Development of a Platform-independent Renderer for the Rendering of OpenStreetMap Indoor Maps in Flutter

Julia Richter[a], Robin Thomas[b], David Lange[c], Thomas Graichen[d] and Ulrich Heinkel[e]
Professorship Circuit and System Design, Chemnitz University of Technology, Reichenhainer Straße 70, Chemnitz, Germany

Keywords: OpenStreetMap, Indoor Maps, Rendering, Mapsforge, Flutter, Cross-platform, Open Data, Open Source.

Abstract: With the development and spread of new localisation technologies, the construction of bigger buildings, as well as the continuous growth of digitalisation, the importance of indoor maps rises. However, developers who want to make use of indoor maps face several obstacles. Among them are the often costly data acquisition, the increased development overhead for diverse platforms, plus the missing support of indoor rendering in many libraries. In this work, the development of a free solution for platform-independent rendering of indoor data is presented based on public geodata that is provided by OpenStreetMap. For this goal, existing open source technologies such as the Flutter toolkit and the Mapsforge library were used in order to develop a flexible and freely accessible rendering engine that directly integrates in outdoor maps and leads to a seamless rendering of both indoor and outdoor in one map view. To prove platform independence and to measure performance, a prototype app was developed in Flutter. Finally, possibilities and limitations of this approach were examined in more detail.

1 INTRODUCTION

Indoor maps and navigation are particularly relevant for **larger buildings** such as airports, railway stations, hospitals, multi-storey car parks or shopping centres (OpenStreetMap Contributors, 2021a). In many indoor applications, various interdisciplinary areas intertwine. These include localisation, but also the **acquisition and provision of geodata**. This is usually done by companies such as Mazemap (MazeMap, 2020), Here (Here, 2021) or Carto (CARTO, 2021). Such processes and services are **complex and associated with costs**, so that this data is typically **not made freely available**. An alternative to the companies mentioned is the community-managed **OpenStreetMap** (OSM) (OpenStreetMap Contributors, 2021b). Since all data is made **freely available**, this is especially interesting for smaller companies, non-commercial organisations and scientific research.

In addition to data acquisition, the area of **information visualisation** is also relevant. **Indoor maps** are a typical form of visualisation for indoor geodata and can be displayed in 2-D or 3-D depending on the application. ”Indoor visualisation differs from outdoor visualisation in the necessity of representing different floors in the same geographical space. Therefore, some concepts of outdoor visualization can be extrapolated to indoor visualisation, while others must be reconsidered.” - (Amat et al., 2014). A computer-generated visualisation is typically based on a **renderer**. This is a programme that converts data, in this case geodata, into pixels based on defined rules.

A general hurdle in software development, however, is the **cross-platform** development of programmes that can run on different devices such as smart phones or desktop computers with different architectures and platforms.

For a few years now, there has been another attempt to solve this problem and to realise **cross-platform** applications with the **Flutter** Software Development Kit (SDK) initiated by Google (Google, 2020). However, there is currently **no library for Flutter that can render OpenStreetMap indoor data**. Still, there already exists the open-source library **Mapsforge Flutter** (Schwartz, 2020) that can render **outdoor maps**. So far, developers who want

[a] https://orcid.org/0000-0001-7313-3013
[b] https://orcid.org/0000-0003-4998-6774
[c] https://orcid.org/0000-0002-9738-042X
[d] https://orcid.org/0000-0003-1861-6033
[e] https://orcid.org/0000-0002-0729-6030

Richter, J., Thomas, R., Lange, D., Graichen, T. and Heinkel, U.
Development of a Platform-independent Renderer for the Rendering of OpenStreetMap Indoor Maps in Flutter.
DOI: 10.5220/0010781500003124
In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 1: GRAPP*, pages 147-154
ISBN: 978-989-758-555-5; ISSN: 2184-4321

to use indoor maps in their application have to implement their indoor rendering themselves.

Therefore, our aim in the presented work, is the **extension of Mapsforge Flutter with 2-D indoor rendering**. Special focus was placed on the **seamless integration** of the indoor maps into already existing outdoor maps or the outdoor rendering. The indoor rendering was also kept customisable through the existing themes in Mapsforge. A prototype was developed in Flutter to demonstrate the functionality of the indoor renderer. In this prototype, the user is able to explore outdoor areas as well as **different levels of indoor areas** in selected maps. At the same time, the prototype was compiled and tested on various end devices and platforms to prove its **platform independence** and to measure **performance**. The extension of the **open source** library Mapsforge Flutter will also allow other developers to re-use the code.

The paper is structured as follows: Section 2 contemplates extant works. This is followed by Section 3 where our indoor rendering process, the resulting prototype as well as used concepts, data structures and libraries are explained in detail. Section 4 presents and discusses results concerning performance, platform independence and the rendering output. The last chapter summarises the work, points out limitations and provides an outlook for future work.

2 RELATED WORK

Indoor rendering of OSM data is still an under-researched topic in the **academic literature**. Many works mainly address the problem of indoor localisation with the rendering of indoor maps often playing only a subordinate role. The work by Amat et al. uses OSM data to develop an Android app with voice control for indoor positioning, navigation and information retrieval in smart cities (Amat et al., 2014). The building data is mapped using the now obsolete IndoorOSM scheme (OpenStreetMap Contributors, 2020a) and is later converted into the GeoJSON format for better processing. The outdoor rendering is implemented using the Mapsforge library (Mapsforge Contributors, 2020a), while the indoor rendering was implemented separately and is then displayed above the map in an overlay. This does not guarantee a seamless integration of indoor and outdoor maps, because text of an outdoor map might be cut off. Moreover, the original Mapsforge uses Java, which per se allows only desktop and Android applications and therefore is **not platform-independent**.

A work by Graichen et al. also uses the IndoorOSM scheme and the Mapsforge library to create a map view that renders indoor as an overlay over outdoor map data, which is also not seamless so that text gets cut off, see Figure 1, (Graichen, 2014). For this, the authors wrote an own extension for indoor rendering. However, this **code is not open source**. Their implementation of level selection is based on the zoom level and displayed depending on whether a building with indoor data is visible in the viewport. Then the level selection bar is dynamically fading in and out. Further work by Graichen et al. builds up on this work and extends it with new indoor tagging approaches (Graichen et al., 2017b) or indoor positioning and navigation (Graichen et al., 2017a).

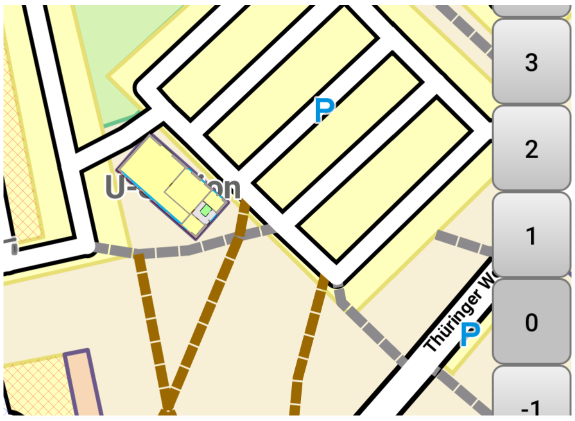

Figure 1: Example of a map where text of outdoor map is cut off due to the overlay.

In addition to academic work, there is also a variety of **community projects**, that have taken up the topic in the form of web apps **without focusing on cross-platform development** aspects. *OpenLevelUp!* accesses the OSM indoor data directly using the Overpass API and then renders them over the standard OSM pixel tiles (Pavie, 2020). The indoor maps are additionally highlighted by fading the outer areas with increasing zoom level. The selection of the levels is done as in previous works via a bar which contains all visible buildings. A special feature also allows the selection of non-integer levels. *Indoorequal* uses Mapbox vector tiles and Mapbox-GL (Mapbox, 2020) for map rendering (indoor=, 2020). Since the standard Mapbox tiles do not contain indoor data, a separate set of indoor vector tiles is created beforehand. Later, these can be reloaded and rendered on a layer above using Mapbox-GL. The level selection consists of separate buttons and, similar to *OpenLevelUp!* also displays non-integer levels. In contrast to other applications, *Openindoor* also renders buildings with 3-D information and hides the outdoor map completely when entering the indoor visualisation (openindoor, 2020).

Commercial map services such as *Google Maps* also offer indoor maps of shopping centres, airports or stadiums (Google, 2021). For general map rendering, vector tiles are preferably used. Pixel tiles only serve as a fallback, e. g. because the browser does not fulfil certain requirements. However, due to the proprietary code, details about the rendering are not available. Although *Google Maps* does not work with OSM data, it is at least worth taking a brief look at their interaction design. In contrast to other applications, the indoor maps and level bar are not displayed automatically. Instead, the building must be selected e. g. by a click or a search query. The level display preferably shows the exact level designation of the respective building instead of a sequence of numbers. There are many other companies that focus on indoor mapping, location and navigation (MazeMap, 2020), (esri, 2020), (Favendo, 2019), (mapwize, 2020). None of these companies, however, allow an insight into their rendering processes, because they are **not open source**. Similarly the few provided demos do not offer novel design techniques that might be of interest for later prototypical implementation. Consequently, commercial solutions **do not offer free libraries that can be re-used** for the development of new map services.

3 METHODS

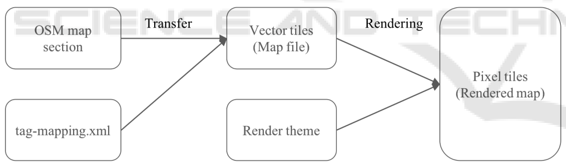

Figure 2: Overview: From OSM indoor data to rendered maps (pixel tiles).

Figure 2 demonstrates the rendering process in which we integrated indoor rendering. Starting from OSM map sections, OSM indoor elements are filtered out by means of the so-called *tag-mapping* file, transferred and stored as vector tiles in a so-called map file. Thereupon, the selected indoor elements are rendered by using a defined render theme, resulting in a map visualisation based on pixels, so-called pixel tiles. In the following sections, every component of this figure is presented in more detail. We contributed to the advance of indoor rendering by realising the following aspects, which will be explained in detail in the subsequent sections as well and highlighted with numbers **(1)** to **(4)** in the next sections:

(1) Transfer of required indoor data with *level* or *repeat_on* tag into Mapsforge vector tiles

(2) Render theme extension with render rules for indoor elements

(3) Extension of the rendering library with an additional match check to an indoor level variable

(4) Extension of the tile identification by an indoor level value for retrieval from the cache

3.1 OpenStreetMap Data Structure

OSM data is commonly represented in the widespread eXtensible Markup Language (XML) and comprises the basic elements *node*, *way*, *relation* and associated *tags*. A *node* defines a point on the earth by the geographical coordinates longitude and latitude. A *way* or path contains an ordered sequence of references to *nodes* and can be understood as a polygon course. For a *way* to describe an area the first and the last *node* must be identical. The *relation* tag can be used to describe a relationship between two or more elements The *tag* element may be used within *node*, *way* or *relation* elements and is necessary to assign additional meanings to them.

For mapping the interior of buildings, the Simple Indoor Tagging scheme (SIT) (OpenStreetMap Contributors, 2020c) was developed and is now the established standard for OSM indoor mapping.

3.2 Simple Indoor Tagging

Figure 3 shows a schematic representation of all indoor elements. SIT is based on the OSM elements described in the previous section. For example, closed *ways* or *areas* are used for the representation of rooms. These elements are then described and classified in more detail using additional *tags* (Graichen, 2018). With the *indoor* key, surfaces can first be assigned to one of the following four indoor elements: *room*, *area*, *wall* or *corridor*. Further details such as room name or room number can be added with the keys *name* and *ref*. All indoor elements can additionally be connected with doors. A door is a single *node* that is shared by both rooms or a *way* adjacent to both rooms and is described by the key *door*. To define over which levels a room or point of interest (POI) extends, the *level* key is used. By means of this key, an elevator extending from the ground level to the top level of a building, can be mapped for example. In the special case where geometrically identical indoor elements can be found repetitively on multiple levels they may only be defined once supplemented by the *repeat_on* key.

To create a map data visualisation from this data, so-called "renderers" are programmed, which process the data and convert it into a visual representation.

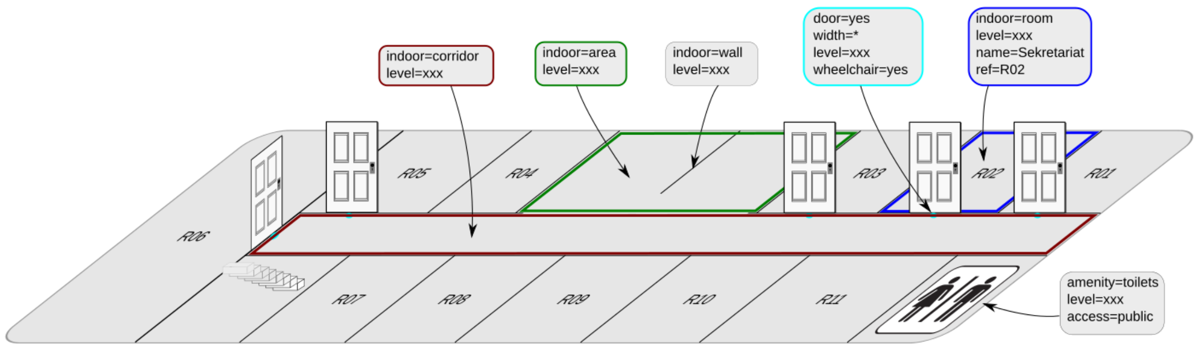

Figure 3: Illustration of the four basic indoor elements in OSM. Graphic taken from OSM Wiki (OpenStreetMap Contributors, 2020c).

3.3 Rendering

In most cases, maps are not rendered as a single continuous image, but as individual tiles, often called **pixel tiles**. A tile represents a square map section of e. g. 256 x 256 pixels and is uniquely described by its position and its zoom level. These tiles are later reassembled into a map in a second much more parsimonious rendering process to create a map.

However, map services such as Google Maps or Mapbox have recently started to use so-called **vector tiles** instead of pixel tiles. These are also map sections, but in the form of geometric data such as paths, shapes and points with additional information such as labels or symbols.

The **advantages of vector tiles over pixel tiles** are as follows: They require less memory, allow the map to be rendered in a custom defined style, are resolution independent, and allow intermediate zooming.

However, both vector and pixel tiles are dependent on a tile server and therefore on an internet connection. As an alternative approach **offline maps** can be used, which store and serve vector tiles directly from the device. Various libraries already exist for the integration of offline maps into own apps (OpenStreetMap Contributors, 2020d). These include open source libraries such as MapboxGL and Mapsforge. Since no uniform vector tile or offline map format exists, the use of a library also leads to a binding to the respective tile format. Although MapboxGL is open source, there is a charge for providing the tiles for larger projects. Thus the Mapsforge library currently offers the best completely free solution.

3.4 Mapsforge

Mapsforge is a library programmed in Java for the simple integration and display of OSM offline maps. With Osmosis, a tool for processing OSM data (OpenStreetMap Contributors, 2020b), and the Mapsforge Writer plugin (Mapsforge Contributors, 2020b), the data can be converted into the **Mapsforge file structure**, i. e. the **map file**. The Mapsforge-Writer resolves all references and relations to concrete values and elements, so that POIs and Ways are stored directly with their associated coordinates and tags. Furthermore, the data is grouped into zoom intervals. This allows a more efficient rendering later on as objects that are not visible are already pre-filtered. Additionally, map files require much less memory than OSM files (factor approximately 50), which is an advantage in terms of downloading and an efficient access of data.

Additionally many OSM tags, which are not needed for the map visualisation, are filtered out when creating the **map file**. The tags to be transferred are stored in the tag-mapping.xml file. This allows the customisation of tags that are finally written into the POI and way data block. **(1)** At this point, we have adjusted the creation of the map file to additionally enable the transfer of indoor data that contains the *level* or the *repeat_on* tag to this very file. We moreover had to specify indoor-relevant tags to be transferred such as *room, area, wall, corridor, entrance, door, stair*.

This map data contains coordinates and tag information, but no data about how the elements should be displayed later. This information is defined separately in a render theme.

As mentioned previously, most vector tile renderers offer the possibility to define the map appearance by means of **render themes**. Mapsforge offers its own XML-based style sheet format. In contrast to OSM files, a fixed XML schema exists here that can be used to validate each theme. To determine how a *way* or *node* element is displayed, it must first be selected. For this there are so called rules in Mapsforge. These rules determine which properties an element must meet, so that it is selected. They are defined by the rule element, which also allows nesting, in which case several rules must apply. All criteria by which an element can be selected are defined as attributes in the rule elements. The e-attribute allows the three values *node*, *way* or *any* and defines which element should be selected. The k and v attributes define which *tag key* and *tag value* the element must have for the rule to apply. After the rule-based selection of the map elements, they must now be rendered with the appropriate **render commands**. Render commands consist, similar to scaleable vector graphics (SVG) elements, of an element name (the render- command) and a list of attributes such as colour, line thickness or font. **(2)** In the presented work, we extended the render theme by defining colour codes for building elements and by creating icons for POIs such as toilets, shops and restaurants as can be seen in the figures in Section 4.3.

The **renderer** is the program component that processes vector tiles into pixel tiles based on a render theme. The Mapsforge map visualisation is initially divided into two separate layers: The tile layer, where the pixel tiles are merged, and the so called label layer, where symbols and labels are rendered and which is rendered on top of the pixel layer. On the one hand, this makes it possible to render text elements that protrude beyond tile boundaries without them being cut off. On the other hand, it further allows an efficient map rotation without having to rotate the labels and symbols or re-render the tiles each time. In addition, this division allows the implementation of a placement algorithm for labels and symbols. Since the rendering of vector tiles to pixel tiles is a computationally intensive process, pixel tiles are stored in a cache once they have been generated. For each tile to be rendered, the elements it contains are checked against the tags defined in the render theme: Only the tags contained in the theme are rendered. **(3)** Since the vector tiles contain indoor elements for multiple floors, elements that do not match the currently selected indoor level must be filtered by their respective *level* or *repeat_on* tag. In this way, only the floor selected by the user is rendered and then written to the cache for faster reuse. **(4)** To enable access to tiles representing different floors in a building, we extended the tile cache by a *level* component.

As mentioned earlier, Mapsforge is written in Java. Although Java is available on many platforms, it is not available for iOS or web app development by default. This lack of platform independence is compensated by other libraries such as Mapbox with platform-specific SDKs (Mapbox, 2020), which must be developed separately for each platform. An alternative solution is provided by the Flutter SDK, which allows to compile apps with the same code base for different platforms (Google, 2020).

3.5 Flutter

Flutter is an **open source** SDK from Google for the development of **cross- platform** applications using the object-oriented programming language Dart. Flutter uses its own rendering engine to create custom widgets. Flutter widgets are modular and developed directly in Dart. This allows them to be extended or even to be edited and thus offer significantly fewer limitations than native widgets. This and the existence of an early Mapsforge port for Flutter (Schwartz, 2020) is why **Flutter was used as the development toolkit in our work**. The present work built on this foundation with the goal of integrating indoor rendering into Mapsforge Flutter in order to develop cross-platform indoor map apps. The developed prototype integrates the described indoor extensions in Mapsforge Flutter and demonstrates the rendering of example OSM indoor maps, which is shown in the next section.

4 RESULTS AND DISCUSSION

The code for rendering indoor maps is publicly available at (Schwartz, 2020).

4.1 Evaluation Setup

The app is tested on a total of four different platforms: Android, iOS, Windows, and Linux. Table 1 shows the more detailed specifications of the individual terminal devices.

4.2 Performance Analysis and Platform Independence

Since all tiles are rendered one after the other, the time required for this is measured per tile. This has the advantage that the tile size is always the same and therefore independent of the screen/viewport of the respective platform. It should be noted that the rendering performance depends on many factors, such as the theme, the used map or the displayed map section. The performance is tested on the four zoom levels 14, 16, 18 and 20 for one example map. A higher zoom level means having zoomed further into the map, i. e. being closer to the ground. The average values measured for each platform and zoom level are shown in from Figure 5.

The first result to notice is that the rendering time increases significantly at a lower zoom level, i. e. when you have zoomed out. This is due to the fact

Table 1: Terminal devices and platform specifications for cross-platform and performance evaluation.

Device	Desktop computer		Fairphone 3 (GSMArena, 2021b)	iPhone 6 (GSMArena, 2021a)
OS	Windows 10	Linux 5.4.101-1; Wayland - GNOME - Manjaro	Android 10 Kernel 4.9.218	iOS 12.5.1
CPU	Intel(R) Core(TM) i7-7700K CPU @ 4.20 GHz		Octa-core (4x1.8 GHz Kryo 250 Gold & 4x1.8 Ghz Kryo 250 Silver)	Dual-core 1.4 GHz Typhoon (ARM v8-based)
GPU	AMD Radeon RX 590		Adreno 506	PowerVR GX6450 (quad-core graphics)

that at lower zoom levels significantly more elements per tile have to be rendered than at higher zoom levels. In a comparison of the individual systems, the desktop applications perform better with a maximum average render time of 120 ms per tile. However, it should be noted that desktops usually have a larger viewport than mobile devices and have to render more tiles than mobile devices.

With regard to cross-platform development with Flutter, the tests showed an all-round positive picture. The app was executable on all tested systems without any particular changes, even though the desktop builds are still in the early stages. Only for the mobile platforms some app permissions had to be added to the code. It has also been confirmed that Flutter automatically adapts the app with regard to platform-specific functions and design patterns, such as the swipe back on iOS or the presentation of icons such as the ”More Options Button”.

4.3 Rendering Output

Figure 4 presents the rendering output for a variety of open and freely available OSM indoor maps. This figure demonstrates that the aim of this work, defined in Section 1, could be reached: We extended the Mapsforge Flutter with a 2-D indoor rendering that allows the **seamless integration** of indoor into outdoor maps without cutting off text. Moreover, the user is now able to select the **different floors** of a building. By means of the implemented features, indoor rendering is now **freely available** for usage in **cross-platform** research, community and commercial projects.

5 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

So far, all the applications and works presented that have dealt with the rendering of OSM indoor data render the indoor visualisation separately over the outdoor visualisation. The disadvantages associated with this are the overlapping of text and icon elements or an increased caching effort. In this work, indoor elements are not considered separately from outdoor elements. They are only subject to the indoor level parameter and receive a higher weighting by being rendered in the theme over outdoor elements such as buildings and streets. Icons or lettering can thus generally be rendered last and are therefore not occluded by indoor elements. Regardless of whether they originate from outdoor or indoor elements, they are always subject to the same layout algorithm, which enables consistent distribution and presentation. As a consequence, this work solves a fundamental problem of previous indoor rendering approaches. A final application with core requirements such as flexible level selection or automatic recognition of the currently available levels was developed in Flutter as a proof of concept, showing various building scenarios. Moreover, by testing the application on different end devices, the platform independence could be proven.

The Mapsforge port to Flutter is currently still at an early stage, which is why a number of errors were discovered in the course of the work that also affect the indoor rendering. These include, for example, labels or symbols being cut off or not visible. The reason for this is that they are currently still rendered directly into the tiles, causing them to be cut off whenever they extend beyond the tile boundaries. Another limitation arises from the use of Flutter: it does currently not allow rendering in a separate thread or isolate (Flutter, 2021). As a result, parallel rendering of multiple tiles is not possible and the rendering process can block the UI thread. Therefore, the flutter engine cannot reliably update the UI at 60 frames per second during interactions with the application, which can be noticeable to the user as stuttering (so-called jank) or, in the worst case, as a brief freeze of the application.

Further work could therefore look at developing a hardware accelerated renderer for Mapsforge Flutter. This could outsource the entire map rendering to the graphics card using OpenGL or Vulkan. In ad-

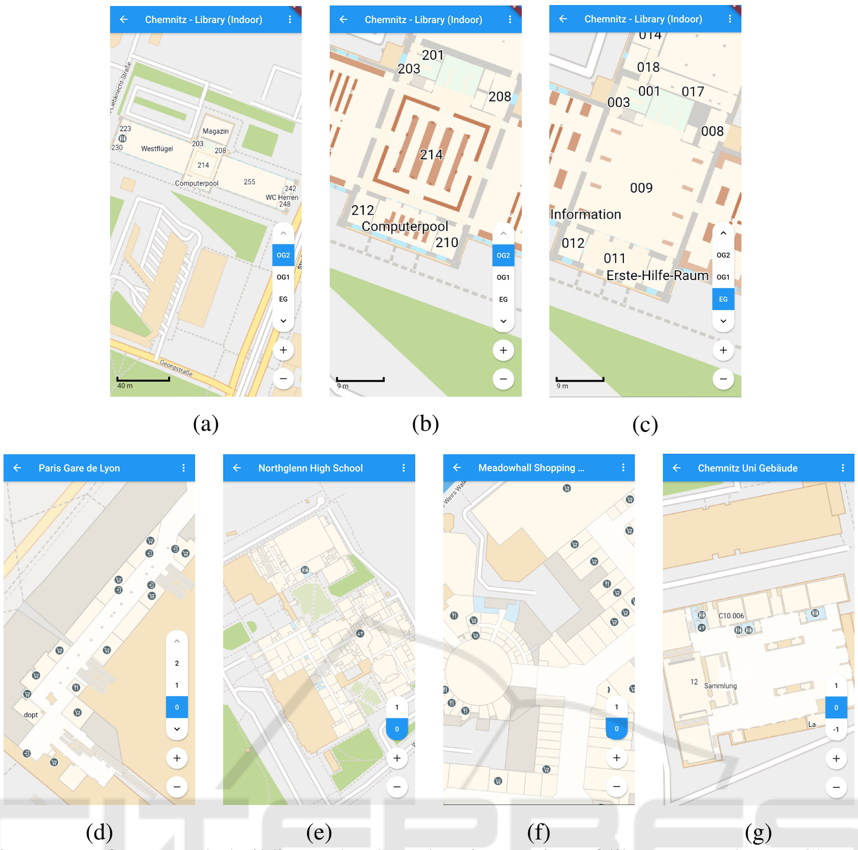

Figure 4: Rendering output for example buildings. 4a: Seamless integration of library map data at Chemnitz University of Technology into outdoor map. 4b: Library map section at lower zoom level. 4c: A level selection bar at the bottom right of the display allows user to view different floors of the building. 4d - 4g: Further examples of generated indoor maps.

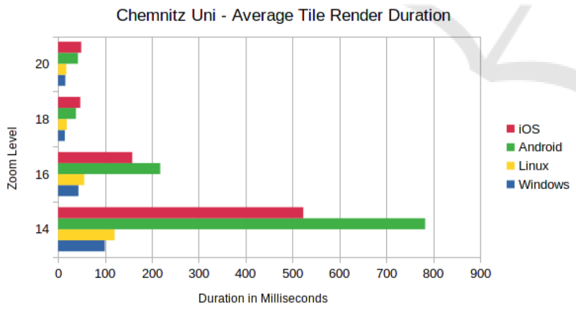

Figure 5: Performance analysis: Average render duration for an example building at Chemnitz University of Technology.

dition to accelerated rendering, this could solve UI thread blocking and result in other benefits such as reduced caching overhead. Other topics relevant to indoor maps are the elaboration and extension of OSM schemas such as doors and windows and, if necessary, the development of suitable Mapsforge render commands. The determination of the building associations of indoor data has also not been conclusively clarified. This is especially true for indoor elements that extend beyond the outline of the building or lie completely outside it. In this context, it is also worthwhile to examine alternative forms of representation for underground indoor maps in more detail. Currently, these are displayed seamlessly above the outdoor map, which means that an underground station can be located above roads or other buildings. However, this representation may be confusing because it contradicts real spatial relations.

It is likely that Flutter will become an increasingly popular and relevant tool for creating cross-platform applications in the future. With the extension of Mapsforge Flutter, this work provides a flexible renderer for OSM indoor data that can be used in a variety of different indoor-related applications.

ACKNOWLEDGEMENTS

This project is funded by the European Social Fund (ESF) with the project ID 100382183.

REFERENCES

Amat, G., Fernandez, J., Arranz, A., and Ramos, A. (2014). Using open street maps data and tools for indoor mapping in a smart city scenario.

CARTO (2021). Indoor Mapping & Analytics — CARTO. https://carto.com/solutions/indoor-mapping/, visited 2021-03-15.

esri (2020). Indoor Navigation Map & App Making | Build Custom Indoor Mapping Applications. https://www.esri.com/en-us/arcgis/products/arcgis-indoors/capabilities/indoor-map-app-making, visited 2021-03-15.

Favendo (2019). Indoor maps & mapping software for professional needs. https://www.favendo.com/indoor-maps, visited 2021-03-15.

Flutter (2021). Unable to call a platform channel method from another isolate · Issue #13937 · flutter/flutter. https://github.com/flutter/flutter/issues/13937, visited 2021-03-15.

Google (2020). Flutter - Beautiful native apps in record time. https://flutter.dev/, visited 2021-03-15.

Google (2021). Google Maps - Indoor-Karten. https://www.google.com/intl/de/maps/about/partners/indoormaps, visited 2012-03-26.

Graichen, T. (2014). A Combined In- and Outdoor Map for Android. https://www.youtube.com/watch?v=65hY4a9ObZI, visited 2021-03-15.

Graichen, T. (2018). Simple Indoor Tagging - An Indoor Mapping Approach for OSM.

Graichen, T., Gruschka, E., and Heinkel, U. (2017a). A map framework using crowd-sourced data for indoor positioning and navigation. In *2017 IEEE International Workshop on Measurement and Networking (M N)*, pages 1–6.

Graichen, T., Quinger, S., Heinkel, U., and Strassenburg-Kleciak, M. (2017b). A Novel, User-Friendly Indoor Mapping Approach for OpenStreetMap.

GSMArena (2021a). Apple iPhone 6 - Full phone specifications. https://www.gsmarena.com/apple_iphone_6-6378.php, visited 2021-03-15.

GSMArena (2021b). Fairphone 3 - Full phone specifications. https://www.gsmarena.com/fairphone_3-10397.php, visited 2021-03-15.

Here (2021). Indoor Mapping Solutions | HERE. https://www.here.com/platform/tracking-positioning-solutions/indoor-mapping-solutions, visited 2021-03-15.

indoor= (2020). indoorequal.org. https://github.com/indoorequal/indoorequal, visited 2021-03-15. Programmers: _:n309 original-date: 2019-12-23T11:58:55Z.

Mapbox (2020). Mapbox Documentation. https://www.mapbox.com/, visited 2021-03-15.

Mapsforge Contributors (2020a). Mapsforge. https://github.com/mapsforge/mapsforge, visited 2021-03-15. Programmers: _:n229 original-date: 2014-11-06T09:13:20Z.

Mapsforge Contributors (2020b). Mapsforge - Map-Writer. https://github.com/mapsforge/mapsforge/blob/master/docs/Getting-Started-Map-Writer.md, visited 2021-03-15.

mapwize (2020). Indoor mapping & Wayfinding for Smart Buildings. https://www.mapwize.io/, visited 2021-03-15.

MazeMap (2020). Indoor Maps. https://www.mazemap.com/indoor-maps, visited 2021-03-15.

openindoor (2020). OpenIndoor – for indoor maps. https://www.openindoor.io/, visited 2021-03-15.

OpenStreetMap Contributors (2020a). IndoorOSM - OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Proposed_features/IndoorOSM, visited 2021-03-15.

OpenStreetMap Contributors (2020b). Osmosis - OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Osmosis, visited 2021-03-15.

OpenStreetMap Contributors (2020c). Simple Indoor Tagging - OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging, visited 2021-03-15.

OpenStreetMap Contributors (2020d). Using OpenStreetMap offline - OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Using_OpenStreetMap_offline, visited 2021-03-15.

OpenStreetMap Contributors (2021a). Indoor/use cases - OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Indoor/use_cases, visited 2021-03-16.

OpenStreetMap Contributors (2021b). OpenStreetMap. https://www.openstreetmap.org/, visited 2021-03-15.

Pavie, A. (2020). OpenLevelUp! https://openlevelup.net, visited 2021-03-15.

Schwartz, M. (2020). GitHub - mikes222/mapsforge_flutter: A port of mapsforge for flutter. https://github.com/mikes222/mapsforge_flutter, visited 2021-03-15.