has been selected to review this ER, as its defined scope of work best encompasses the scope of the experiments in this testbed activity.
- A future portrayal working group, which could be established separately from the Styled Layer

Descriptor/Symbology Encoding (SLD/SE) Standards Working Group (SWG) [https://www.opengeospatial.org/projects/groups/sldse1.2swg], would find recommendations relating to portrayal relevant to their ongoing work.

- The CityGML SWG [http://www.opengeospatial.org/projects/groups/citygmlswg] will find the integration of CityGML contents into AR applications informative.
- The CDB SWG [http://www.opengeospatial.org/projects/groups/cdbswg] will be interested in the integration, efficient storage, delivery and use in AR applications of content originally sourced from a CDB datastore.
- The Interoperable Simulation and Gaming DWG [http://www.opengeospatial.org/projects/groups/isgdwg] might also have an interest in the applicability of this report's findings to modeling, simulation, and gaming.
- Other groups might also find these recommendations relevant for various aspects of OGC standards work.

1.4. Recommendations for Future Work

- Conducting more Augmented and Mixed Reality initiatives would be highly beneficial. The work done during this testbed barely scratched the surface of many different topics.
  - A new initiative simply continuing the work done in relation to the scenario selected for this Testbed would be very valuable. Such an initiative could:
    - Perform more field tests;
    - Ensure better registration between virtual objects and the real world;
    - Aim to achieve better client performance;
    - Support additional transfer formats and mechanisms (e.g. glTF, 3D Tiles, I3S);
    - Test the efficiency of tiled content delivery while moving with the AR device over large distances;
    - Develop better batching of 3D data;
    - Perform experiments with additional geospatial data sets;
    - Integrate more annotations and interaction capabilities into applications;
    - Experiment with more powerful devices such as those supporting Android's ARCore and *6 Degrees of Freedom* sensors or Apple's ARKit;
    - Investigate the use of hardware enabling Mixed Reality experiments.
  - Ideally, requirements for any new AR/MR Innovation Program initiative should focus on clear functional objectives of limited scope. The current initiative requirements specified many 3D data standards that should be used for the experiments, but did not present a detailed picture of what needed to be achieved. The initiative ended up dealing simultaneously with multiple 3D data format conversions, transmitting 3D data between services and clients, and rendering 3D objects and annotations as AR content.
  - When describing AR applications and related initiatives, a distinction should be made between different types of AR content such as text and symbol annotations, overlaid 3D features (e.g. to compensate for poor visibility), anchored 3D models, etc.

    - In general, a requirement for using a specific data standard (e.g. CityGML) should ensure an appropriate data set for such standard is readily available before the initiative begins. This would avoid diverting efforts to generate an intermediate dataset for the sole purpose of satisfying a potentially non-essential requirement, when a more direct path may be possible.
    - For AR, data sets should be available covering the location where potential participants are located to properly test AR functionality.
    - One new initiative should look into the specifics of AR based on Computer Vision, such as ways to anchor visual assets based on pattern recognition as was the case in ARML, or recognizing geospatial features for increasing location accuracy to better support in-door scenarios.
    - A new initiative could leverage a highly detailed dataset of a small area, possibly from IndoorGML, point clouds, or other source.
    - Mixed Reality initiatives should consider the use of specialized hardware such VR headsets equipped with stereoscopic cameras and/or new generation AR glasses such as the MagicLeap (or the ODG-R9 if it ever becomes available).
- The WFS extensions for querying 3D models and textures, as well as the concept of harmonizing services (so that these can be retrieved in addition to vector data, imagery, coverages from a single service and end-point) should be considered as part of future initiatives and the ongoing development of the next iteration of OGC services standards. This approach enabled the efficient serving of full CDB datasets in this initiative. An overview of client/server data exchange is presented in Chapter 8, and a full description of the services in Chapter 9.
- The E3D format (described in Appendix A), with its ability to represent 3D models in a very compact manner and requiring very little processing before uploading to a GPU for hardware accelerated rendering should be considered for inclusion as part of future Innovation Program initiatives.
- Contributing support for loading and writing the E3D model format to the Open Asset Importer Library [http://assimp.org/] would be beneficial, as it would result in that format being a lot more interoperable due to the widespread use of that library in modeling and 3D visualization tools.
- GNOSIS Map Tiles (described in Appendix B), with their potential for describing compact tiled vector data, imagery, coverages and now also referenced and embedded 3D models as well as point clouds should be considered for inclusion as part of future Innovation Program initiatives as well as a potential OGC standard.
- AR applications leveraging geospatial data should adopt a classic GIS styling conceptual model which can scale to 3D views (such as described in Appendix C), with extensions specific to AR. This approach allows linking source attributes with the presentation of different types of AR content (e.g. annotations, styled 3D geometry). The conceptual model used in this initiative leverages work done in the Testbed-14 Portrayal Task.

1.5. Document contributor contact points

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

Contacts

Name	Organization
Jérôme Jacovella-St-Louis *(editor)*	Ecere Corporation
振宇 Hao, Chen-Yu (How)	GIS Research Center, Feng Chia University
柏淳 Wen, Bo-Chun (Cherry)	GIS Research Center, Feng Chia University
Marcus Alzona	Keys
Carl Reed	Carl Reed & Associates
Scott Serich	Open Geospatial Consortium

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.

- IETF: RFC 7946, The GeoJSON format [https://tools.ietf.org/html/rfc7946]
- OGC: OGC 18-053r2, OGC® 3D Tiles Specification 1.0 [http://docs.opengeospatial.org/cs/18-053r2/18-053r2.html]
- OGC: OGC 12-132r4, OGC® Augmented Reality Markup Language 2.0 (ARML 2.0) [http://docs.opengeospatial.org/is/12-132r4/12-132r4.html]
- OGC: OGC® CDB Standard 1.1 (multiple volumes) [https://www.opengeospatial.org/standards/cdb]
- OGC: OGC 09-025r2, OGC® City Geography Markup Language (CityGML) Encoding Standard 2.0 [https://portal.opengeospatial.org/files/?artifact_id=47842]
- OGC: OGC 10-129r1, OGC® Geography Markup Language (GML) - Extended schemas and encoding rules [https://portal.opengeospatial.org/files/?artifact_id=46568]
- OGC: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: OGC 14-005r5, OGC® IndoorGML [http://docs.opengeospatial.org/is/14-005r5/14-005r5.html]
- OGC: OGC 05-078r4, OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification [http://portal.opengeospatial.org/files/?artifact_id=22364]
- OGC: OGC 05-077r4, OpenGIS Symbology Encoding Implementation Specification [http://portal.opengeospatial.org/files/?artifact_id=16700]
- OGC: OGC 09-025r2, OGC® Web Feature Service 2.0 [http://docs.opengeospatial.org/is/09-025r2/09-025r2.html]
- OGC: OGC 07-057r7, OpenGIS Web Map Tile Service Implementation Standard [http://portal.opengeospatial.org/files/?artifact_id=35326]
- OGC: OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]

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.

Augmented Reality (AR)

The integration of visual objects (e.g. labels, virtual 3D objects) with a viewer's actual surroundings.

Virtual Reality (VR)

Virtual environment experienced by a user in a system (e.g. head mounted display) integrating a stereoscopic display and sensors allowing to re-orient the view with the head's rotation (3DoF), and optionally position as well (6DoF).

Mixed Reality (MR)

The combination of Augmented and Virtual Reality in a HMD or smart glasses combining a stereoscopic display, motion sensors and (ideally stereoscopic) cameras.

3.1. Abbreviated terms

3DPS	3D Portrayal Service
ADE	Application Domain Extension
API	Application Programming Interface
AR	Augmented Reality
ARML	Augmented Reality Markup Language
ASSIMP	Open Asset Import Library
CDB	OGC CDB (a datastore standard for 3D environments, formerly Common Database standard)
CMSS	(GNOSIS) Cartographic Map Style Sheets
COLLADA	Collaborative Design Activity (3D interchange data format)
CPU	Central Processing Unit
CV	Computer Vision
DAE	Digital Asset Exchange (COLLADA)
DoF	Degrees of Freedom
DWG	Domain Working Group
E3D	Ecere 3D Model Format
ER	Engineering Report
ETC2	Ericsson Texture Compression, version 2
FoV	Field of View

glTF	GL Transmission Format
GIS	Geographic Information System
GIS-FCU	GIS Research Center, Feng Chia University
GML	Geography Markup Language
GMT	GNOSIS Map Tiles
GPS	Global Positioning System
GPU	Graphical Processing Unit
HMD	Head mounted display
HTTP	Hypertext Transfer Protocol
IBO	Index Buffer Object
ID	Identifier
IFC	Industry Foundation Classes
MR	Mixed Reality
NDK	Native Development Kit (Android)
OGC	Open Geospatial Consortium
OSM	OpenStreetMap
PBR	Physically based rendering
SDK	Software Development Kit
SE	Symbology Encoding
SLD	Styled Layer Descriptor
SWG	Standard Working Group
UMS	Unified Map Service
VBO	Vertex Buffer Object
VR	Virtual Reality
WFS	Web Feature Service
WMTS	Web Map Tile Service
XML	eXtensible Markup Language

Chapter 4. Overview

Chapter 5 explores the possibilities of Augmented Reality in a geospatial context, as they apply to the various features of interest for the experiments. It delves into the portrayal aspect of geospatial data to augment reality, and the similarities and particularities of styling geospatial features for AR, for 3D views and in general, proposing a common conceptual model. It describes how the presentations can link annotations to attributes of the source data using classic GIS approaches. It also skims over the challenges of in-door Augmented Reality.

Chapter 6 provides a description of the various sensors found in mobile devices making Augmented Reality possible, and explains how they can be used to situate the position and orientation of the viewer in the real world.

Chapter 7 describes the two mobile Augmented Reality clients that were developed in this testbed. The first is an Android client, built by Ecere, using GNOSIS and OpenGL and accessing the Android sensors and Camera 2 API directly. The second was built by GIS-FCU, using ARKit, and runs on iOS.

Chapter 8 details the challenges of transmitting large geospatial contents, including 3D contents, efficiently from extended Web Feature Services. An overview of the extended WFS requests used for exchanging data is presented. The rationale behind opting for the E3D model format to deliver 3D meshes to the AR clients in this initiative is explained. The topic of texture compression is briefly covered. Because how 3D data is organized and transmitted by the server to the client, 3D performance considerations are also discussed. Tiling and caching strategies as well as support for clients working completely offline are explored. An attempt is made to describe a potential relationship between the work of this initiative and the 3D Portrayal Service.

Chapter 9 describes the two geospatial services that were built to support these Augmented Reality experiments, by Ecere and GIS-FCU.

Chapter 10 describes in details the datasets used for these experiments, their formats, as well as how they were processed and consumed by the services before being served to the mobile AR clients.

Appendix A details the specifications of the E3D model format used by the experiments.

Appendix B describes the latest specifications of the GNOSIS Map Tiles with support for referencing or embedding 3D models.

Appendix C presents an overview of a conceptual model and GNOSIS Cascading Map Style Sheets language for styling geospatial features, including describing AR annotations.

Chapter 5. Augmented Reality

In order to provide a useful Augmented Reality experience, choosing the right type of data to present, as well as how to present that content for the best AR experience is crucial. This initiative focused on presenting geospatial data, and factors such as the limited availability of data and specific pre-established requirements for the initiative (but no pre-identified data sets) posed some challenges in identifying relevant data to present. Many aspects of AR work best with highly detailed data, which is typically more difficult to obtain.

5.1. Augmenting reality with geospatial data

As the use of CityGML was an important objective of this initiative, attempts were made to identify useable data sets, as well as CityGML Application Domain Extensions (ADEs) which may provide information more particularly useful for Augmented Reality experiments. Existing CityGML ADEs identified included:

- The GeoBIM ADE [http://www.citygmlwiki.org/index.php?title=CityGML_GeoBIM_ADE], based on Industry Foundation Classes (IFC). [2]
  (Also interesting is this open-source tool [https://github.com/tudelft3d/ifc2citygml] for automatically converting IFC data to CityGML)
- The Energy ADE [http://www.citygmlwiki.org/index.php/CityGML_Energy_ADE] [3]
- The Utility Network ADE [http://www.citygmlwiki.org/index.php/CityGML_UtilityNetworkADE]

Also relevant are the open source BIM collective [4], Indoor Spatial Data Model Using CityGML ADE [5], application of large 3D CityGML models on smartphones [6] and Augmented Reality Games with CityGML [7].

However, as it was already difficult to find a basic usable CityGML data set in an area of interest, let alone one defining any of these ADEs, no such data sets were identified.

Being able to see "hidden" features is a very useful aspect of AR, displaying underground utilities such as pipelines, is a good use case scenario. A data set of a pipeline in Kaohsiung, Taiwan available in the COLLADA format was used for some experiments in the Testbed. The location of the content however, proved problematic as no testbed participant was normally located in Kaohsiung.

The Testbed sponsors originally specified requirements for supporting 3D Tiles and/or I3S. They also hoped for a continuity of the work performed in Testbed-13 on 3D client performance. As a result, the focus shifted on the presentation of 3D buildings in an AR application. One problem with the presentation of 3D buildings in AR is that the actual buildings are normally directly visible, without the need for augmentations. As such, a 3D model would simply obstruct a real, better view of the actual building. However, having the 3D model could potentially be useful in an emergency response scenario where thick fog or smoke could severely restrict visibility. Alternatively, presenting the buildings with a reduced opacity setting may be useful to verify the accuracy of a data set with the real world, and/or for mapping one’s surroundings. The vast majority of the experiments performed in this initiative dealt with this 3D buildings scenario.

Unfortunately, the data sets used had little attribute information available. However, the approach

implemented for transforming and transmitting buildings data kept into consideration the capability for preserving attributes when better attributed data is available.

In order to experiment with the capacity to display actual annotations apart from 3D models, separate and combined experiments were done using OpenStreetMap data for presenting street maps, including buildings and streets names.

Although desktop experiments were done with terrain elevation and aerial imagery, the very limited nature of mobile hardware, as well as the desire to not hide the majority of the camera feed made it difficult to find a proper use case for these data in a mobile Augmented Reality scenario. In one case however, information from the elevation model data was incorporated within the buildings position information.

See Chapter 10 for a detailed description of the data sets used for all experiments, as well as how they were processed and integrated for use by the services and Augmented Reality clients.

In this initiative, generic geospatial datasets, with no special consideration for Augmented Reality, were used so that AR applications developers could readily make use of any geospatial datasets such as those defined or transmitted using OGC standards. Similarly, current GIS applications developers can leverage their existing tools, data and expertise to start deploying Augmented Reality solutions.

5.2. Similarities and particularities of styling for 3D views

Many of the same concepts for styling geospatial features apply whether dealing with a cartographic or 3D view (whether doing Augmented Reality or not). As an example, any flat feature can be simply draped on the ground, and the same stroke or fill symbolizers can be applied to the feature(s). Labels and markers (annotations) typically still face the user as billboards, even if their 3D position serves as a basis for projecting to a screen position.

Some additional capabilities are possible however, such as defining styles specific to solid shapes / volumes, or using 3D models as markers. There is also the concept of depth of objects, which is sometimes useful to have objects sorted back to front with actual depth values.

With this in mind, rather than defining something entirely distinct, it makes sense to extend the classic GIS styling model with 3D capabilities.

5.3. A common flexible geographic features styling conceptual model

A great advantage of unifying styling representations is to be able to easily integrate any styled GIS content within an AR application.

This is done through the same mechanism as specifying cartographic styling for vector features in a typical cartographic 2D view, with a symbolizer concept associated with a feature described in a generic manner which accommodates 3D views just as well as 2D views.

As part of the ongoing next iteration of defining a standard conceptual model for styling capabilities, these possibilities were also considered within the Testbed-14 portrayal thread activities.

This conceptual model for styling features, as well as an encoding for it, is detailed in Appendix C.

5.4. Linking to attributes from source data in annotations

By configuring how the Augmented Reality content is displayed on either the camera feed, or the rendering of a virtual 3D view of the integrated urban model represented by the CityGML, entirely on the client, a great level of flexibility is gained. This linking can be established by specifying styling rules, and referencing available attributes associated with the geospatial data. In the approach used in this Testbed, the description of annotations elements was not hardcoded in either the source CityGML or the transmission format.

5.5. Extended capabilities applying specifically to AR

Capabilities very specific to Augmented Reality could be extensions to a conceptual model for styling geospatial data. This would provide a mechanism by which to re-integrate ARML capabilities within this conceptual model.

5.6. Indoor localization challenges: pattern matching, computer vision

At a very close range, the GPS precision is not enough to accurately register the camera with the surroundings. Indoor, the GPS signal is blocked, and devices cannot communicate with satellites. Virtual sensors such as the Android fusion sensor providing 6 degrees of freedom integrating the gyroscope, accelerometer, magnetometer in addition to input from the camera may solve some of these challenges. However, potentially recognizing features from a pre-existing data set and/or leveraging more advanced computer vision techniques on the camera feed could help solve some of these challenges.

Access to detailed in-door data sets is key for looking at these challenges in the context of Augmented Reality. Perhaps insights from the Indoor mapping and Navigation pilot [https://www.opengeospatial.org/projects/initiatives/indoor-pilot] would prove useful. Additionally, hardware with stereoscopic cameras might offer better ways to more accurately register the view. It is also possible to apply pattern recognition, and place markers in the real world at specific locations where augmentations should appear.

Finally, augmentation of reality at a smaller scale is generally more interesting and useful, making it possible to display more relevant content which is part of the user's immediate surroundings. The heavy reliance on the complex topic of computer vision however implies more involved processes, and for this reason this was reserved for a future initiative.

Chapter 6. Situating reality in a geospatial context

Augmented Reality applications leverage many embedded mobile device capabilities to merge the user's real surroundings together with virtual elements (augmentations). These include the camera, the Global Positioning System (GPS) chip, as well as sensors such as the gyroscope, accelerometer, magnetometer and even the barometer (as an altimeter).

In the context of a geospatial Augmented Reality application, the user's location is also related to an absolute global frame of reference, typically through the location information provided by GPS coordinates.

This chapter provides an overview of these capabilities and how they each play their role in making AR possible on a mobile device. This chapter is written from the perspective of the Ecere Android client making direct use of the camera and sensors, but the same fundamental concepts are utilized by toolkits such as ARKit on iOS.

6.1. The camera

The (rear-facing) camera of the mobile device provides a live feed of the user's surroundings. It supplies an application with the backdrop in which augmentations can be integrated so that the user feels as though these are part of their actual surroundings. In computer vision-based Augmented Reality, the camera feed is not merely a backdrop but is analyzed in real time to identify anchoring patterns or to precisely locate features. Through the use of virtual fusion sensors such as Android's six degrees of freedom sensor (TYPE_POSE_6DOF [https://developer.android.com/reference/android/hardware/Sensor.html#TYPE_POSE_6DOF]), the camera can also play an integral part in maintaining accurate information about the position and orientation of the device.

The Ecere client directly accessed the Android camera using the Android Camera 2 API [https://developer.android.com/reference/android/hardware/camera2/CameraDevice]. The Ecere client is built using the native eC programming language [http://ec-lang.org] rather than Android's Java development language. Further the NDK API for the Camera 2 was only added in Android API Level 24 (corresponding to Android 7.0 — Nougat). As there was a desire to support devices with an earlier version of Android, the Java Native Interface (JNI) had to be used. This decision required writing extra code to make use of that Camera API, thus complicating the task of implementing camera support. By using the Camera 2 API through JNI, it was possible to support Android devices starting from API Level 21 (corresponding to Android 5.0 — Lollipop). This code will likely be published under an open-source license as part of the Ecere SDK, and may be re-usable in other projects (including in projects written in other native languages such as C or C++, with minor modifications).

That camera code uses a capture session set up with repeating requests to continuously capture images. Using a lower resolution than the camera is capable of capturing can provide smoother performance. The performance of the current setup is still being investigated as it is believed that performance can be further improved, potentially by taking a more direct path between the captures and displaying the image on the device.

6.2. The GPS (and other means of determining locations)

Obtaining the GPS coordinates is the most accurate means for retrieving the device's location. However, using the on-board GPS chip only works outdoors, and requires significant power. The network location mechanism can use the cell tower and Wi-Fi triangulation, but these computed locations are fairly inaccurate. [1: By using cell tower triangulation (3 towers), it is possible to determine a phone location to within an area of about ¾ square mile. https://transition.fcc.gov/pshs/911/Apps%20Wrkshp%202015/911_Help_SMS_WhitePaper0515.pdf]

Ideally for a geospatial AR application, the GPS unit should be used to provide coordinates. This could prove a real challenge for indoor Augmented Reality applications (possibly the location could first be obtained as close as possible from outside the building). Using the GPS provides the absolute frame of reference for the device's position in relation to the Earth.

Because the Android NDK does not provide an API for obtaining the GPS coordinates and location updates, the JNI also had to be used to set up location updates and access the GPS chip in the Ecere Android client, through the LocationManager [https://developer.android.com/reference/android/location/LocationManager] class.

In addition to latitude and longitude, the on-board GPS chip can provide altitude information. The altitude returned by Location.getAltitude() will be the altitude above the WGS84 reference ellipsoid, not the mean sea level. The latter can be obtained by taking into account the difference between the EGM96 geoid and the WGS84 reference ellipsoid, which varies between a difference of -100 and 80 meters).

This altitude information from the GPS has not yet been used in the Ecere client.

6.3. Motion sensors

A guide to the Android motion sensors can be found here [https://developer.android.com/guide/topics/sensors/sensors_motion].

6.3.1. The magnetometer

A magnetometer senses the Earth's magnetic field to establish the direction of magnetic north, much like a classic compass. The magnetometer is the sensor most likely to suffer from distortion errors, due to a failure to initially be calibrated properly, or to be de-calibrated by the presence of magnetic materials nearby whose magnetic field could be much more significant than the Earth's. It is possible to re-calibrate the magnetometer by waving the device following a figure-8 pattern over one's head.

Remember that even when properly calibrated, the magnetometer indicates the magnetic North, and not the Geographic north. The local angular difference between the two, the magnetic declination, varies from one place to another on the globe, as well as over time as the magnetic pole moves (due to magnetic changes in the Earth's core). This may have partially contributed to the difficulty of registering augmentations with the real world during the Testbed experimentation. For example, the magnetic declination is only approximately 4 degrees in Taiwan, but around 13.5

degrees in Eastern North America. Natural Resources Canada provides a convenient online calculator [http://www.geomag.nrcan.gc.ca/calc/mdcal-en.php] for the magnetic declination based on a given date, latitude and longitude.

The magnetometer, together with the gyroscope and accelerometer, allows establishing the 3D orientation of the device.

6.3.2. The gyroscope

The gyroscope sensor reports angular velocity. Together with the magnetometer and the accelerometer sensing the Earth's gravity, which is used to establish the ground plane, an absolute 3D orientation can be calculated.

6.3.3. The accelerometer

An accelerometer senses linear acceleration as the device is moved. The sensor perceives the acceleration due to gravity at the Earth's surface (*g*, roughly 9.8 m/s^2), which must be accounted for, but which also provides the horizontal frame of reference from which the conceptual electronic compass will interpret the magnetometer input.

6.3.4. The barometer (as an altimeter)

Mobile devices featuring a barometer that measure atmospheric pressure can use this information to establish a much more accurate altitude (at least in relative terms), based on a reference sea level pressure. The challenge is to obtain an accurate reference sea level pressure in order to get an accurate absolute altitude.

6.3.5. 3D orientation

By fusing the information measured by the magnetometer, gyroscope, and accelerometer, an absolute 3D orientation of the device can be determined.

Because the 6 degrees of freedom pose sensor was not available on Android devices used for the experiments, the TYPE_ROTATION_VECTOR [https://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ROTATION_VECTOR] was used. This Android sensor type fuses information from the magnetometer, gyroscope and accelerometer to provide a rotation vector in the form of a quaternion. [2: When used to represent rotation, unit quaternions are also called rotation quaternions as they represent the 3D rotation group. When used to represent an orientation (rotation relative to a reference coordinate system), they are called orientation quaternions or attitude quaternions. (Wikipedia, 2019)] The 4th component (w) of the quaternion is optionally provided on some devices (always provided from SDK Level 18 onwards), but it can otherwise be implied from the other 3 components as the quaternion is normalized. During the experiments, it was found that special care has to be given to remap the coordinate system based on the default rotation of a device, to which the sensors information is relative. Most mobile devices have a portrait default rotation, but some tablets (such as the Nexus 10) default to a landscape orientation.

In the Ecere client, further transformation had to be done to convert this orientation to the GNOSIS coordinate system, requiring a 90 degrees pitch rotation, then a negation of the roll component of the orientation. Finally, the magnetic declination has to be taken into account for the yaw

component.

6.3.6. 3D position

By performing a first integration of linear acceleration, velocity can be determined, and by a second integration, a position can be determined. Such a position would not be very useful in a geographic context, as it would only be relative to the user's original position. However, by relating this position at the precise time where a location update from the GPS was received, the relative movement provides the necessary intermediate position updates until the next location update is received. More involved calculations can smooth these calculations over time based on past discrepancies and previsions. A virtual fusion sensor could also leverage computer vision using a device's camera, such as an Android 6 degrees of freedom pose fusion sensor may do on recent devices [8].

On Android devices, it would seem that this type of comprehensive fused position and orientation information is only readily available through this particular sensor. As no device supporting it was available for the experiments, and as performing these calculations manually is quite involved, there was no time during the initiative to implement this intermediate position aspect for the Ecere client (positions currently only get updated when location updates are received).

6.4. Size, weight and power usage considerations

The use of location updates, sensors and the camera capabilities is particularly costly in terms of power. Turning off such capabilities when not required is essential to minimize this impact. This is in addition to potentially heavy use of the Graphical Processing Unit (GPU) for rendering a large number of items and/or 3D geometry. This makes Augmented Reality applications rather power hungry, as they can quickly drain a device's battery. Correspondingly, it also produces a lot of heat, and the size of a device can be related to both its ability to dissipate this heat and the capacity of its battery. As purely anecdotal information, the Testbed-14 participants saw two laptops and one tablet perish during this initiative!

These are things to consider for field operations where devices would be required to be used for long periods of time without the ability to charge the devices, especially if operations increasingly rely on such Augmented Reality applications. It was beyond the scope of this testbed initiative to perform an analysis of factors such as power usage, heat and battery duration of devices. The small team was kept busy for the entire duration of the testbed by the already significant workload of the primary objectives. These objectives included processing and serving relevant data sets, achieving interoperability between the different services and clients, as well as displaying augmentation properly registering with the real world. This could be investigated in future AR initiatives. However, this would require strictly determining many variables and evaluation criteria so as to produce useful results.

Chapter 7. Augmented Reality Clients

Two mobile clients were built by Ecere and GIS-FCU during this Testbed-14 Augmented Reality initiative.

See also Chapter 6 for more details about how the sensors were used to situate the view in the clients, and Chapter 8 for information about how the 3D contents was streamed from services.

7.1. GNOSIS / Android mobile client by Ecere

Ecere built a client targeting Android mobile devices, leveraging their cross-platform Ecere and GNOSIS SDKs and making use of OpenGL ES, the Android sensors, and the Android Camera 2 API directly. The client would automatically connect to both services and request 3D contents data in a tiled manner based on the current view position. It provided buttons to jump to fixed locations for the datasets (Washington, D.C. and New York City), as well as to the last known GPS position, or to constantly reposition the camera based on GPS location updates.

The client currently requires an Android 5.0+ device, as well as support for OpenGL ES 3.0. It will be possible to lower this requirement to OpenGL ES 2.0 after some fallbacks are properly implemented in the Ecere graphics engine. Application packages (APKs) were built for ARM CPUs, both 32-bit (armeabi v7a) and 64-bit (arm64-v8a), and tested on multiple devices.

There are four buttons in the application:

- **Follow**: Toggling (it is off by default) automatically repositions the camera (latitude, longitude) based on GPS updates. The altitude will also currently be maintained at 2 meters above sea level.
- **Here**: Clicking this button repositions the camera at the last received GPS position
- **NYC**: Clicking this button moves the camera right above the Statue of Liberty
- **DC**: Clicking this button moves the camera inside the Capitol.

If *Follow* mode is **not** toggled on, the user can:

- Pinch the screen with two fingers to zoom in & out (changing altitude and position as well if not looking straight down).
- Slide with one finger to move the camera around
- Double-click to smoothly transition the camera closer where the clicks were performed

Whether *Follow* is on or not, the user can simply rotate the device around freely to update the camera view orientation (yaw/pitch/roll).

These experiments focused on rendering 3D buildings, as streamed by the services.

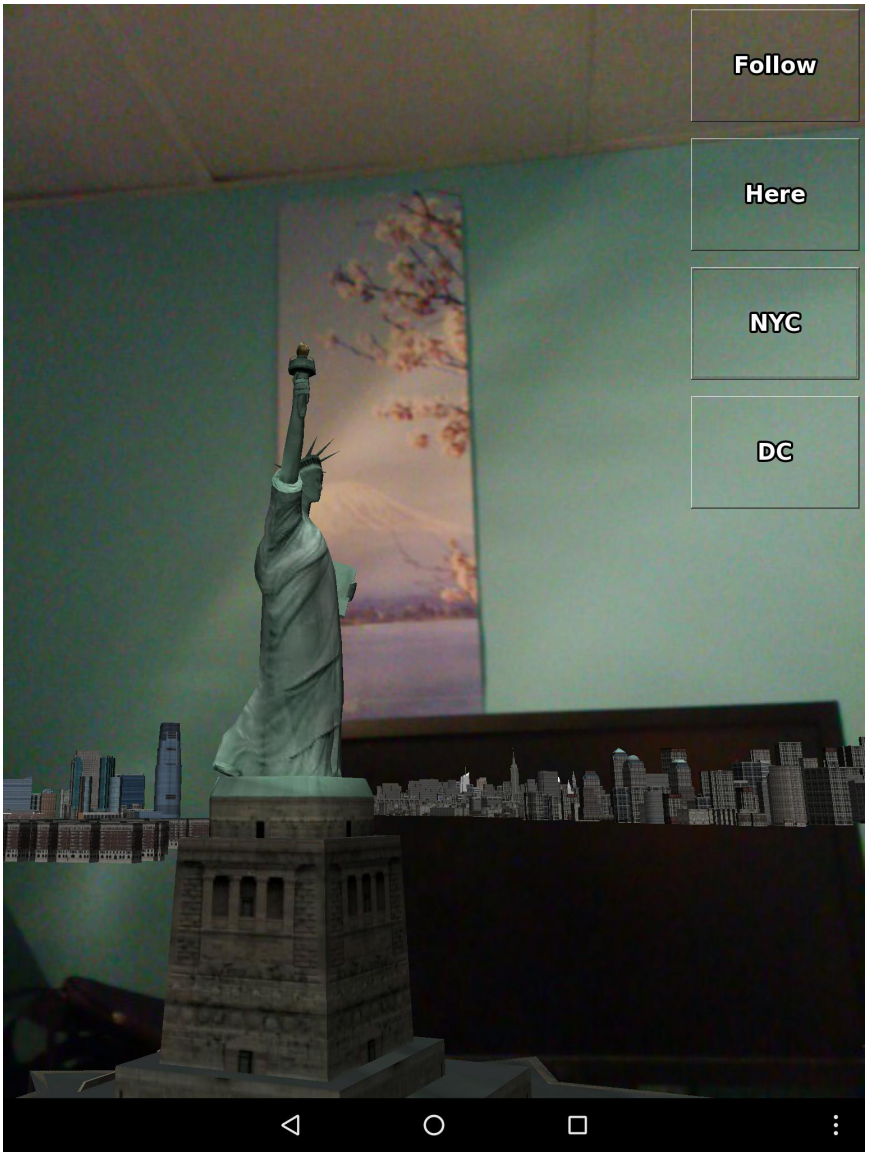

Figure 2. View of Statue of Liberty, with Manhattan in the background, retrieved as E3D from Ecere service, over camera (faking location)

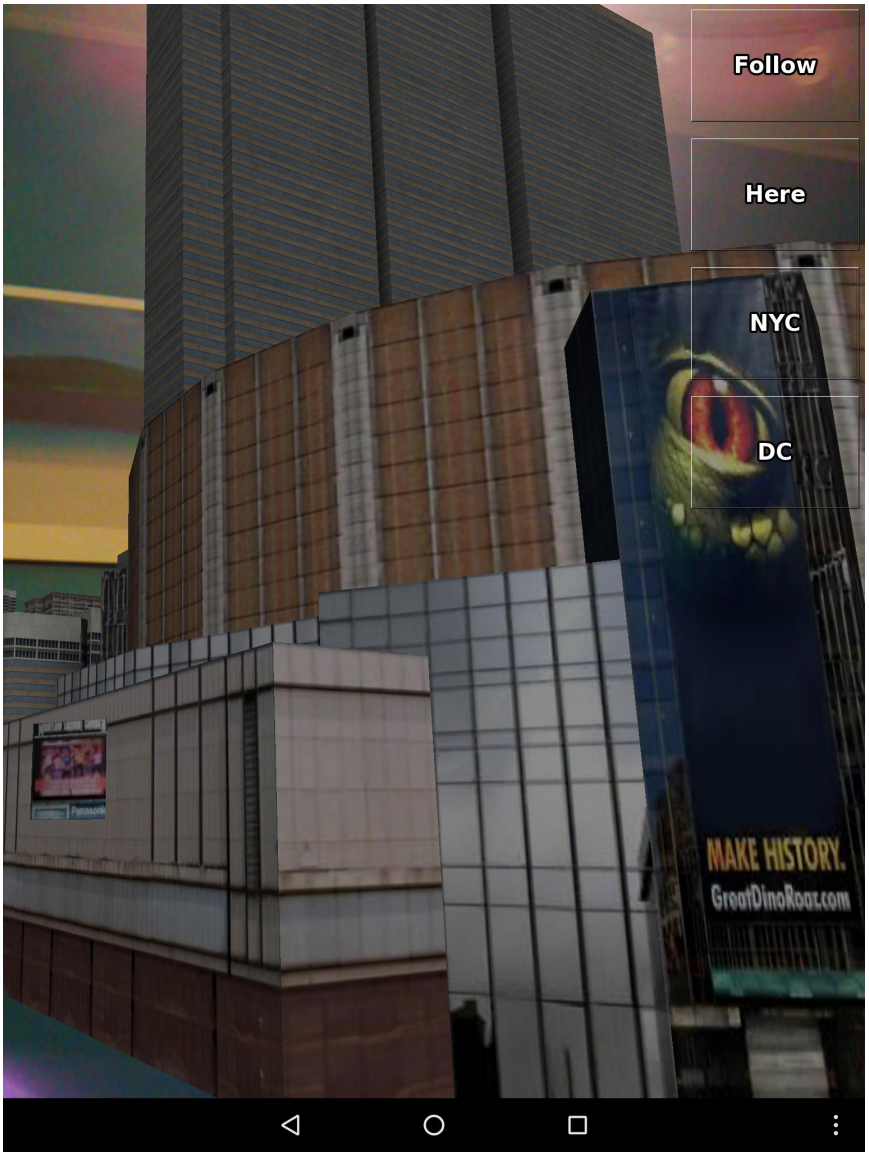

Figure 3. View of New York City theater, retrieved as E3D from Ecere service, over camera (faking location)

Figure 4. View of New York City buildings, retrieved as E3D from Ecere service, over camera (faking location)

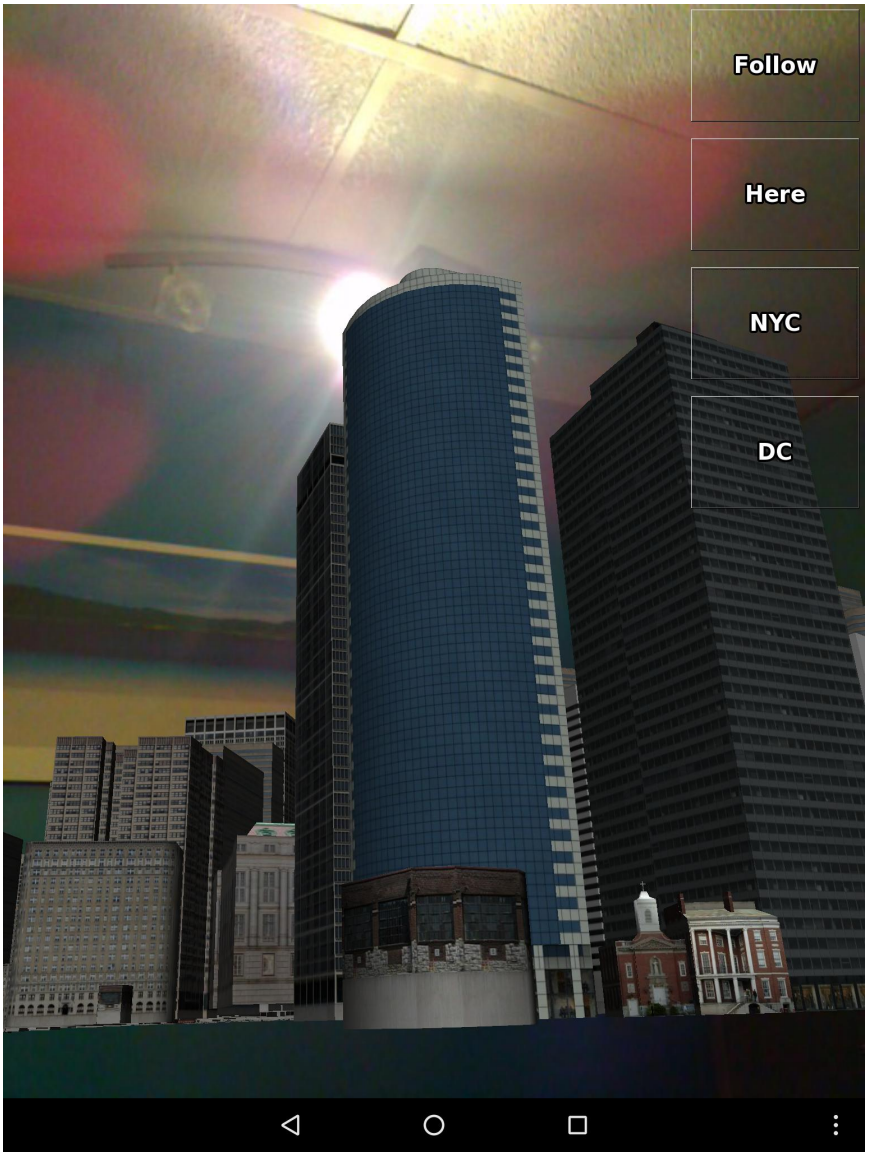

Figure 5. View of New York City buildings, retrieved as E3D from Ecere service, over camera (faking location)

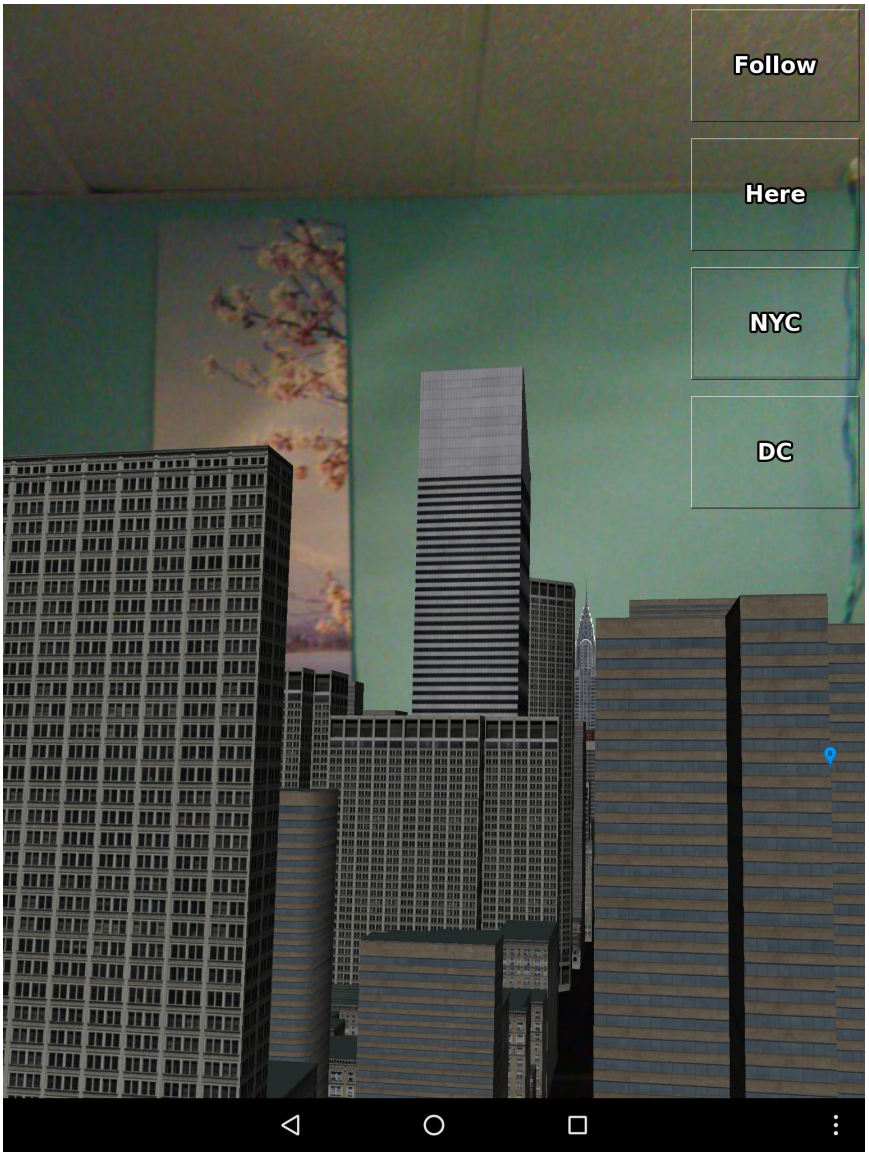

Figure 6. View of New York City buildings, retrieved as E3D from Ecere service, over camera (faking location)

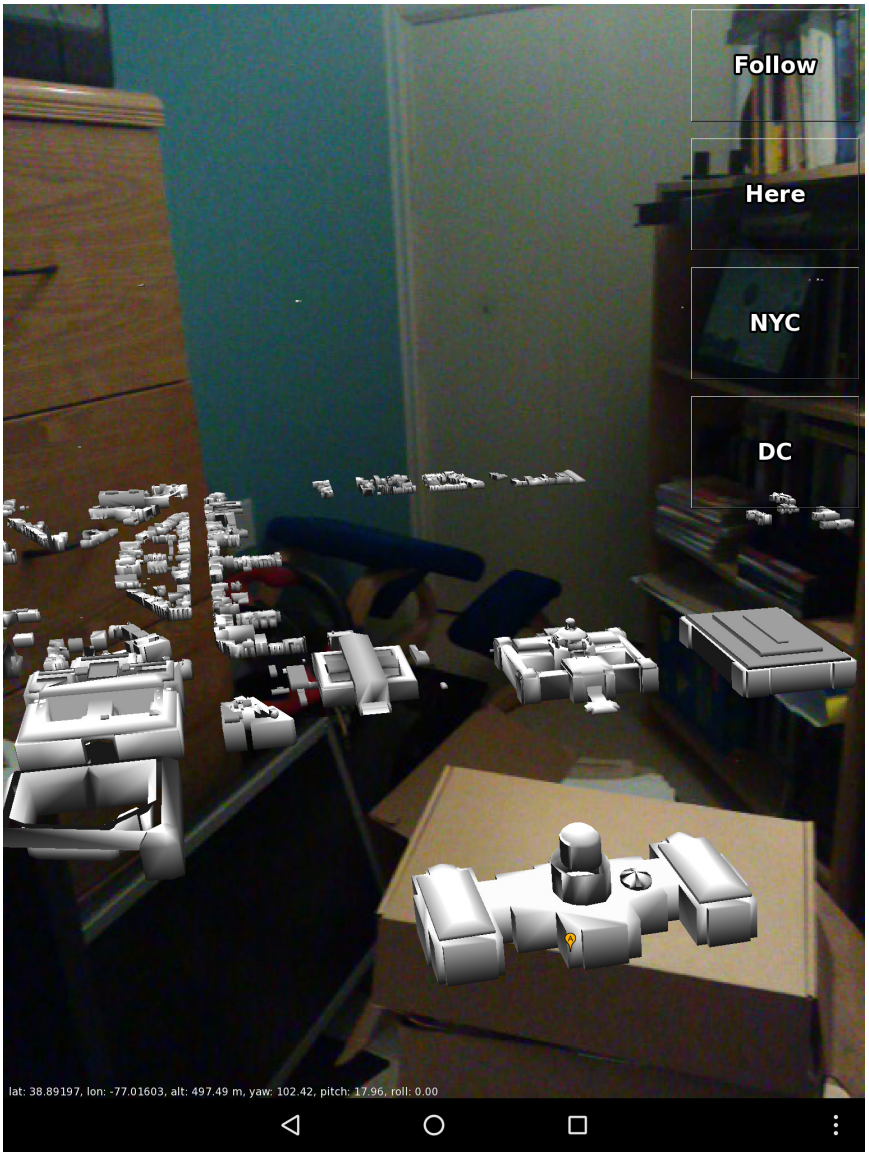

Figure 7. View of Washington, D.C. retrieved as E3D from GIS-FCU service over camera (faking location)

Other experiments were done with OpenStreetMap data, which featured more interesting attributes for use as annotations. Portrayal rules were established using the *GNOSIS Cascading Map Style Sheets* (described in Appendix C) to display labels for roads and buildings, as well as the roads themselves. These rules also included extrusion of buildings footprints (polygon layers) to render them as 3D buildings based on the OpenStreetMap *Simple 3D Buildings* [https://wiki.openstreetmap.org/wiki/Simple_3D_buildings] attributes. [9]

Figure 8. OpenStreetMap data (roads and extruded 3D buildings) around EPRI location

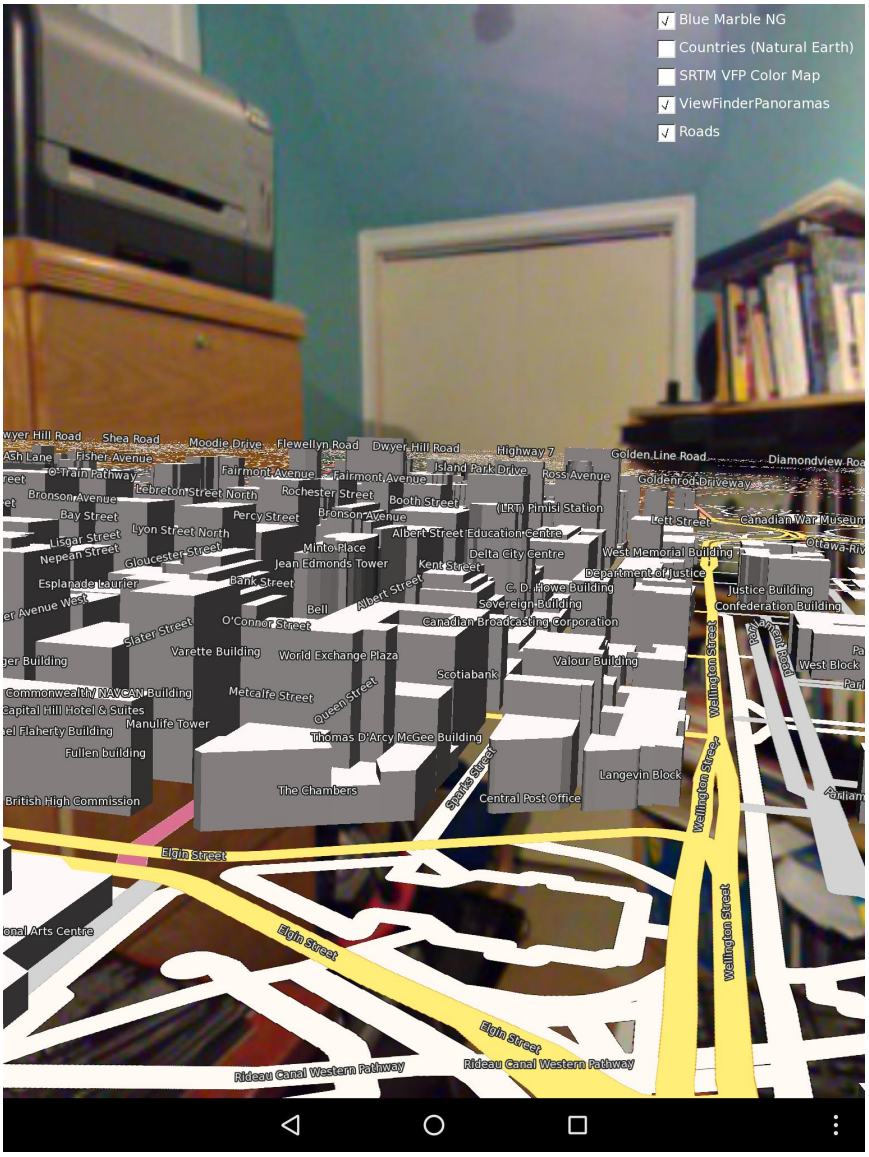

Figure 9. OpenStreetMap data (roads and extruded 3D buildings) in Ottawa (fake location)

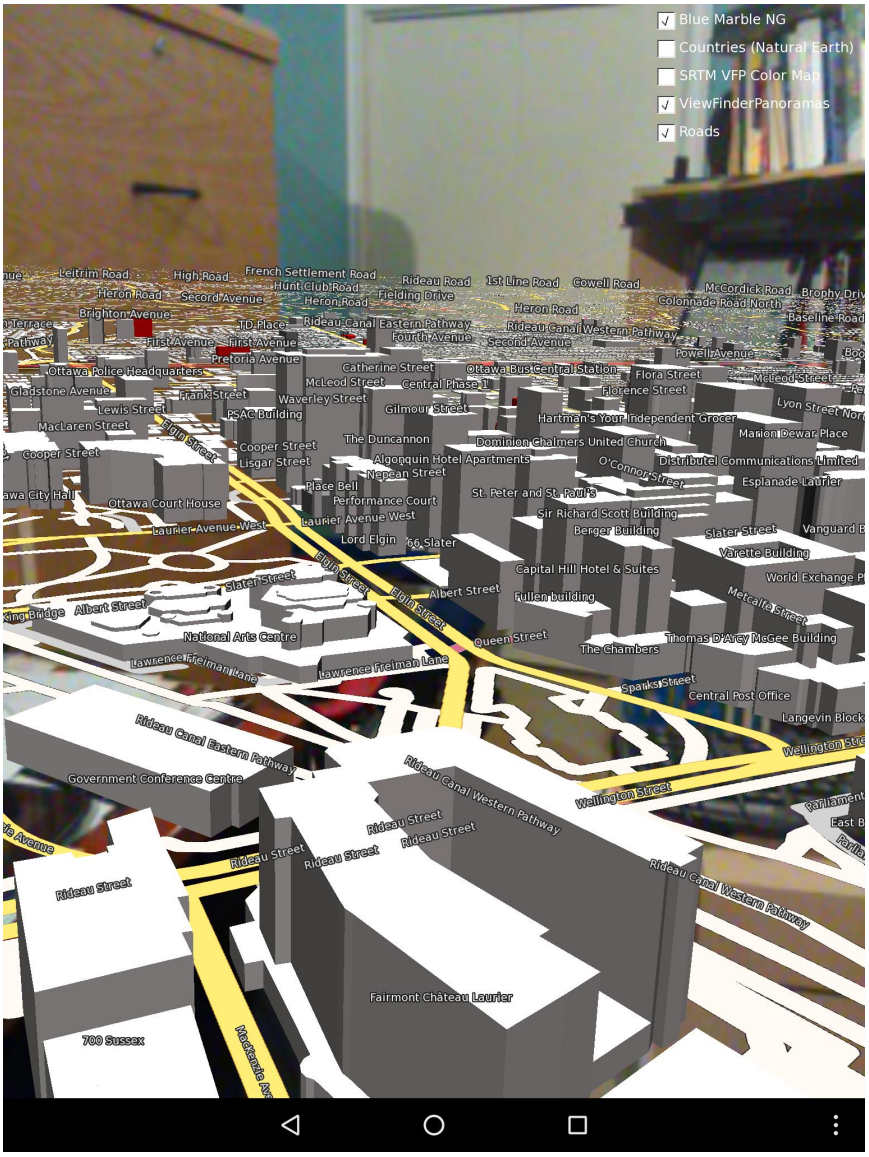

Figure 10. OpenStreetMap data (roads and extruded 3D buildings) in Ottawa (fake location)

OpenStreetMap data from Washington, D.C, was also combined with 3D buildings data from a GIS-FCU service to verify the correctness of the location.

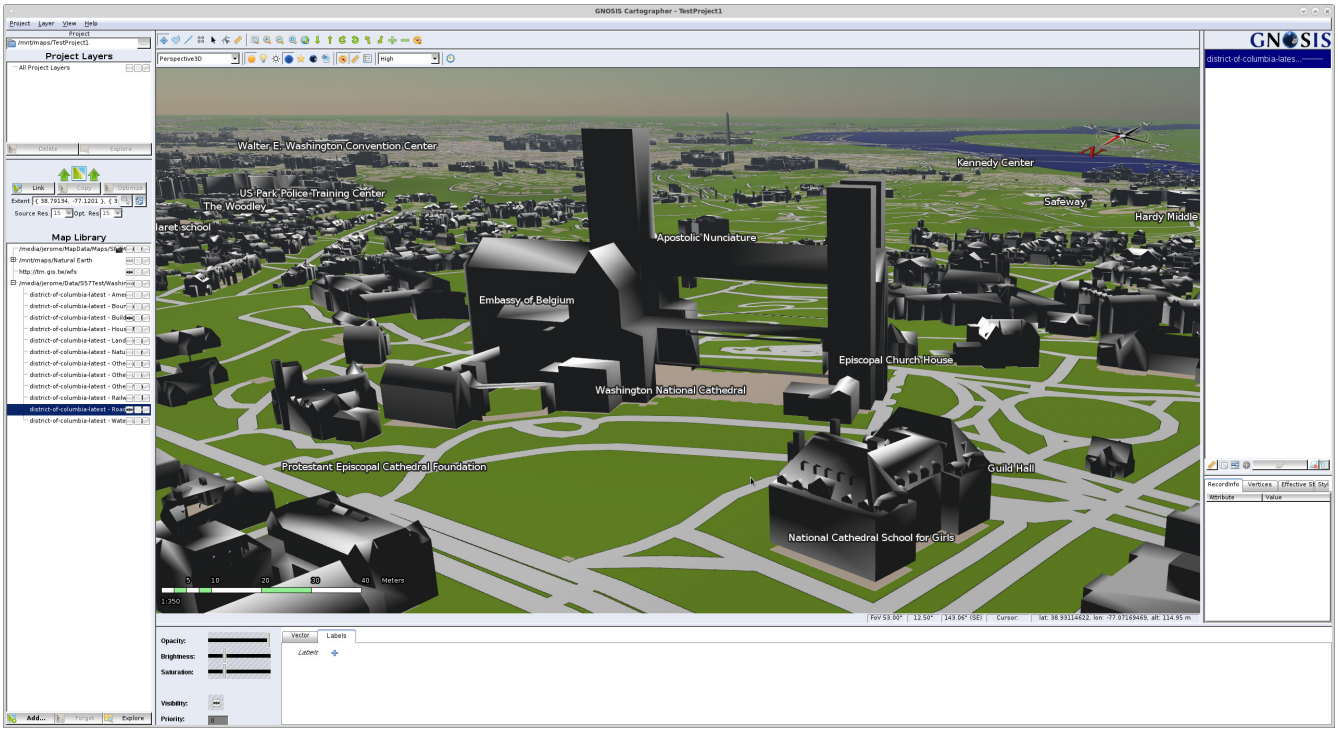

Figure 11. Accessing Washington, DC data from GIS-FCU service in GNOSIS client (together with OSM data)

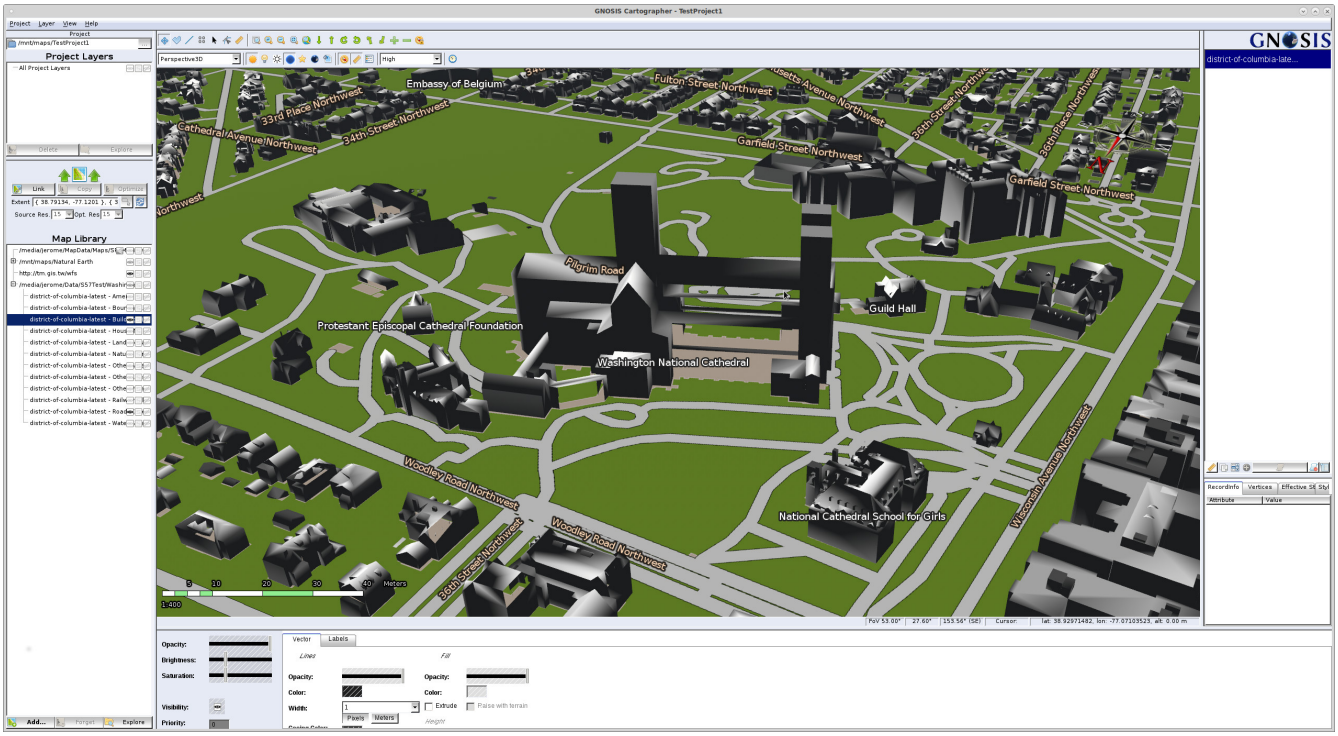

Figure 12. Accessing Washington, DC data from GIS-FCU service in GNOSIS client (together with OSM data)

Some experiments were also done in a desktop version of the client, rendering the vast majority of the New York City CDB data set content (buildings, trees, 3D terrain elevation, high-resolution imagery). Support for CDB geotypical models, such as the trees, was added during this initiative. Those were most notably missing from Central Park in Testbed-13 demonstration. Much effort was also spent optimizing the GNOSIS graphics engine, and work is still on-going to further improve performance.

Figure 13. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (a)

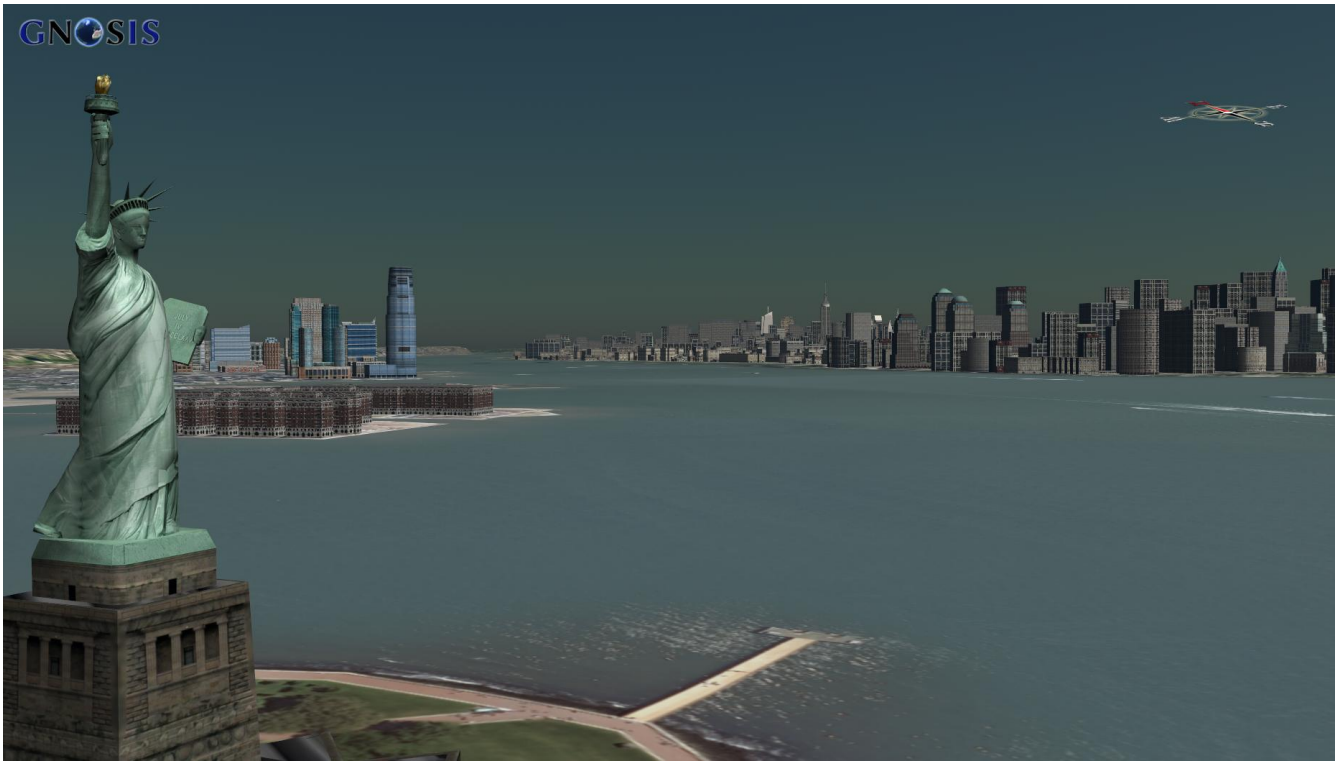

Figure 14. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (b)

Figure 15. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (c)

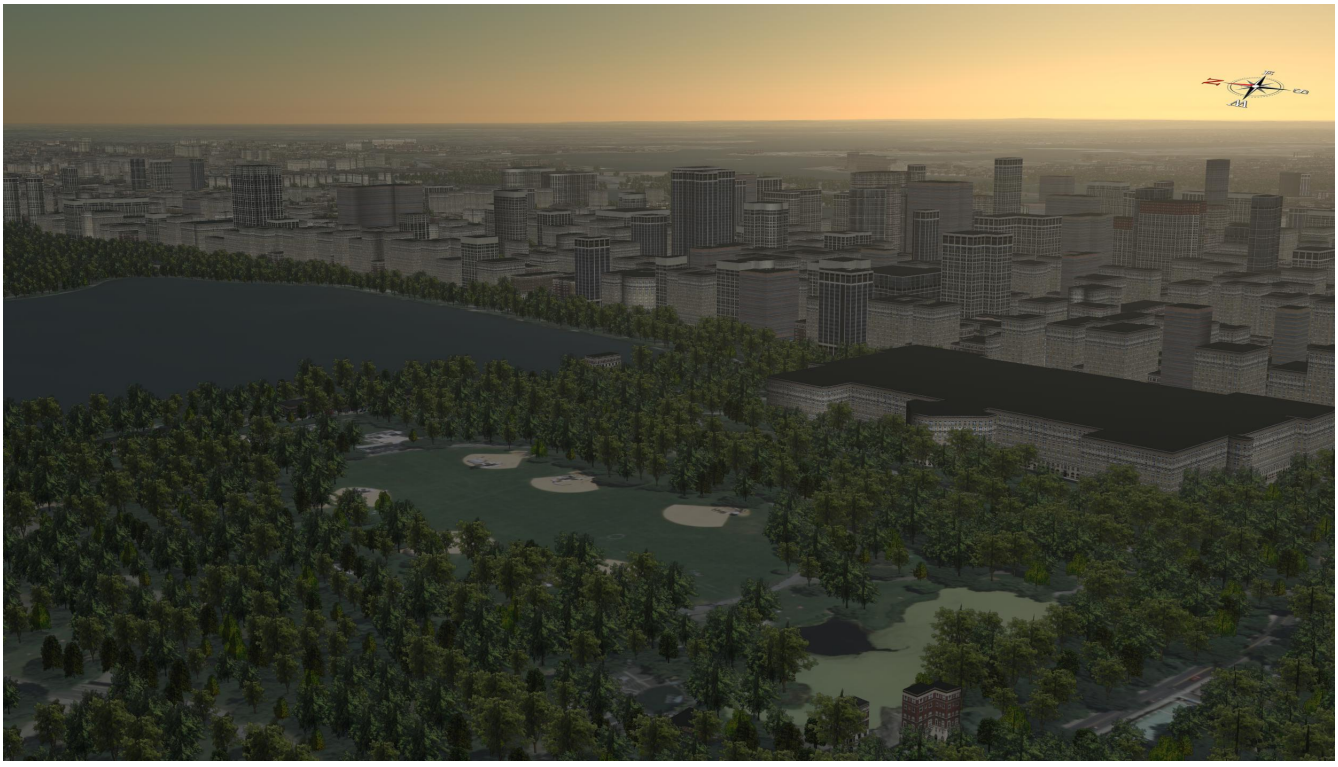

Figure 16. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (d)

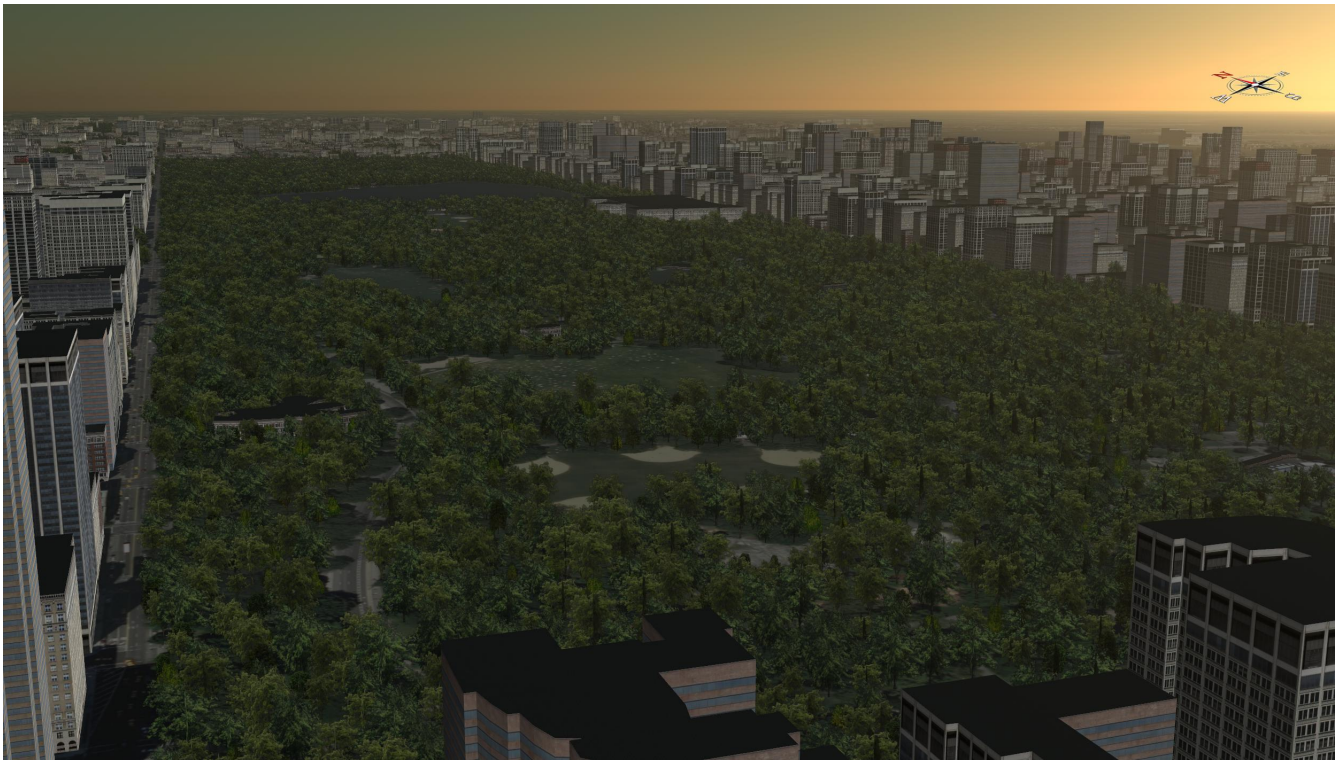

Figure 17. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (e)

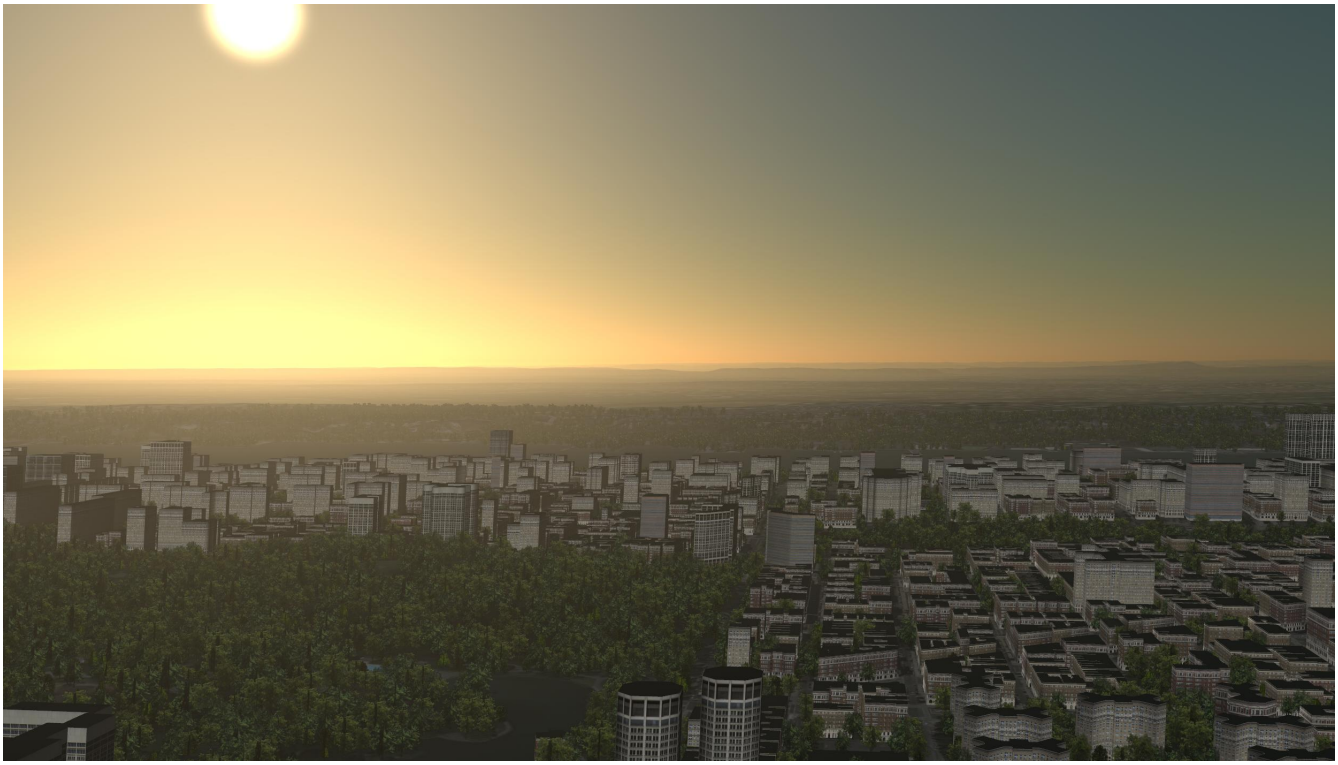

Figure 18. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (f)

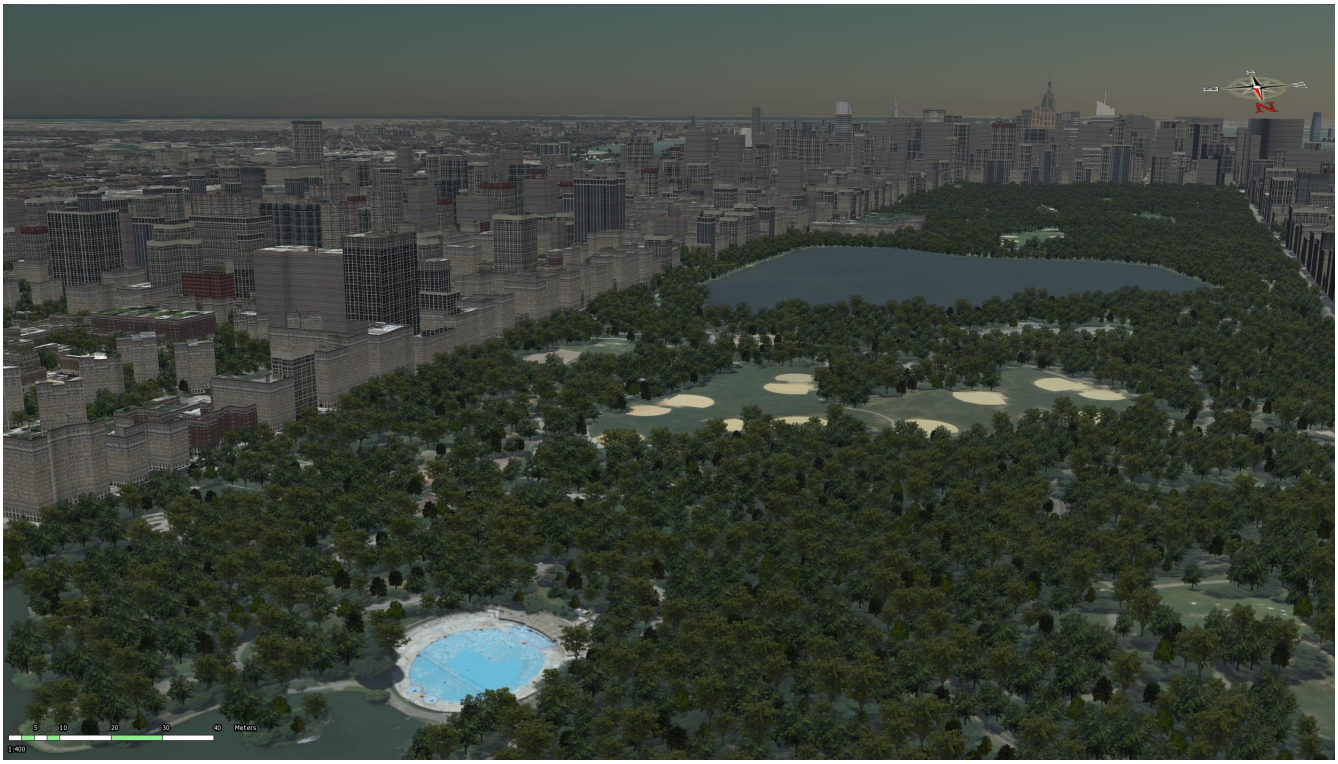

Figure 19. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (g)

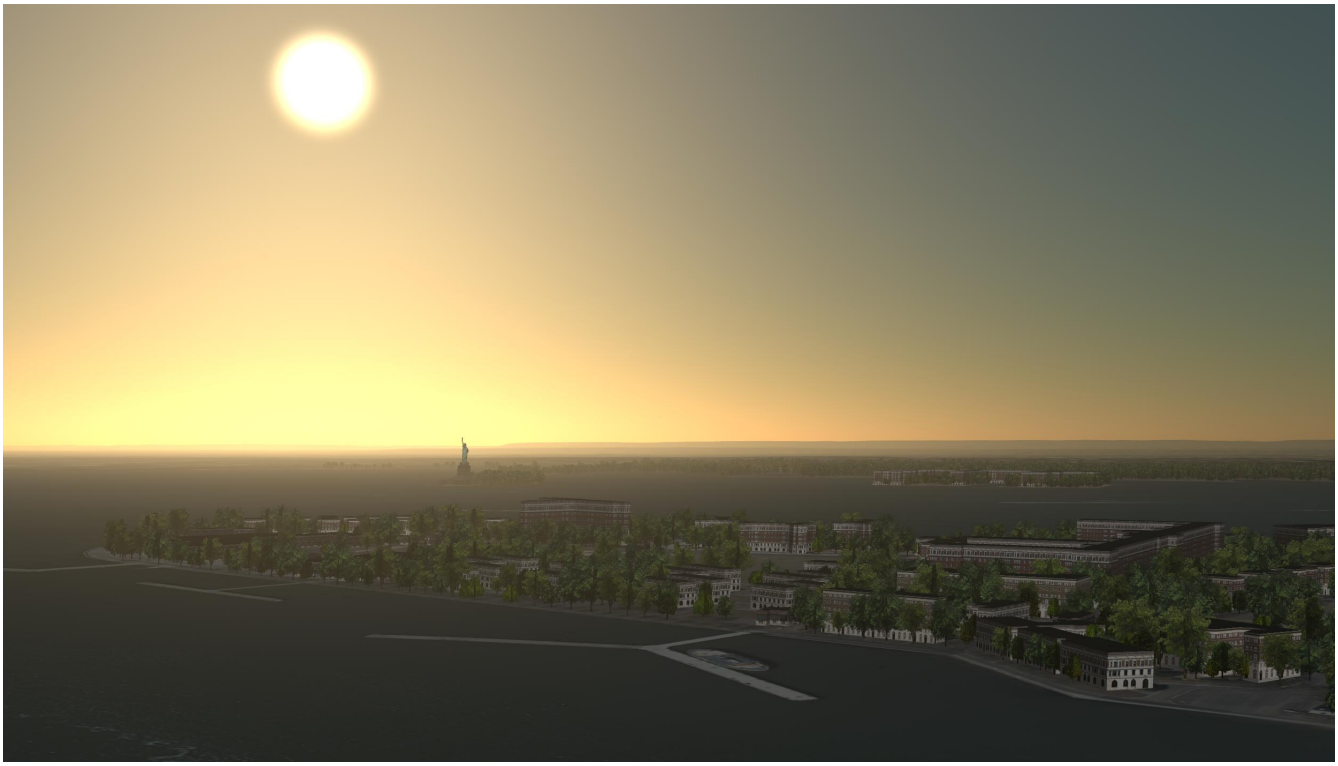

Figure 20. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (h)

Figure 21. Visualizing New York City CDB data (FlightSafety) converted to GNOSIS Map Tiles / E3D Models (i)

7.2. ARKit / iOS mobile client by GIS-FCU

A mobile client targeting iOS, using ARKit, was built by GIS-FCU.

7.2.1. Using ARKit

Here is a brief description of how ARKit and SceneKit were configured within the application:

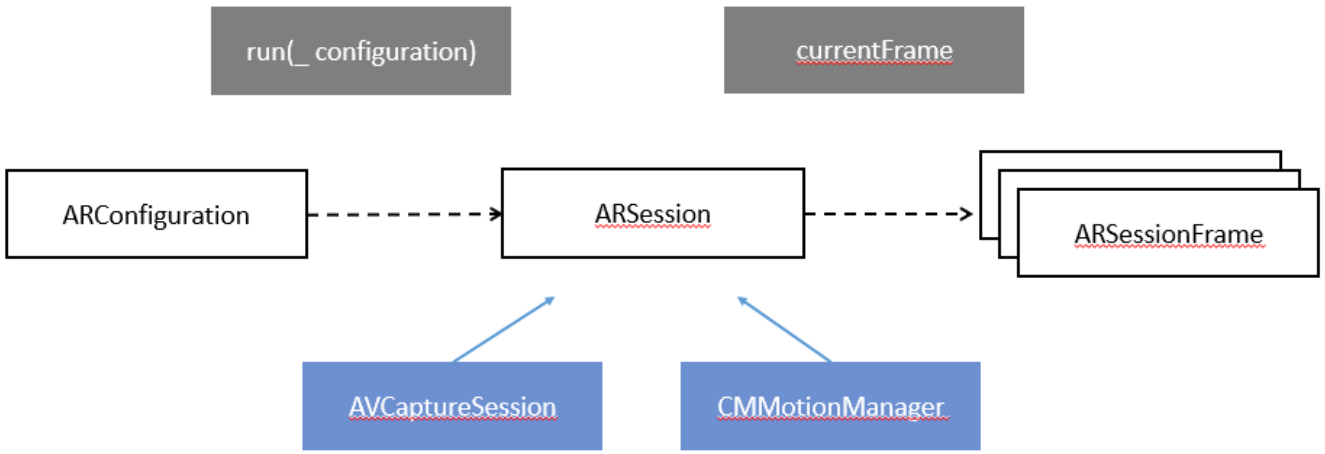

Figure 22. Diagram illustrating ARKit usage in GIS-FCU client

1. Set up the ARKit ARConfiguration to be "ARWorldTrackingConfiguration".
   ARWorldTrackingConfiguration: Use the rear camera to track the direction and position of the device and detect the configuration of the real world plane.
2. Activate an ARSession.(include AVCaptureSession and CMMotionManage)
   AVCaptureSession: Start gathering odometry

CMMotionManage: Start gathering inertial odometry

3. Use Core Location to obtain geographic position.
4. Instantiate the 3D virtual object as an SCNGeometry object having SCNMaterial, SCNLight, SCNCamera.
5. Attach the SCNGeometry object (visible to user) as a child SCNNode of the ARAnchor.
6. Render the scene
7. ARKit will automatically move the 3D virtual objects as the camera moves.

7.2.2. Rendering 3D Models

The actual drawing of E3D models in the client was done as follows:

1. Open the phone application and initialize it

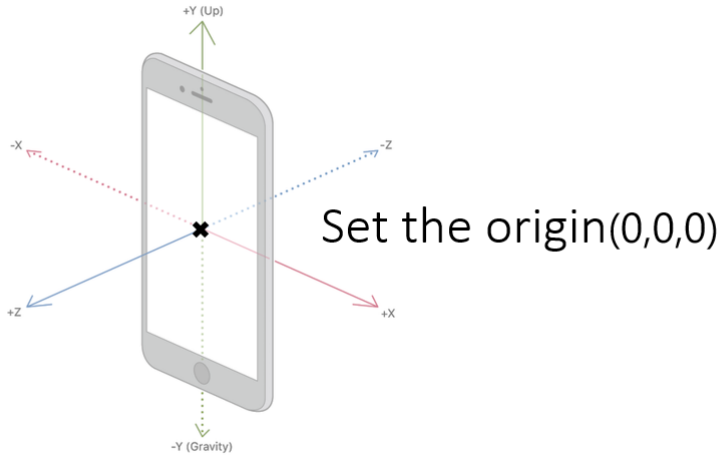

Figure 23. Diagram illustrating mobile device origin

2. Set the origin (0,0,0) as the reference point, and then get all vertex from each E3D building model.

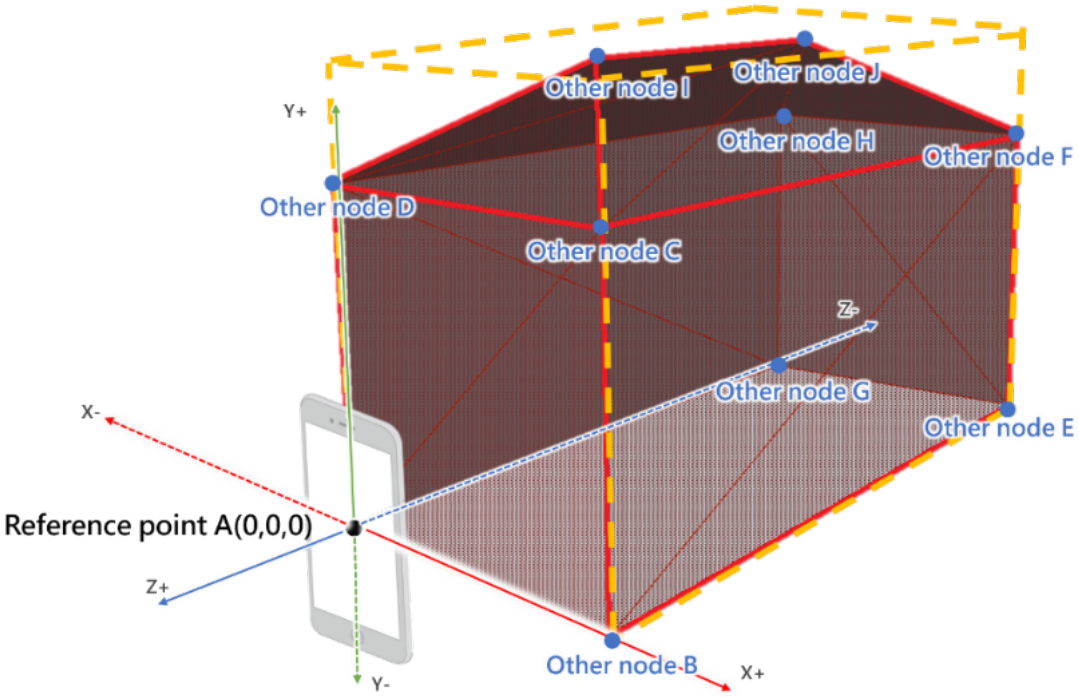

Figure 24. Diagram illustrating building local coordinate system, as if it were positioned at mobile device origin

3. Put all the nodes in SCNVector3 as an array.

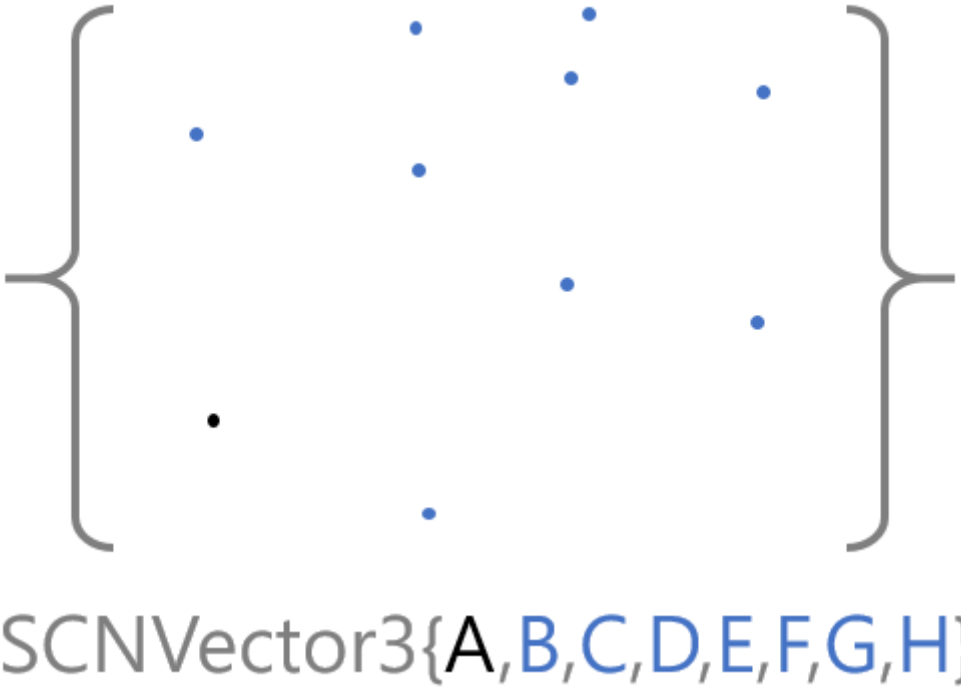

Figure 25. Diagram illustrating an array of vertices

4. Set up the geometry module object according to SCNGeometry (Completed), and then drawing triangles for all faces of the building.

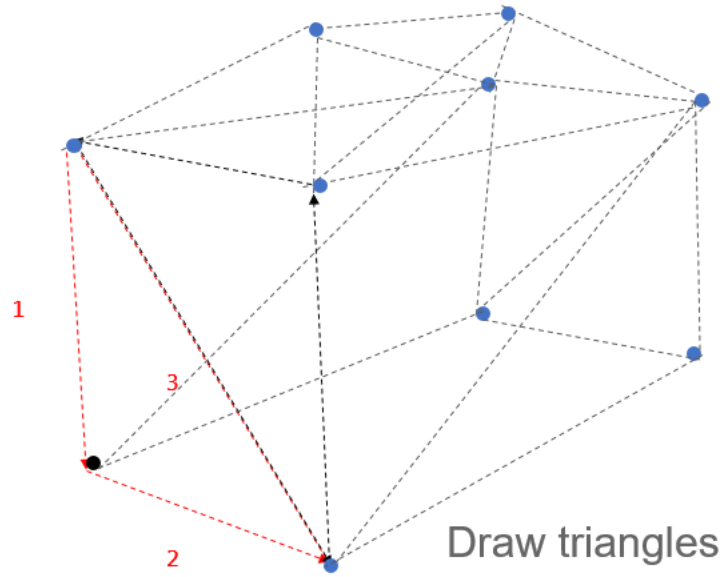

Figure 26. Diagram illustrating setting up building geometry as triangular faces

5. Use the reference point to put the entire building onto correct place of the user's camera.

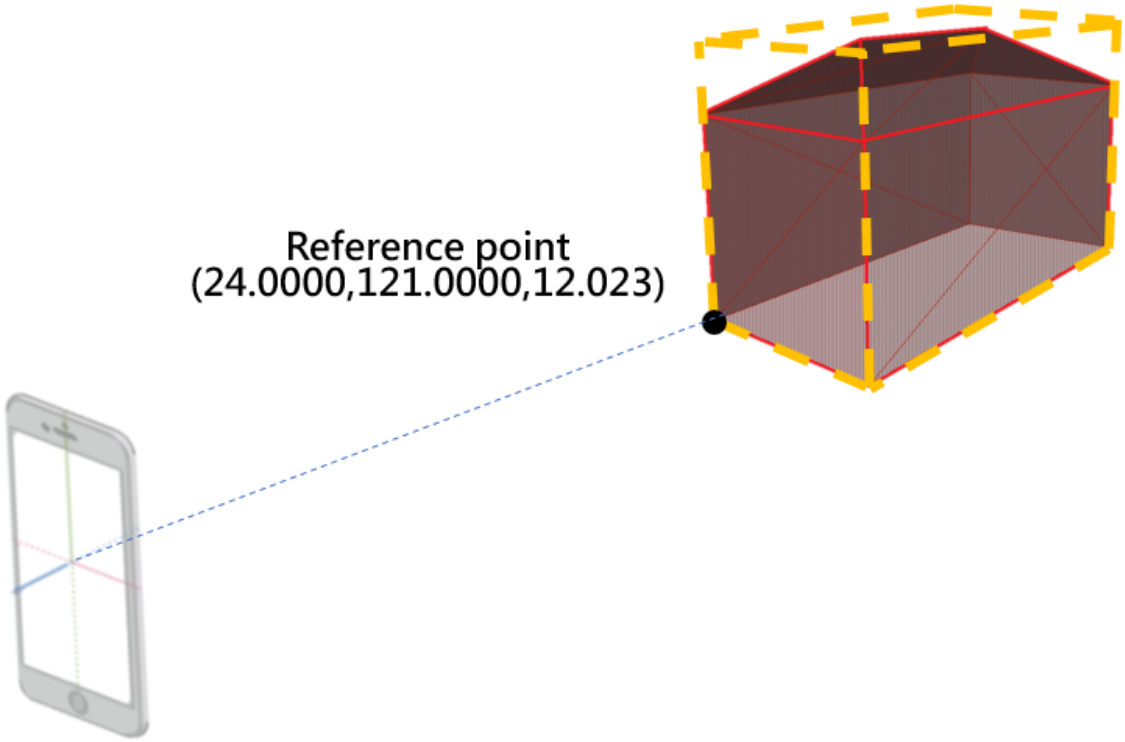

Figure 27. Diagram illustrating positioning building at proper location based on building reference point and the user position (camera)

7.2.3. Client Demonstration

Taichung City Demo

Video:

https://drive.google.com/file/d/1CRtaSW44hzcS0BJMWTq-kCoMGu-8t5wi/view

Service:

http://tm.gis.tw/wfs?SERVICE=WFS&REQUEST=GetFeature&centerPosition=120.64981667547605,24.178336886020727&outputFormat=json&Distance=0.1

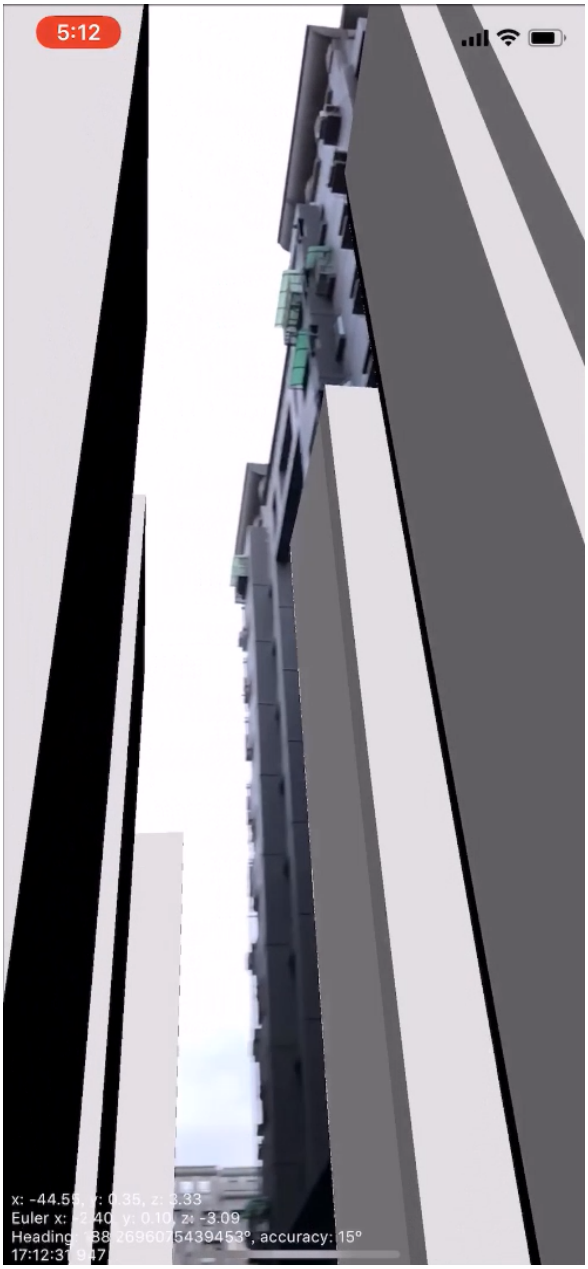

Figure 28. Screenshot of 3D buildings in GIS-FCU client accessing GIS-FCU service with data from in Taichung City

Figure 29. Picture of the building represented by the 3D geometry overlaid on next figure by itself

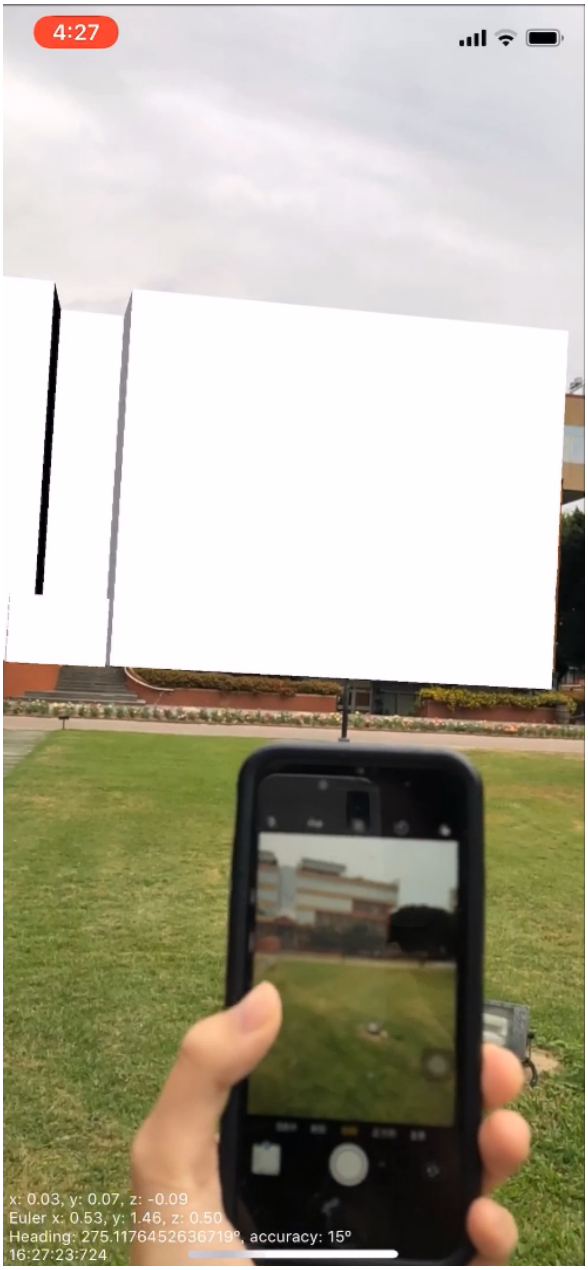

Figure 30. Screenshot of 3D buildings in GIS-FCU client accessing GIS-FCU service with data from in Taichung City

New York City data set (Ecere Service)

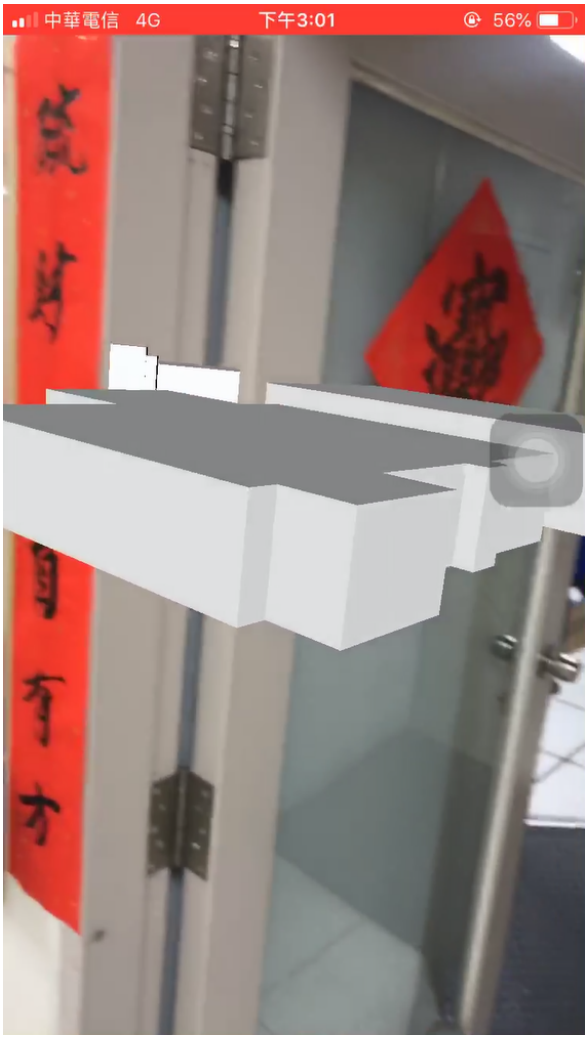

Figure 31. Visualizing data from Ecere Service (New York City)

Video: https://drive.google.com/open?id=1mG-CUeb0n514sjU3Luhqfq-a7JkZj8nC

Field tests in Washington DC (GIS-FCU Service)

Figure 32. 3D axes and overlay information shown in GIS-FCU client during testing by Marcus Alzona (Keys) in Washington, DC, in front of the Capitol

Figure 33. 3D building geometry overlaid on top of camera near National Cathedral in GIS-FCU client accessing GIS-FCU service, tested by Marcus Alzona (Keys) in Washington, DC (computed camera orientation and/or position may be slightly off, possibly partially due to magnetic declination)

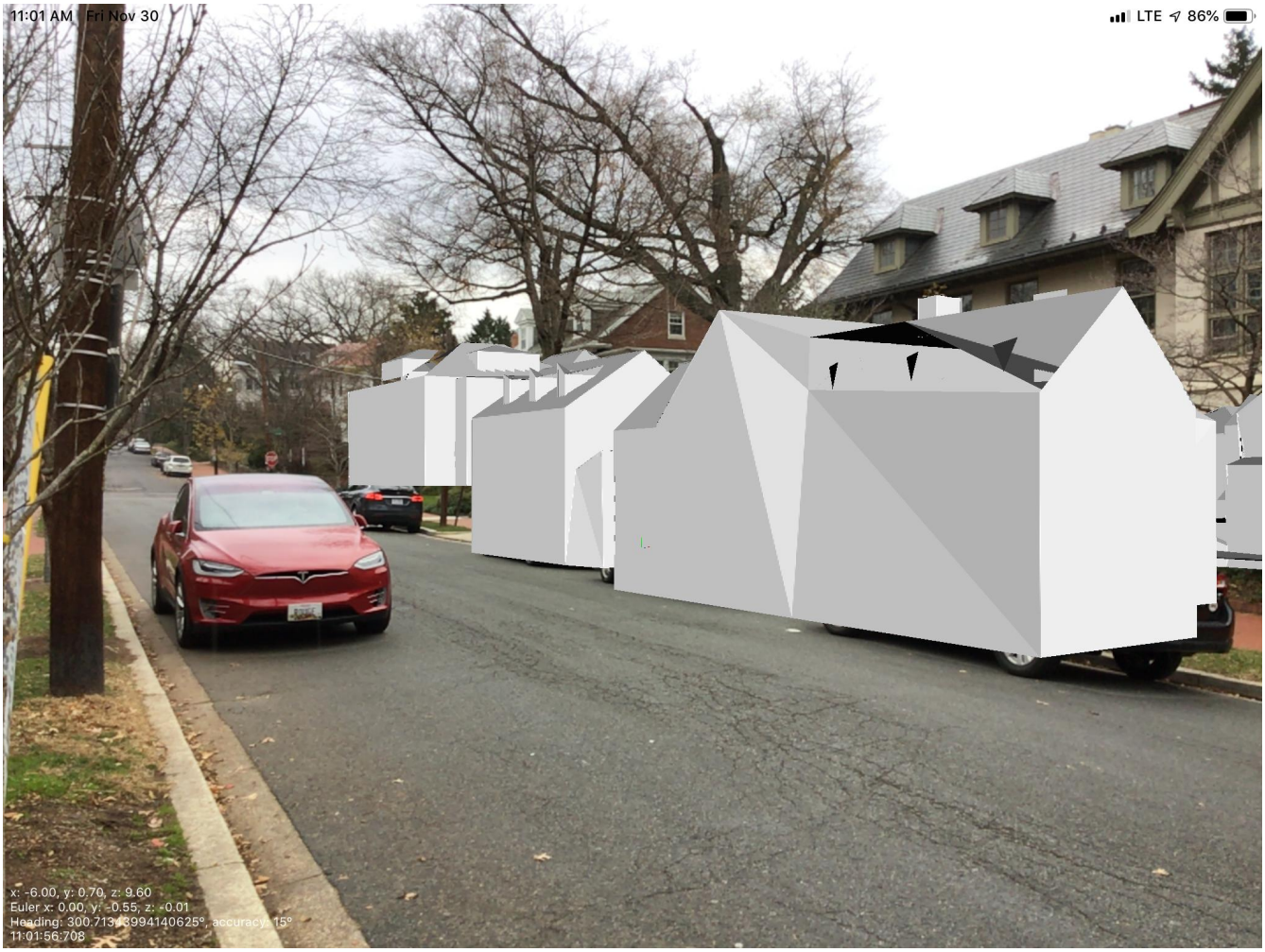

Figure 34. 3D building geometry overlaid on top of camera in GIS-FCU client accessing GIS-FCU service, tested by Marcus Alzona (Keys) in Washington, DC (computed camera orientation and/or position may be slightly off, possibly partially due to magnetic declination)

Chapter 8. Client-Server Communication

A simple approach to transmit 3D geospatial contents was developed during this initiative to facilitate the rapid implementation and interoperability of both clients and services. Considerations were given to a number of aspects contributing to the client performance and efficient use of bandwidth.

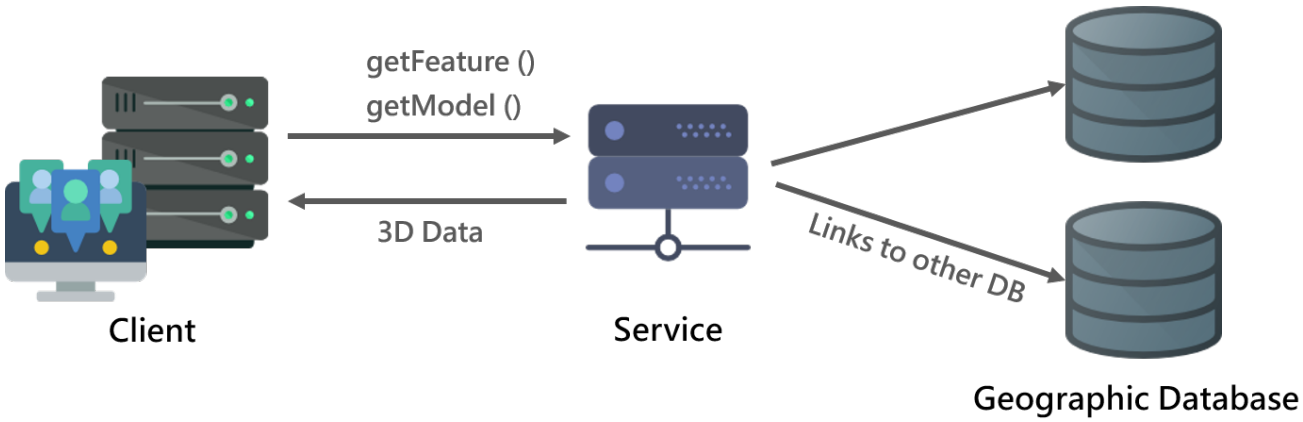

Figure 35. Diagram summarizing Client-Server interaction for requesting and retrieving 3D data

8.1. Services overview

The services implemented were mainly based upon and extended the Web Feature Service. Both GIS-FCU and Ecere provided a Key/Value Pair WFS service. The supported requests included:

- **GetCapabilities** (Ecere) To do the initial handshake with the service, and retrieve the list of available layers.
- **DescribeFeatureType** (Ecere) To describe attributes for a given layer.
- **GetFeature** (Ecere, GIS-FCU) To retrieve the points referencing 3D models. For Ecere service, it can also be used to retrieve other vector features, imagery or elevation data. The Ecere service supported the regular WFS 'bbox' parameter (which can be used to cut out tiles for any tiling scheme based on a WFS84 tiling scheme), as well as an alternate tile-oriented API similar to a WMTS interface (tilingScheme, zoomLevel, tileRow, tileCol). GIS-FCU opted instead for 'centerPosition' (longitude, latitude) and 'distance' parameters (in kilometers).
- **GetModel** (Ecere, GIS-FCU) Retrieve a specific 3D model (a triangulated mesh, which could be instantiated more than once). E3D was used as the output format for its compactness, and ease of implementation by the participants (Ecere provided sample E3D loading & writing code). glTF or OpenFlight would be good examples of alternative output formats that could eventually be requested to the services. Considerations were also given to have the ability to request attributes associated with sub-elements within a single 3D model (with partID and propertyName parameters). This would have allowed to batch more objects together in a single model without losing the attribution of these different entities. However the time for the initiative was too limited to investigate batching approaches, and the datasets used had a very limited set of useful attributes.
- **GetTexture** (Ecere) The Ecere service additionally implemented a GetTexture request.

Additionally, the Ecere service provided a WFS3/Next Generation-style API. It could be used to retrieve all elements of a CDB data set, including the terrain elevation models (a coverage) and imagery, either tiled or not, showing how Next Generation OGC services could be harmonized as a single service. A Unified Map Service interface (a service type developed during Testbed 13) was also made available, although support for 3D models and textures was not completed.

Using a direct socket connection (e.g. direct socket or WebSockets) was considered, but there was no time to include this in the experiments. Proper used of persistent HTTP connection and request queueing might also offer similar advantages.

See also Chapter 9 for a detailed description of both services and their requests.

8.2. Transmission data formats

The formats used to transmit data during this initiative included:

- E3D (for 3D models) (described in details in Appendix A: E3D 3D Model Specifications)
- GeoJSON (for non-tiled reference points)
- GNOSIS Map Tiles (points referencing 3D models; terrain elevation; imagery; street maps; buildings footprints)
  (described in details in Appendix B: GNOSIS Map Tiles Specifications)
- JPG, PNG and ETC2 compressed textures

The participants opted for E3D for its simplicity, implementing both loaders and writers from the ground up, as well as for the slight compactness advantage and readiness to load 3D model data onto the GPU, without any required parsing. Although during these experiments E3D was used to describe 3D models by both services and both clients, glTF could also be used with the same types of requests as an alternative output format. OpenFlight or COLLADA, would also be valid candidates for 3D models output formats, although the latter would be much heavier due to its XML and textual nature.

8.2.1. 3D Tiles & I3S

From these two formats mentioned in the original requirements for the initiative, 3D Tiles was agreed upon as being the easiest standard transmission format to support based on the participants capabilities, proposal commitments & resources availability. During Testbed 13, Ecere had developed a client supporting 3D Tiles encoded as batched geometry, using glTF 1. It was noted that AssimpKit [https://assimpkit.readthedocs.io/en/latest/] can import many formats, including glTF 1 (the guts of 3D Tiles) for use within SceneKit in conjunction with ARKit. Given more time, glTF support, and then a transmission pathway as 3D Tiles would also have been implemented. However, the stakeholders opted for focusing on the data transformation such as creating 3D meshes for buildings (which would still have been required for 3D Tiles), and the augmented reality aspect.

For similar reason, there was no time to implement I3S support.

8.2.2. Rationale behind decision to use E3D for this initiative

In order to validate interoperability between multiple client/server combination, and given the limited resources and very short time frame for the Testbed, a single transmission format had to be agreed upon (a requirement in the CFP: *The service/client instance shall be tested with **both** client/service applications.*).

No existing 3D Tiles or I3S data sets was identified to be used for the experiments; rather the data would be sourced from CityGML and served in a transmission format. Both 3D Tiles and I3S do present the advantages of being such a transmission format for streaming detailed 3D models of a large urban area (well suited especially for the Virtual Reality scenario) over CityGML.

The rationale for implementing a simple 3D transmission format was mainly to reduce the burden of implementation for the clients and services (minimizing the efforts spent on producing and consuming 3D data), so as to focus on the Augmented Reality aspect, given the short development time frame and the existing capabilities of the participants involved. The technical value of 3D Tiles and/or I3S was not being disputed.

None of the participants of these experiments are the initiators of 3D Tiles (or I3S), nor did they plan to leverage the original clients/services for which these formats were initially designed. For both services, and for the GIS-FCU client to run on iOS using AR Kit, however, there was no readily available support for 3D Tiles or glTF production or consumption. ARKit natively supports COLLADA (DAE) and OBJ/MTL (https://stackoverflow.com/questions/48190891/what-3d-model-formats-are-supported-by-arkit).

In addition to these practical reasons, some advantages of the approach of using E3D (and in the case of the Ecere client, GNOSIS Map Tiles to reference these E3D) are noted in relation to minimizing bandwidth usage and processing:

- By using a simple fixed tiling scheme allowing the client to know exactly which tiles are needed, they would not have to deal with the iterative client/server round-trip process of querying 3D tiles tile sets and figuring out whether they must be refined based on the geometric errors of the tiles.
- By avoiding having any parsing to do (3D Tiles and glTF, even in the binary 'glb' form, are still made up of JSON text that must be parsed), data can be processed (and transmitted) more efficiently.

Given the simplicity of E3D and the participants' familiarity with the solution, a minimal set of options helped facilitate interoperability. 3D Tiles and glTF cover a very large set of capabilities, which all components being implemented during the short span of the Testbed would not have been able to cover entirely; and neither did the experiments have a need for most of them. There is also a large variation of how things can be done within 3D Tiles / glTF (e.g. starting with glTF version), which might have caused interoperability issues if the clients/services implement a different subset of these capabilities, or if they end up being interpreted differently.

Finally, as an additional 3D data representation format kept simple but with performance in mind, E3D might inspire a conceptual model to which both 3D Tiles (and/or glTF) and I3S could be mapped, which could in turn facilitate interoperability between different 3D formats.

8.2.3. ETC2 Texture Compression

Due to the very limited memory available to mobile applications (CPU and GPU memory is typically shared on mobile devices, and e.g. the Android operating system itself can easily take up half of total memory available), displaying a large textured urban environment is challenging. Texture compression can offer welcomed significant gains in that regard, and a particularly appealing compression is the version 2 of Ericsson Texture Compression [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression], because support for it is mandatory in both OpenGL ES 3.0 and OpenGL 4.3, which solves the complex problem of having to use different compression mechanisms for different hardware. This problem was exacerbated by the fact that compressing textures on-the-fly to load them on the GPU is impractical due to the high amount of CPU processing required, and so compression should be done offline as preprocessing. Although two standard formats exist to store and transmit ETC2 textures (KTX [https://www.khronos.org/opengles/sdk/tools/KTX/file_format_spec/] and PKM [https://github.com/Ericsson/ETCPACK/blob/master/source/etcpack.cxx#L9189]), due to very limited time, a simplistic custom format was used in this initiative. It can be described as follows (in little-endian byte order):

Table 1. Simple format encoding used for ETC2 texture data

Offset	Type	Size	Name	Description
0	uint	4	count	Number of mipmaps
count images follow (offset from start of each image)				
0	uint	4	width	Width of the image
4	uint	4	height	Height of the image
8	uint	4	size	Size of the compressed data
12	byte[*]	size	data	Compressed data (to be loaded with GL_COMPRESSED_RGBA8_ETC2_EAC)

8.3. Performance considerations

Several factors affect the performance of displaying 3D contents, and some of these were highlighted in *Testbed 13 3D Performance Client ER* [http://docs.opengeospatial.org/per/17-046.html]. [10] Those aspects were also considered in this initiative to facilitate achieving good performance in the clients. Ideally, 3D data sent to a mobile or web client is optimized through pre-processing so as to minimize the size of a transmitted payload, and it is batched by materials so as to reduce rendering overhead. However, only limited efforts could be spent implementing these optimizations and batching due to the short duration of the Testbed and the focus on Augmented Reality experiments.

Nevertheless, the interface described herein would still be appropriate for services delivering an optimized payload, and for clients to take full advantage of them. The E3D model format also has special considerations to facilitate the batching of materials.

A key goal of achieving good performance is to minimize the number of drawing calls and state changes, and batching geometry by materials makes this possible. The E3D specifications also introduce the concept of a *Material Group*, which conceptually can encompass multiple compatible materials for which geometry could be rendered within the same drawing call, using the same texture object. Ideally, all models within a single 3D data layer (e.g. 'Buildings' or 'Trees') use the same material group, and potentially multiple layers could use the same material group as well, allowing to render an entire scene with a single drawing call. This also implies using a single Vertex Buffer Object for storing all of this batched geometry.

For textured environments, this also requires the use of either or both texture atlases and array textures, which allow using a large quantity of texture data within a single texture object. While a texture atlas can be used to regroup smaller textures within a single larger texture, array textures allow to define multiple layers of large textures.

Since the batching of geometry follows visual properties, but not necessarily the logical structure of the entities represented by the models (e.g. buildings), it is essential to maintain this original organization so as to preserve selection and attribution capabilities. The E3D format allows this with the concept of parts, establishing a relationship between faces and a part ID, which in turn can be used with this service interface's for querying parts attributes. This can be used, for example, to transport original properties from a CityGML dataset.

Another important aspect, especially relevant to geotypical models, is the use of geometry instancing. With instancing, an entire forest made of thousands of trees for example can be rendered by lighter drawing commands with minimal position information for each tree which can reference one of a few complex models e.g. for trees of different species.

More basic ways in which re-use helps gain performance include indexing the same vertices through the use of indices to describe faces, as well as having multiple models sharing the same textures.

8.3.1. Tiling and Caching

The use of tiles has several benefits as they allow to retrieve and display a specific area of interest, and of a specific detail if implemented with multiple zoom levels.

Tiles also facilitate the implementation of a caching mechanism, as they identify discrete chunks of data(of a limited size). The implementation of such a cache could help keep the information in GPU memory, in CPU memory and/or on disk.

With 3D models data, finer zoom levels can either refine models with denser geometry (an approach better suited to batching), or integrate additional smaller models (as done in CDB). When denser geometry is available at finer zoom levels, 3D mesh simplification algorithms can be applied to the lower zoom levels to produce lightened generalized tiles covering the larger extent. When tiling a layer of 3D models, especially with a fixed tiling scheme as used within this Testbed, the problem of handling models crossing tile boundaries arises.

Two tiling approaches were devised: one where models are actually cut clean on the tile boundaries and embedded within the tile, and one where the models are allowed to spill onto the neighboring tiles and referenced. Support for both of these approaches was incorporated as new

capabilities of GNOSIS Map Tiles (see Appendix B).

For the clean-cut approach, the embedded3DModel (0xB0) type can be used, where each tile will contain a single 3D model. This implies a geospecific model, and is best suited for large-scale geospecific features (e.g. pipelines) or 3D terrain models (although the GMT specifications also provides a coverage-based mechanism to store 3D terrain tiles). Such large features would also benefit from describing vertices quantized to the model's bounding box, e.g. using the quantized vertices attribute block (verticesQ: 0x2018) of E3D specifications.

For the referenced/spilling models approach, the models3D (0xA0) or models3DGround (0xA1) types can be used. The tile then describes a collection of points, each with a *Model ID* referencing an external 3D model to be accessed separately. In the case of models3DGround, the origin of the model is intended to be dropped to a separate elevation model, whereas for models3D each point also includes an altitude value relative to the WGS84 ellipsoid. An orientation or scaling can also be included for each point, as well as data attributes as for regular point features. In this case, models can be either geospecific or geotypical. A given model instance is defined only in one tile of a particular zoom level. In order to ensure drawing models of nearby tiles which may not have been selected as visible (based on a layer-independent selection mechanism), additional information has also been included in the tiles about the bounding box of all referenced models within, as well as about neighboring tiles (of the same zoom level) and their models which are spilling onto the current tile.

Because in a 3D view tiles of multiple zoom levels might be mixed (lower zoom levels for tiles further away), some additional complexity is involved as refined tiles may contain a duplicate model to a lower resolution neighboring tile. To handle these scenarios, the *Model ID* can contain a 'zoom level' portion in addition to the ID of the model itself. Models representing the same entity would always have the same base ID, while the level is the coarsest zoom level at which this model should be displayed: it can be the same as the one for a lower level tile if this finer tile does not refine the model, or does so with additions.

In addition to GNOSIS Map Tiles, some experiments were also done with tiles of GeoJSON point feature tiles carrying the same key information. In both cases, the GNOSIS Global Grid, defining tiles approximating equal area for the entire globe, was used as the tiling scheme to define the tiles. The GIS-FCU service also provided similar GeoJSON point feature data, however it was based on a center position and a radius rather than pre-defined rectangular tiles.

Unfortunately, due to time constraints, only the reference/spilling models approach was implemented. It was also not possible to complete the special handling of models defined in neighboring tiles spilling onto selected tiles, and therefore in experiments buildings close to the viewer would sometimes not appear when they should.

Another important aspect not covered in depth is the selection of different resolutions for textures. A resolution parameter was considered for the GetTexture request, but the mechanism by which a tile of a finer zoom level selects a higher resolution has not been adopted. This is a rather important aspect, as in some cases the texture data may be even heavier than the geometry and have a great impact on bandwidth, performance and GPU memory usage (especially limited in the case of mobile devices).

8.4. Disconnected environment and intermittent connectivity

Some experiments were done entirely offline based on a local data store. The GNOSIS Data Store was used to organize tiled layers (in GNOSIS Map Tiles pyramid format), with sub-folders for models (in E3D format) and textures (in JPEG, PNG or ETC2 format). This demonstrated the possibility to retrieve, store, publish, and directly visualize a partial or complete dataset of terrain elevation, imagery and 3D model data based on the same principles as the client/server interaction mechanism described herein. Furthermore, this embodiment of the dataset represented a significant reduction in terms of storage size (e.g. 3D model and terrain data layers easily fit within the OGC portal's limited allowed file size) as well as count of files, as well as performance improvements, compared to the original CDB.

8.5. Relationship with 3D Portrayal Service

During this initiative, questions and thoughts regarding the potential relationship with the 3D Portrayal Service came up, but there was no time devoted specifically to consider what that relationship could be. Therefore investigating any overlap or complimentary aspects, and integrating any of the capabilities prototyped during this testbed activity within the 3D Portrayal Service framework would be reserved for future work. Some suggestions included considering the integration of the additional 3DPS capabilities within a general harmonized or unified next generation map service, which we tried to demonstrate as something feasible with this interface. Additionally, we feel one of the initial of the objective of the 3DPS which was to bridge the gap between different model formats, allowing a service to offer multiple supported formats, is within the grasp of this interface through a format parameter for the GetModel and/or GetTile request, e.g. allowing to return an E3D or glTF file; or a GNOSIS Map Tile or 3D Tile. The E3D specifications, as an alternative yet simple way of describing 3D models, may also inform the formulation of a conceptual model and/or simple profile to which both glTF (the foundation of 3D Tiles) and I3S could be related as well, and this may help achieve additional format interoperability.

Chapter 9. Geospatial Services for Augmented Reality

Two services were built as part of this initiative to serve primarily 3D buildings data.

Both services served 3D models following the E3D specifications (Appendix A), to be interoperable with both clients.
See Chapter 8 for a high level overview of the services/client communication.

The data was loaded from various sources, including OpenStreetMap, Multipatch Shapefiles, CityGML and CDB.
See Chapter 10 for a detailed description of data sources and how they were processed.

9.1. Ecere Service

The service provided by Ecere was based on the *GNOSIS Map Server* and implemented primarily a Web Feature Service, with extensions for retrieving 3D models and textures. Both a classic Key/Value pair WFS as well as a next generation REST-based (WFS3) interfaces were used in the experiments. Additionally, an interface for the Unified Map Service (UMS), as prototyped for Testbed-13 Vector Tiles work package was used in the experiments.

For all requests, an **authKey** parameter provides security authentication, giving access to specific features. This authorization key is not included in requests herein, but is required for data derived from the New York CDB, which was provided solely for the purpose of this Testbed-14 AR initiative. As a result, many of the requests listed will result in a *404 - Resource Not Found* error. The usage of the data set is subject to a license agreement with FlightSafety. OGC can be contacted in case these working resources would be useful and provide information on how to agree to this license and obtain the necessary authorization key.

9.1.1. GetCapabilities

A GetCapabilities request lists all layers available from the WFS end point.

Example request:

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetCapabilities

9.1.2. DescribeFeatureType

A DescribeFeatureType request provides a schema of all attributes supported by the layer, including special attributes for referencing and positioning 3D models (called ums::feature::modelID, ums::feature::altitude and ums::feature::orientation, and ums::feature::scale).

Example request:

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=DescribeFeatureType&
typeName=NewYork:NewYork_Buildings

9.1.3. GetFeature

In addition to standard *service* and *request* parameters, the following parameters were used for the GetFeature requests:

Table 2. GetFeature request parameters (Ecere WFS service)

Parameter	Description
typeName	Select a specific type name (layer, feature collection) from which to retrieve data.
outputFormat	Select a specific format according to which the data to retrieve should be encoded. Supported format currently includes GNOSIS Map Tile (gmt), GeoJSON, Mapbox vector tiles, GML and GeoECON.
bbox	A bounding box describing the extent of the geometry to be retrieved. Normally in a WFS this returns the entire feature intersecting the bounding box (without clipping). The Ecere service currently performs clipping. Because points are either in or out of the box, this is not an issue with points layers such as those referencing 3D models.
propertyName	Provides a way to select specific data attributes and/or geometry to retrieve.
tilingScheme	Identifies a tiling scheme which zoomLevel, tileRow and tileCol refers to. In these experiments, the GNOSIS Global Grid was used as the tiling scheme, but the service also supports tiling schemes based on the WMTS well-known scale sets (GoogleMapsCompatible, GlobalCRS84Pixel, GlobalCRS84Quad, GoogleCRS84Quad). If unspecified, the GNOSIS Global Grid is assumed.
zoomLevel	Identifies a zoom level for which to retrieve data, based on the selected tiling scheme. This provides a mechanism to retrieve generalized data for large areas. The service could provide a different different set of points based on the zoom level (e.g. smaller/less important 3D models filtered out at lower zoom level and/or using alternate lower resolution models).
tileRow	Identifies a tile row for which to retrieve data, based on the selected tiling scheme.
tileCol	Identifies a tile column for which to retrieve data, based on the selected tiling scheme.

Parameter	Description
authKey	Provides basic security authentication, giving access to specific features. The use of the authorization key listed in the requests herein associated with data derived from the New York CDB, provided solely for the purpose of this Testbed-14 AR initiative, is subject to a license agreement from FlightSafety.

Some example requests retrieving features from various data layers:

Buildings

(tile-oriented)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&tilingScheme=GNOSISGlobalGrid&zoomLevel=15&tileRow=47588&tileCol=38589&outputFormat=geo+json

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=geo+json

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=gmt

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=mvt

(non-tile)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&outputFormat=geo+json&propertyName=geometry

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Buildings&bbox=40,-75,41,-74&outputFormat=geo+json

Trees

(tile-oriented)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Trees&tilingScheme=GNOSISGlobalGrid&zoomLevel=15&tileRow=47588&tileCol=38589&outputFormat=geo+json

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Trees&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=geo+json

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Trees&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=gmt

(non-tile)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Trees&outputFormat=geo+json&propertyName=geometry

(omitting the extra original CDB attributes)

Imagery

(tile-oriented)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=png

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=gmt

(non-tile)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&outputFormat=png

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&outputFormat=png&zoomLevel=7

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&outputFormat=png&zoomLevel=9&bbox=40.47336645,-74.277405,40.92364737,-73.68470065

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&outputFormat=png&zoomLevel=11&bbox=40.47336645,-74.277405,40.92364737,-73.68470065

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&typeName=NewYork:NewYork_Imagery&outputFormat=png&zoomLevel=12&bbox=40.68735853,-74.04963142,40.73704513,-73.97036899

NOTE	The highest resolution imagery from the CDB (level 17 rather than 12) was not made available, as it was to be re-encoded in JPEG2000 first to use up less server space.

Elevation

(tile-oriented)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&

typeName=NewYork:NewYork_Elevation&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&tileRow=5948&tileCol=4823&outputFormat=png

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_Elevation&tilingScheme=GNOSISGlobalGrid&zoomLevel=12&
tileRow=5948&tileCol=4823&outputFormat=gmt

(non-tile)

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_Elevation&outputFormat=png

Although there was no time to experiment with the embedded 3D model approach during the initiative, a GetFeature request could have been used for this as well. In this case each tile would contain a single 3D model, and the output format could either be GNOSIS Map Tiles each embedding a single 3D model, or directly the model itself (as E3D, glTF, COLLADA...). The following are would-be examples of a terrain mesh served this way:

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_TerrainMesh&tilingScheme=GNOSISGlobalGrid&zoomLevel=4&
tileRow=6&tileCol=11&format=gmt

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_TerrainMesh&tilingScheme=GNOSISGlobalGrid&zoomLevel=4&
tileRow=6&tileCol=11&format=e3d

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_TerrainMesh&tilingScheme=GNOSISGlobalGrid&zoomLevel=4&
tileRow=6&tileCol=11&format=glb

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetFeature&
typeName=NewYork:NewYork_TerrainMesh&tilingScheme=GNOSISGlobalGrid&zoomLevel=4&
tileRow=6&tileCol=11&format=collada

9.1.4. GetModel

Table 3. GetModel request parameters (Ecere WFS service)

Parameter	Description
typeName	Select a specific type name (layer, feature collection) from which to retrieve a 3D model.
modelID	The ID identifying the 3D model to be retrieved. This is a numeric ID, with the high 5 bits corresponding to the model level, while the lower 27 bits correspond to a model. The model level is the coarsest zoom level at which this model is used: it can be the same as the one for referencing points data at a lower zoom level tile if this finer level does not refine the model, or does so with additions.

Parameter	Description
format	The format in which to encode the 3D model data (or the attributes when partID is used) being retrieved. Currently, only E3D is supported, but the output for this could later be binary glTF (glb) or COLLADA/DAE model.
partID	Rather than retrieving the 3D model itself, retrieve attributes asociated with specific part(s) of the model (not yet implemented).
propertyName	Specific sub-model attributes to retrieve (for use together with partID, not yet implemented).

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetModel&
typeName=NewYork:NewYork_Buildings&modelID=1476406989&format=e3d

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetModel&
typeName=NewYork:NewYork_Buildings&modelID=1342193159&format=e3d

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetModel&
typeName=NewYork:NewYork_Buildings&modelID=1342193159&format=glb

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetModel&
typeName=NewYork:NewYork_Buildings&modelID=1342193159&format=collada

Sub-model attribution example:

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetModel&
typeName=NewYork:NewYork_Buildings&format=geo+json&modelID=1342193159&partID=4&
propertyName=name

9.1.5. GetTexture

The GetTexture request makes it possible to share textures between models of the same layer (as opposed to only being able to embed textures within the 3D models).

Table 4. GetTexture request parameters (Ecere WFS service)

Parameter	Description
typeName	Select a specific type name (layer, feature collection) from which to retrieve a texture.
textureID	The (numeric) ID identifying the texture to be retrieved.
format	The format in which to encode the texture being retrieved. Currently, jpg, png and etc2 (a custom simple encoding of Ericsson Texture Compression version 2) are supported.
resolution	Resolution at which to retrieve the texture.

Parameter	Description
propertyName	Specific sub-model attributes to retrieve (for use together with partID, not yet implemented).

Some example requests:

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=1

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=1&resolution=256

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=56&format=jpg&resolution=256

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=56&format=jpg&resolution=512

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=56&format=jpg&resolution=1024

http://maps.ecere.com/wfs?SERVICE=WFS&REQUEST=GetTexture&
typeName=NewYork:NewYork_Buildings&textureID=56&format=png&resolution=1024

9.1.6. WFS3 / Next Generation / Harmonized Map Service (REST API)

A WFS3 / Next Generation service interface is also provided. Some example requests follow.

NOTE	OpenAPI description remains in progress.

Capabilities

http://maps.ecere.com/hms

List of layers (with support to drill into hierarchy of layers being served)

http://maps.ecere.com/hms/layers/NewYork/

Listing attributes for a layer (schema)

http://maps.ecere.com/hms/layers/NewYork/NewYork-Buildings/schema.xml

Listing and describing tiling schemes

http://maps.ecere.com/hms/tiles

http://maps.ecere.com/hms/tiles/GNOSISGlobalGrid

Retrieving features (tile-oriented)

http://maps.ecere.com/hms/layers/NewYork/NewYork-Buildings/tiles/GNOSISGlobalGrid/12/5946/

4823.json

Retrieving features (bounding box)

http://maps.ecere.com/hms/layers/NewYork/NewYork-Buildings/items.json?BBOX=40,-75,41,-74

Retrieving a 3D model

http://maps.ecere.com/hms/layers/NewYork/NewYork-Buildings/models/1342193159.e3d

Retrieving a texture

http://maps.ecere.com/hms/layers/NewYork/NewYork-Buildings/textures/56.jpg

9.1.7. Unified Map Service

A Unified Map Service (UMS, a service type developed during Testbed-13) interface is also provided. Some example requests feature below.

Capabilities

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetCapabilities

List of layers (with option to specify collection to drill into hierarchy of layers being served)

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetLayersList&collection=NewYork

Listing attributes for a layer (schema)

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetAttributesList&layer=NewYork/NewYork-Buildings

Describing a tiling scheme

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetTilingScheme&tilingScheme=GNOSISGlobalGrid

Figuring out which tile a given point is part of (useful for testing)

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetTileAtPos&zoomLevel=12&position={40.70579927,%20-74.00932322}

Retrieving features for a given tile

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetTile&layer=NewYork/NewYork-Buildings&tilingScheme=GNOSISGlobalGrid&tileKey={15,47588,38589}&format=geo+json

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetTile&layer=NewYork/NewYork-Buildings&tilingScheme=GNOSISGlobalGrid&tileKey={15,47588,38589}&format=gmt

Retrieving features (bounding box)

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetFeatures&layer=NewYork/NewYork-

Buildings&extent={{40,-75},{41,-74}}&format=geo+json

Retrieving attributes

54199 [http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetAttributes&layer=NewYork/NewYork-Buildings&features=]&on=gml

54199 [http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetAttributes&layer=NewYork/NewYork-Buildings&features=]&attributes=[%22CNAM%22,%22FACC%22]&on=json

Support for retrieving 3D models and textures remains to be added to UMS, but they could take the form of:

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetModel&layer=NewYork/NewYork-Buildings&modelID=1342193159&format=E3D

http://maps.ecere.com/ums?SERVICE=UMS&REQUEST=GetTexture&layer=NewYork/NewYork-Buildings&textureID=56&format=jpg

9.2. GIS-FCU Service

All E3D files are stored in a geospatial database (PostGIS) in the backend. The client can send a RESTful request with latitude and longitude of an on-site user via the ***GetFeature*** method. The service will retrieve the corresponding ID of the E3D and return a list of buildings to the client in GeoJSON format. The client can send separate ***GetModel*** requests with each of these model IDs in order to retrieve the E3D models.

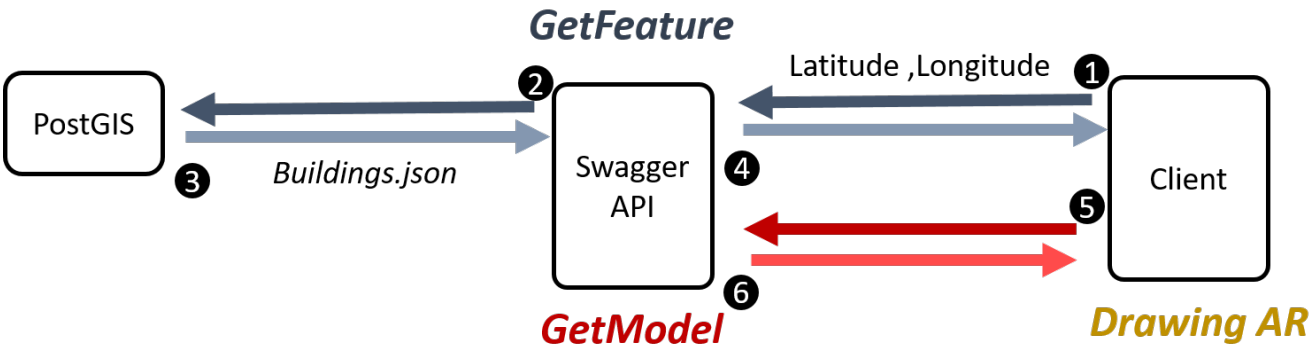

Figure 36. Diagram illustrating interaction between AR client, GIS-FCU service and back-end PostGIS database.

NOTE	Unlike a typical WFS, this GIS-FCU service is currently sensitive to case in parameter names.

9.2.1. GetFeature

Table 5. GetFeature request parameters (GIS-FCU WFS service)

Parameter	Description	Remarks

SERVICE	An open API following style of WFS	-
REQUEST	GetFeature request	-
centerPosition	Reference point of an user.	Longitude, then latitude separated by comma (,)
outputFormat	Output format	Only support Geojson
Distance	Querying area	0 to 5 KM(float)

Example request:

http://tm.gis.tw/wfs?SERVICE=WFS&REQUEST=GetFeature&centerPosition=-76.97663465999995,38.86509976200006&outputFormat=json&Distance=0.1

9.2.2. GetModel

Table 6. GetModel request parameters (GIS-FCU WFS service)

Parameter	Description	Remarks
SERVICE	An open API following style of WFS	-
REQUEST	GetModel request	-
outputFormat	Output format	only support E3D
modelID	model ID of E3D that a client wants to retrieve.	-

Example requests:

http://tm.gis.tw/wfs?SERVICE=WFS&REQUEST=GetModel&outputFormat=e3d&modelID=DC0000001

http://tm.gis.tw/wfs?SERVICE=WFS&REQUEST=GetModel&outputFormat=e3d&modelID=DC00047921

Chapter 10. Geospatial data

Before 3D data could be served to the clients as Augmented Reality content by the services developed for this initiative, data sets had to be pre-loaded. For both of the services, this involved some pre-processing to convert the data into a format ready for efficient delivery through the client/server interface described in Chapter 8. Because investigating the use and applicability of CityGML to Augmented Reality was an essential requirement of this Testbed activity, CityGML was used in one such conversion process as an intermediate format between Shapefiles (multipatch) [11] and E3D models, even though a more direct conversion pathway could have otherwise been adopted.

10.1. Original data sources (and their formats) used for the experiments

The two main source data sets that were used for the experiments were:

- New York CDB data provided by FlightSafety (the same data set which was used in Testbed-13 for *3D Performance* work package)
- 3D Buildings of Washington, D.C. provided as an ESRI multipatch shapefile by OpenData DC [http://opendata.dc.gov] — 103,073 buildings; available from here [https://drive.google.com/a/dc.gov/file/d/0B1Wt8FRXoFfJamhPUkEtdEh0TGs/view].

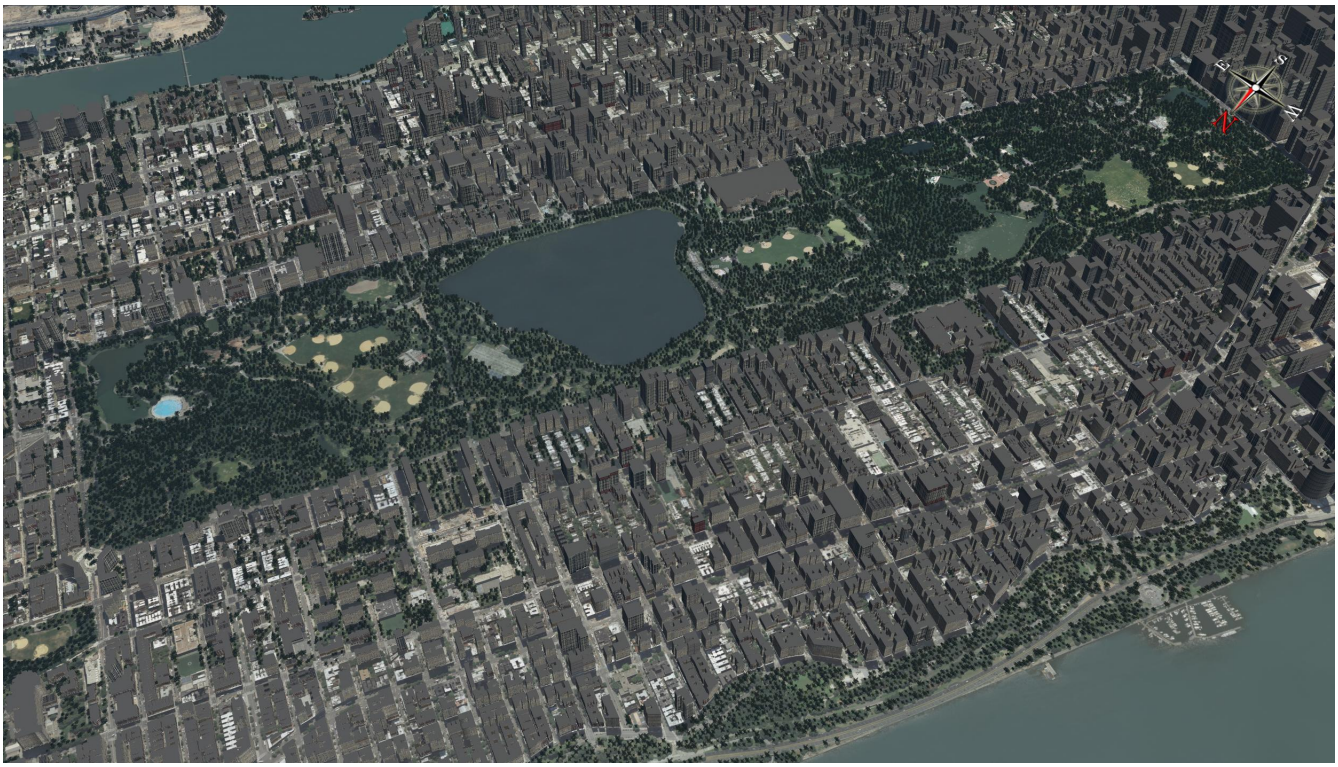

Figure 37. New York CDB dataset from FlightSafety, as visualized in Ecere's 3D client.

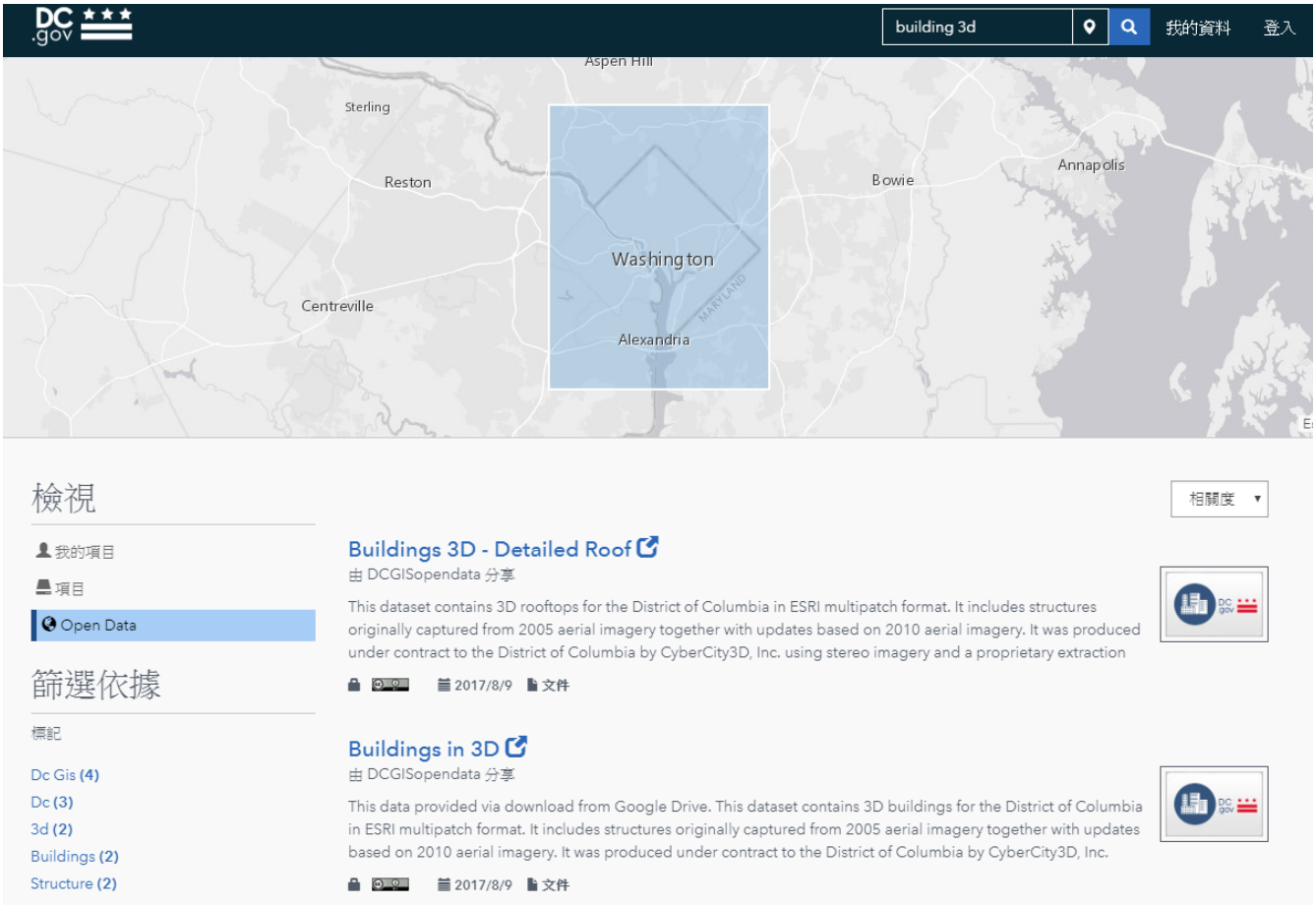

Figure 38. Washington, D.C. 3D buildings dataset from OpenData DC

Some experiments made use of OpenStreetMap data (extracts in Google Protocol Buffer encoding of OpenStreetMap data model retrieved from Geofabrik [https://www.geofabrik.de/] and extract.bbbike.org [https://extract.bbbike.org/]). As well, a COLLADA model of Kaohsiung pipelines was also used in early experiments, converting the model to E3D format (available here [https://portal.opengeospatial.org/files/?artifact_id=80066]).

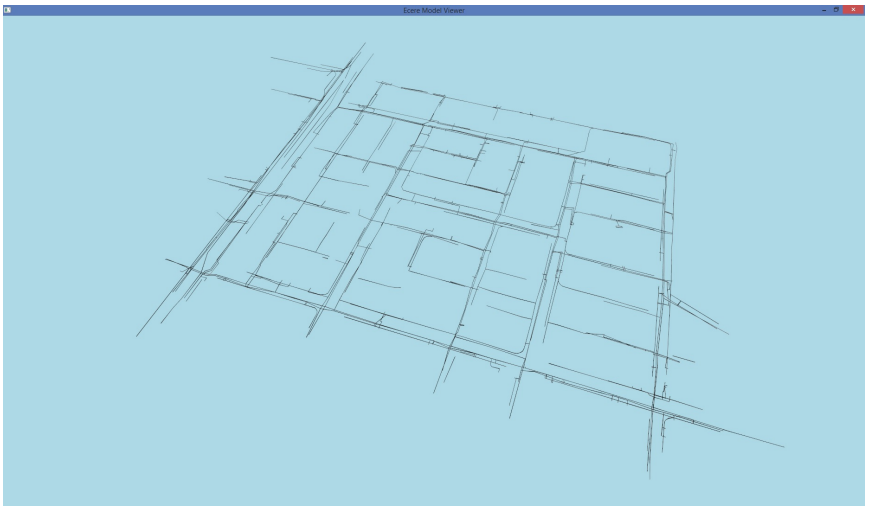

Figure 39. An overview screenshot of the Kaohsiung pipelines, converted to E3D format and visualized in E3D viewer

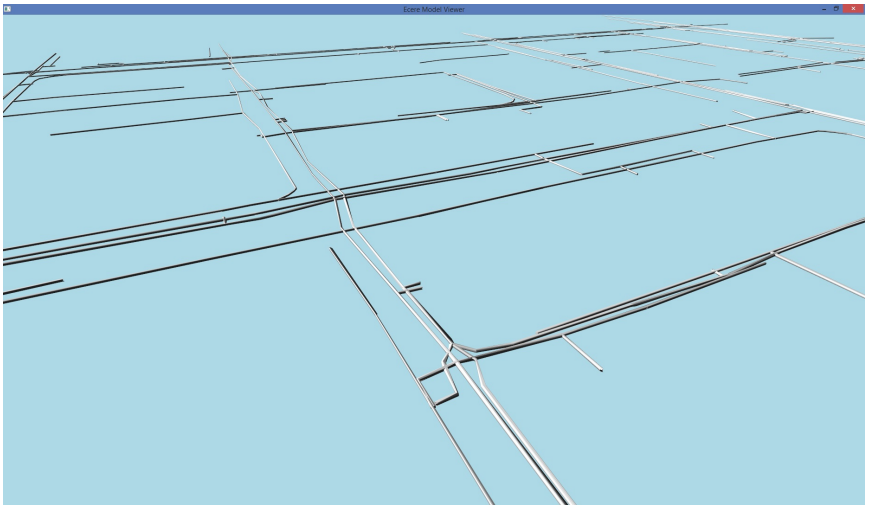

Figure 40. A close-up screenshot of the Kaohsiung pipelines, converted to E3D format and visualized in E3D viewer

Other data sources were identified as potentially useful for conducting additional experiments, but were not experimented with due to time constraints:

- Imagery and Elevation data (from LiDAR survey) available on OpenData DC [http://opendata.dc.gov]
- Textured CityGML data set of Berlin (available from businesslocationcenter.de [https://www.businesslocationcenter.de/en/downloadportal])
- Geo-referenced detailed texture models available as KML / COLLADA models from 3D Warehouse [http://3dwarehouse.sketchup.com]

10.2. Intermediate CityGML data model (GIS-FCU)

An intermediate CityGML data model was produced from Washington, D.C. 3D buildings multipatch shapefiles by GIS-FCU, primarily using Safe Software's FME data integration platform.

Because the original building IDs were not unique, which would cause problems with how these were used later on in the conversion process, ESRI's ArcGIS software was first used to associate a unique ID to each building, prior to the conversion from multipatch shapefiles to CityGML.

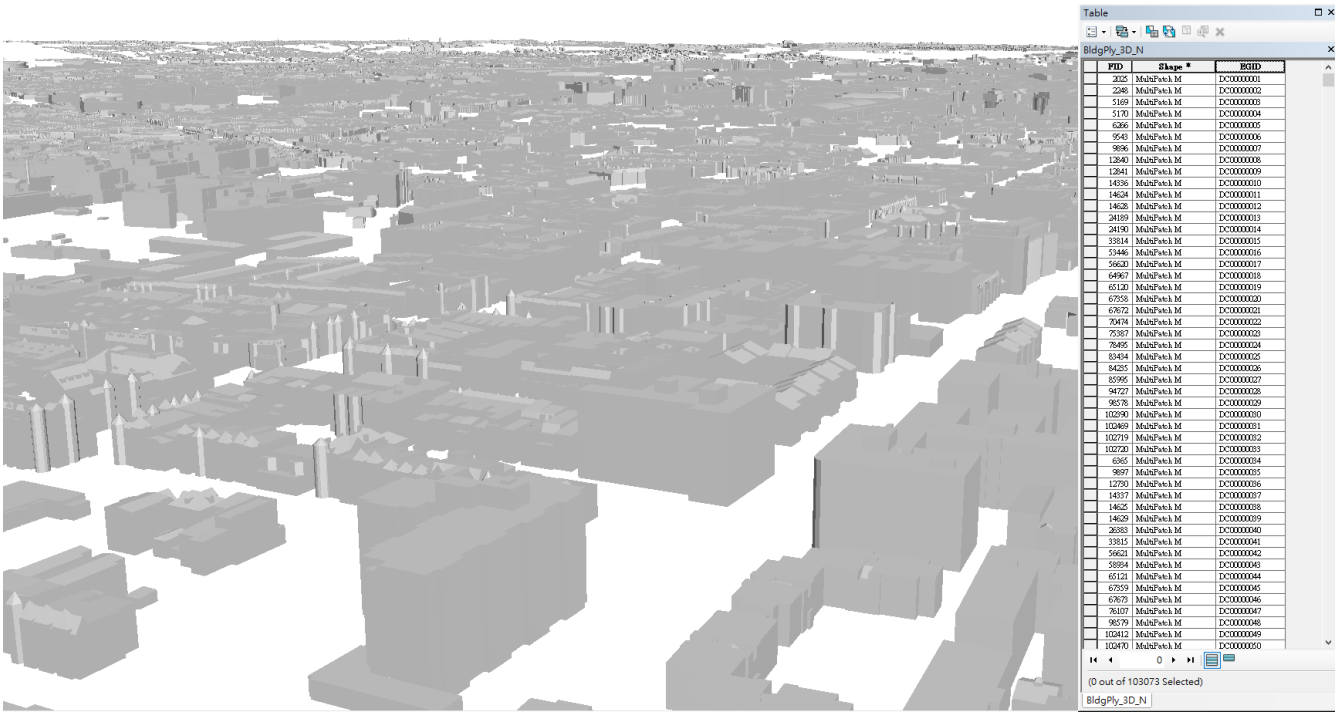

Figure 41. Washington, D.C. data set in FME

Previous research [12] from Technische Universität München (TUM) was instrumental in making this possible, as they shared an FME configuration file specifically for the purpose of creating such a building data model, which was used for setting up the conversion process.

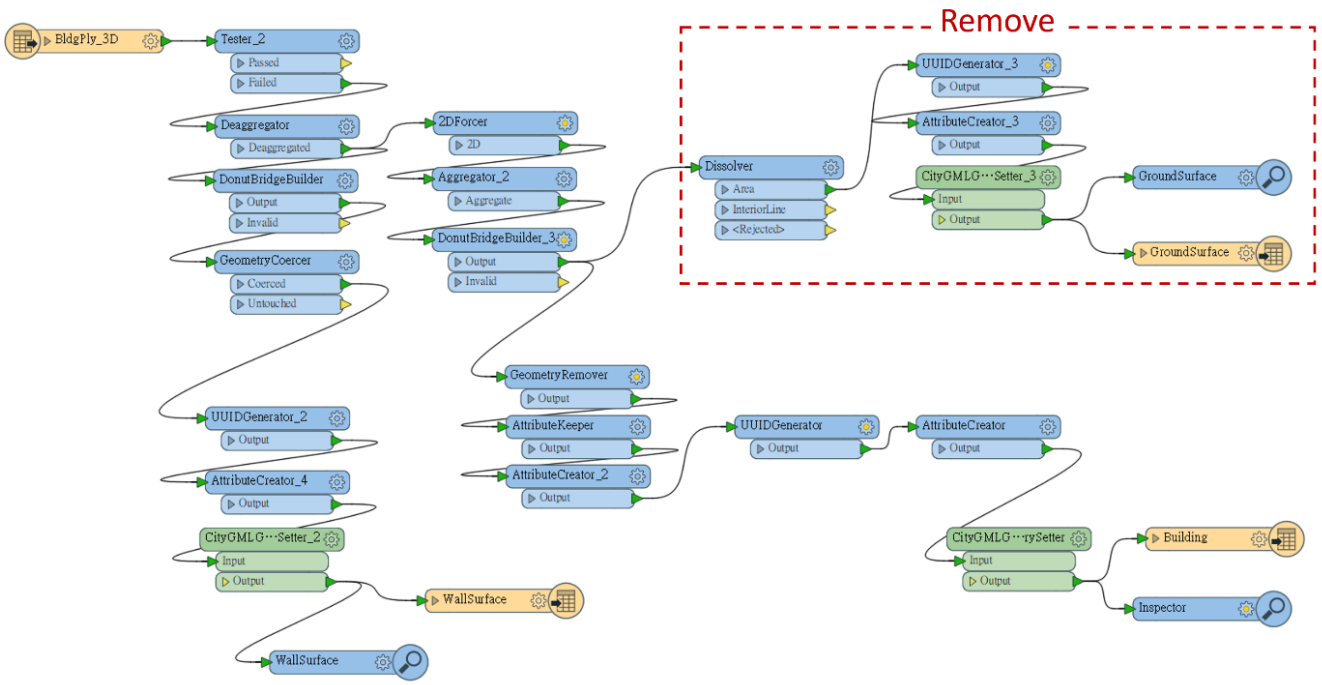

Figure 42. Shapefile to CityGML conversion process

This FME configuration file can be downloaded from here [https://drive.google.com/file/d/19-C5AekVC7QE7cfrdUOws3U2BNVmQO9R/view?usp=sharing].

```
<core:cityObjectMember>
    <bldg:Building gml:id="id_79223bdb-dfe1-4253-9ce6-9a40914ad8a9">
        <gen:stringAttribute name="EGID">
            <gen:value>DC00043056</gen:value>
        </gen:stringAttribute>
        <bldg:lod2MultiSurface/>
    </bldg:Building>
</core:cityObjectMember>
<core:cityObjectMember>
    <bldg:WallSurface gml:id="id_b6afc7e1-adad-45d9-85b9-dc4e5e025b4e">
        <gen:stringAttribute name="EGID">
            <gen:value>DC00043056</gen:value>
        </gen:stringAttribute>
        <bldg:lod2MultiSurface>
            <gml:MultiSurface srsName="EPSG:26985" srsDimension="3">
                <gml:surfaceMember>
                    <gml:Polygon>
                        <gml:exterior>
                            <gml:LinearRing>
                                <gml:posList>399900.4085083008 139717.5048828125 48.81609999999637 39
                            </gml:LinearRing>
                        </gml:exterior>
                    </gml:Polygon>
                </gml:surfaceMember>
            </gml:MultiSurface>
        </bldg:lod2MultiSurface>
    </bldg:WallSurface>
</core:cityObjectMember>
```

Figure 43. Sample data from resulting CityGML data set

The resulting CityGML data set is available here [https://drive.google.com/file/d/1My5n8rycHlIXFjqjCNJtlUjdKObcpLoN/view?usp=sharing].

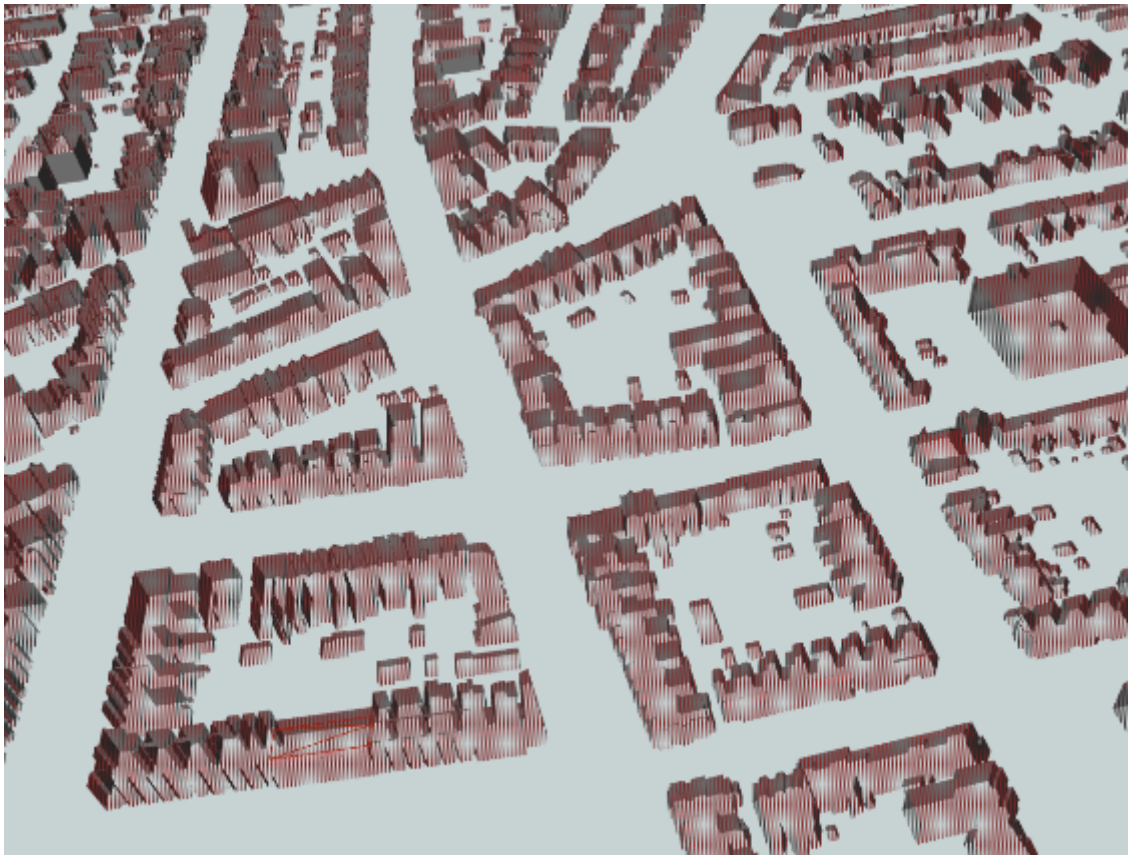

Figure 44. CityGML data set

The final CityGML data set can be previewed and visualized using FME's CityGML inspector.

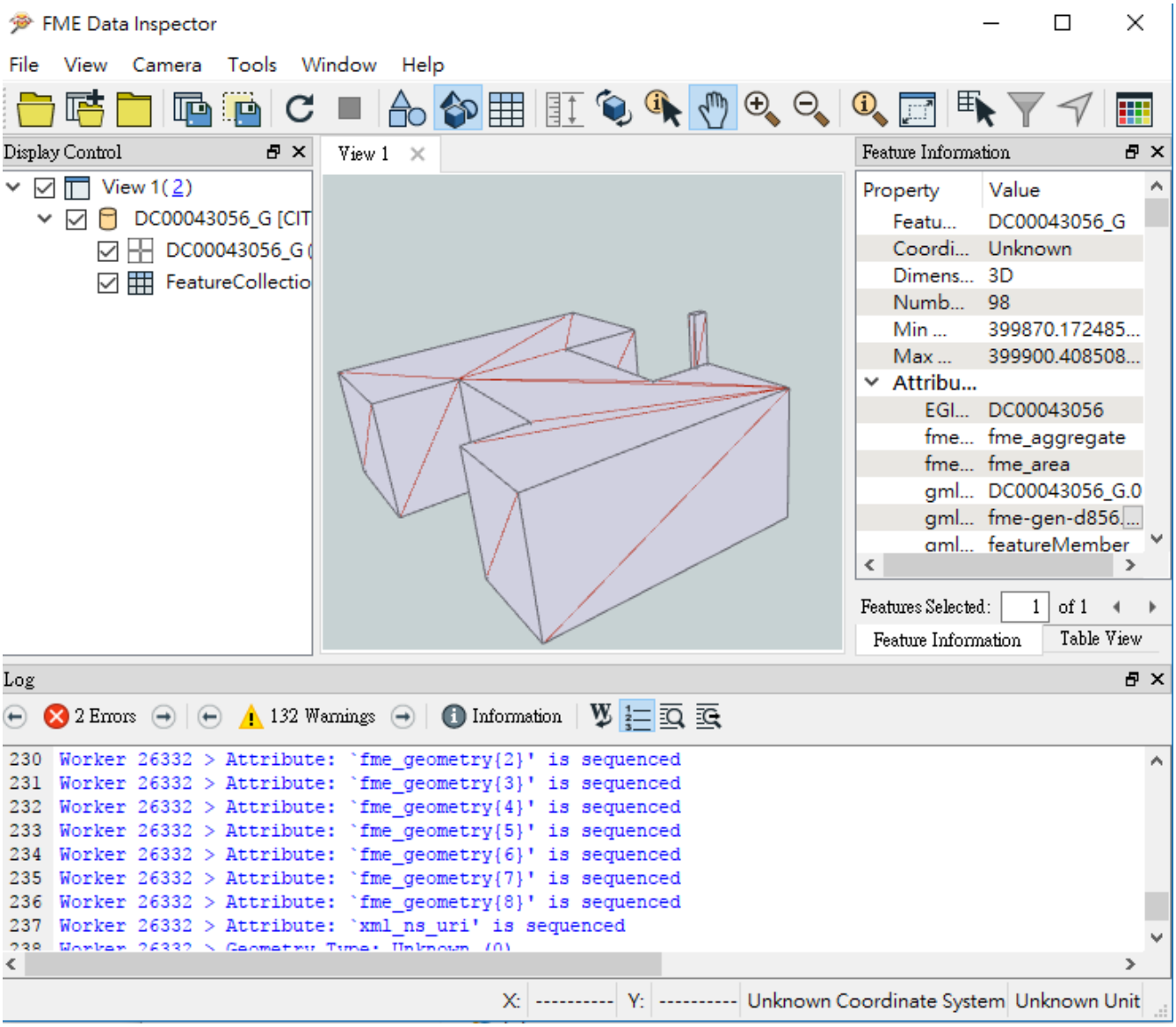

Figure 45. FME CityGML preview

In the resulting CityGML data set, all of the 3D geometry is described using the following GML tags (themselves within CityGML tags such as ***bldg:WallSurface***):

gml:MultiSurface → gml:surfaceMember → gml:Polygon → gml:exterior/interior → gml:LinearRing → gml:posList

The entire preprocessing of the Washington, D.C. 3D buildings data set, from multipatch shapefiles to E3D models through CityGML, is summarized in the following figure:

Figure 46. Washington, D.C. processing steps (from Shapefiles to E3D, through CityGML)

10.3. Conversion from CityGML to E3D models data set (GIS-FCU)

To facilitate the conversion process from CityGML to E3D models, as well as the visualization of E3D models in the GIS-FCU client, Ecere provided GIS-FCU with source code for reading and writing E3D models. This source code, implementing the E3D 3D Model Specifications (Appendix A), will become part of the open-source Ecere cross-platform SDK [http://ecere.org]. Additionally, support for E3D may be contributed to the Open Asset Import Library [http://assimp.org/]. GIS-FCU opted for making use of the Ecere E3D API in its native eC programming language [http://ec-lang.org] to implement the transformation from CityGML to E3D (eC was designed by Ecere, who also maintain the open-source compiler and tools for the language).

After parsing the CityGML to resolve the structure of each building, GIS-FCU implemented batch processing for outputting separate E3D models for each building. These E3D models get stored in a PostGIS database, while establishing reference positions for the buildings. A tool called 'm2S' tool was developed to perform this processing (transforming one input CityGML file into several E3D model files).

Figure 47. A PostGIS (PostgreSQL) database serves as the data store for the GIS-FCU service, storing the E3D files and reference points

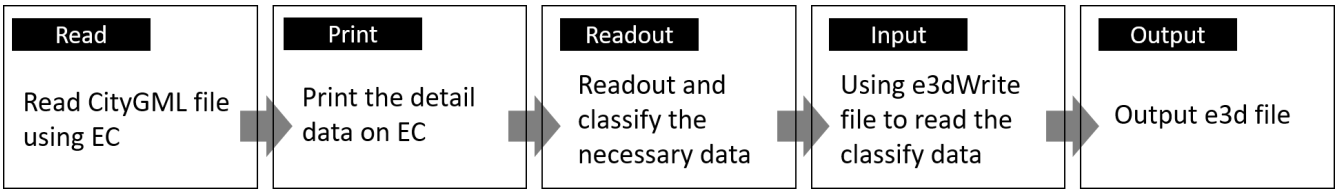

Figure 48. A summary of the approach used by GIS-FCU to generate E3D models for the entire CityGML data set

Ecere also provided a 3D model viewer (e3dView) to visualize and validate the correctness of E3D models and it was used for checking the result of the conversion from CityGML to E3D.

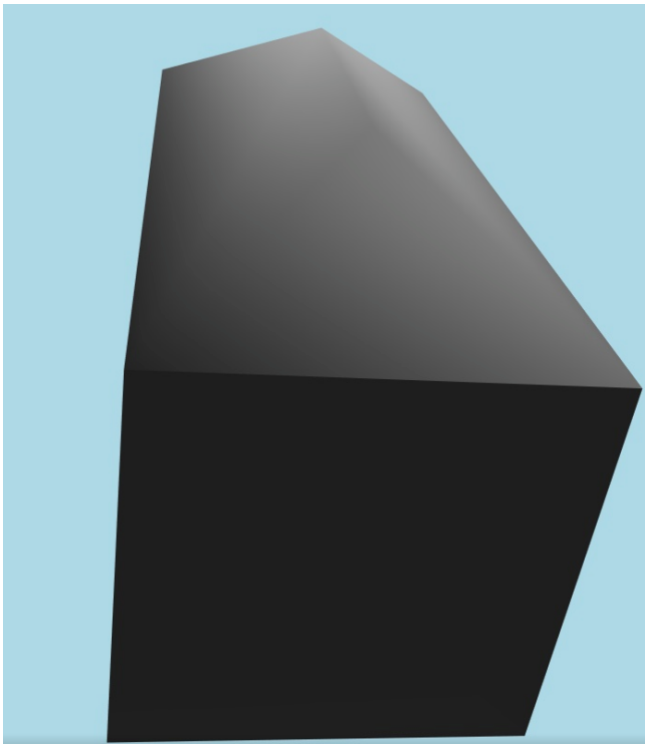

Figure 49. A screenshot of e3dView displaying a single building

10.4. Conversion of CDB data set to GNOSIS data store (Ecere)

The GNOSIS data store serves as input from which the Ecere service (based on the *GNOSIS Map Server*) can directly serve geospatial data in a number of OGC services (e.g. WMTS, WFS) as well as UMS. It can also be used directly for high performance visualization by applications built using Ecere's GNOSIS SDK, such as the mobile Augmented Reality client built for this initiative, or Ecere's GNOSIS Cartographer GIS tool.

The data store (described in detail in Testbed-13 Vector Tiles Engineering Report [http://docs.opengeospatial.org/per/17-041.html] - Appendix D) [13] consists of a simple directory structure with layers of tiled geospatial data in the GNOSIS Map Tiles format (Appendix B) organized by tile pyramids so as to keep a small file count and reduce file system overhead, while providing fast access to geometry without the overhead of a database. Data attributes are stored separately in an SQLite database with spatial indexing. For referenced 3D models as used in this initiative, a *models* subfolder contains separate E3D files, with a *textures* subfolder within for textured models. A *layerInfo.econ* description file contains information such as geospatial data type, geospatial and temporal extent. The data store can store any types of geospatial data supported by GNOSIS Map Tiles, which currently include imagery, coverage, vector data, point clouds, as well as referenced or embedded 3D models.

In order to produce a GNOSIS Map Tiles / E3D data set out of CDB, GNOSIS Cartographer was used. The CDB data set is simply pointed to using the 'Add' button of the data sources library panel, and then with the top-level node of the CDB data set, clicking on the *Optimize* button starts the conversion process.

Within a CDB 1.1 data set, vector data is represented using shapefile tiles. This also applies to points referencing 3D models. The 3D models themselves are stored separately as OpenFlight models,

while their textures are stored using SGI RGB format. Imagery is stored using JPEG-2000 encoding, while GeoTIFF is used for elevation data.

When processing the complete New York CDB data set (as seen in the screenshots on the clients page), the following source layers were used:

- The imagery layers (2 resolutions: ~10 meters / pixel for a larger overview, and ~30 centimeters / pixel for a smaller extent)
- The elevation data layer (~20 meters / pixel)
- The geospecific buildings layer (100_GSFeature / S001 - Man-made point features, along with supporting 300_GSModelGeometry for the OpenFlight 3D models, and 301_GSModelTexture for the RGB textures)
- The geotypical trees layer (101_GSFeature / S002 - Tree point features, along with supporting 500_GTModelGeometry for the OpenFlight 3D models, and 501_GTModelTexture for the RGB textures)

In addition to these layers, the original CDB data set also contains vector roads and railroads layers, but these were not used.

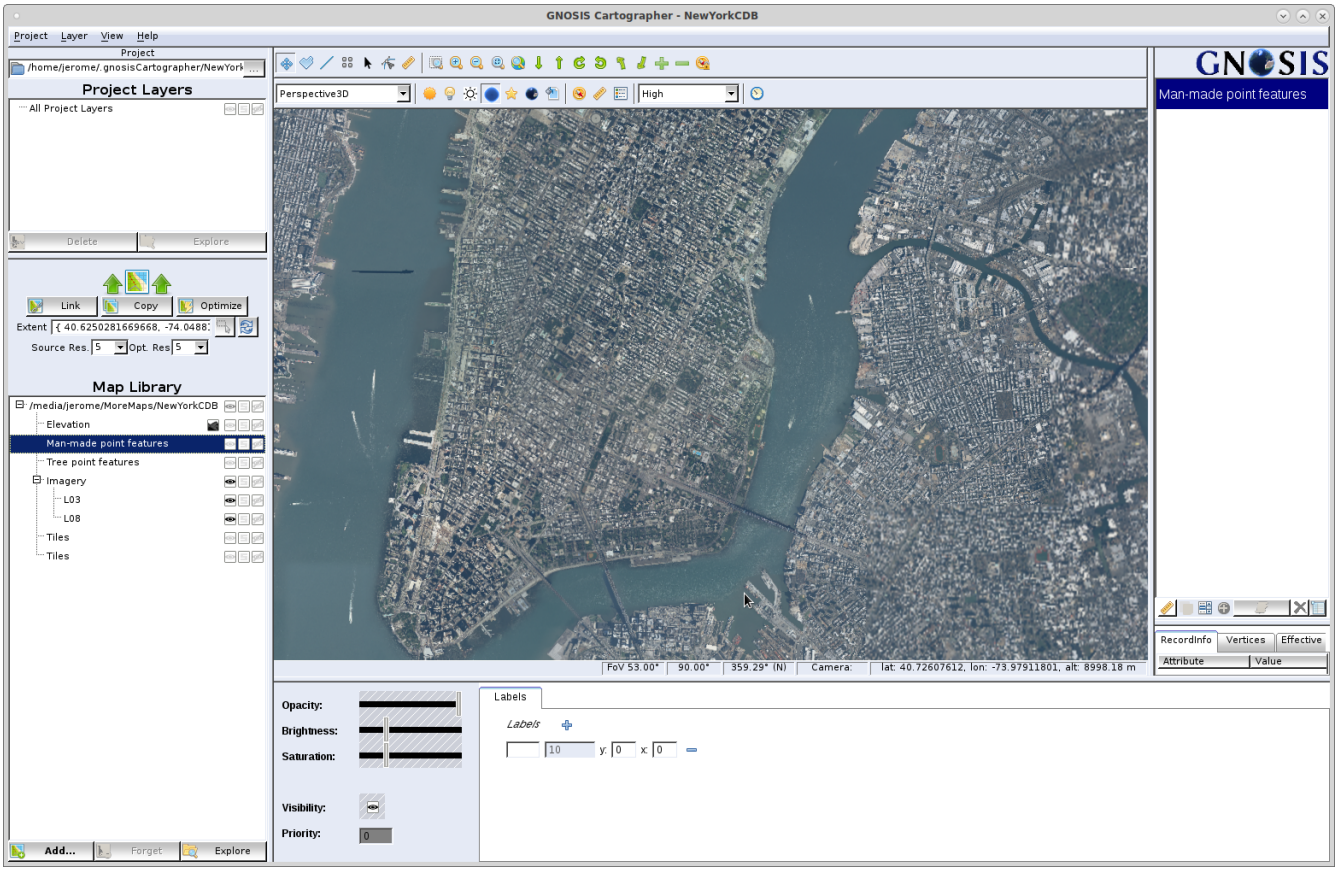

Figure 50. New York CDB data set loaded into GNOSIS Cartographer for conversion process

When converting the 3D models layers, the elevation information from the elevation layer was incorporated into the referencing points of the GNOSIS Map Tiles.

The resulting data set in GNOSIS Map Tiles / E3D format is quite compact, compared with the original source layers, and contains much fewer files:

- Buildings [https://portal.opengeospatial.org/files/?artifact_id=80490] (166 mb)
- Trees [https://portal.opengeospatial.org/files/?artifact_id=80491] (12.51 mb)
- Elevation [https://portal.opengeospatial.org/files/?artifact_id=80492] (187 mb)

The imagery tiles were produced, but not yet using JPEG-2000 encoding within GNOSIS Map Tiles, so they are considerably larger. For this reason, only the lower resolution imagery layer is currently available from the Ecere service.

10.5. Importing OpenStreetMap data (Ecere)

Some experiments used OpenStreetMap data to compensate for two important problems of both the main CDB and shapefiles/CityGML data sets:

- Neither had worldwide coverage where participants were located throughout this initiative (a fundamental issue for Augmented Reality experiments)
- Neither had particularly useful information to display as annotations, not even so much as the name of a building (another major issue for annotating reality)

Luckily, OpenStreetMap provides continuously updated and readily available data with worldwide coverage and a vast amount of textual attributes.

Again, GNOSIS Cartographer was used to convert various OpenStreetMap data sets (retrieved as extracts from the OSM database in OSM PBF format) of where participants happened to be during the Testbed initiative to GNOSIS Map Tiles (as regular vector data layers):

- Ottawa / Gatineau, Canada
- Washington, D.C.
- Stuttgart, Germany
- North Carolina

In addition to displaying roads and information such as building and street names, some experiments used the OpenStreetMap building footprints together with their data attributes to extrude these into 3D buildings. The OpenStreetMap *Simple 3D Buildings* [https://wiki.openstreetmap.org/wiki/Simple_3D_buildings] schema allows to describe 3D geometry (up to a certain amount of details) out of regular polygons. This however requires both mappers to invest considerable time, as well as a lot of efforts from the client to fully and properly support all of the possible attributes (e.g. special roof shapes).

In one experiment, the OpenStreetMap data was used together with 3D data from the GIS-FCU service to add streets and building names.

In other experiments, the data was used on mobile devices for displaying information relevant to the location, based on the device's position and orientation.

The Washington, D.C. OpenStreetMap layers are being served from the Ecere service e.g. through its WFS 3 / Next Generation service [http://maps.ecere.com/hms/collections/osmDC].

Appendix A: E3D 3D Model Specifications

NOTE	The latest version of these specifications is maintained at http://ecere.com/E3D.pdf

A.1. Coordinate system

The coordinate system for E3D models is left-handed:

- x is positive to the right
- y is positive up
- z is positive going away into the distance

A.2. A simple cube example

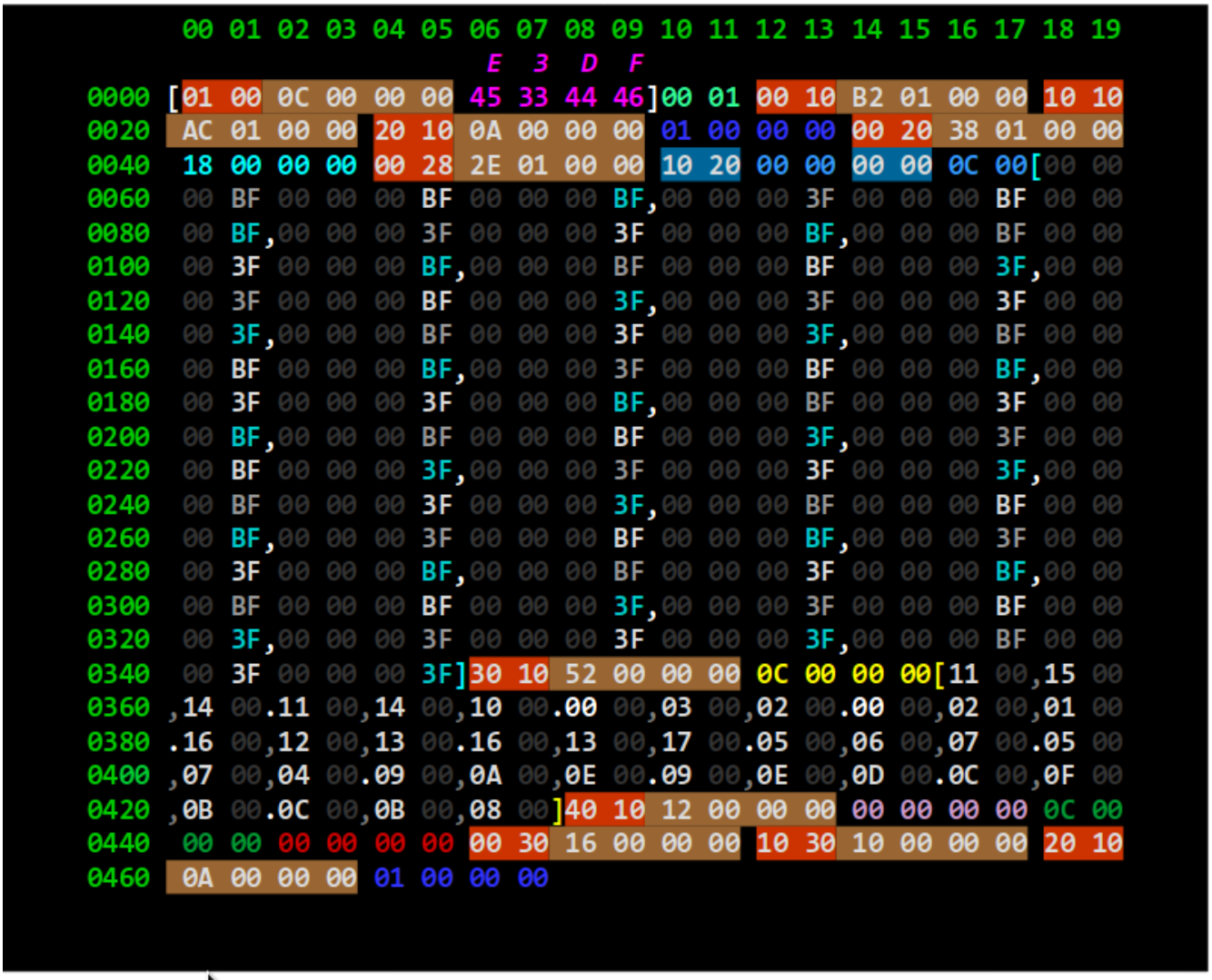

Figure 51. cube1.e3d (468 bytes)

Little-endian throughout (least significant bytes first)

Table 7. Example E3D encoding of a simple cube

Offset	Block Type	Block Length	Contents	Description
0	0x0001 ***(Version)***	0x0000000C (12) bytes [0..11]	E3DF 0x0100 (1, 0)	E3D Version 1.0
12	0x1000 (Meshes)	0x000001B2 (434) bytes [12..445]	sub-blocks: Mesh	1 mesh in this file

Offset	Block Type	Block Length	Contents	Description
18	► 0x1010 (Mesh)	0x000001AC (428) bytes [18..445]	sub-blocks: MeshID, Attributes, TriFaces16, FacesMaterials	Mesh description
24	►►0x1020 (MeshID)	0x0000000A (10) bytes [24..33]	0x00000001	Mesh ID: 1
34	►► 0x2000 (Attributes)	0x00000138 (312) bytes [34..345]	0x00000018 (24) sub-blocks: Interleaved	24 vertices, interleaved attributes for each vertex
44	►►► 0x2800 (Interleaved)	0x0000012E (302) bytes [44..345]	0x2010 (Vertices) 0x0000 (0) 0x0000 (0) 0x000C (12) [...vertex data...] (24 vertices)	The interleaved attributes contain only (x,y,z) 32-bit floating-point vertices at offset 0 (0 type ends list of attribute types); for a total of 12 bytes per vertex.

Offset	Block Type	Block Length	Contents	Description
58	Vertex data	(288 bytes) bytes [58..402]	BF000000,BF000000,BF000000, 3F000000,BF000000,BF000000, 3F000000,3F000000,BF000000, BF000000,3F000000,BF000000, BF000000,BF000000,3F000000, 3F000000,BF000000,3F000000, 3F000000,3F000000,3F000000, BF000000,3F000000,3F000000, BF000000,BF000000,BF000000, 3F000000,BF000000,BF000000, 3F000000,3F000000,BF000000, BF000000,3F000000,BF000000, BF000000,BF000000,3F000000, 3F000000,BF000000,3F000000, 3F000000,3F000000,3F000000, BF000000,3F000000,3F000000, BF000000,BF000000,BF000000, 3F000000,BF000000,BF000000, 3F000000,3F000000,BF000000, BF000000,3F000000,BF000000, BF000000,BF000000,3F000000, 3F000000,BF000000,3F000000, 3F000000,3F000000,3F000000, BF000000,3F000000, 3F000000,	{-0.5,-0.5,-0.5 }, { 0.5,-0.5,-0.5 }, { 0.5, 0.5,-0.5 }, {-0.5, 0.5,-0.5 }, {-0.5,-0.5, 0.5 }, { 0.5,-0.5, 0.5 }, { 0.5, 0.5, 0.5 }, {-0.5, 0.5, 0.5 }, {-0.5,-0.5,-0.5 }, { 0.5,-0.5,-0.5 }, { 0.5, 0.5,-0.5 }, {-0.5, 0.5,-0.5 }, {-0.5,-0.5, 0.5 }, { 0.5,-0.5, 0.5 }, { 0.5, 0.5, 0.5 }, {-0.5, 0.5, 0.5 }, {-0.5,-0.5,-0.5 }, { 0.5,-0.5,-0.5 }, { 0.5, 0.5,-0.5 }, {-0.5, 0.5,-0.5 }, {-0.5,-0.5, 0.5 }, { 0.5,-0.5, 0.5 }, { 0.5, 0.5, 0.5 }, {-0.5, 0.5, 0.5 }

Offset	Block Type	Block Length	Contents	Description
Although a cube only has 8 vertices, this cube describes 24 vertices so as to be ready for adding additional attributes such as normals, which will differ depending on which face it is being referenced by (because a cube has faces at a square angle and the normals pointing away from the faces are very different therefore not averaged at the shared vertices / corners).				
346	► 0x1030 (TriFaces16)	0x00000052 (82) bytes [346..427]	0x0000000C (12) [...16-bit tri indices...]	Count of 12 triangle faces described as triplets of indices into attributes (12 faces, 36 indices)
356	Faces data	(72 bytes) bytes [356..427]	0x11, 0x15, 0x14, 0x11, 0x14, 0x10, 0x00, 0x03, 0x02, 0x00, 0x02, 0x01, 0x16, 0x12, 0x13, 0x16, 0x13, 0x17, 0x05, 0x06, 0x07, 0x05, 0x07, 0x04, 0x09, 0x0A, 0x0E, 0x09, 0x0E, 0x0D, 0x0C, 0x0F, 0x0B, 0x0C, 0x0B, 0x08	{ 17, 21, 20 }, { 17,20, 16 }, { 0, 3, 2 }, { 0, 2, 1 }, { 22, 18, 19 }, { 22, 19, 23 }, { 5, 6, 7 }, { 5, 7, 4 }, { 9, 10, 14 }, { 9, 14, 13 }, { 12, 15, 11 }, { 12, 11, 8 } (2 triangles per cube square faces)
428	► 0x1040 (Faces Materials)	0x00000012 (18) bytes [428..445]	0x00000000 (0) 0x0000000C (12) 0x00000000 (0)	First face: 0 (indices: × 3) Faces count: 0 (indices: × 3) Material ID: 0 (none)
446	0x3000 (Nodes)	0x00000016 (22) bytes [446..467]	sub-blocks: MeshNode	1 node in this file (instance of a mesh)
452	► 0x3010 (MeshNode)	0x00000010 (16) bytes [452..467]	sub-blocks: MeshID	This node references a mesh in the meshes list by ID. Because no transformation is specified, the defaults apply: (1,1,1) scaling; (0,0,0) offset; (w=1,0,0,0) quaternion orientation
458	►► 0x1020 (MeshID)	0x0000000A (10)	0x00000001 (1)	This references mesh ID 1.

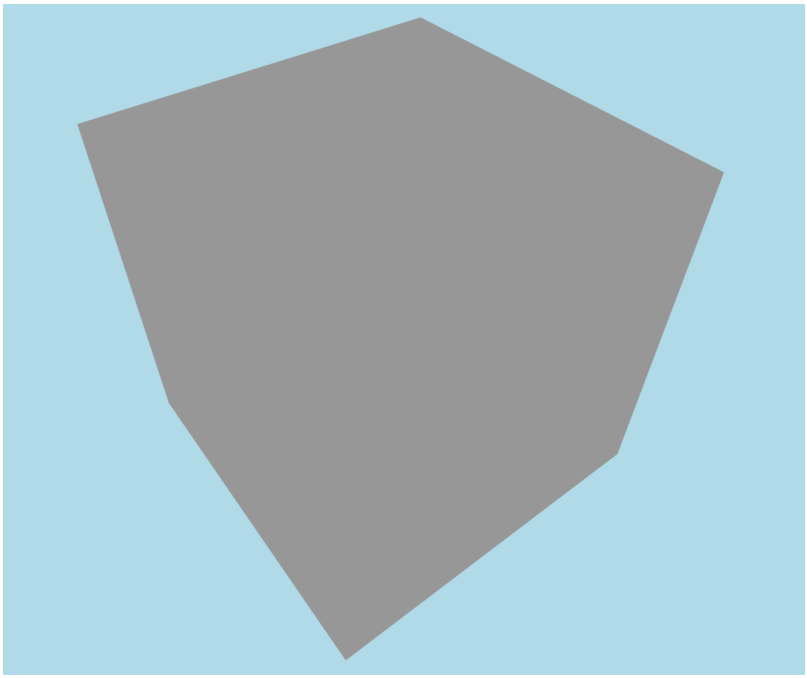

Figure 52. cube1.e3d

A.3. Adding normals attributes

This version adds normals to the interleaved attributes (with x,y,z packed using 10 bits each).

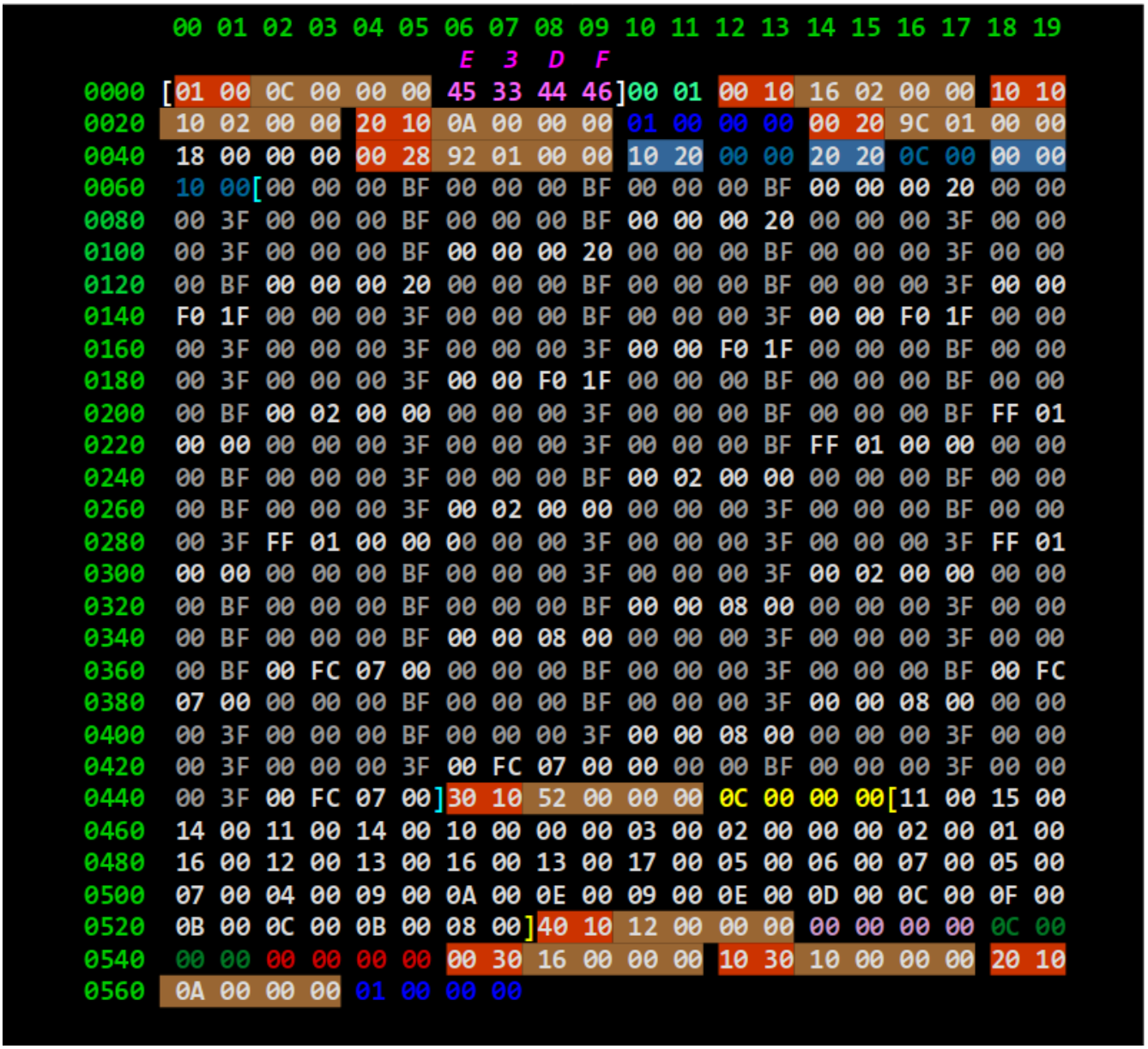

Figure 53. cube2.e3d (568 bytes) — A cube with normals

The floating-point normal values for the normals are (0 is implied for non-specified component values):

{ z = -1.0 }, { z = -1.0 }, { z = -1.0 }, { z = -1.0 },

{ z = 1.0 }, { z = 1.0 }, { z = 1.0 }, { z = 1.0 },

{ x = -1.0 }, { x = 1.0 }, { x = 1.0 }, { x = -1.0 },

{ x = -1.0 }, { x = 1.0 }, { x = 1.0 }, { x = -1.0 },

{ y = -1.0 }, { y = -1.0 }, { y = 1.0 }, { y = 1.0 },

{ y = -1.0 }, { y = -1.0 }, { y = 1.0 }, { y = 1.0 }

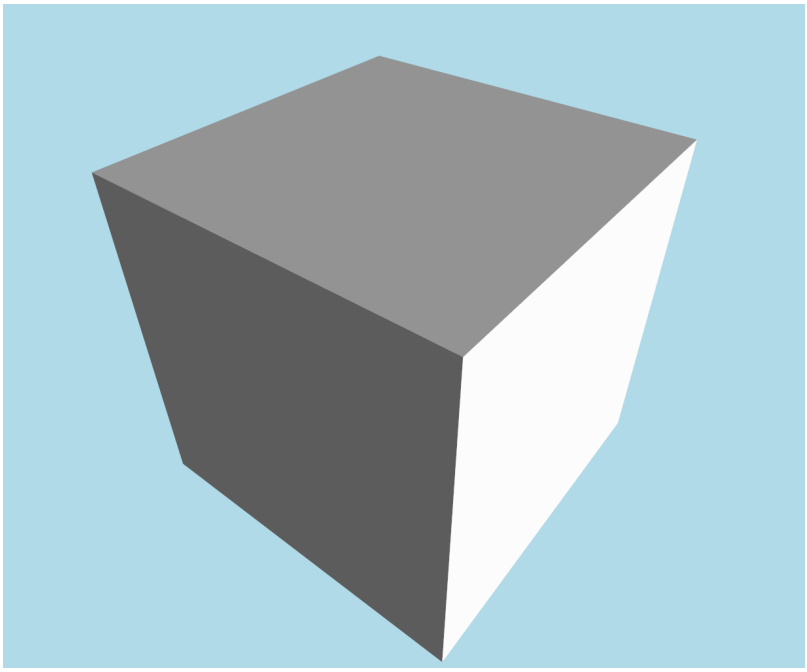

Figure 54. cube2.e3d & cube3.e3d

A.4. Compression

This version compresses the data using LZMA (any series of blocks, except the version header block, can be compressed inside an LZMA block). Here the top blocks are compressed in one LZMA block.

```
        00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19
                              E  3  D  F
0000    01 00 0C 00 00 00 45 33 44 46 00 01 10 00 BD 00 00 00 2C 02
0020    00 00[DC 00 00 00 04 00 00 04 02 B0 15 03 82 28 18 3A 38 A6
0040    99 EC BF 18 10 47 E3 91 48 98 8F 50 22 61 93 3E 8C 6C 88 45
0060    1B FF 72 35 4E 14 9C 47 EF 2C CB 0E F9 9D 2B C7 34 5A 3C 03
0080    20 A6 38 D2 95 83 46 86 AE B5 3D 4C 83 22 EE 11 1A 46 42 C7
0100    52 53 28 12 83 13 D9 BF 2B ED AD B1 68 BF B9 60 DA C7 9D 23
0120    07 73 3A A9 65 B8 6F 82 6D E5 43 57 0B 1D 51 A6 EB 19 D3 68
0140    14 9F DC 26 50 92 3D 9E D4 58 90 95 EF 3B C2 11 C1 85 11 73
0160    D8 4F D2 DC A7 86 8D 3C FA 91 00 8C AB 68 C1 90 73 D7 C9 95
0180    C9 8C A1 60 7F 13 1D 93 5A EC AE AA E9 62 22 F1 06 CB A5 51
0200    41]
```

Legend

[10 00	Block Type	0x0010 (*LZMA*)
BD 00 00 00	Block Size	
45 33 44 46]	Format Signature	
00 01	Version Number	
2C 02 00 00	Uncompressed size	0x022C (556 bytes)
[...]	Compressed data (a file with blocks)	*12 bytes less (Version block) than 568 bytes of cube2.e3d*

Figure 55. cube3.e3d (201 bytes) — compressed with LZMA.

A.5. Detailed description of all block types

Table 8. Detailed description of all E3D block types

Block Type	Value	Description
version	0x0001	uint16: major (high), minor (low)
lzma	0x0010	size: uint16, compressed data Compression can be applied done at any block-level.
meshes	**0x1000**	Section to describe meshes.
►mesh	0x1010	Describe a single mesh with a unique set of attributes.
►►meshID	0x1020	(uint) Defines (within mesh) or refers to (within meshNode) a unique ID for the mesh.
►►meshBBox	0x1021	float: loX, loY, loZ, hiX, hiY, hiZ
►►attributes	0x2000	uint count (limit of 65,536 vertices, multiple meshes should be used for more); one interleaved and/or multiple attributes sub-blocks.
►►►vertices	0x2010	3x (x,y,z) 32-bit floats
►►►verticesDbl	0x2011	3x (x,y,z) 64-bit doubles
►►►verticesQ	0x2018	3x (x,y,z) 16-bit signed integer (quantized to meshBBox)
►►►normals	0x2020	3 components (x,y,z) stored as signed 10_10_10_2 [https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_vertex_type_2_10_10_10_rev.txt] format (10 bits per component: -1..1 range mapped to -511..511)
►►►texCoords	0x2030	Texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1 for covering the entire texture (beyond that range for tiling)
►►►texCoords2	0x2031	Second set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►texCoords3	0x2032	Third set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1

Block Type	Value	Description
►►►texCoords4	0x2033	Fourth set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►texCoords5	0x2034	Fifth set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►texCoords6	0x2035	Sixth set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►texCoords7	0x2036	Seventh set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►texCoords8	0x2037	Eighth set of texture coordinates — 2 components (u,v) as 32-bit floats ranging from 0..1
►►►colors	0x2070	4 components (r,g,b,a) as 8-bit integers (0..1 range mapped to 0..255)
►►►tangentsSign	0x2080	Tangents as 3 signed components (x,y,z) in 10_10_10_2 [https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_vertex_type_2_10_10_10_rev.txt] format (10 bits per component: -1..1 range mapped to -511..511), with the first extra bit used to indicate sign for re-constructing the co-tangent (orthogonal to normal and tangent)
►►►tangentsBi	0x2081	Tangents and bi-tangents as 6 components 2x (x,y,z) signed 10_10_10_2 [https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_vertex_type_2_10_10_10_rev.txt] format (10 bits per component: -1..1 range mapped to -511..511)
►►►skin	0x2090	Reserved for defining skins (bone IDs and weights)

Block Type	Value	Description
►►►interleaved	0x2800	Define multiple attributes interleaved First a list of included attributes as: uint16 type, offset. 0 (uint16) is used to end the list, followed by the total size of attributes per vertex (also uint16).
►►►custom	0x4000:0x5FFF	Custom attributes definitions.
►►triFaces16	0x1030	Triangles (3 indices per triangles — each unsigned, 16-bit)
►►triFaces32	0x1031	Triangles (3 indices per triangles — each unsigned, 32-bit)
►►facesMaterials	0x1040	Start triangle (uint), count of triangles (uint), material ID (uint) (this could be a reference to an external materials table, e.g. if description is omitted)
►►bones	0x1050	Reserved for bones definition
►►parts	0x1060	For each part: (uint) Part ID; (uint) Start triangle within triFaces list; (uint) Count of triangles For intra-model attribution. Attributes stored/queried separately in a database (or embedded as special block type).
nodes	**0x3000**	Section to define nodes instancing meshes, cameras and lights.
►meshNode	0x3010	A node to instance a mesh.
►►nodeID	0x3020	(uint) Defines or refers to a node ID.
►►nodeName	0x3021	(String) Defines or refers to a node name.
►►scaling	0x3030	Defines scaling transformation for a node as 3x (x,y,z) 32-bit float scale factors.

Block Type	Value	Description
►►orientation	0x3031	Defines orientation transformation for a node as a quaternion 4x (w,x,y,z) 64-bit doubles.
►►position	0x3032	Defines translation transformation for a node as 3x (x,y,z) 64-bit doubles.
►cameraNode	0x3011	Reserved for defining a camera.
►lightNode	0x3012	Reserved for defining a light.
materials	**0x8000**	Section to define materials.
►material	0x8010	Describes the real-world appearance of this material in the scene. Both classic Phong shading model and Physically Based Rendering (PBR) properties can be specified. All properties are optional, defaulting to white non-textured.
►►materialID	0x8011	(uint) Defines the ID for the material (referenced by FacesMaterials bock).
►►materialName	0x8012	(String) Defines the name of the material.
►►materialGroup	0x8013	(uint) An ID which can be used to regroup compatible materials where a given map (e.g. phongDiffuseMap) is of identical dimensions, allowing to leverage array textures. One material can then correspond to a layer of the array texture (e.g. the material ID could be used as the layer ID).

Block Type	Value	Description
►►materialFlags	0x8020	bit 0: double-sided *(default to true)* ; bit 1: partly transparent (true if textures not entirely opaque) *(default to false)* bit 2: translucent flag (true if textures contain siginificant semi-opaque portions) *(default to false)* *The distinction between partly transparent and translucent will suggest very different approaches to handling transparency.* bit 3: wrapU: tile texture horizontally if set; clamp otherwise *(default to clamp)* bit 4: wrapV: tile texture vertically if set; clamp otherwise *(default to clamp)*
►►opacity	0x8021	(float) 1 meaning fully opaque (default); 0 fully transparent
►►refractionRelIndex	0x8022	(float) Relative refraction index *(Default to 1.0)* (refractive index / container refractive index) *examples of refraction indices: vacuum : 1.0, glass: 1.5; water: 1.333*
►►reflectivity	0x8023	(float) Reflectivity *(default to 0.0 — non-reflective)*
►►phongShininess	0x8024	(float) Shininess (Phong Model exponent: sharpness of specular highlight)
►►diffuse	0x8030	3 floats (r,g,b) Diffuse *(default to white)*
►►specular	0x8031	3 floats (r,g,b) Specular *(default to diffuse color or white)*
►►emissive	0x8032	3 floats (r,g,b) Emissive *(default to black — non-emissive)*
►►ambient	0x8034	3 floats (r,g,b) Ambient *(default to diffuse color or white)*
►►emissiveMap	0x8100	Emissive map (reference to a sharable texture via a textureID sub-block)

Block Type	Value	Description
►►normalMap	0x8101	Normal map (reference to a sharable texture via a textureID sub-block)
►►heightMap	0x8102	Height displacement map (reference to a sharable texture via a textureID sub-block)
►►ambientOcclusionMap	0x8103	Ambient occlusion map (reference to a sharable texture via a textureID sub-block)
►►phongDiffuseMap	0x8200	Diffuse & opacity map (reference to a sharable texture via a textureID sub-block)
►►phongSpecularMap	0x8201	Specular map (reference to a sharable texture via a textureID sub-block)
►►phongAmbientMap	0x8202	Ambient map (reference to a sharable texture via a textureID sub-block)
►►pbrRMAlbedo	0x8300	Albedo texture for Roughness/Metalness PBR model (reference to a sharable texture via a textureID sub-block)
►►pbrRMRoughnessMetalness	0x8301	Roughness/Metalness texture for Roughness/Metalness PBR model (reference to a sharable texture via a textureID sub-block)
►►pbrSpecDiffuseMap	0x8400	Diffuse Map for Specular/Glossiness PBR model (reference to a sharable texture via a textureID sub-block)
►►pbrSpecSpecularGlossMap	0x8401	Specular/Glossiness Map for Specular PBR model (reference to a sharable texture via a textureID sub-block)
textures	**0x9000**	Section to define textures.
►texture	0x9001	Definition of a single texture.
►►textureID	0x9002	Defines or refers to a sharable texture using a unique ID.
►►textureName	0x9003	(String) Defines or refers to a sharable texture by name.
►►texturePNG	0x9101	Embeds a PNG-encoded texture.

Block Type	Value	Description
► ►textureJPG	0x9102	Embeds a JPEG-encoded texture.
► ►textureJPG2K	0x9103	Embeds a JPEG2000-encoded texture.
animations	**0xA000**	Section reserved to define animations.

A.6. Sample E3D models

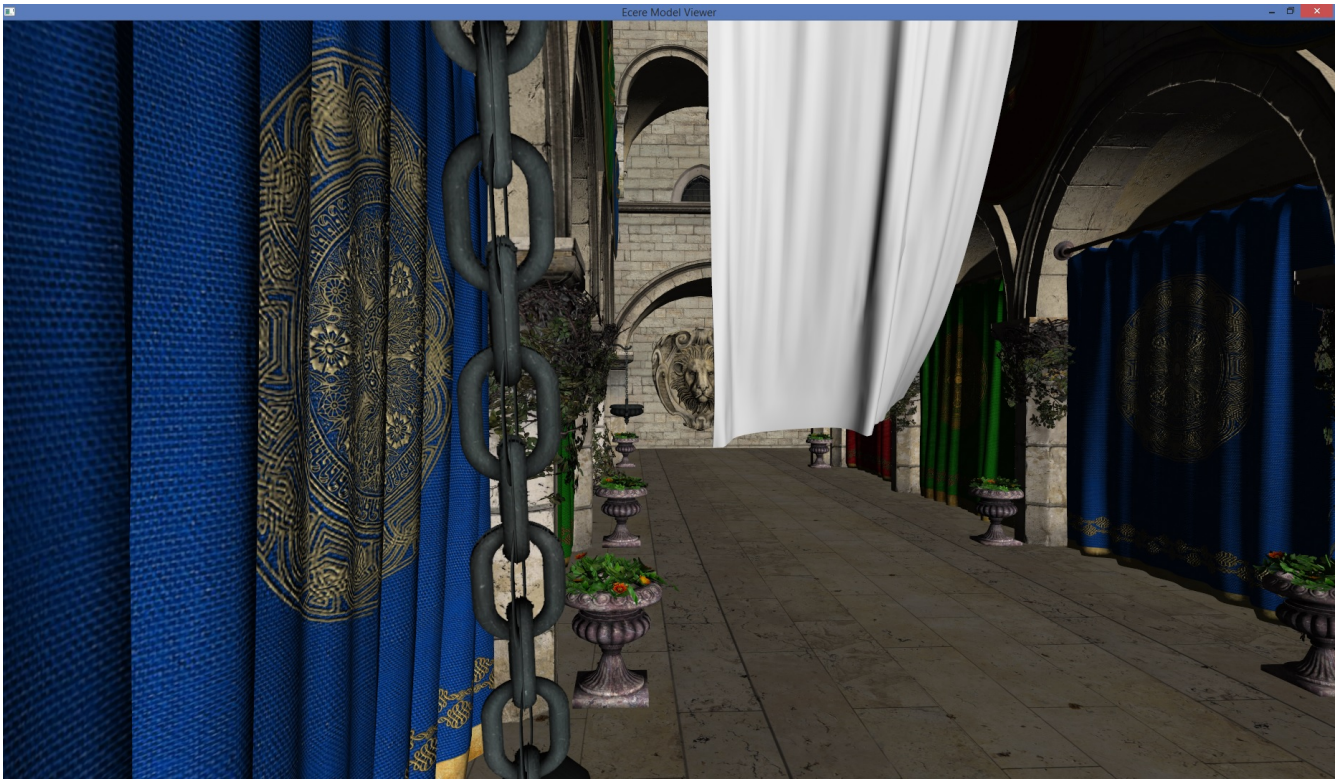

Figure 56. sponza.e3d (14.9 mb – with all textures embedded) — Crytek Sponza Atrium from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

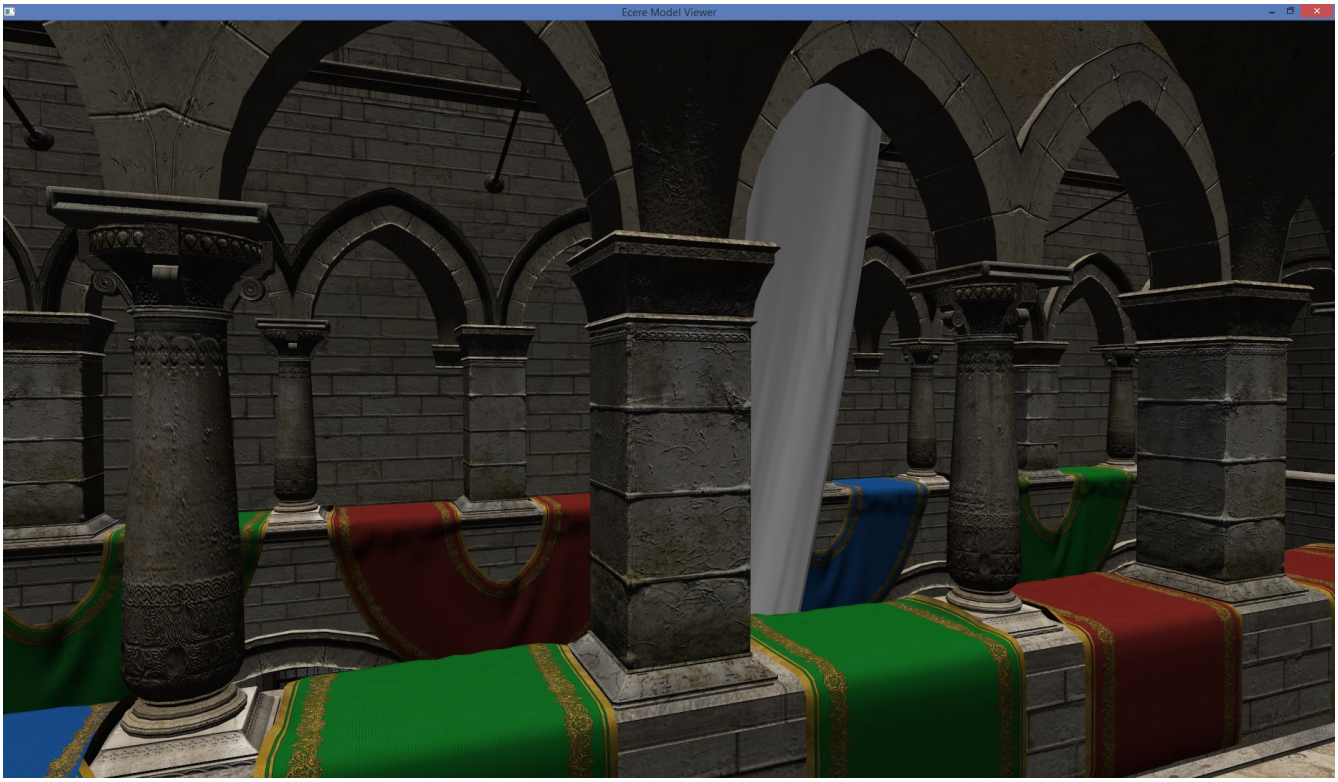

Figure 57. sponza.e3d (14.9 mb – with all textures embedded) — Crytek Sponza Atrium from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

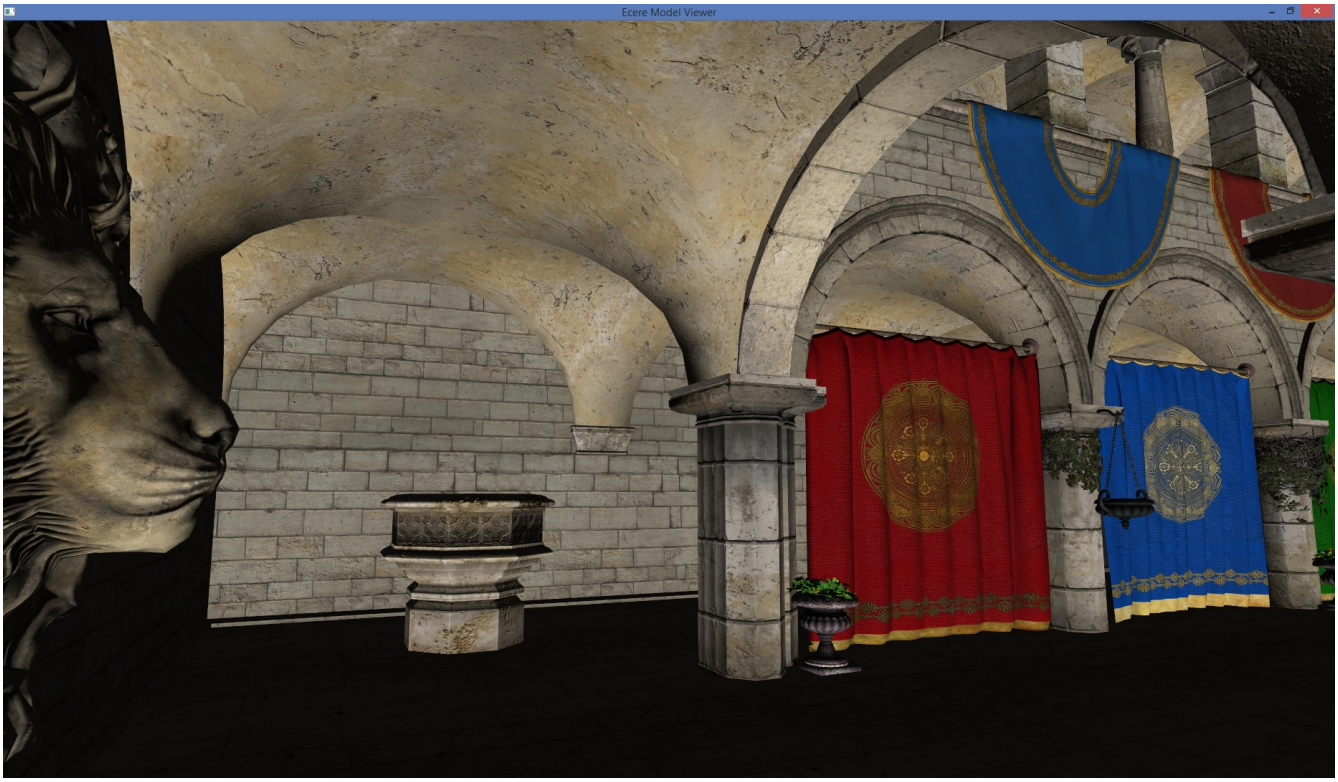

Figure 58. sponza.e3d (14.9 mb – with all textures embedded) — Crytek Sponza Atrium from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

NOTE	PBR (Physically Based Rendering) textures for Sponza Atrium available from http://www.alexandre-pestana.com/pbr-textures-sponza/

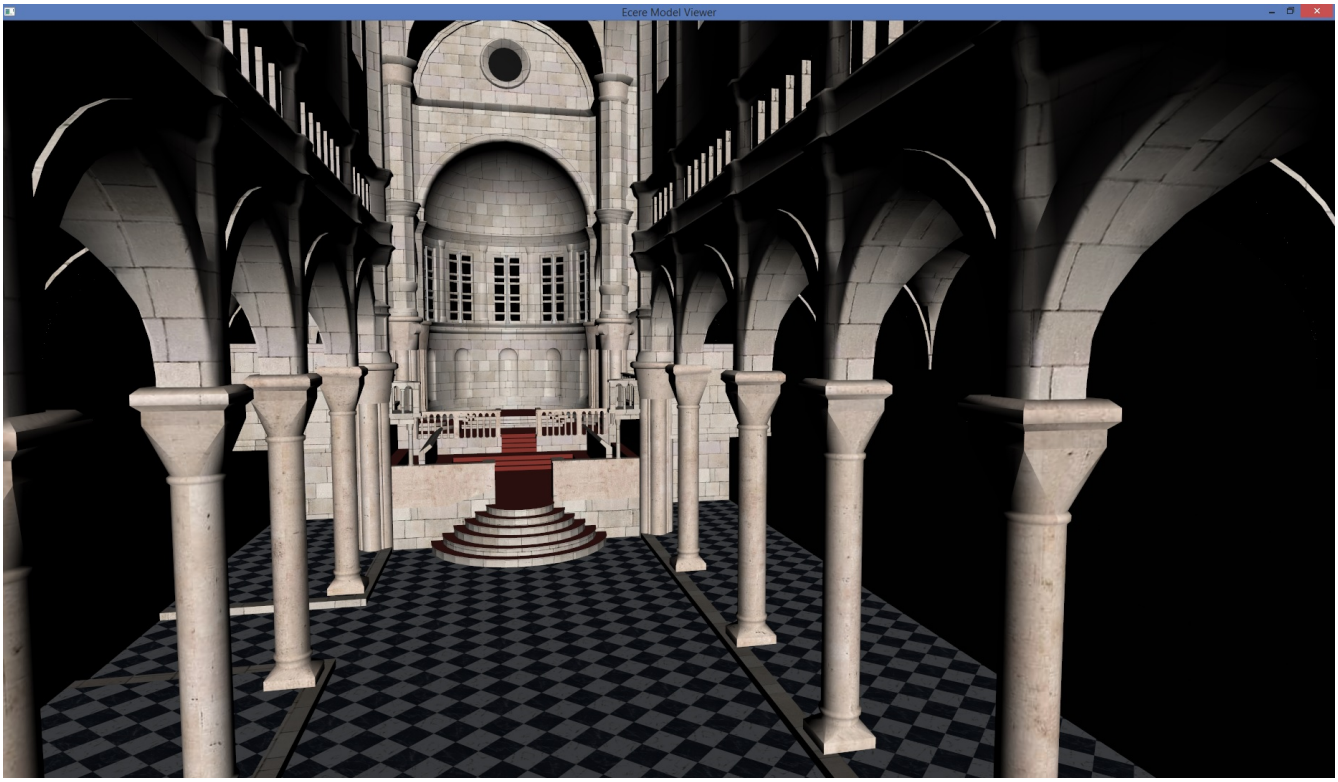

Figure 59. sibenik.e3d (880 kb) — Sibenik Cathedral from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

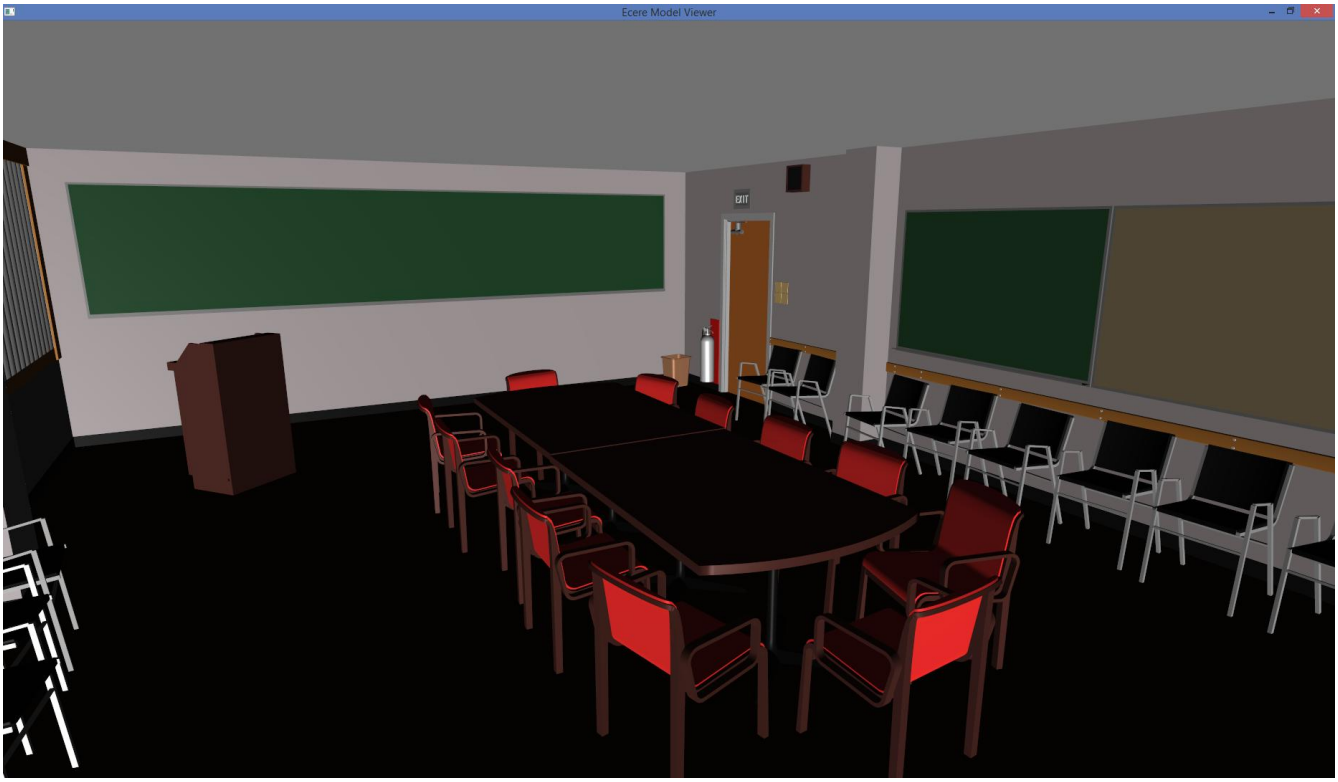

Figure 60. conference.e3d (1.61 mb) — Conference Room from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

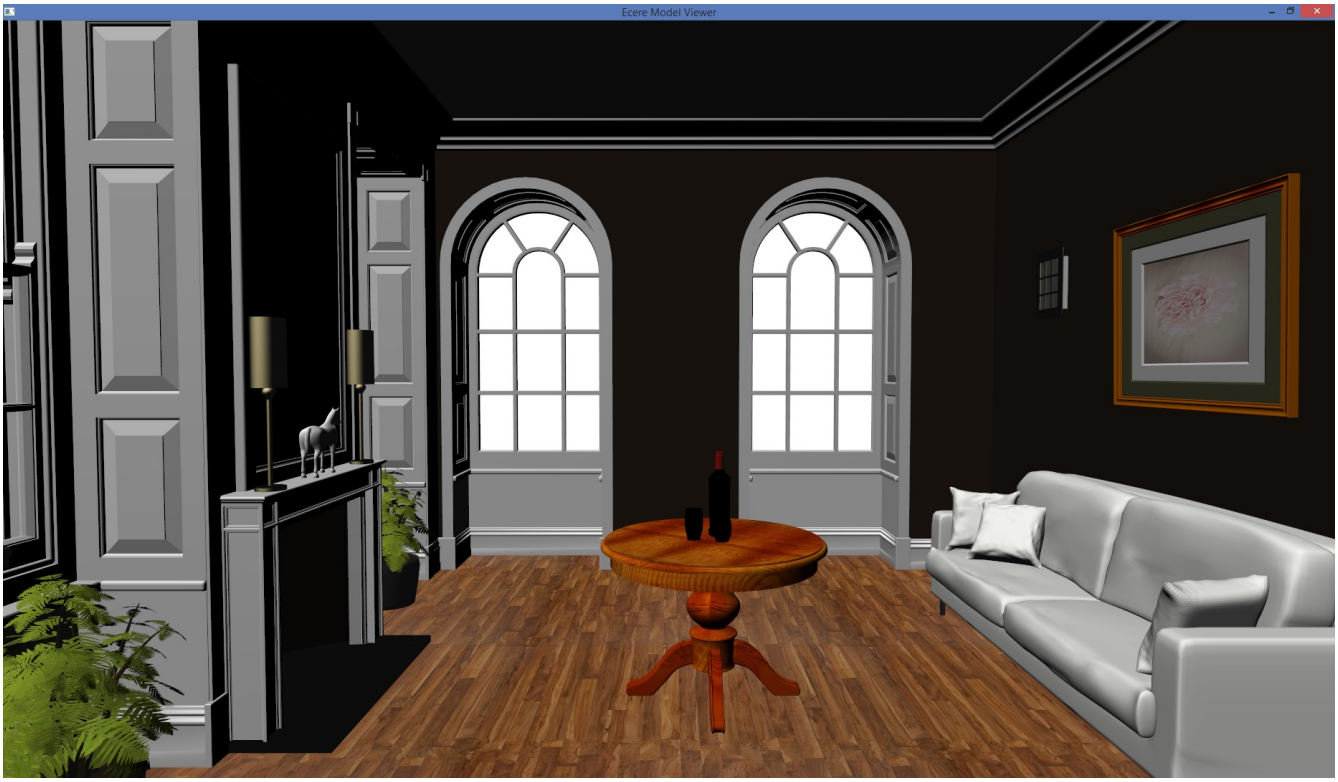

Figure 61. fireplace.e3d (2.44 mb) — Conference Room from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

Figure 62. bedroom.e3d (21.9 mb) — Bedroom from Morgan McGuire's Computer Graphics Archive [http://casual-effects.com/data/]

Appendix B: GNOSIS Map Tiles

(with new ability to reference or embed 3D models, store point clouds)

Some of the experiments for this testbed initiative made use of GNOSIS Map Tiles to describe a variety of tiled geospatial data. This included:

- Terrain elevation
- Imagery layers (which can be draped on terrain)
- Tiled vector data for street maps
- 3D buildings referenced from tiled points data
- 3D buildings extruded from 2D footprints polygons layers extruded based on attributes rules (OpenStreetMap Simple 3D Buildings [https://wiki.openstreetmap.org/wiki/Simple_3D_buildings])

An earlier version of these specifications was originally described in Testbed 13 Vector Tiles ER. [13] They can now also notably describe point clouds and tiles embedding a single 3D model.

NOTE	The latest version of these specifications is maintained at http://ecere.com/gmt.pdf

These specifications allow for a binary representation of tiled geospatial data of different geospatial data types:

- Tiled vector data
- Embedded and referenced 3D models
- Point clouds
- Imagery
- Coverages

Agnostic of CRS and Tiling Scheme

- Although only WGS84 and the GNOSIS Global Grid [http://docs.opengeospatial.org/per/17-041.pdf#page=118] has been used so far in conjunction with these specifications (and some of the terminology herein may assume this), nothing prevents the use of other CRS or tiling schemes.

NOTE	
	• Offsets and sizes are specified in decimal bytes. • Despite MSB being network byte ordering, values are encoded as little-endian (Least Significant Bit first) to avoid a very significant amount of byte swapping, accommodating today's most common architectures.

The tile data is prefixed by a 24 bytes header:

B.1. GNOSIS Map Tile Header

Table 9. GNOSIS Map Tile Header

Offset	Type	Size	Name
0	char	3	Signature: The 3 characters **GMT**
3	uint8	1	Major (Currently 1)
4	uint8	1	Minor (Currently 0)
5	uint8	1	Type (See Types table below)
6	uint16	2	Flags (See Flags table below)
8	uint64	8	Tile Key (tiling scheme specific) GNOSIS Global Grid layout (from high to low bits): level (5 bits: 0..28), latitude index (29 bits), longitude index (30 bits)
16	uint32	4	Size of uncompressed data excluding header
20	uint8	1	Encoding (See Encodings table below)
21	uint	3	Compressed size (note: 3 bytes only)
24	**Total size of header**		

If the tile is not flagged as empty or full, the actual tile data follows, based on geospatial data type.

B.1.1. Geospatial data types

The following ***Types*** are currently defined:

Table 10. Geospatial data types

Type	Value	Notes
Vector types		
vectorPoints	0x10	Points vector type
vector3DPoints	0x11	This implies a 16-bit Z value per point
vector3DPoints32	0x12	This implies a 32-bit Z value per point
vectorLines	0x14	Lines vector type

vector3DLine	0x15	This implies a 16-bit Z value per point
vector3DLines32	0x16	This implies a 32-bit Z value per point
vectorPolygons	0x18	Polygons are stored as a list of triangles (CDT)
vector3DPolygons	0x19	This implies a 16-bit Z value per point
vector3DPolygons32	0x1A	This implies a 32-bit Z value per point
vectorContours	0x1C	Polygons are stored as contours
vector3DContours	0x1D	This implies a 16-bit Z value per point
vector3DContours32	0x1E	This implies a 32-bit Z value per point
vectorTopoContours	0x20	Polygons are stored as contours with shared segments
vector3DTopoContours	0x21	This implies a 16-bit Z value per point
vector3DTopoContours32	0x22	This implies a 32-bit Z value per point
Imagery types		
rasterARGB	0x30	Alpha, Red, Green, Blue (the alpha in high order bit). 4 bytes per pixel (262,144 bytes per 256x256 tile).
raster16Bit	0x31	signed 16-bit integer (131,072 bytes per 256x256 tile)
raster8Bit	0x32	1 unsigned byte per pixel (65,536 per 256x256 tile)
Gridded coverage types		
coverage8Bit	0x50	unsigned, 1 byte per pixel (67,081 total)
coverage16Bit	0x51	signed, 2 bytes per pixel (134,162 total)
coverageInt32	0x52	signed, 4 bytes per pixel (268,324 total)
coverageFloat32	0x53	floating-point, 4 bytes per pixel (536,648 total)
coverageDouble64	0x54	floating-point, 8 bytes per pixel (1,073,296 total)

coverageQuantized16	0x70	- 2x 64-bit double to specify range (min, max) — Paeth filter, image encoding e.g. PNG does not apply to range - signed 16-bit integer per pixel (134,178 bytes per tile) representing grid values quantized to the min-max range. 0 represents (min+max)/2.
3D environment types		
The geometry of pointCloud and models3D(Ground) is still technically vector/points:		
pointCloud	0x90	For e.g. LAS files whereas ESRI Shapefiles pointZ/S57 Sounding Points will be vector3DPoints
models3D(Ground) reference or embed 3D models (e.g. E3D, glTF, COLLADA, FLT, 3DS, OBJ…)		
models3D	0xA0	This references external 3D models — 16-bit altitude per point, 32-bit LevelModelID in point data Models can be geotypical or geospecific.
models3DGround	0xA1	This references external 3D models — dropped to ground (no Z), 32-bit LevelModelID in point data Models can be geotypical or geospecific.
embedded3DModel	0xB0	This embeds a single 3D model and is purely 3D geometry (treated similarly to vector3DPolygons) Model is geospecific.

B.1.2. Flags

The following ***Flags*** are currently defined:

Table 11. Flags

Flag	**Position** *(from lsb)*	**Description**
All types		
full	0	The tile is full (finer zoom levels within the pyramid need not be defined).
empty	1	The tile is empty (finer zoom levels within the pyramid need not be defined).

Flag	Position *(from lsb)*	Description
Points, Referenced 3D Models & Point Clouds		
pointsDataId	2	Separate ID instead of elements list (more efficient when most points are individual). Duplicates are allowed (quantization can bring to same point).
rgb	3	Color information
alpha	4	Opacity information
All vector types, Point Clouds (for intensity)		
measure	5	Measurement information (e.g. PointM)
measure32	6	Measurement information (32 bit)
3D Polygons and Embedded 3D Model		
texCoords	8	Texture coordinates
normals	9	Normals information
tangents	10	Tangents information
Referenced 3D Models		
yaw	8	Orientation information
yawPitchRoll	9	Separate yaw, pitch, roll orientation information
scale	10	Scaling information
xyzScale	11	Separate x, y, z scaling information
Point clouds *(note: id is used for classification)*		
scanInfo	8	Scan information

The following ***Encodings*** are currently defined:

Table 12. Encodings

Encoding	Value	Description
uncompressed	0x00	The raw data without any special encoding.
deflate	0x01	The data is compressed with deflate (zlib) algorithm.
lzma	0x02	The data is compressed with LZMA.
jpeg2000	0x80	The data is compressed as a JPEG-2000 image.

Encoding	Value	Description
png	0x81	The data is compressed as a PNG image.
paethLZMA	0x82	A Paeth filter* is applied; then the data is compresed with LZMA.

B.2. Compact tiled vector data representation

For more details of how the compact tiled vector data works, see also its ECON textual representation in the OGC Testbed-13: Vector Tiles Engineering Report [http://docs.opengeospatial.org/per/17-041.pdf] - Appendix C [13].

B.2.1. Compact storage as localized vertices with accuracy proportional to scale

- Coordinates are specified as two 16-bit signed integer per vertex, the first integer representing the latitude, and the second the longitude — like ISO 6709:1983 [https://en.wikipedia.org/wiki/ISO_6709]. The full range (-32,767..32,767) of these integers are linearly mapped to the geospatial extent of the tile.
- Preserving proper topology with varying accuracy was a major challenge which has been solved in the GNOSIS vector pipeline.
- All points used by the tile are specified in one single array.

B.2.2. Pre-triangulated for high performance GPU rendering and optimal service-to-display processing

- Polygons are described as triangles since tesselation is a required step for hardware accelerated rendering of polygons which are either concave or feature inner holes. The tesselation process can add to the initial loading/processing time before incoming geometry can be visualized on the screen, and therefore this delay is minimized.
- Constrained Delaunay Triangulation [https://en.wikipedia.org/wiki/Constrained_Delaunay_triangulation] is performed to produce an optimal tesselation which maximizes the fill rate.

B.2.3. Enforced topologically correct representation (shared vertex indices)

- Lines and polygons provide a list of 16-bit indices into the array of vertices to be re-used by multiple elements sharing the same edges, or by multiple pieces of the same element connecting. This ensures proper topology as common edges and the spatial relationship between different elements are preserved, and makes the representation suitable for both high performance visualization as well as analysis.
- For lines, the indices of one single element make up a single line string
- For polygons, the indices of one single element make up a list of triangles (3 indices per triangle).
- The polygon indices making up triangles are always specified in a **counter-clockwise** manner.

B.2.4. Unique feature identifier

- Elements listing indices making up a given feature are uniquely identified by a 64-bit ID.
- Elements are specified by the ID, the start index (in the list of indices for lines and polygons; in the list of points for points) and the count of indices/points used.

B.2.5. Center lines for curved area labels

- Since computing the center lines of curved polygons is better done in regard to the overall shapes before tiling occurs, this information can optionally be included together with polygon geometry.
- This is useful for example to render labels following the curve of those areas, such as typically seen on lakes and large rivers.

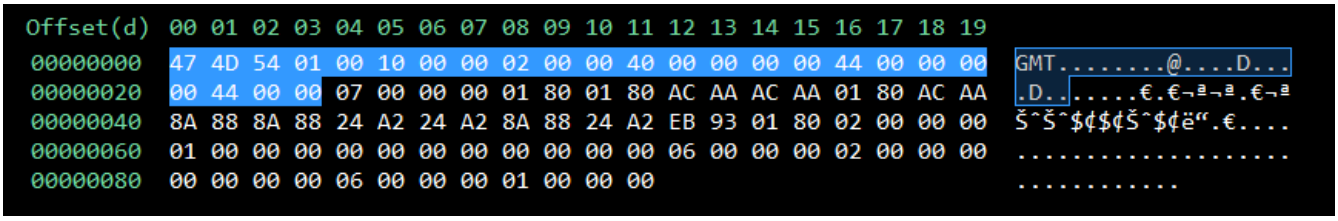

Figure 63. Hexadecimal view of a sample GMT encoded vector tile with points geometry

B.3. Binary layout for Points, Point Clouds and Positioned & Oriented 3D Models tiles

Table 13. Binary layout for Points, Point Clouds and Positioned & Oriented 3D Models tiles

Offset	Type	Size	Name	Description
0	int	4	numPoints	The number of vertices in the tile
Vertices *(numPoints occurrences)*				
4+n*12	int64	8	id	*(If pointsDataId flag is set)* ID identifying the feature the following point is part of (in the data store's geometry table).
pointsDataId flag set: 4+n*12 *pointsDataId flag not set:* 4+n*4	int16	2	latitude	Latitude mapped from the tile's latitude extent to -32,767 to 32,767, with the bottom (south) edge being at -32,767

Offset	Type	Size	Name	Description
6+n*4	int16	2	longitude	Longitude mapped from the tile's longitude extent to -32,767 to 32,767, with the left (west) edge being at -32,767
(end of vertices data)				
*If points dataId flag is not set)**				
4+numPoints*4	int	4	numElements	The number of elements in the tile
Elements *(numElements occurrences)*				
8+numPoints*4 +n*16	int64	8	id	ID identifying the feature the points within this element are part of (in the data store's geometry table).
16+numPoints*4 +n*16	int	4	start	Index to the first vertices for this element
20+numPoints*4 +n*16	int	4	count	Number of consecutive vertices making up element
If altitude data is available				
	double	8	loAlt	Minimum altitude (relative to geoid).
	double	8	hiAlt	Maximum altitude (relative to geoid).
	int16 / int32	(2 or 4) * numPoints	altitudes	min..max as -32767..32767 *(32 bit if vector3DPoints32)*
If measurement data is available				
	double	8	loMeasure	Minimum measurement.
	double	8	hiMeasure	Maximum measurement.

Offset	Type	Size	Name	Description
	int16 / int32	(2 or 4) * numPoints	measures	min..max as -32767..32767 *(32 bit if measure32 flag set)*
If color and/or alpha information is available (repeats numPoints times)				
	byte	1	alpha	Opacity value *(if alpha flag is set)*
	byte	3	r, g, b	Red, Green and Blue. *(if color flag is set)*
If model flag is set, this points layer references 3D models				
	uint32	4 * numPoints	modelIDs	High 5 bits: model level Low 27 bits: model The model level is the coarsest zoom level at which this model is used: it can be the same as the one for a lower level tile if this finer tile does not refine the model, or does so with additions.
If model flag is set and orientation information is available (repeats numPoints times)				
	uint16	2	yaw	-32767 (-360°) .. 32767 (360°) (North -→ East rotation of a compass, clockwise when looking down towards the surface of the globe, counter-clockwise when looking up towards the sky (positive y axis))

Offset	Type	Size	Name	Description
	uint16	2	pitch	*(if separate ypr flags is set)* Counter-clockwise when looking towards the East (positive x axis) Rotation angles are applied in the yaw, pitch, roll order.
	uint16	2	roll	*(if separate ypr flags is set)* Counter-clockwise when looking towards the North (positive z axis) Rotation angles are applied in the yaw, pitch, roll order.
If model flag is set and scaling information is available (repeats numPoints times)				
	uint16	2	sx or scale	*(mul'ed by 256, e.g. 512=2x)*
	uint16	2	sy	*(if xyz scaling flag is set)*
	uint16	2	sz	*(if xyz scaling flag is set)*
If model flag is set				
	double	8 * 6	extent	Overall extent of referenced models (not including models from other tiles spilling into this one) loLat, loLon, loAlt, hiLat, hiLon, hiAlt
	int	4	numSpillTiles	The number of tiles whose models extend over this tile
For each tile (of same level) whose models spill onto this tile: *(they will only need to be considered if not already selected for display)*				
	uint64	8	spillTileKey	Identifier of tile spilling onto this one.

Offset	Type	Size	Name	Description
	*	*	spillData	All information for those points whose models spill onto this tile (i.e. all fields fro 'numPoints' to 'extent' inclusively).
If point cloud flag is set and scan information is available				
	uint16	2 * numPoints	scanInfo	*(from lsb to msb)* returnNumber (3 bits) numberOfReturns (3 bits) scanDirection (1 bit) edgeOfFlight (1 bit) angle (8 bit)

B.3.1. 3D Models

The following rules control how embedded or referenced 3D models are positioned relative to the Earth, when the WGS84 CRS is in used.

- The units are assumed to be in meters.
- The **up** axis of the model aligns with going upwards from the ground.
- The **right** axis of the model aligns with the East.
- The axis going away into the distance aligns with the North.
- The East and North axis are understood to be tangent to the Earth (not following the curvature) at the local origin of the mesh (0, 0, 0), where they are instantiated (the tile origin in the case of embedded models, or the specified geospatial point if referenced)
- Unless animation is needed, embedded or referenced 3D models should consist of a single mesh and not apply any transformations. Most importantly, the origin of the mesh data should be relative to its geospatial position, not absolute in relation to the world.

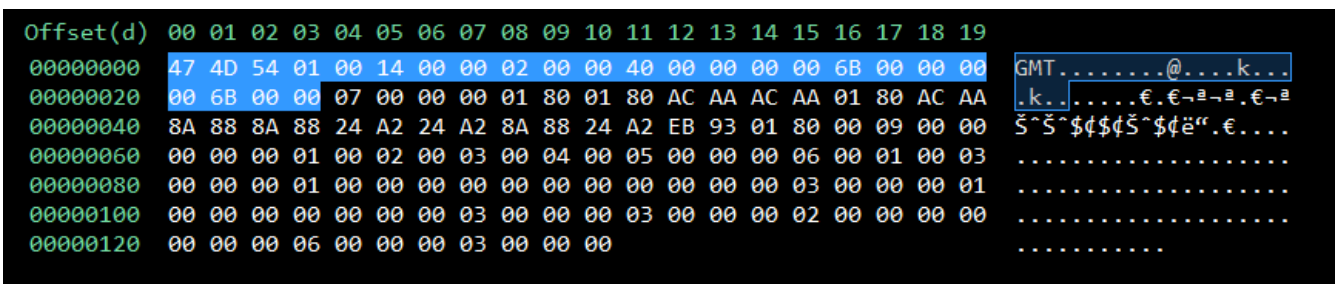

Figure 64. Hexadecimal view of a sample GMT encoded vector tile with lines geometry

B.4. Binary layout for Lines tiles

Table 14. Binary layout for Lines tiles

Offset	Type	Size	Name	Description
0	int	4	numPoints	The number of vertices in the tile
Vertices *(numPoints occurrences)*				
4+n*4	int16	2	latitude	Latitude mapped from the tile's latitude extent to -32,767 to 32,767, with the bottom (south) edge being at -32,767
6+n*4	int16	2	longitude	Longitude mapped from the tile's longitude extent to -32,767 to 32,767, with the left (west) edge being at -32,767
(end of vertices data)				
4+numPoints*4	uint8	(numPoints+7)/8	flags	A compact bits array of flags (1 bit per vertex) indicating whether the vertex is artificial (i.e not present in source data). The least significant bit represents the first of the up to 8 vertices mapped to each byte of flags.
4+numPoints*4 +(numPoints+7)/8	int	4	numIndices	The number of indices in the tile
8+numPoints*4 +(numPoints+7)/8	uint16	numIndices * 2	indices	16-bit indices into the vertex table to be referenced by elements
8+numPoints*4 +(numPoints+7)/8 +numIndices*2	int	4	numElements	The number of elements in the tile
Elements *(numElements occurrences)* Each element defines a line string as a series of indices.				

Offset	Type	Size	Name	Description
12+numPoints*4 +(numPoints+7)/8 +numIndices*2 +n*16	int64	8	id	ID identifying the feature the lines within this element are part of (in the data store's geometry table).
20+numPoints*4 +(numPoints+7)/8 +numIndices*2 +n*16	int	4	start	Index to the first index making up the lines for this element
24+numPoints*4 +(numPoints+7)/8 +numIndices*2 +n*16	int	4	count	Number of consecutive indices making up the lines for this element
28+numPoints*4 +(numPoints+7)/8 +numIndices*2 +numElements*16	**Total Size**			

NOTE	This table does not yet describe the layout for altitude and measurement values.

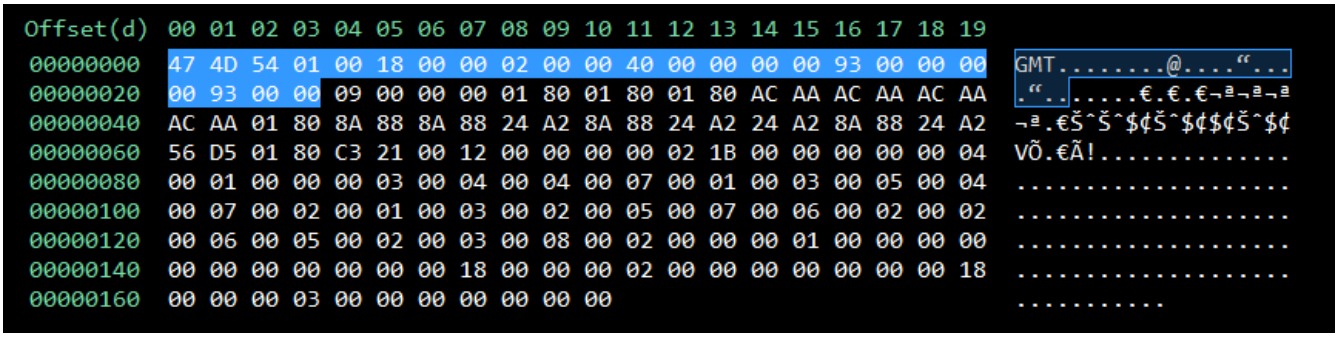

Figure 65. Hexadecimal view of a sample GMT encoded vector tile with polygons geometry

B.5. Binary layout for Polygons tiles

Table 15. Binary layout for Polygons tiles

Offset	Type	Size	Name	Description
0	int	4	numPoints	The number of vertices in the tile
Vertices *(numPoints occurrences)*				

Offset	Type	Size	Name	Description
4+n*4	int16	2	latitude	Latitude mapped from the tile's latitude extent to -32,767 to 32,767, with the bottom (south) edge being at -32,767
6+n*4	int16	2	longitude	Longitude mapped from the tile's longitude extent to -32,767 to 32,767, with the left (west) edge being at -32,767
(end of vertices data)				
Polygon Vertex Flags *(numPoints occurrences)* Each vertex has an associated flag indicating whether it lies on the tile boundary and whether edges stemming from it were in the source data (see section below explaining vertex flags).				
4+numPoints*4+n (& 0x01)	bit	single bit	onBottomEdge	Set if this vertex lies on the **bottom** tile boundary
4+numPoints*4+n (& 0x02)	bit	single bit	onLeftEdge	Set if this vertex lies on the **left** tile boundary
4+numPoints*4+n (& 0x04)	bit	single bit	onTopEdge	Set if this vertex lies on the **top** tile boundary
4+numPoints*4+n (& 0x08)	bit	single bit	onRightEdge	Set if this vertex lies on the **right** tile boundary
4+numPoints*4+n (& 0x10)	bit	single bit	downIn	Set if an edge from this vertex going **down** originates from source data
4+numPoints*4+n (& 0x20)	bit	single bit	leftIn	Set if an edge from this vertex going **left** originates from source data
4+numPoints*4+n (& 0x40)	bit	single bit	upIn	Set if an edge from this vertex going **up** originates from source data
4+numPoints*4+n (& 0x80)	bit	single bit	rightIn	Set if an edge from this vertex going **right** originates from source data

Offset	Type	Size	Name	Description
(end of vertex flags)				
4+numPoints*5	int	4	numIndices	The number of indices in the tile
8+numPoints*5	uint16	numIndices * 2	indices	16-bit indices into the vertex table to be referenced by elements
8+numPoints*5 +numIndices*2	int	4	numElements	The number of elements in the tile
Elements *(numElements occurrences)* Each element defines polygons as a series of triplets of indices, each defining counter-clockwise triangles.				
12+numPoints*5 +numIndices*2 +n*16	int64	8	id	ID identifying the feature the polygons within this element are part of (in the data store's geometry table).
20+numPoints*5 +numIndices*2 +n*16	int	4	start	Index to the first index making up the polygons for this element
24+numPoints*5 +numIndices*2 +n*16	int	4	count	Number of consecutive indices making up the polygons for this element
(end of elements data)				
28+numPoints*5 +numIndices*2 +numElements*16	int	4	numCenterLines	The number of center lines defined for the tile. 0 if center lines are not defined.
Center Lines *(numCenterLines occurrences)* Each center line defines a line string as a series of vertices and thickness for the polygon at each vertex. A center line of a single vertex is used if the centerline falls outside the tile, where the thickness can be thought of as a radius from that point. In this case the normalized values of the vertex will extend beyong -1.0..1.0.				

Offset	Type	Size	Name	Description
32+numPoints*5 +numIndices*2 +numElements*16 +n*16	int64	8	id	ID identifying the feature for which a center line is being defined (in the data store's geometry table).
40+numPoints*5 +numIndices*2 +numElements*16 +n*16	int	4	count	Number of vertices making up this centerline. 1 for a point + radius.
44+numPoints*5 +numIndices*2 +numElements*16 +n*16	float	4	length	Length of this segment of the center-line in units relative to the tile's latitude and longitude extent: sqrt ((proportion of tile's latitude delta)^2 + (proportion of tile's longitude delta)^2)) 0 for a point + radius describing a centerline outside this tile.
Vertices for all centerlines *(number of occurrences based on cummulative vertices count)*				
32+numPoints*5 +numIndices*2 +numElements*16 +numCenterlines*16 +n*12	float	4	latitude	Latitude mapped from tile's latitude extent to the -1.0 .. 1.0 range, with the bottom (south) edge being at -1.0. This value can be outside the -1.0..1.0 range when describing a centerline outside of the tile, in which case a single point is used.

Offset	Type	Size	Name	Description
36+numPoints*5 +numIndices*2 +numElements*16 +n*12	float	4	longitude	Longitude mapped from tile's longitude extent to the -1.0 .. 1.0 range, with the left (west) edge being at -1.0. This value can be outside the -1.0..1.0 range when describing a centerline outside of the tile, in which case a single point is used.
40+numPoints*5 +numIndices*2 +numElements*16 +n*12	float	4	radius	Distance to the edges of the polygon at this point of the centerline (half thickness or radius for a single point). Relative to the tile's latitude and longitude extent: sqrt ((proportion of tile's latitude delta)^2 + (proportion of tile's longitude delta)^2))
32+numPoints*5 +numIndices*2 +numElements*16 +numCenterlines*16 +totalCenterlineVertices*12	**Total Size**			

NOTE This table does not yet describe the layout for altitude and measurement values. Contours and Topological Contours representation remain to be defined as well.

B.5.1. Vertex flags for identifying tile boundaries and artificial edges

- In order to avoid rendering unwanted edges at the tile boundaries of polygons, flags are marked at each vertex.

- This feature is also used to avoid similar edges problems at the dateline with global datasets
- Each vertex actually has two sets of flags, represented in the Tiles API by the *PolygonVertexFlags* class.
- The first set of flags indicates whether a vertex is on any of a tile's 4 boundaries (top, left, bottom, right). These flags are also useful for recombining tiles, by identifying vertices at a tile's border. If an edge links two vertices flagged as being on the same edge, it is deemed to be an artificial edge, unless explicitly marked as being an actual edge by the second set of flags.
- The other *direction flags* as they are named in the Tiles API indicate whether there is actually a real edge (i.e. a segment of a polygon contour not introduced by tiling or by wrapping around the dateline) leaving from the flagged vertex going into each 4 directions (up, left, down, right). These flags should only set or inspected in relation to the corresponding set of edge flags:
  - For *on the right edge* and *on the left edge* flags, the up and/or down *edge is not artificial* flags can be set.
  - For *on the top edge* and *on the bottom edge* flags, the left and/or right *edge is not artificial* flags can be set.
- The *PolygonVertexFlags* provides a simple draw() method to determine whether an edge from one point to another should be drawn or not. The ordering of the vertices matter: the method should be called with a point counter-clockwise to the object on which it is invoked. This is because the flags mark whether an actual edge from the source data passed through each vertex coming from a certain direction.
- The *PolygonVertexFlags* class is implemented as such:

```
public class PolygonVertexFlags : byte
{
public:
   EdgeFlags onEdge:4;
   DirFlags d:4;
   bool draw(PolygonVertexFlags b)
   {
      bool drawEdge = true;
      EdgeFlags cf = onEdge & b.onEdge;
      if(cf && (
         (cf.right  && (!d.upIn    && !b.d.downIn )) ||
         (cf.top    && (!d.leftIn  && !b.d.rightIn)) ||
         (cf.left   && (!d.downIn  && !b.d.upIn   )) ||
         (cf.bottom && (!d.rightIn && !b.d.leftIn ))))
         drawEdge = false;
      return drawEdge;
   }
};
```

- For line features, a single flag is used, set to *true* if the vertex was **not** in the original data. In the binary representation, a single bit is used per vertex.

NOTE	When encoded using an image compression format such as PNG or JPEG-2000, which already defines a way to encode the image and dimensions, only the geospatial mapping and geometry of the image is relevant from what is described below.

B.6. Embedded 3D Models

The *embedded3DModel* (0xB0) data type can be used to describe geospecific features, such as 3D terrain as pre-triangulated alternative to compressed coverage heightmaps, or other models covering a large area (e.g. infrastructure) to be split in tiles.

- Texture coordinates can be optionally automatically computed from tile (e.g. imagery).
- The payload of the GNOSIS Map Tile is directly the mesh definition.
- The local origin (0, 0, 0) of the model is relative to the center of tile.

B.7. Binary layout for Imagery tiles

Table 16. Binary layout for Imagery tiles

Offset	Type	Size	Name	Description
0	uint16	2	width	Width of the data (typically 256)
2	uint16	2	height	Height of the data (typically 256)
4	*(based on format)*	width * height * sizeof(type) (typically 262,144)	data	The first pixel has its upper-left corner at the upper-left (north-west) corner of the tile, and the next pixels fill a scanline to the East. The next scanline is south of the first one, and so on. Each pixel represents a color for the entire pixel sampled from the center or average, with the 256 x 256 squares to be entirely within the tiles

B.8. Binary layout for gridded Coverage tiles

Table 17. Binary layout for gridded Coverage tiles

Offset	Type	Size	Name	Description
0	uint16	2	width	Width of the data (typically 259)
2	uint16	2	height	Height of the data (typically 259)

4	*(based on format)*	width * height * sizeof(type) (typically 67,081)	data	The first value reflects a sample 1/256th of the tile's latitude difference (height) and longitude difference (width) away towards the north-west direction from the upper-left (north-west) corner. The next values fill a scanline to the East, going 1/256th past the tile to the East, for a total of 259 samples across. The next scanline is south of the first one, and so on for a total of 259 scanlines, with the last scanline 1/256th of the tile's latitude difference south of the bottom (south) edge. The value are expected to be sampled at exact location (e.g. at the corners of the imagery 'pixels'). The values in different cells for the same geospatial location (e.g. on the tile boundary, as well as for the 1 value buffer around each tile) should match exactly, and facilitate dealing with partial data during visualization or analysis (e.g. to

For coverages, NODATA values are encoded as -32,767.

B.9. Paeth filtering

Imagery and coverages can be filtered using a Paeth filter before being compressed with LZMA. For 16 bit rasters such as the coverageQuantized16 format used for encoding terrain elevation data, this achieves significantly better compression than the Paeth filter within the PNG format because it treats the 16 bit integers as a whole rather than as individual bytes. It also seems to compress ARGB rasters better than PNG.

The following is an eC reference implementation for the Paeth filter encoding:

```
// a: left, b: above, c: upper left
static inline int paethPredictor(int a, int b, int c)
{
   int p = a + b - c;
   int pa = Abs(p - a);
   int pb = Abs(p - b);
   int pc = Abs(p - c);

   return pa <= pb && pa <= pc ? a : pb <= pc ? b : c;
}

static inline uint16 intToUint16(int x)
{
   x = (short)(uint16)(((uint) x) & 65535);
   return x < 0 ? ((-x-1)*2 + 1) & 65535 : (x * 2) & 65535;
}

static inline int uint16ToInt(uint16 x)
{
   return (x & 1) ? -((int)x-1)/2-1 : (int)x/2;
}

static inline byte intToByte(int x)
{
   return (byte)(((uint) x) & 255);
}

static inline int byteToInt(uint16 x)
{
   return (byte)(((uint) x) & 255);
}

static void encodePaeth(void * src, uint16 width, uint16 height, Format format)
{
   int x, y;

   if(format == raster16)
   {
```

```
      short * in = src;
      uint16 * temp = new uint16[width * height];
      uint16 * out = temp;

      for(y = 0; y < height; y++)
      {
         short a = 0, c = 0;

         for(x = 0; x < width; x++, out++, in++)
         {
            short d = *in, b = (y > 0) ? in[-width] : 0;
            int r = (int)d - paethPredictor(a, b, c);

            *out = intToUint16(r);
            a = d, c = b;
         }
      }
      memcpy(src, temp, width*height*2);
      delete temp;
   }
   else if(format == rasterARGB)
   {
      char * in = src;
      byte * temp = new byte[width * height * 4];
      byte * out = temp;

      for(y = 0; y < height; y++)
      {
         byte a = 0, c = 0;
         for(x = 0; x < width*4; x++, out++, in++)
         {
            byte d = *in, b = (y > 0) ? in[-width*4] : 0;
            int r = (int)d - paethPredictor(a, b, c);

            *out = intToByte(r);
            // NOTE: These are the values for the next iteration
            if(x >= 3) { a = in[-3]; if(y > 0) c = in[-width*4-3]; }
         }
      }
      memcpy(src, temp, (int)width*height*4);
      delete temp;
   }
}

static void decodePaeth(void * src,  uint16 width, uint16 height, Format format)
{
   int x, y;

   if(format == raster16)
   {
      uint16 * in = src;
```

```
      short * temp = new short[width * height];
      short * out = temp;

      for(y = 0; y < height; y++)
      {
         short a = 0, c = 0;

         for(x = 0; x < width; x++, out++, in++)
         {
            short b = (y > 0) ? out[-width] : 0;
            int r = (int)uint16ToInt(*in) + paethPredictor(a, b, c);
            short d = (short)r;

            *out = d;
            a = d, c = b;
         }
      }
      memcpy(src, temp, width*height*2);
      delete temp;
   }
   else if(format == rasterARGB)
   {
      byte * in = src;
      byte * temp = new byte[width * height * 4];
      byte * out = temp;

      for(y = 0; y < height; y++)
      {
         byte a = 0, c = 0;
         for(x = 0; x < width*4; x++, out++, in++)
         {
            byte b = (y > 0) ? out[-width*4] : 0;
            int r = (int)byteToInt(*in) + paethPredictor(a, b, c);
            byte d = (byte)(char)r;

            *out = d;
            // NOTE: These are the values for the next iteration
            if(x >= 3) { a = out[-3]; if(y > 0) c = out[-width*4-3]; }
         }
      }
      memcpy(src, temp, (int)width*height*4);
      delete temp;
   }
}
```

Appendix C: Geospatial features styling with support for 3D and AR

C.1. Conceptual Styling Model

This annex presents the conceptual model used for portraying Augmented Reality content in this initiative, which is based on a classic approach to style GIS features, but also applies to portraying geospatial data in 3D views. The usage scenarios covered during the testbed focused on geospatial Augmented Reality rather than computer vision Augmented Reality, and thus did not require any extensions beyond those capabilities. It was suggested that ARML could inspire future extensions specific to AR to integrate to this portrayal conceptual model, covering other scenarios with requirements specific to AR.

A similar conceptual model and the the *GNOSIS Cartographic Map Style Sheets (CMSS)* description of styles was also used during Testbed-14 portrayal work package (See 18-029 *OGC Testbed-14: Symbology Engineering Report*), as well as for the Vector Tiles Pilot. Objectives of the GNOSIS CMSS encoding (which was improved upon during Testbed-13 and Testbed-14) include maximizing expressiveness. A key concept in Testbed-14 work package is that styles described by the model could be encoded in different ways (GNOSIS CMSS being just one example), and the conceptual model exists to facilitate interoperability between multiple encodings, applications and rendering engines.

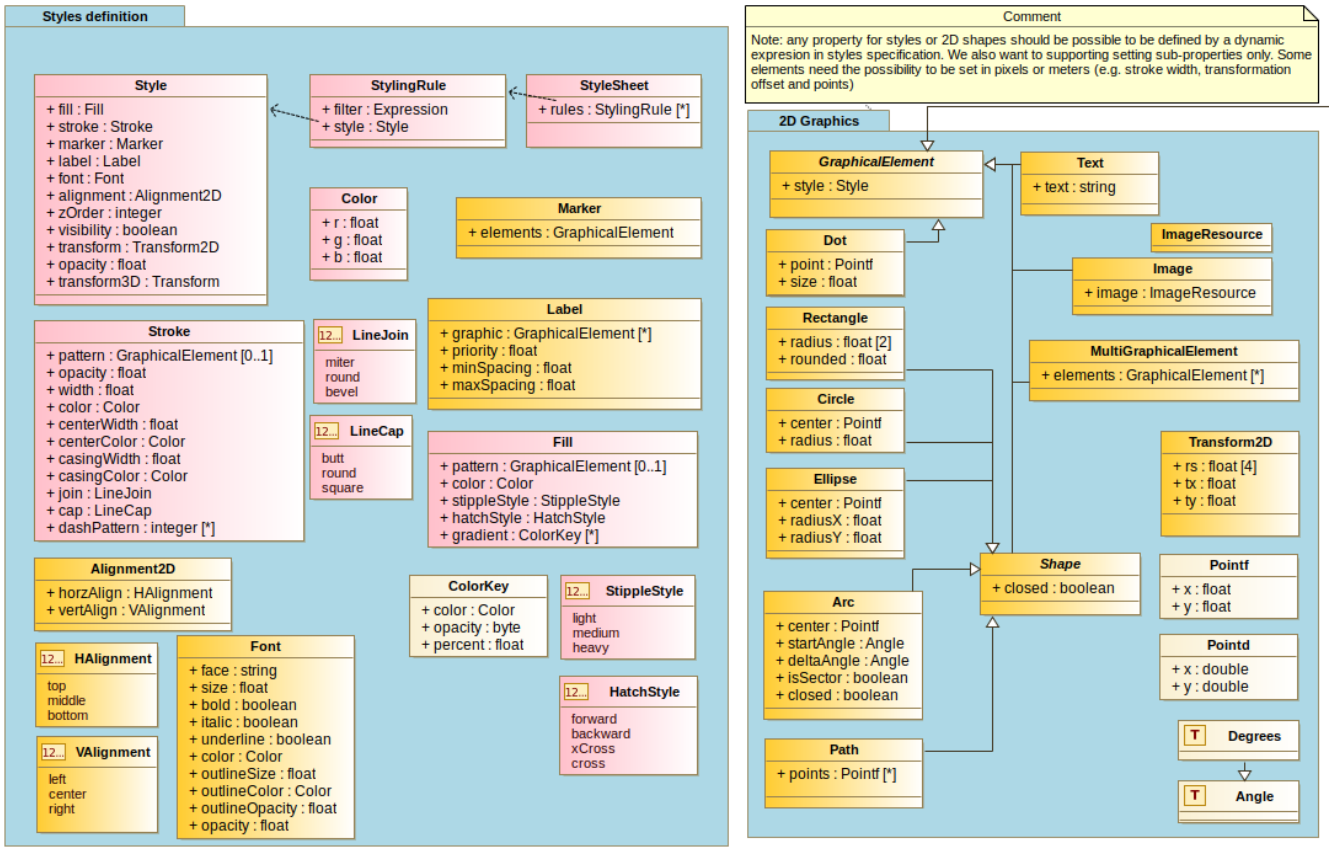

Figure 66. UML diagram of Styling Model

C.1.1. Portrayal Rules

Fundamentally, the conceptual model is a list of portrayal (styling) rules.

This list of rules can be made up of concatenated 'sheets' (each made up of a number of rules) stacked in a specific order, with the sheets being applied last (on top of the stack) ending up at the end of the list. This enables users to further customize the default styles from maps provided to them with default styles (e.g. one default style sheet in a GeoPackage or from a WFS GetStyles request, and a style sheet in the application).

The order of the rules in that list determines the order in which the styles are being applied, so that if a same style (symbolizer) is applied again from a rule later in the list, the style is overridden by the new value.

NOTE	Unlike e.g. SLD/SE, this order of rules is completely unrelated to the visual priority or z-order, and repeating a rule with different styles do not result in different things being rendered a second time

A rule is made up of a series of styles to be applied (see Styles), as well as an optional expression defining a set of selectors (see expressions) which determine (by resolving to a boolean value) whether the associated rule should be applied (if it does not have any selectors, or if the filtering expression resolves to a TRUE value) or not (if the filter resolves to FALSE) to a given data layer and data record (if applicable).

In addition to the possibility of listing rules in a flat layout, the model supports nested rules. By nesting a rule, these sub-rules are only considered if the selectors for the parent rule has been determined to match. These nested rules specify additional selectors, which if they also resolve to TRUE will trigger additional styles to be applied. This tremendously facilitates organizing cartographic style sheets and allows making them considerably more expressive. Furthermore, it facilitates implementing an optimal rendering engine by completely ignoring these sub-rules if the parent rules are ignored.

Conceptually, nested rules could be represented in a flat layout by repeating all selectors of the parent rules for each dependent rules. To translate a flat layout back into a nested layout however will result in a different organization based on the assumptions made. For example, one could produce a nested layout based on the order in which the multiple selectors for a given rule are listed; or one could prioritize certain types of selectors (e.g. layer identifier first, then scale, then record values…). This may result in a nested layout less representative of the intent of the cartographer; or less optimal for evaluation at run-time unless a preliminary analysis is done by the engine. This is the rationale behind including rules nesting capability as part of this conceptual model, to be capable of preserving this intent.

C.1.2. Rendering styles (a.k.a. symbolizers)

A rendering style defines a visual attribute whose value can be modified.

Multiple styles can be assigned within a single rule.

A given style applies to a determined set of feature classes.

The value for a given style is of a specific expression type.

It should be possible to set only the style of a sub-element of an object type, e.g. line.color = red .

If the type of the value is a list, it should be possible to add to it, rather than only replace it entirely, e.g. label.elements += Text { Name }.

Furthermore, adding to a null list should result in a list containing only that added element.

C.1.3. Markers, Labels, and GraphicalElements

Markers are typically used together with point features, and have precise positions where some symbol (whether an image, or vector shapes) will be displayed.

Labels on the other hand have some flexibility and will be positioned by some label placement algorithms, following certain rules (e.g. priority of importance, minimum and maximum distance between them etc.). Different types of vector features (points, lines, polygons) will follow different placement methodologies, which will offer different configuration parameters.

In this styling model, both markers and labels can be associated with a list of GraphicalElements (visual assets) which can be text, an image, vector shapes or a hierarchy combination of any or all these. These elements make up the visual representation of the marker or label at its position. For 3D views, 2D elements would typically always face the user, as a billboard. It could also be possible to associate 3D elements such as 3D models or 3D shapes with these markers or labels. For all the properties of these graphical elements, notably for textual content, expressions can be used which can refer for example to data attributes associated with the feature being styled.

C.1.4. Expressions

Expressions are used both for selectors (filters determining whether a given rule applies) as well as for styles values.

A **selector** is an expression which shall resolve to either TRUE or FALSE, and determines whether a rule is to be applied or not.

CMSS Encoding examples

```
#Roads // Equivalent to [lyr.id='Roads']

[viz.sd>10M][FEATCODE=123]
```

When resolving to a boolean value an expression which is either:

- null (unset)
- a numeric 0 (whether integer or real)
- or an enumeration value assigned the integer value 0 (e.g. false)

  evaluates to false;

all other expressions evaluate to true.

Primary Expressions

Table 18. Primary Expressions

Kind of expression	Description	CMSS Encoding examples
identifier	An identifier of a certain kind resolving to a value ([l|L][l|L|d]*) (*see Identifiers*)	FEATCODE
text	A fixed UTF-8 character string	'Parking'
integer	An integral number (up to 64 bit)	10
real	A real number (IEEE floating-point, up to double precision)	3.14159
object	If some elements are omitted, they inherit default values (0 or null, but which can be interpreted in a special way based on the object class) Objects classes can support inheritance and the definition of an object can include the specification of a derived class for this object	{ hour = 16, minutes = 30 } Text { Name } Circle { radius = 5, color = red }
list	A list of expressions	[1, 2, 3]
operation	A number of operands combined by an operator (*see Operators*)	viz.sd > 10M
variable	If variables are supported, the encoding should define default value for them which should produce reasonable styling.	@colorScheme

Identifiers

Table 19. Identifiers

Kind of identifier	Description	CMSS Encoding examples
null	An identifier representing an unset value	[someValue!=null]
enumValue	A valid value for expected for an enumeration type	[true]
layer	An object describing the layer being styled	lyr

Kind of identifier	Description	CMSS Encoding examples
identifier	A unique identifier to reference the layer	[lyr.id = Roads] (equivalent: #Roads)
featureClass	The type of geospatial data	[lyr.fc = vector]
vectorType	The type of vector data	[lyr.vt = points]
geometry	The geometry of the entire layer (a VectorFeatureCollection)	lyr.geom
visualization	An object describing the current state of the visualization	viz
scaleDenominator	The denominator for a fraction representing the scale / resolution at which the data is being looked at	[viz.sd > 10M]
time	The current time of visualization (This contains the time elements of both date as well as timeOfDay)	[viz.time > { 2018, june, 30, 16, 30 }]
date	The current date of visualization (year, month, day)	[viz.date > { 2018, june, 30 }]
timeOfDay	The current time of visualization (hours, minutes, seconds)	[viz.timeOfDay > { 16, 29, 59 }]
record	An object describing the current record (vector data)	rec
id	The unique ID identifying the record	[rec.id = 1234]
geometry	The actual geometry of the record (a single VectorFeature)	centroid(rec.geom)
(data attributes)	Data attributes associated with the record	FEATCODE

Kind of identifier	Description	CMSS Encoding examples
records	A list of all records in the layer	[featcode = 'bay' & iterate(or, records, it.featcode = 'ocean' & intersect(it, rec.geom))] True if this is a bay intersecting with an ocean in another record

Operators

Table 20. Operators

Operator	Operands Count	Description	CMSS Encoding	SLD/SE
Logical				
and	2	Logical conjunction	&	And
or	2	Logical disjunction	|	Or
not	1	Logical negation	!	Not
Comparison				
equal	2	Equality	=	PropertyIsEqualTo
notEqual	2	Non-equality	!=	PropertyIsNotEqualTo
greater	2	Greater than	>	PropertyIsGreaterThan
lesser	2	Lesser than	<	PropertyIsSmallerThan
greaterEqual	2	Greater or equal to	>=	PropertyIsGreaterThanOrEqualTo
lesserEqual	2	Lesser or equal to	<=	PropertyIsLessThanOrEqualTo
Text-specific				
stringContains	2	Contains the sub-string	~	
stringStartsWith	2	Starts with the sub-string	^	
stringEndsWith	2	Ends with the sub-string	$	
stringNotContains	2	Does not contain the sub-string	!~	

Operator	Operands Count	Description	CMSS Encoding	SLD/SE
stringNotStartsWith	2	Starts with the sub-string	!^	
stringNotEndsWith	2	Does not end with the sub-string	!$	
Arithmetic				
add	2	For numeric values: addition For text values: concatenation	+	Add
sub	2	Subtraction	-	Sub
mul	2	Multiplication	*	Mul
div	2	Division	/	Div
intDiv	2	Integer division	idiv	
mod	2	Modulo (remainder)	%	Mod
Others				
parentheses	1	Operation priority	( )	
conditional	3	condition-then-else expression	viz.sd>10M?1:3	
in	2	True if the left operand is within the right operand list	TYPE in [1,2,3]	
functionCall	fn,param*	(*See Functions*)	length(r.geom)	

Functions

Examples of functions:

- Text Manipulation: e.g. format(), replace()
- Geometry Operation: area(), length(), centroid()
- Spatial Operations: intersects(), contains()
- Iteration: iterate() to iterate over a list to evaluate an expression (*it* representing the current iterator)

C.2. GNOSIS Cascading Map Style Sheet encoding example

An example of cascading portrayal rules used for styling OpenStreetMap roads, buildings and annotations (labels) in the AR experiments

```
#buildings
```

```
{
   visibility = false;
   zOrder = 8;
   [viz.sd <= 127000]
   {
      fill = { 0xe2d8d8 },
      stroke = { width = 0 },
      visibility = true
   }
   [viz.sd <= 16000]
   {
      stroke = { darkGray, width = 0.5 },
      visibility = true,
      label = { [
         Text { name, color = white, font = { 'Tahoma', size = 14, outline = { black,
size = 3 } } }
      ] }
   }
   // This rule enables 3D extrusion of buildings, with a default height of 1.75
meters,
   // but enabling the recognition of the OSM Simple Buildings attributes description
   // (e.g. building:height, building:levels, building:min_level, building:min_height)
   [viz.sd <= 4000]
   {
      visibility = true,
      fill = { 0xe2d8d8 },
      extrude = { height = 1.75, raiseWithTerrain = true, osmBuildings = true }
   }
}

#roads
{
   stroke = { lightGray, width = Meters { 3 }, casing = { black, width = 1.0 } };
   label = { [
      Text { name, color = white, font = { 'Tahoma', size = 14, outline = { black,
size = 3 } } }
   ] };

   [viz.sd > 100000] { visibility = false }
   [viz.sd <= 100000] { visibility = true }
   [highway in ('primary','primary_link')]
   {
      stroke.color = 0xffce5f, stroke.casing.color = 0xd18d1a;
      zOrder = 13;
   }
   [highway in ('secondary', 'secondary_link')]
   {
      stroke.color = 0xffee78, stroke.casing.color = 0xe9a432;
      zOrder = 12;
   }
   [highway in ('motorway', 'motorway_link')]
```

```
   {
      stroke.color = 0xf35d63, stroke.casing.color = 0xc5262d;
      zOrder = 14;
   }
   [highway in ('tertiary', 'tertiary_link')]
   {
      stroke.color = paleVioletRed, stroke.casing.color = 0xfcfbe7;
      zOrder = 11;
   }
   [highway in ('trunk', 'trunk_link')]
   {
      stroke.color = 0xe9b798, stroke.casing.color = 0xc6262d;
      zOrder = 14;
   }
   [highway in('residential', 'unclassified', 'road', 'living_street', 'service',
'track', 'construction')]
   {
      stroke.casing.color = 0xb1b0a8;
      zOrder = 10;
   }
   [highway in ('path', 'cycleway', 'footway', 'pedestrian', 'steps', 'bridleway',
'rest_area')]
   {
      stroke.color = 0xfffbf4;
      zOrder = 9;
   }
}
```

Appendix D: Revision History

Table 21. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
May 31, 2018	J. St-Louis	.1	all	initial version

Appendix E: Bibliography

1. Matney, L.: An AR glasses pioneer collapses, https://techcrunch.com/2019/01/10/an-ar-glasses-pioneer-collapses/, (2019).
2. GeoBIM extension for CityGML and IFC BIM, https://vimeo.com/12462613.
3. Wendel, J., Núñez, J.M.S., Simons, A.: Urban Energy Modelling: Semantic 3D City Data as Virtual and Augmented Reality. GIM International Magazine. (2017).
4. Collective, T.O.S.B.I.M.: Open Source BIM, https://github.com/opensourceBIM.
5. Kim, Y.-J., Kang, H.-Y., Lee, J.: Development of Indoor Spatial Data Model Using CityGML ADE. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XL-2/W2, (2013).
6. Blut, C., Blut, T., Blankenbach, J.: CityGML goes mobile: application of large 3D CityGML models on smartphones. International Journal of Digital Earth. 12, 25–42 (2019).
7. Zamyadi, A., Pouliot, J., Bédard, Y.: Enriching Augmented Reality Games with CityGML, http://yvanbedard.scg.ulaval.ca/wp-content/documents/publications/635.pdf.
8. Basics of 6DoF and 9DoF sensor fusion, http://www.embedded-computing.com/embedded-computing-design/basics-of-6dof-and-9dof-sensor-fusion.
9. OpenStreetMap: Simple 3D Buildings, https://wiki.openstreetmap.org/wiki/Simple_3D_buildings.
10. Coors, V.: OGC Testbed-13: 3D Tiles and I3S Interoperability and Performance Engineering Report. OGC 17-046,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-046.html (2018).
11. ESRI: ESRI Shapefile Technical Description, https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.
12. Beil, C., Kolbe, T.: CityGML and the streets of New York - A proposal for detailed street space modeling. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences. IV-4/W5, 9–16 (2017).
13. Cavazzi, S.: OGC Testbed-13: Vector Tiles Engineering Report. OGC 17-041,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-041.html (2018).