International Journal of Digital Earth

ISSN: 1753-8947 (Print) 1753-8955 (Online) Journal homepage: http://www.tandfonline.com/loi/tjde20

CityGML goes mobile: application of large 3D CityGML models on smartphones

Christoph Blut, Timothy Blut & Jörg Blankenbach

To cite this article: Christoph Blut, Timothy Blut & Jörg Blankenbach (2017): CityGML goes mobile: application of large 3D CityGML models on smartphones, International Journal of Digital Earth, DOI: 10.1080/17538947.2017.1404150

To link to this article: https://doi.org/10.1080/17538947.2017.1404150

Published online: 20 Nov 2017.

INTERNATIONAL JOURNAL OF DIGITAL EARTH, 2017
https://doi.org/10.1080/17538947.2017.1404150

CityGML goes mobile: application of large 3D CityGML models on smartphones

Christoph Blut, Timothy Blut and Jörg Blankenbach

Department of Computing in Civil Engineering and Geo Information Systems, Geodetic Institute, RWTH Aachen University, Aachen, Germany

ABSTRACT

CityGML, a semantic information model for digital/virtual city models has become quite popular in various scenarios. While the data format is still actively under development, it is already supported by different software solutions, especially GIS-based desktop applications. Mobile systems on the other hand are still neglected, even though the georeferenced objects of CityGML have many application fields, for example, in the currently popular area of location-based Augmented Reality. In this paper we present an independent multi-platform CityGML viewer, its architecture and specific implementation techniques that we use to realize and optimize the process of visualizing CityGML data for use in Augmented Reality. The main focus lies in improving the implementation on mobile devices, such as smartphones, and assessing its usability and performance in comparison to web-based approaches. Due to the constrained hardware resources of smartphones, it is a particular challenge to handle complex 3D objects and large virtual worlds as provided by CityGML, not only in terms of memory and storage space, but also with respect to mobile processing units and display sizes.

ARTICLE HISTORY
Received 17 May 2017
Accepted 8 November 2017

KEYWORDS
CityGML; city model; android; 3D visualization; real-time rendering

1. Introduction

Often virtual 3D city models are used for a general overview of large-scale environments or in areas such as urban planning, -management and -simulation. Some examples for analyses tasks are solar potential analyses, shadow analyses and disaster analyses. Biljecki et al. (2015) present a state-of-the art review for the use of 3D city models with around 30 use cases and more than 100 applications which underlines the increasing importance of 3D city models. While the majority of virtual 3D city models was purely graphical in the past and prevented more sophisticated applications, the growing popularity of semantic information models in recent years has created new opportunities. For example in the AEC industry Building Information Modeling (BIM) is becoming more and more important. With the Industry Foundation Classes (IFC) a semantic data model for the description and exchange of building data in the context of BIM already exists. One of the most influential models in geographic information science is City Geographic Markup Language (CityGML) which is an XML-encoding schema based on Geography Markup Language (GML) for storing and exchanging virtual 3D urban objects and their semantic- and topological information. It uses

CONTACT Christoph Blut christoph.blut@gia.rwth-aachen.de Department of Computing in Civil Engineering and Geo Information Systems, Geodetic Institute, RWTH Aachen University, Mies-van-der-Rohe-Str. 1, Aachen 52074, Germany

the ISO 19107 standard (Herring 2001) and a number of other ISO 191xx standards. Geometries are defined using Boundary Representation (B-Rep) which uses collections of observable surfaces (boundaries) to model a solid. CityGML's module-based structure and Level of Detail (LOD) system ranging from LOD0 to LOD4 allow for problem-specific modeling. While the mandatory core module contains basic definitions, the extension modules, like the building or appearance module, add specific thematic information. The LOD system allows modeling objects in different granularities (Kolbe 2008; Schall et al. 2011; Gröger et al. 2012).

Commonly, virtual 3D city models are employed and visualized on desktop computers. The success of mobile devices such as smartphones has introduced alternative ways of visualization such as virtual reality or visual augmented reality (AR). This allows for new use cases for city models. For example, AR combined with the semantic 3D city- and landscape models, as provided by CityGML, enables the georeferenced visualization of planned buildings on parcels, particularly interesting for real estate and planning offices. The virtual buildings can be freely inspected on-site from a point of view perspective and attributes like color, size or overall integration into the cityscape can be evaluated. Some realized examples for combining such geographic information with AR are approaches by Vlahakis et al. (2001), Ghadirian and Bishop (2008), Woodward and Hakkarainen (2011), Schall, Zollmann, and Reitmayr (2013) and Zamyadi, Pouliot, and Bédard (2013).

As part of our mobile AR research project to realize and analyze the mentioned use cases amongst others, it is necessary to visualize CityGML models. For this, multiple issues must be addressed, for instance, the increased complexity of the CityGML models in comparison to purely graphical models and in contrast the limited hardware of mobile devices. Also the screen size must be taken into account. With less display space than on desktop monitors it is essential to only display important information on the screen. Furthermore, different than in stationary desktop environments where a network connection is usually available at all time, mobile devices cannot rely on such an assumption due to the incomplete coverage of wireless high-speed networks. Thus, to process large CityGML models and display their 3D objects from a point of view perspective in an AR system, a specialized CityGML viewer which takes the named issues into account is necessary. We propose an offline, cross-platform CityGML visualization architecture for use in mobile AR which applies selective visualization methods that take advantage of CityGML's data structure to minimize the data load.

The research in this paper focuses on the methodology for handling the above-mentioned requirements. The solutions are investigated empirically to evaluate the final proposed offline approach as an alternative to web-based solutions and community standards such as 3D Tiles in terms of usability and performance. Since the number of use cases and availability of CityGML data are most promising for buildings, we focus on the building module of CityGML. An important goal of our application is to preserve all available information of the building models and make it accessible at runtime. This includes geometries, visual information like colors and additional semantic-, topological- and descriptive information.

For proof of concept we developed an Android™ app. The advantage is that the operating system (OS) uses the Java™ programming language which easily allows porting the application to other platforms. The structure of the paper is as follows: Followed by the introduction, background information on processing and visualizing virtual 3D city models is presented in Section 2. Section 3 discusses different approaches for using CityGML on mobile devices with a focus on our implementation aimed towards AR. In Section 4, the realized solution is analyzed and evaluated against alternative approaches, followed by the conclusion and future work in Section 5.

2. Background

While frameworks for the visualization of 3D models based on common graphics formats such as COLLADA, OBJ or X3D are widely available (e.g. Unity3D), direct support for CityGML is still rare, especially for mobile devices. Commercial software for CityGML visualization like ArcGIS™ (ESRI), Bentley Map™ (Bentley Systems) or the CityEditor (3DIS GmbH) and freeware/open source

software like the CityGML SpiderViewer (GEORES), the FZKViewer (KIT Karlsruhe) or 3D City Database (3DCityDB) are only available for desktop computers.

For visualizing CityGML on mobile devices current developments mainly focus on web-based solutions using client/server architectures and tiling systems that load chunks of data according to the current view. Gesquière and Manin (2012) and Gaillard et al. (2015) presented a WebGL™-based architecture that handles the data on a server and exports tiles with data in JSON files. Prieto, Izkara, and Delgado del Hoyo (2012), Giovannini et al. (2014) and Simões, Prandi, and De Amicis (2015) pursue similar concepts by using client/server approaches and a conversion to graphics formats such as X3D™, OBJ or KML for more efficient visualization. The increased focus on HTML5 (WebGL) solutions has introduced entire frameworks for 3D geospatial data visualizations such as Cesium and iTowns. They feature an open source JavaScript and WebGL-based virtual globe and map engine that can display terrain, image data and 3D models. Since Cesium and iTowns do not offer direct support for loading and visualizing CityGML, approaches have been described by Chaturvedi (2014) using KML or Schilling, Bolling, and Nagel (2016) using the GL Transmission Format (glTF) which are natively supported by Cesium. The initial release of glTF by the Khronos Group was in late 2015 with the goal of minimizing transmission and loading times of 3D scenes for WebGL applications. While using formats such as COLLADA, OBJ, X3D or glTF makes the rendering process using existing visualization frameworks particularly simple these pure graphics formats cannot directly store CityGML's semantic information which eliminates what makes CityGML distinctive. Therefore, a new specification named 3D Tiles based on glTF is being developed that promises the efficiency of glTF and the possibility to store additional information. Specifically the format Batched 3D Model (B3DM) aims at displaying large city models by using batching methods while preserving the per-object properties.

An advantage of a web-based solution in comparison to a native application is its platform independency that allows using an application on any device given a compatible web browser. This enables access to a larger user base and provides the same experience on every device. Also, by employing a client/server structure data and workloads can be distributed. For instance, data storage, management and queries can be handled by the server while the client device is used for rendering. However, the usability of such web-based solutions heavily depends on the availability of a network connection. While stable and fast network connections are commonly present in desktop environments, mobile connections are often unreliable, especially in rural areas due to poor network coverage. Furthermore, bandwidth is another critical issue to consider. CityGML models are often multiple gigabytes (GB) large which is problematic since mobile data plans are typically limited. In CityGML, especially the LOD has a strong impact on file sizes. While the same building modeled with LOD1 is only a few kilobytes (kB) small, its LOD4 representation can be multiple GB large. Another performance critical issue which is especially important for our research towards AR is low-level access to the smartphone's hardware, like the sensors. Specifically using HTML5 (WebGL) the functionality and performance of the web applications heavily depend on the type of web browser and its capabilities. Full HTML5 support generally still is uncommon, especially on mobile devices (HTML5TEST, 7 August 2017, https://html5test.com/index.html). For instance, highly specialized WebGL applications, such as Unity WebGL are not supported at all on mobile devices or only perform insufficiently (Unity3D, 7 August 2017, https://docs.unity3d.com/Manual/webgl-browsercompatibility.html). How well the application performs ultimately depends on the web browser's JavaScript engine and capabilities to allocate memory. As a consequence of the limitations of web-based solutions and the lack of existing native mobile solutions we developed a custom solution.

3. CityGML viewer

In general, to display any type of data on a screen a step-wise process is required. Dos Santos and Brodlie (2004) summarize this process in the form of a visualization pipeline that involves four intermediate steps to create image data from raw input data. In the first step data analysis is performed in which, for example, erroneous data are corrected. The second and third steps involve selecting data

of interest (filtering) and preparing it for rendering (mapping). Finally, image data is created from the geometric data and their attributes (e.g. color) in the last step (rendering). These steps are also applied for visualizing CityGML data. In the following we assume the data to be free of errors and focus on the remaining three steps. The filtering step is described as part of data processing, while mapping and rendering are described in the visualization section.

As Android has been one of the most dominant OS on the mobile market in recent years, it is favorable to base developments on it. Android additionally has the advantage towards iOS that it is fully Java-based and therefore allows easily running applications on any other major OS. Therefore, our application can be executed on Android (ranging from early to the most recent versions) and Windows directly, but also on iOS using cross-compilation. As depicted in Figure 1, our application architecture consists of two main packages, the processing and the visualization component. We utilize a custom CityGML pull parser, a local SpatiaLite database and specialized database export algorithms. For rendering we utilize the jMonkey game engine.

3.1. Processing

A straightforward approach for processing data for visualization is to load it directly from an exchange format (e.g. CityGML) into an in-memory model. The advantage is that no further infrastructure is necessary. Some low-level libraries such as citygml4j that can process CityGML are already available. However, careful attention must be paid to the compatibility of such libraries with the Android™ system, since Java™ code as such can be run without problems, but not all libraries of the standard desktop-based Java™ package are part of the system. For instance, citygml4j utilizes the Java Architecture for XML Binding (JAXB) interface, which is not compatible with Android™. Therefore, citygml4j could not be used, so instead we implemented a custom solution.

To evaluate an in-memory model approach we developed two Java-based CityGML parsers that are able to run on mobile devices, one using a Document Object Model (DOM) and another using a pull parser. Both parsers create per-feature Java objects with their corresponding attributes. They were compared in terms of time and memory usage to read the CityGML data from the file and

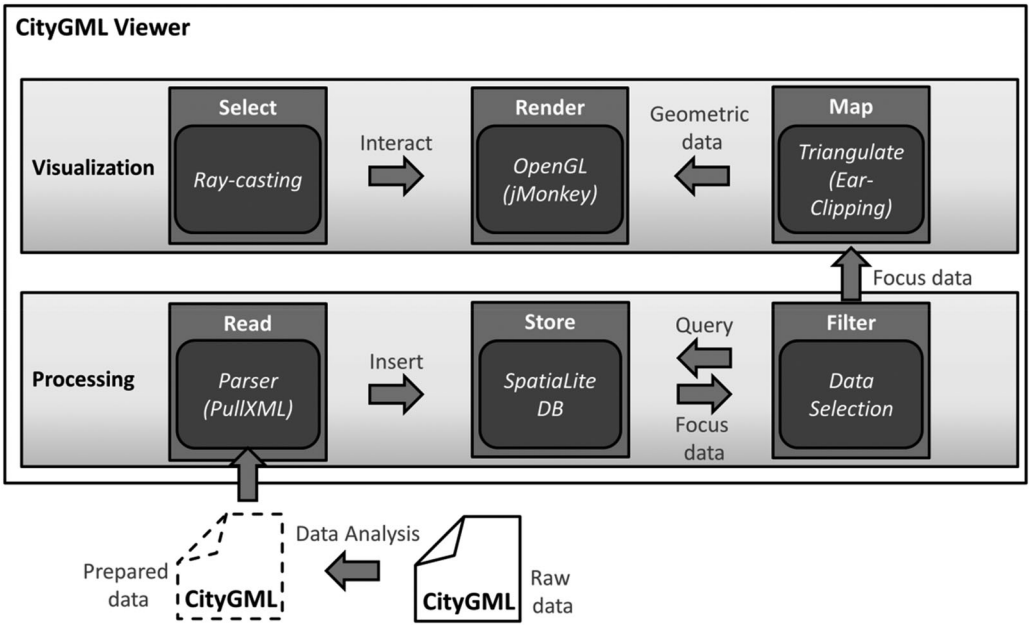

Figure 1. Architecture of the CityGML viewer.

transfer it into a Java rendering model. Multiple CityGML models were used to confirm that the results are reproducible.

Using the DOM, RAM consumption generally ranged between five and six times of the source file size, independently of the LOD (Figure 2). Thus, given a file size of around 500 MB already 3 GB of RAM in average were necessary. Consequently, for 1500 MB roughly 9 GB RAM are required. The pull parser on the other hand had a consistent RAM consumption of around 10 MB for running, regardless of the LOD or file size, plus the size of the actual in-memory model which was about the half to the same size of the file.

Furthermore, the measured loading times for both parsers are not applicable for real-time use in AR environments. The loading times can be seen in Figure 3. Generally, the pull parser was about two to three times faster than the DOM, but still too time-consuming for real-time applications.

The overall outcome was that CityGML is not suited for loading directly using only the limited hardware of smartphones without any pre-selection of objects. To solve this, some specialized (data) architecture is required. As a solution we employ an embedded database with specialized import/export capabilities which enable complex geometric and semantic analysis tasks using SQL. It consists of three parts, the CityGML pull parser to read CityGML files, a SpatiaLite database and the semantic-geometric data selection and export methods (Figure 1). In comparison to, for instance, B3DM the data are not restructured, but rather left in its original form which allows exploiting the semantic relationships.

3.1.1. Import and storage

Since a client/server solution is not feasible for a mobile CityGML viewer aimed at AR, server-based databases like Oracle Spatial or PostGIS cannot be utilized. Instead a fully embedded local database is required. While there is a solid amount available (e.g. Oracle Berkeley DB, Couchbase Lite) for Android devices, only a few have spatial capabilities.

Android contains the software library SQLite™ that implements a self-contained, serverless, zero-configuration, transactional SQL database engine (SQLite, 1 August 2017, https://www.sqlite.org). However, like the majority of embedded databases, it natively does not support spatial features. SpatiaLite™ solves this by extending the core functionalities of SQLite to facilitate vector geodatabase functionalities. Multiple types of geometries such as points, lines and polygons, but also more complex types such as MultiPolygons are supported. Most important for CityGML is that it is capable of storing 3D geometries. Unfortunately it only provides a few functions such as ST_3DDistance that

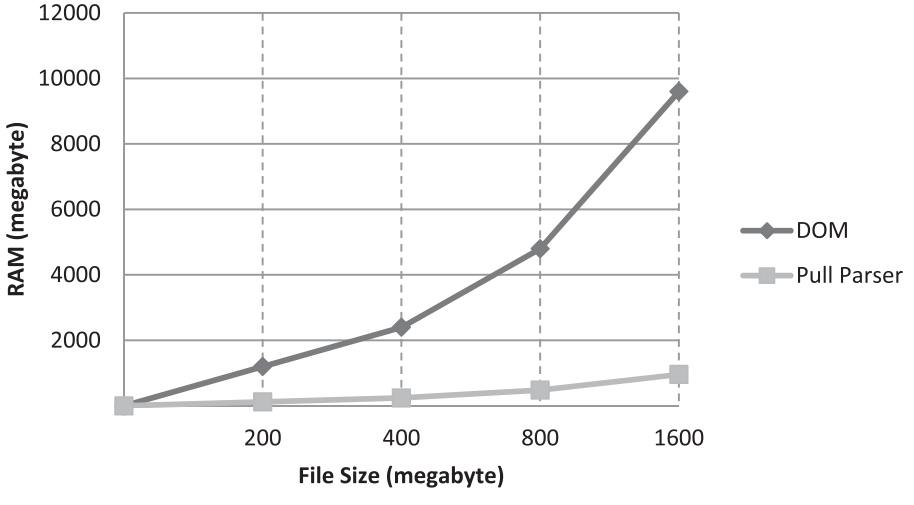

Figure 2. Comparison of the amount of RAM necessary for a CityGML parser using DOM and a pull parser.

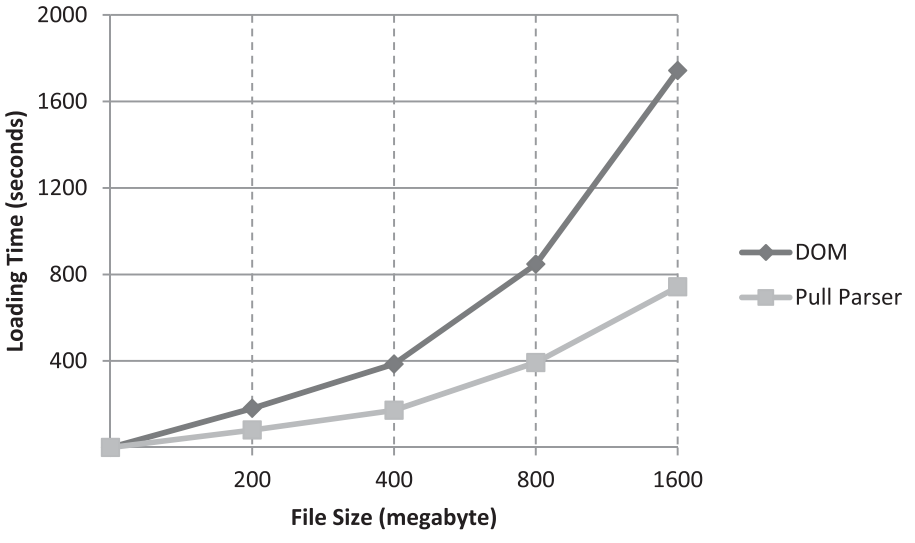

Figure 3. Loading times for the LOD2 and LOD4 model with a CityGML parser using DOM and a pull parser.

actually consider the *z*-value of the geometries during calculations. Oracle Spatial and PostGIS on the other side each have a more extensive list (e.g. PostGIS, 7 August 2017, http://postgis.net/docs/manual-dev/PostGIS_Special_Functions_Index.html#PostGIS_3D_Functions).

Nevertheless, SpatiaLite represents a strong solution for storing CityGML data. Just as Oracle Spatial and PostGIS, it is capable of utilizing spatial indices, which are crucial for fast querying of geometries in large database tables. The spatial index uses the SQLite R*Tree (Guttman 1984; Beckmann et al. 1990) algorithm which employs a tree-like approach in which geometries are represented by their minimum bounding rectangles. It allows attaching multiple databases to a single database connection, enabling them to operate as one. Using this feature, it has the following maximum capabilities (Table 1).

By being able to utilize multiple databases it is not only possible to store massive models, but also to handle models with different coordinate systems. For example, models with the same coordinate system can be hosted in one database while others are stored in another database.

In general, SpatiaLite's 2D/3D spatial queries perform well against PostGIS and Oracle Spatial. This was assessed by evaluating typical spatial functions such as topological operators relevant for querying CityGML data (Figure 4).

SpatiaLite's performance and functions make it a suitable option for working with CityGML, even with the limited amount of spatial functions that support true 3D. A solution to the missing support is to utilize the in-database 2D functions, pre-select data to receive only data of interest and then to apply 3D queries outside of the database to particularize the selection. The alternative would be to export the entire database to apply external functions which is not accomplishable for massive city models. The self-contained file-based design of SpatiaLite and our platform-independent CityGML database importer allow creating a new database on the smartphone and directly importing data. This also enables the possibility for a semi web-based solution so that data can be important during an available network connection and later provided from the local storage when the network is

Table 1. Maximum capabilities of SpatiaLite (SQLite, 1 August 2017, https://www.sqlite.org).

Max string/BLOB length	2.147483647 GB
Max pages	2,147,483,646 pages with size 65.536 kB
Max attached databases	125
Max database size	140 terabytes (TB)

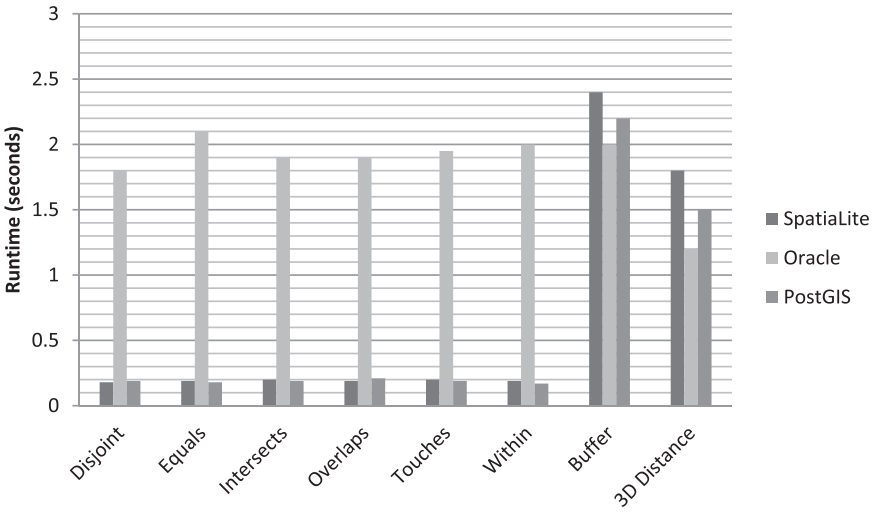

Figure 4. Runtimes for typical spatial functions. A selected polygon was queried against 30,761 polygons on an Intel® Core™ i7 – 4770 with 8 GB RAM.

unavailable. Alternatively, the database file can be created on an external device and then transferred to the smartphone. Therefore, large city models can be imported easily using more powerful desktop computers.

A database schema was created for the CityGML modules core, building and appearance and for the geometry and address model according to the CityGML Encoding Standard Version 2.0 (Gröger et al. 2012). All classes, attributes and relationships relevant to buildings are preserved. In comparison to, for example, 3DCityDB the schema is more generalized. For instance, multiple relationships are represented as one (e.g. lod1Solid, lod2Solid, etc. → lodXSolid). Similar to 3DCityDB, subclasses such as Window and Door are merged into single tables (e.g. OPENINGS) (see Table 2 for a complete overview). For optimized storing and accessibility we simplify complex object attributes and datatypes (e.g. multi-valued attributes) to reduce the number of database tables. For instance, multiple functions of a building are represented as a concatenated string divided by a separator, so these can be stored in a single column. For preliminary research towards use cases of CityGML and AR, the appearance model was reduced to the X3DMaterial feature. Generally, textures consume more memory than simple color information therefore their overall benefit for AR at the cost of the application's performance first must be assessed in further research.

CityGML geometries are extracted from the source file and converted to native SpatiaLite geometries. For this SpatiaLite provides multiple functions such as the OGC compliant

Table 2. Mapping of CityGML classes to database tables.

Database tables	CityGML classes
ADDRESSES	Address
BUILDINGS	_CityObject;_AbstractBuilding;Building
BUILDINGINSTALLATIONS	_CityObject;BuildingInstallation;IntBuildingInstallation
BUILDINGPARTS	_CityObject;_AbstractBuilding;BuildingPart
CITYMODELS	CityModel
BUILDINGFURNITURE	_CityObject;BuildingFurniture
OPENINGS	_CityObject;_Opening;Window;Door
GEOMETRIES	GML
ROOMS	_CityObject;Room
BOUNDARYSURFACES	_CityObject;_BoundarySurface;RoofSurface;WallSurface;GroundSurface;ClosureSurface;CeilingSurface; InteriorWallSurface;FloorSurface;OuterCeilingSurface;OuterFloorSurface
APPEARANCES	Appearance;_SurfaceData;X3DMaterial

GeomFromText or GeomFromWKB functions which expect a geometry in Well-known Text (WKT) or Well-known Binary (WKB) correspondingly, along with an optional Spatial Reference System Identifier (SRID). These functions allow us to easily transfer the spatial data from a CityGML file to the database.

Furthermore, it is important that the hierarchy of the objects is preserved. The primary keys of the database entries are set according to the identifiers of the CityGML objects. If no identifier is available, a Universally Unique Identifier is generated. Foreign keys are used to combine the data according to the data hierarchy defined by the CityGML schema, enabling smart object selections later in the application. So, the bottommost objects of the data hierarchy are the geometries, as shown in the example in Figure 5.

The database tables are created at runtime of the application with the help of SQL-Scripts and filled by utilizing our lightweight CityGML pull parser. We apply the XMLPULL API-based XmlPullParser, which is included in Android™ and is also available in the Windows®-based Java libraries. Event types can be triggered by progressing through the XML structure with the method next. The method registers inter alia the event types START_TAG, TEXT, END_TAG and END_DOCUMENT. This has the major advantage that only relevant tags and therefore data of interest are handled while non-relevant XML blocks are skipped entirely.

After the actual parsing process is finished, some post-processes such as the calculation of bounding boxes for the CityGML objects are invoked.

3.1.2. Data selection

The implemented selection method is explained using the AR example of visualizing a planned building on a parcel. The typical user simply needs to see a limited selection of buildings or the room he is currently in. To achieve this, the hierarchical structure of CityGML offers promising possibilities. Given the current position of the user, relevant objects for this view point can be determined by exploiting the semantic relationships of the CityGML objects.

The selection process is performed in the following way (Figure 6), given a LOD4 CityGML model with multiple buildings: With a position, first the relevant city model is selected in which further

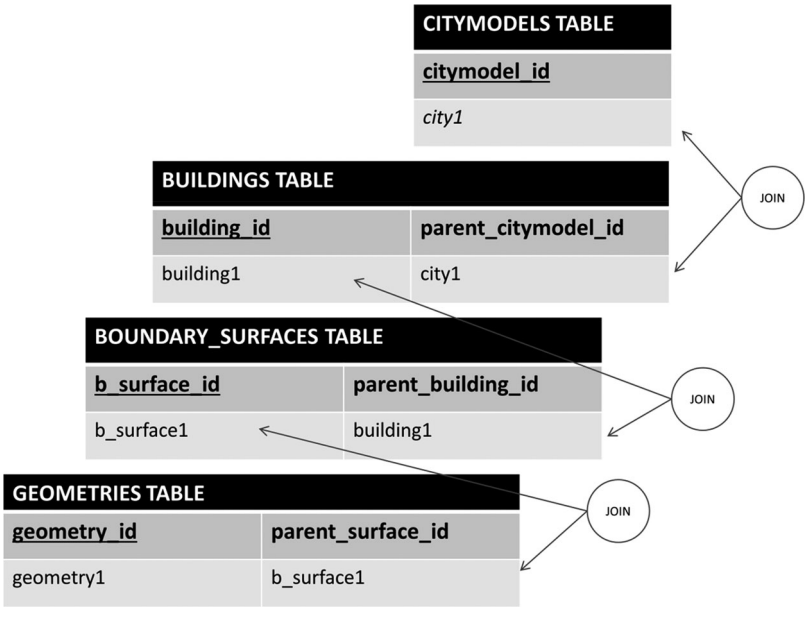

Figure 5. An example for the hierarchy of CityGML classes in database.

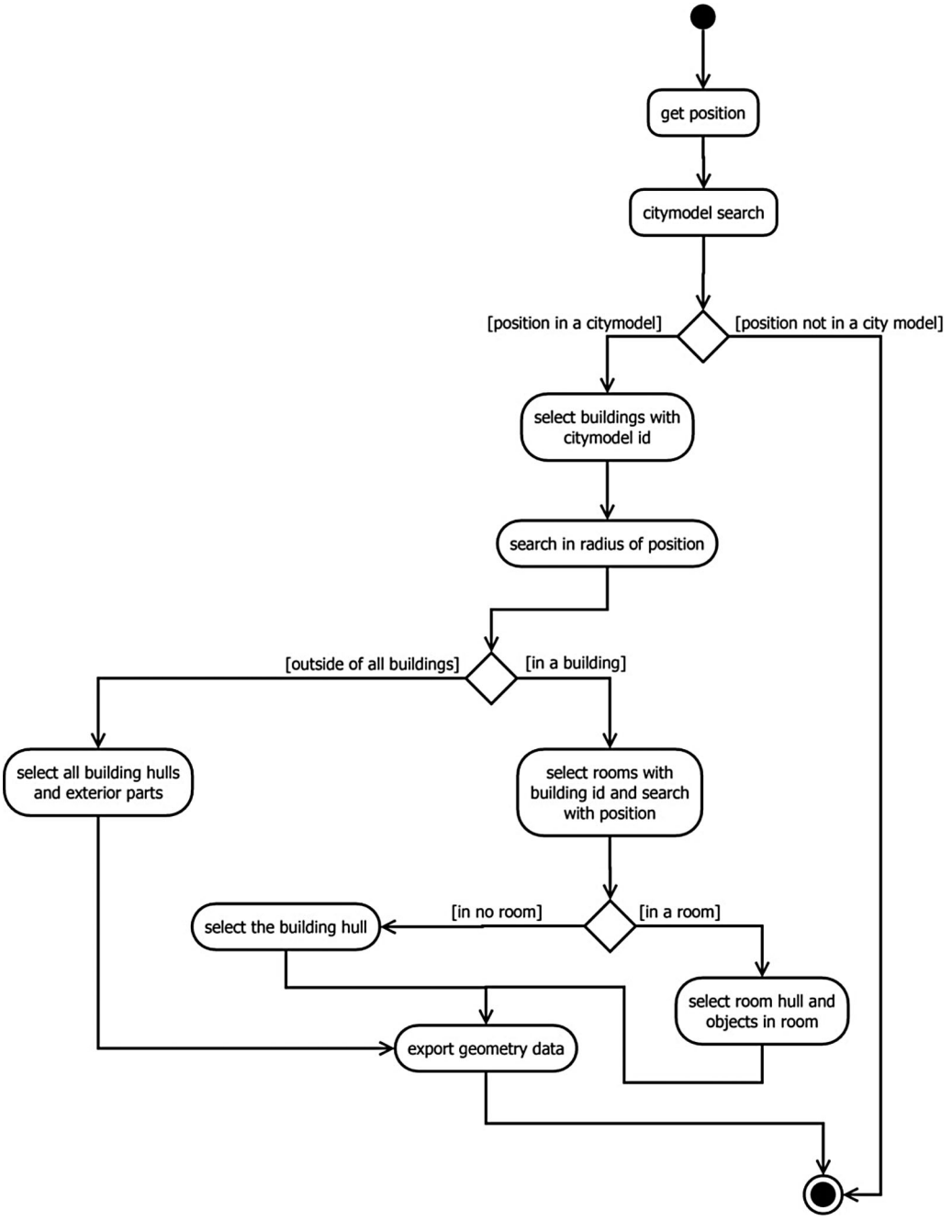

Figure 6. Selection process using the CityGML data query algorithm.

queries are carried out. In the next step, all buildings of the city model are selected by using the city model id and narrowed down by a geometric search in a defined buffer around the position. The unit type of the radius is set according to the current spatial reference system of the database. The temporary collection of buildings is used for a further position-based query. If the position is located outside of all buildings, only the building hulls (walls) and exterior parts, such as outer building installations, are exported, so basically LOD3 buildings are created from the LOD4 data. If the

position is located in the interior of a building, only the relevant room with its contained objects is exported. If no rooms are found, in case of a LOD1-3 model, only the hull of the building without the exterior objects (e.g. outer building installations or other buildings) is loaded. Like this the data volume can be kept at a minimum.

SpatiaLite facilitates the export of geometries in WKT, WKB or other geometry representations such as Scalable Vector Graphics (SVG), Keyhole Markup Language (KML) or GML. We export the geometries as byte arrays and parse the output according to the documentation of the internal BLOB-Geometry format which is similar to the WKB representation.

Finally, the exported data are passed to the visualization package.

3.2. Visualization

The visualization of the output CityGML geometries is accomplished with the platform-independent graphics specification Open Graphics Library (OpenGL™) and Open Graphics Library for Embedded Systems (OpenGL ES™) on mobile devices. By default, Graphics Processing Units (GPU) are optimized for processing triangles and thus expect triangulated geometry data (Luebke and Humphreys 2007), therefore a polygon mesh must be created beforehand.

3.2.1. Data preparation

Polygon triangulation is not a new problem. Consequently, a wealth of libraries with corresponding methods for polygon triangulation exists. These are typically written in C++ such as cgal. Only few such as JTS Topology Suite (JTS), Poly2Tri, jDelaunay or delaunay-triangulator are available in Java. We evaluated these and found that none consider all special cases that can occur in CityGML data. For instance, holes might not be considered or holes touching the outline of the polygon or another hole. Therefore, we used an existing in-house triangulation library that is implemented according to Eberly's (2002) Ear-clipping algorithm and specifically covers possible special cases. Since the Ear-clipping method is an algorithm for polygons in 2D space, the 3D CityGML polygons must be transformed into the second dimension before applying the algorithm.

For this we compute the local basis of a polygon and transform its points, which are assumed to be in world space coordinates, to the polygon's local space coordinate system. The basis requires three orthonormal vectors, one of which is given by the polygon's normal vector $\vec{n}$ which is treated as the z-axis of the basis. The x-axis is determined by projecting the x-axis of the standard basis $\vec{e}_1$ onto the polygon's plane. This is achieved with Equation (1) which projects $\vec{e}_1$ onto the plane determined by the polygon's normal vector $\vec{n}$ and is then subsequently normalized. If the scalar product of $\vec{e}_1$ and $\vec{n}$ equals 1, then $\vec{e}_1$ and $\vec{n}$ must be equal, in which case the negative z-axis of the standard basis $\vec{e}_3$ is used. The polygon's y-axis is determined by taking the cross product of $\vec{n}$ and $\vec{u}$ which is done in Equation (2). When the local basis of the polygon is found, the points can be transformed to it by using the formula in Equation (3).

$$\vec{u} = \begin{cases} \dfrac{\vec{e}_1 - \vec{n} * (\vec{e}_1 \cdot \vec{n})}{\|\vec{e}_1 - \vec{n} * (\vec{e}_1 \cdot \vec{n})\|} & \text{if} \quad \vec{e}_1 \cdot \vec{n} \neq 1 \\ \vec{e}_3 & \text{otherwise} \end{cases}, \tag{1}$$

$$\vec{v} = \vec{n} \times \vec{u}, \tag{2}$$

$$\mathrm{proj}_{u,v}(\vec{p}) = \begin{pmatrix} \vec{p} \cdot \vec{u} \\ \vec{p} \cdot \vec{v} \\ 0 \end{pmatrix}. \tag{3}$$

The original geometry is kept in 3D space, while references to the points of the triangles in 2D space are made.

Whether or not the Ear-clipping method is necessary depends on the shape of the polygon. If all vertices of the polygon are convex, the triangulation of the polygon is trivial and can be accomplished in linear time $O(n)$ by connecting an arbitrary vertex with all other vertices, except its immediate neighbors. If one of the vertices is concave, Ear-clipping must be used for triangulation which requires $O(n^2)$ when using a two-step approach of searching for ears and cutting these off according to Eberly (2002). We evaluated publically available CityGML data and custom models and found that 85–95% of all polygons are typically convex leaving only 5–15% that are concave. Furthermore, the number of vertices for each concave polygon is relatively low with an average of 7. Other triangulation algorithms with $O(n*log(n))$ as described by Held (2001) exist, but only have an advantage when the number of concave polygons or vertices is larger. Therefore, the Ear-clipping algorithm chosen for this work is sufficient.

3.2.2. Rendering

For rendering we use the game engine jMonkey. The advantage of such a software framework is that low-level methods are already implemented and are provided to the developer in an encapsulated form. Commonly, well established and cutting-edge technologies are integrated. Typically, a game engine provides support for scene graph-based rendering of 2D and/or 3D graphics, physical simulations, sound, animations, artificial intelligence and networking (Fugger 2004). Therefore, employing a game engine for CityGML is not only advantageous for efficient visualizations, but also for carrying out further tasks, as for example, complex simulations using realistic physics. For our use case the jMonkey (2017) game engine is best suited since it is a fully customizable Java™-based open-source framework that provides all necessary high-level functions while simultaneously facilitating low-level access. From experience it delivers the most complete package. It is stable, comprehensibly structured and offers all necessary features. Additionally it is well documented and has an active community. Though some commercial game engines such as Unity3D or Unreal Engine indeed contain some more sophisticated development tools and features, they typically restrain the developer to use engine specific concepts that make low-level adjustments particularly difficult or even impossible. There are several smaller non-commercial open-source game engines similar to jMonkey such as OGRE3D or libGDX, but these are typically focussed on a specific use case, limited in functionality or poorly documented.

Using jMonkey's scene graph we are able to structure CityGML's objects in the engine according to their hierarchy in the original source file. An element of jMonkey's scene graph is referred to as a Spatial. A Spatial can store transformations and arbitrary custom data and can be instantiated as a Node or Geometry. These are used for different purposes. While Geometries describe visible objects and carry a polygon mesh and material information, Nodes are invisible and are used for grouping Spatials. Therefore, we use a Geometry object to represent a single CityGML object and a Node to group these according to CityGML's schema. The identifier of the CityGML object is stored in the custom data of the Node or Geometry and is later used to query additional object data such as the name or description. While Geometries allow us to create lossless representations of CityGML objects, the Nodes enable us to preserve spatial topology relations between these objects. Figure 7 shows an example for this. The root node is the main Spatial of jMonkey's scene graph and is automatically part of an application. Any Geometry attached to the root node at any level is rendered. Using Nodes a tree is created and relationships between objects established, so that, for example, walls are part of a specific building which again is part of a specific city model.

These relationships enable us to select specific objects from the scene, for instance, all walls of a specific room belonging to a certain building. This is useful for visualizing only specific objects. Another advantage is that transformations that are applied to a parent Node are also applied to all child elements. This, for example, allows us to treat a building with all its sub-elements as a single object.

The polygon meshes of a geometry and the corresponding material are passed to the renderer, which places them in a predefined virtual environment with lighting and a camera. Careful attention

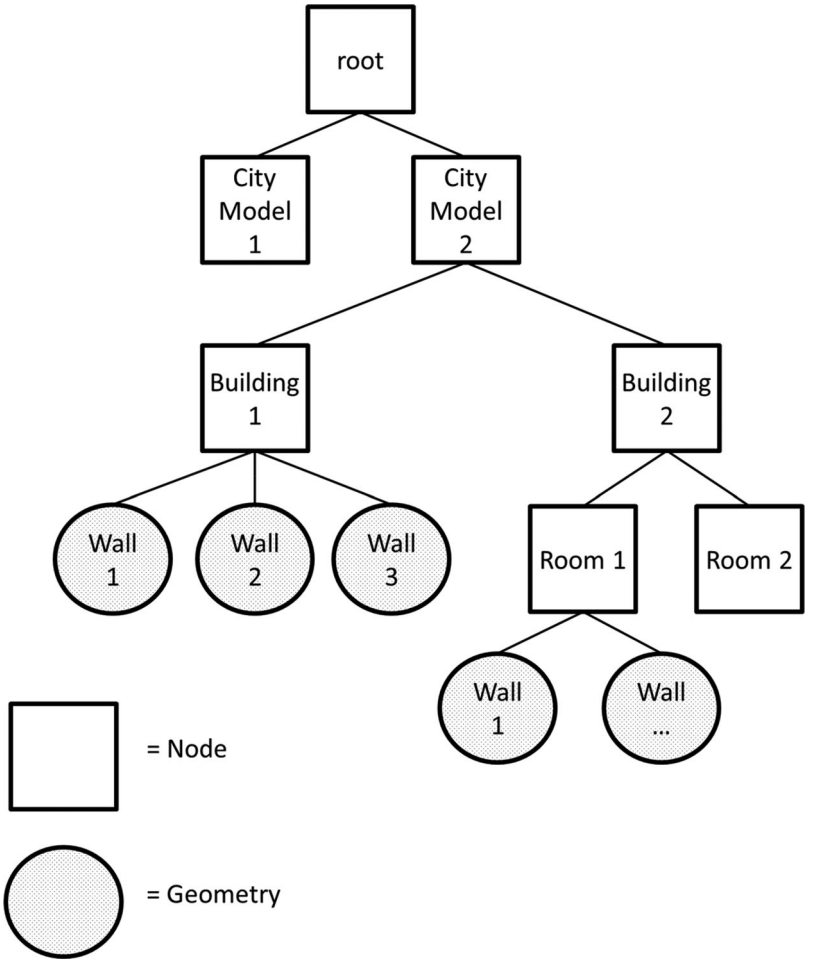

Figure 7. Example of the jMonkey scene graph model used to render CityGML.

must be paid to the placement of objects in the virtual world. The farther the geometries lie from the virtual coordinate system's origin, the more imprecise they are handled, resulting in visible jumping of the geometries (spatial jitter). This is due to the floating point number-based calculations of the GPU (Thorne 2005). One of our test data sets was a LOD2 CityGML model, which contains coordinates in the ETRS89 UTM32 reference system with six digits for the east value and seven digits for the north value plus multiple digits behind the decimal point each, creating the problem of jittery objects. To overcome this, we place the geometries as close as possible to the virtual world's origin, in order to minimize the coordinate values. We achieve this by a geometric translation, i.e. we determine the center of the CityGML model and subtract this value from all vertices of the models geometries, so that relative coordinates are created and the objects are shifted towards the origin. The transformations are carried out in-memory, so that the geometries stay un-edited in the database.

The number of separately rendered objects in a virtual scene (Figure 8) has a significant impact on the frame rate. To visualize a mesh the renderer issues a draw call to the graphics API. Draw calls are particularly expensive due to validation and translation steps (e.g. switching to a different material) which are executed in between each call. Rendering performance can be optimized by grouping multiple meshes into one batch to reduce the number of meshes sent to the renderer and therefore reducing the number of draw calls. This is typically referred to as Draw Call Batching (Akenine-Möller, Haines, and Hoffman 2008). We realize this by batching all meshes with the same X3D™ material.

Figure 8. LOD2 CityGML building model.

For example, all objects (e.g. walls, doors, windows, etc.) with the color gray or all objects with the color red are handled as one batch. So basically this method creates a single mesh from many separate meshes.

3.2.3. Object selection

To access additional semantic and descriptive information stored in the objects we apply a picking algorithm to the 3D scene and query our SpatiaLite CityGML database. This enables the user to select objects and display information about it.

The algorithm is based on a ray casting approach as described by Lawrence (2010) and Matusik (2012) which uses a directed line to determine the object of choice as depicted in Figure 9. When the user points at an object, either with the mouse on desktop-based systems or by touch screen on mobile devices, the screen space coordinates of the selection point are calculated and transformed into 3D coordinates in virtual world space. A point p_1 is calculated by using the z-value of the near plane. From point p_1 a ray is cast according to the current view of the virtual camera towards

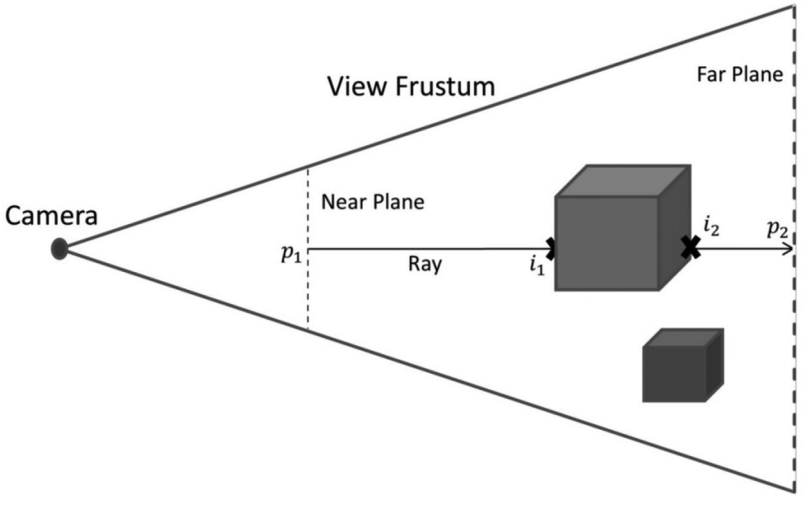

Figure 9. View frustum for selecting and displaying information about CityGML objects in the virtual world.

p_2 on the far plane. Intersections between the ray and objects in world space are calculated and used to determine the closest geometry collision from the camera's point of view, which is intersection i_1 in Figure 9. In turn the object ID of the collision object can be determined. We use the ID for querying the SpatiaLite CityGML database.

Since the picking algorithm is mesh-based it requires every object (e.g. wall) to be associated to a separate mesh to be identifiable. However, by merging multiple meshes this relation is lost. As a solution we assign a unique index number to each triangle of a combined mesh and keep track of which triangles belong to the original separate meshes. The picking algorithm returns the triangle's index which allows us to identify exactly which CityGML object was selected. This method only has the minor additional cost of storing the triangle indices of each CityGML object, but with the benefit of an improved rendering performance.

4. Evaluation

For proof of concept we evaluated the processing and visualization solutions described in this paper. While the proposed application has a generally different use case than web-based solutions, we nevertheless compare it to Cesium and 3D Tiles since these are currently much-discussed technologies in combination with CityGML.

We used a Google\LG Nexus 5™ smartphone with the Android™ 6 (Marshmallow) OS. The hardware specifications are:

- Qualcomm Snapdragon 800 2.26 GHz processor
- Adreno-330 450 MHz GPU
- 2 GB of RAM
- 32 GB internal memory

As of 2017, it can be considered an average smartphone and thus provides an appropriate development and testing environment. For the evaluation we used models with different LODs as shown in Table 3. M1 was extracted from the open data LOD2 model provided by the German state of North Rhine-Westphalia and M2 is a LOD3 model from the CityGML 2.0 test dataset (CityGML, 18 October 2017, https://www.citygml.org). M3 is the publically available LOD4 model FZK Haus (CityGMLWiki, 18 October 2017, http://www.citygmlwiki.org/index.php?title = FZK_Haus) and M4 a custom LOD4 model. The datasets were combined in a single SpatiaLite database.

4.1. Data processing

To handle 881 MB or even 2830 MB of data a geometric tile-based approach as described by the 3D Tiles specification is one possibility. CityGML objects are placed on tiles which then are arranged in a

Table 3. Statistics of test models used for evaluation of CityGML viewer.

	M1	M2	M3	M4
Addresses	43,960	/	1	10
Buildings	43,960	3	1	10
BuildingInstallations	/	56	10	90
BuildingParts	37,700	/	/	/
CityModels	1	1	1	1
Furniture	/	/	15	1300
Openings	/	43	32	9680
Rooms	/	/	7	380
Surfaces	681,240	40	55	7830
Polygons	695,944	666	15,137	3,301,930
Convex/concave	597,572/98,372	598/68	9886/5251	3,203,260/98,670
File size	881 MB	0.94 MB	16 MB	2830 MB

spatial data structure using K-d trees, Quadtrees, Octrees or Grids and loaded when required. This enables filtering objects of interest geometrically according to the current view. But it does not use the full potential of CityGML. In CityGML especially the additional information such as semantics attached to each object is what makes it stand out from other formats such as pure graphical ones. Therefore, it is essential to not only be able to query geometrically, but also semantically. Mixing both query types enables yet more complex queries. For instance, a useful query could be to select all doors with a certain name that belong to buildings in a defined radius that are part of a city model with a specific name. With 3D Tiles this is only partly accomplishable since the attributes of each object are stored in the tiles along with the geometry. To access this information all tiles must be loaded first. Furthermore, by including the additional data in the loaded tiles, file sizes are unnecessarily increased. A better solution is to query these data only when needed.

Our proposed solution realizes this by facilitating efficient complex queries and smart data selection algorithms. The following table lists average runtimes of such queries which each were run 10 times using the created CityGML test database (Table 4).

To evaluate the complete process the overall loading times were measured for querying the database, exporting geometries, preparing geometries for rendering (transformation/triangulation) and displaying triangulated meshes on screen. Positions inside and outside of buildings with different LODs were chosen to display exteriors and interiors of these. A selection/export radius of 100 m and a visualization radius of 50 m were used. The test area had the following objects in a 100 m radius (Table 5).

In Figure 10, B1 is a LOD2 building surrounded only by other LOD2 buildings, B2 is a LOD2 building surrounded by LOD2 and LOD4 buildings and B3 is a LOD4 building surrounded by LOD2 and LOD4 buildings. It can be seen that the LOD4 buildings around B2 influence the loading times only slightly compared to B1. The actual visualization of the LOD4 building B3 on the other hand needs double the time. But, with only around 5 s maximum, the loading times of this solution are a multiple more efficient than loading the data directly from the CityGML file. In all cases average loading times can be considered to be sufficient for loading/reloading data in real-time AR applications.

Using web-based solutions such as Cesium or iTowns the loading times from selection to visualization heavily depend on the quality of the network connection. As a comparison Table 6 lists some average runtimes using mobile cellular networks (3G, 4G) and a wireless computer network (WLAN).

4.2. Data visualization

One of the most influential factors for good rendering performances is the number of draw calls. Typically, everything above 15 frames per second (FPS) can be considered real-time, but with the minimum value of 15 FPS generally some flickering is still observable. We aim for a value of 60 FPS as in 3D real-time rendering this is a typical desired value for a fluent experience. Furthermore, the value was set as a maximum to ensure system stability. The native resolution of the screen (1920×1080) was used with a color depth of 32 bit per pixel (bpp), gamma correction enabled and anti-aliasing turned off. No further post-effects were placed on the objects.

In 3D Tiles (B3DM) every tile requires one draw call (Patrick Cozzi, 1 August 2017, http://cesiumjs.org/2015/08/10/Introducing-3D-Tiles). So the frame rate directly depends on the number of tiles displayed. We on the other side batch all polygons with the same color. The number of draw

Table 4. Average runtimes of some typical queries.

Select all buildings with defined name	111 ms
Select all doors of a building by semantic relationships	115 ms
Select all buildings of a city model in defined buffer	140 ms
Select all openings of buildings in city model contained in defined buffer	240 ms

Table 5. Number of CityGML objects in the evaluation area.

Addresses	87
Buildings	87 (86 LOD2, 1 LOD4)
BuildingInstallations	9
BuildingParts	10
CityModels	1
Furniture	130
Openings	968
Rooms	38
Surfaces	2088
Polygons	330,208

calls therefore mainly depends on the variety of colors in the scene. This has the advantage that scenes in which polygons are colored uniformly require only a few draw calls which is beneficial for the FPS. The relationship between the number of objects, triangles, draw calls and FPS can be seen in the following (Table 7).

For the Nexus 5 we found that with a small amount of triangles a maximum of 100 draw calls per frame is desirable and 150–200 are manageable. Larger numbers of triangles, as for example, in model M4, have a slight impact on the FPS due to the smartphone's GPU. Nevertheless, as a general comparison, without using the mesh batching method, the FPS of the four models ranges from only 15 down to 1, depending on the amount of visible objects.

Generally, the right type of batching scheme, either by tile as done in Cesium or color, heavily depends on the use case as both have their advantages and disadvantages.

5. Conclusion and future work

In this paper we presented our platform-independent offline CityGML viewer for use in mobile AR environments. Possible solutions for processing CityGML data from a file to an in-memory model were presented and evaluated. Due to the high use of RAM of these methods, we proposed a solution based on a mobile spatial database (SpatiaLite). We use the XMLPullParser, which is included in Android™, to read CityGML files. The data are saved to a SpatiaLite database that allows us to

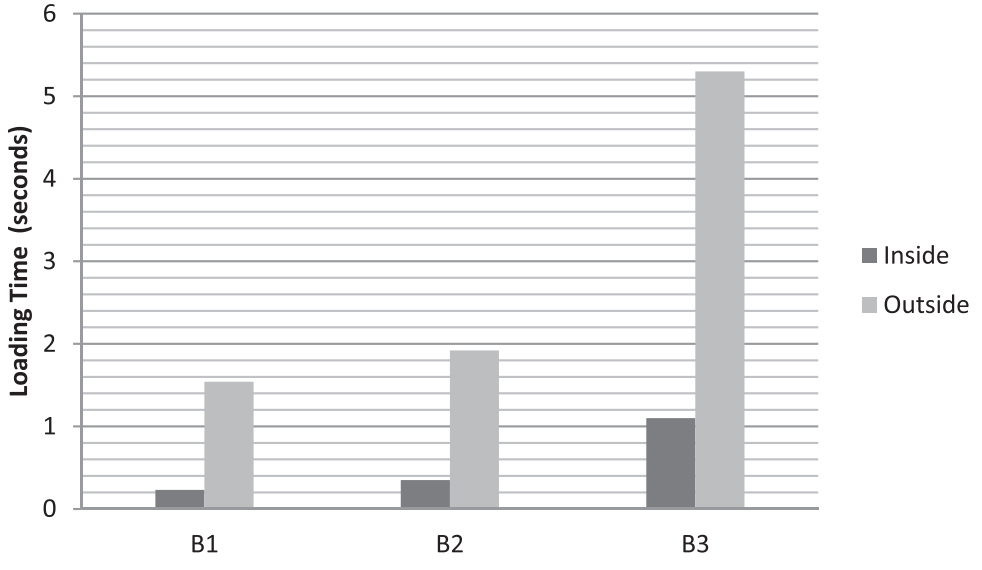

Figure 10. Loading times to visualize buildings B1, B2 and B3.

Table 6. Average loading times from selection to visualization of 15 MB CityGML data.

3G Regular (100 ms ping, 750 kb/s download, 250 kb/s upload)	2.7 min
3G Good (40 ms ping, 1.5 mb/s download, 750 kb/s upload)	1.3 min
4G Regular (20 ms ping, 4 mb/s download, 3 mb/s upload)	30 s
WLAN (2 ms ping, 30mb/s download, 15 mb/s upload)	8 s

Table 7. An evaluation of the rendering performance in the test models.

Model	Objects	Triangles	Draw calls	FPS
M1	10,280	60,274	5	60
M2	139	4832	8	60
M3	68	3108	10	60
M4	1694	283,024	21	26

store spatial data and perform position-dependent queries to load selective data, thereby, keeping the object count low and improve performance on mobile devices. The necessary polygon triangulation is performed by an extended Ear-clipping algorithm. When rendering, we combine meshes to reduce the number of draw calls to the OpenGL™ API which has a significant impact on the frame rate during visualization. The benchmarks showed that the performance is sufficient for a good usability and renders an appropriate solution for utilization in mobile real-time AR applications. Web-based solutions such as Cesium are not applicable in our case due to the lack of network connectivity and required low-level access to the smartphone's hardware for AR. The 3D Tiles specification presents efficient methods for displaying large-scale city scenes. Our use case though is focused on smaller more specific objects and the semantics of the objects, so a custom solution is required.

Future developments include the integration of the CityGML viewer into mobile AR applications, in which the components can be used to generate the visual information overlay for the AR system. The virtual objects are completely georeferenced and can, therefore, especially be used in location-based visualizations (e.g. for pedestrian navigation inside of buildings). Further CityGML object information, such as attributes and semantics, can be accessed by object selection and the ID-based link to the database objects (e.g. for maintenance issues).

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

Akenine-Möller, T., E. Haines, and N. Hoffman. 2008. *Real-time Rendering*. 3rd ed. Natick, MA: A. K. Peters, Ltd.

Beckmann, N., H. Kriegel, R. Schneider, and B. Seeger. 1990. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles." *SIGMOD Conference 1990*, 322–331.

Biljecki, F., J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. 2015. "Applications of 3D City Models: State of the Art Review." *ISPRS International Journal of Geo-Information* 4 (4): 2842–2889.

Chaturvedi, K. 2014. "Web Based 3D Analysis and Visualization Using HTML5 and WebGL." Master thesis, Faculty of Geo-information Science and Earth Observation of the University of Twente.

Dos Santos, S., and K. Brodlie. 2004. "Gaining Understanding of Multivariate and Multidimensional Data Through Visualization." *Computers & Graphics* 28 (3): 311–325.

Eberly, D. 2002. "Triangulation by Ear Clipping." *Geometric Tools, LLC*. Accessed August 1, 2017. http://www.geometrictools.com/

Fugger, T. 2004. *3D-Game-Engine*. Mainz: University Koblenz/Landau.

Gaillard, J., Vienne A., Baume R., Pedrinis F., Peytavie A., and Gesquière G., 2015. "Urban Data Visualisation in a Web Browser." In *Proceedings of the 20th International Conference on 3D Web Technology (Web3D '15)*, 81–88. New York: ACM.

Gesquière, G., and A. Manin. 2012. "3D Visualization of Urban Data Based on CityGML with WebGL." *International Journal of 3-D Information Modeling (IJ3DIM)*, 1 (3): 1–15.

Ghadirian, P., and I. D. Bishop. 2008. “Integration of Augmented Reality and GIS: A new Approach to Realistic Landscape Visualisation.” *Landscape and Urban Planning* 86: 226–232.

Giovannini, L., S. Pezzi, U. Staso, F. Prandi, and R. Amicis. 2014. “Large-scale Assessment and Visualization of the Energy Performance of Buildings with Ecomaps.” In *Proceedings of 3rd International Conference on Data Management Technologies and Applications (DATA 2014)*, edited by Markus Helfert, Andreas Holzinger, Orlando Belo, and Chiara Francalanci, 170–177. Setúbal: SCITEPRESS – Science and Technology Publications, Lda.

Gröger, G., T. H. Kolbe, C. Nagel, and K. H. Häfele. 2012. "OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0, Doc. No. 12-019.".

Guttman, T. 1984. "R-Trees: A Dynamic Index Structure for Spatial Searching." *Proceedings of 1984 ACM SIGMOD International Conference on Management of Data*, 47–57.

Held, M. 2001. “FIST: Fast Industrial-Strength Triangulation of Polygons.” *Algorithmica* 30 (4): 563–596.

Herring, J. 2001. "The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema), Version 5. OGC Document Number 01-101.".

Kolbe, T.H. 2008. “Representing and Exchanging 3D City Models with CityGML.” In *3D GeoInformation Sciences*, edited by S. Zlatanova, and J. Lee, 15–31. Lecture Notes in Geoinformation and Cartography. Berlin: Springer.

Lawrence, J. 2010. *3D Rendering and Ray Casting*.

Luebke, D., and G. Humphreys. 2007. “How GPUs Work.” *Computer* 40 (2): 96–100.

Matusik, W. 2012. *Ray Casting and Rendering*. Accessed August 1, 2017. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-837-computer-graphics-fall-2012/lecture-notes/MIT6_837F12_Lec11.pdf

Prieto I., Izkara J. L., and F. J. Delgado del Hoyo, 2012. "Efficient Visualization of the Geometric Information of CityGML: Application for the Documentation of Built Heritage." *Proceedings of the 12th International Conference on Computational Science and Its Applications*. Vol. Part I, 529–544. Berlin: Springer-Verlag.

Schall, G., J. Schöning, V. Paelke, and G. Gartner. 2011. “A Survey on Augmented Maps and Environments: Approaches, Interactions and Applications.” *Advances in Web-based GIS, Mapping Services and Applications* 9: 207.

Schall, G., S. Zollmann, and G. Reitmayr. 2013. “Smart Vidente: Advances in mobile Augmented Reality for Interactive Visualization of Underground Infrastructure.” *Personal Ubiquitous Computing* 17 (7): 1533–1549.

Schilling, A., J. Bolling, and C. Nagel. 2016. "Using glTF for Streaming CityGML 3D City Models." (October 2015), 109–116.

Simões, B., F. Prandi, and R. De Amicis. 2015. “i-Scope: a CityGML Framework for mobile Devices.” In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft ‘15)*, 52–55. Piscataway, NJ: IEEE Press.

Thorne, C. 2005. "Using a Floating Origin to Improve Fidelity and Performance of Large, Distributed Virtual Worlds." In *CW ‘05: Proceedings of the 2005 International Conference on Cyberworlds*, 263–270. Washington, DC: IEEE Computer Society.

Vlahakis, V., J. Karigiannis, M. Tsotros, M. Gounaris, L. Almeida, D. Stricker, T. Gleue, I. Christou, R. Carlucci, and N. Ioannidis. 2001. “ARCHEOGUIDE: First Results of an Augmented Reality, Mobile Computing System in Cultural Heritage Sites.” In *Proceedings of Virtual Reality, Archaeology, and Cultural Heritage International Symposium (VAST01)*, 131–140.

Woodward, C., and M. Hakkarainen. 2011. “Mobile Mixed Reality System for Architectural and Construction Site Visualization.” In *Augmented Reality – Some Emerging Application Areas*, edited by Andrew Yeh Ching Nee, 115–130. Rijeka: InTech.

Zamyadi, A., J. Pouliot, and Y. Bédard. 2013. “A Three Step Procedure to Enrich Augmented Reality Games with CityGML 3D Semantic Modeling.” In *Progress and New Trends in 3D Geoinformation Sciences*, 261–275. Berlin: Springer.