Occlusion in outdoor Augmented Reality using geospatial building data*

Johan Kasperi
KTH Royal Institute of Technology
Stockholm, Sweden
kasperi@kth.se

ABSTRACT

Creating physical simulations between virtual and real objects in *Augmented Reality* (AR) is essential for the user experience. Otherwise the user might lose sense of depth, distance and size. One of these simulations is occlusion, meaning that virtual content should be partially or fully occluded if real world objects is in the line-of-sight between the user and the content. The challenge for simulating occlusion is to construct the geometric model of the current AR environment. Earlier studies within the field have all tried to create realistic pixel-perfect occlusion and most of them have either required special depth-sensing hardware or a static predefined environment. This study proposes and evaluates an alternative model-based approach to the problem. It uses geospatial data to construct the geometric model of all the buildings in the current environment, making virtual content occluded by all real buildings in the current environment. This approach made the developed function compatible on non depth-sensing devices and in a dynamic outdoor urban environment. To evaluate the solution it was implemented in an sensor-based AR application visualizing a future building in Stockholm. The effect of the developed function was that the future virtual building was occluded as expected. However, it was not pixel-perfect, meaning that the simulated occlusion was not realistic, but results from the conducted user study said that it fulfilled its goal. A majority of the participants thought that their AR experience got better with the solution activated and that their depth perception improved. But any definite conclusions could not be drawn due to issues with the sensor-based tracking. The result is interesting for the mobile AR field since the great majority of smartphones are not equipped with depth sensors. Using geospatial data for simulating occlusions, or other physical interactions between the virtual and real objects, could then be an efficient enough solution until depth-sensing AR devices get more widely used.

KEYWORDS

Augmented Reality, AR, Occlusion, Physical simulation, Geospatial data, Open Street Maps

1 INTRODUCTION

Augmented Reality (AR), is the medium that supplements the reality with virtual content such as sound, video, images or 3D objects [1] by attaching content to a physical location, creating the effect of a virtual layer on top of the reality. The true strength of AR is that the users are enabled to see an enhanced reality with content that they ordinarily cannot see, for example making the construction worker see the beams inside a wall before they put their drill in it or the home decorator see the real size of a sofa before buying it [2, 3]. The technology has the potential of providing us with information about the real world in real time as we see it. This invites to interesting usage areas in fields such as construction, education and gaming among others. The term *Augmented Reality* was first coined in 1992 [4] but the first AR system was made back in 1965 [4]. Early AR systems required expensive hardware and a lot of custom development efforts, making them only available for governments and companies [5]. But the recent years of development in AR hardware, software and applications, as well as its sister technology *Virtual Reality* (VR), has made the technology relevant for everybody. Especially since the first release of the smartphone and its rapid development during the last decade. The smartphone made AR accessible for everyone and AR applications has already reached the consumers, for example the summer 2016 success game Pokmon GO [5]. The smartphone has pushed the development of so called *Mobile AR Systems* (MARS), which are AR systems developed for usage out in the real world and far away from the research labs [6]. However, the field still has some problems until it fulfills its ultimate goal:

> *"The ultimate goal will be to generate virtual objects that are so realistic that they are virtually indistinguishable from the real environment"* [1]

The ultimate goal will not be fulfilled until the transition between the virtual content and the real world is seamless. Both of them needs to coexist and interact realistically [7]. One challenge that needs to be tackled in order to achieve this is simulating physical interactions such as collisions, shadows, lighting and occlusions [8]. This study covers the latest research and proposes a new function for handling occlusion in AR. Occlusion refers to the problem when real objects that are closer to the user than a virtual object is not shown in the line-of-sight due to the virtual objects always overlaying real ones [7]. The virtual content is always displayed upon the reality and the reality act as a background to the virtual content. This artefact may cause a misleading experience for the user (see fig. 1) [9]. Not handling occlusion in systems with only virtual 2D content in screen space might be fine, but when it comes to AR systems with 3D content this is not desired.

2 BACKGROUND AND THEORY

This section covers a historical background and recent advances in occlusion handling for AR. It also provides the purpose and the research questions of this study. It finishes with describing the core technologies used when developing the prototype that was needed to answer this question and definitions of the core concepts.

*Master Degree Project in Computer Science and Communication.

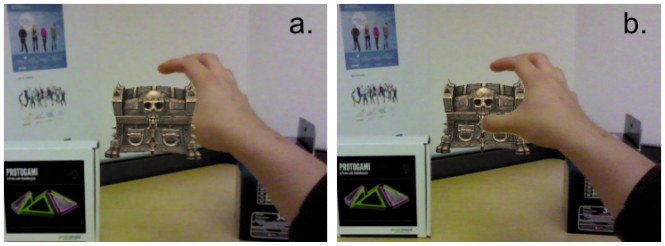

Figure 1: *a) Incorrect occlusion, b) correct occlusion [10].*

2.1 Occlusion handling in AR

As mentioned earlier "occlusion handling" in AR is the name of functions that tries to solve the problem of occlusion between virtual content and real objects. I.e. if real objects are closer in distance to the user than the virtual objects, these objects are supposed to be fully or partially occluded (see fig. 1) . If not, the user may lose their perception, making it harder to sense distances, size and depth of the virtual content [9].

Researchers at European Computer-industry Research Centre (ECRC) introduced a taxonomy for these functions, categorizing them into model-based methods and depth-based methods [7]. Both of these methods tries to solve the occlusion problem but in different ways and they each have their own strengths and weaknesses. The challenge that these methods face is to, in real-time, construct the geometric model of the occluding real objects and estimate the pose and position of the constructed objects as close as possible to the actual real objects [8]. Then the actual occlusion effect will be fairly simple to achieve and similar for both methods. Both of them use some sort of Z-buffering technique, meaning that pixels of the virtual content that should be occluded are pushed down the depth buffer by the information retrieved about the surrounding environment.

The model-based method involves all occlusion handlers that need a 3D model representation of the surrounding environment to function. This model can either be manually created [7, 11], defined by the user [12–14] or retrieved by depth sensors [15]. Breen et al. [7] performed one of the first model-based attempts for occlusion in AR by manually creating models of real objects in the current environment. They found the occlusion to be accurate but they also concluded that manually creating detailed models of complex scene is "difficult or impossible". Fuhrmann et al. [11] tried to achieve accurate occlusion in a collaborative AR setting, which meant that static objects like furniture, dynamic objects like tools as well as the human body needed to be included in their occlusion handler. They achieved this by pre modelling so called "phantoms" of all types of objects, and then aligning them with the corresponding real objects using magnetic trackers. [12, 13] created a semi-automatic method where the user had to trace the silhouette of occluding objects and then their function defined a 3D model out of the traced data. Tian et al. [14] had a similar approach by letting the user trace the occluding objects, but the trace information was used for object tracking instead of conversion into a 3D model. Tian et al. [15] used a RGB-D camera to first construct a 3D model of the environment in an offline stage and then they mapped this 3D model to the depth information retrieved in an online stage. By doing this they achieved a real-time solution that accepted an arbitrary position and orientation of the camera. However it required a static environment and they also concluded that prefetching the 3D model of the environment in an offline stage is not ideal.

Depth based methods include the functions that withdraw depth information of the surrounding environment in real-time. This depth information, or depth map, can then be converted into a 3D model of the environment used for the occlusion effect [7] or the depth information can be applied straight away to the Z-buffer in each rendering cycle [16]. Compared to the model-based method, this method do not require any previous information about the environment [9]. The depth map can be retrieved by using a stereo camera [7, 16–18], RGB-D cameras [10], Time-of-Flight (TOF) cameras [19], and LADAR technology [20]. Wloka et al. [16] conducted one of the earliest attempts of resolving occlusion in AR systems using a depth-based method with a stereo camera setup. They achieved the creation of the first real-time depth-based occlusion handler for AR, and the retrieved depth information was used for Z-buffering in order to create the occlusion effect. However, they found their algorithm to have difficulties in calculating depth on "rectangular image areas that are evenly lit, non-textured and horizontal". Breen et al. [7] was also early in research for occlusion in AR but compared to Wloka et al. [16] they used the depth map to construct a 3D model of the surrounding environment. They found that 3D reconstruction was time consuming since the model became invalid as soon as the camera changed position or orientation. More recent depth-based attempts using a stereo camera include [17] who achieved pixel perfect occlusion almost in real-time by creating dense disparity maps. Zhu et al. [18] fused the depth information from a stereo camera with color and neighboring pixels information. Their fusion algorithm achieved an accurate and robust result since when either the depth, color or neighboring pixel information was poor they could complement each other. Du et al. [10] introduced an "Edge Snapping-Based" depth enhancement to reduce the noise retrieved from RGB-D cameras that are used for occlusion in AR. By adding edge tracking algorithms they enhanced the occlusion effect significantly compared to using only the raw depth data. Fischer et al. [19] combined a regular color camera with a TOF camera to retrieve depth information of the environment, but they struggled with noise from the TOF camera as well as synchronizing both the color camera and the TOF camera. Behzadan et al. [20] achieved an occlusion handler that was functioning in a dynamic AR environment, both outdoors and indoors, in real-time. However, their setup included a heavy and expensive LADAR camera that the user carried on their back.

The model-based method has an advantage in not needing special hardware or big computing power when the application is running, however depth sensors can be used when retrieving the 3D model of the environment [15]. The method's obvious disadvantage is that the model of the environment needs to be acquired before hand [7, 9], making it only usable for predefined static environments. It may also be difficult to create detailed models of complex environments. However when the model is acquired the model-based method has proven to be robust and not having any visual artefacts, given that the model perfectly aligns with the real objects [7].

The advantages of the depth-based method is that it does not require a predefined 3D model, making it possible for usage in a

Figure 2: *Visualization of the "Geolocation Raycasting" technique used for determining the user's current FOV [23].*

dynamic and complex environment [7, 9]. However, it has been proven to be time consuming [15] and that is not ideal that the depth map becomes invalid when the device changes position or orientation [7, 9]. Depth sensing devices also produce a lot of noise, which create visual artefacts such as aliasing [9, 16, 18]. Common for all the previous depth-based attempts is that they require certain conditions, special hardware and/or big computing power. Neither of them can be implemented on regular non depth-sensing devices that the consumers have today. This might change soon as depth sensing devices such as Google Tango and Microsoft Hololens is already available for consumers [21, 22].

Kasapakis et al. [23] proposed an alternative approach to reconstruct the geometric model of the current environment, that did not require any special hardware or a predefined environment. However, it was not used for simulating an occlusion effect. They called their solution "Geolocative Raycasting" which determined the user's field of view (FOV) in an urban AR environment. By first querying geospatial data from *OpenStreetMap* (OSM) and then using ray casting from the user's current geographical position (defined by a smartphone's GPS) onto the surrounding buildings (defined by their corresponding geospatial data from OSM), they could in real-time create a polygon representing the area the user currently could see (see fig. 2). The area that was withdrawn from the ray casting function was then used to determine if the user could or couldn't see a Point-of-Interest (POI), by marking it with a red color. The results were a stable and accurate function, and they concluded that geospatial data can be used to construct the model of the surrounding environment. However, Kasapakis et al. [23] only used geospatial data to determine the user's FOV in 2D (latitude and longitude). It is also important to note that in their function only included buildings, other objects such as trees, cars or people did not affect the detection of if the POIs were visible or not.

2.2 Motivation and research question

Previous studies have tried to create pixel-perfect occlusion for AR systems, but they have all required either special hardware or a predefined static environment. This study has tried to do the opposite. It proposes a new dynamic occlusion handling for outdoor AR systems, functioning on regular smartphones without depth sensing capabilities and in a dynamic outdoor urban environment. However, the proposed solution do not try to achieve pixel-perfect occlusion. It tries to create an occlusion effect that is efficient enough, i.e. it tries to give users an acceptable sense of perception and depth. Also will it not include all real world objects in the occlusion function, it will only include buildings. The study continues the work of Kasapakis et al. [23] by using geospatial data from OSM in AR. But compared to Kasapakis et al., the solution in this study converts geospatial data of buildings from OSM into 3D models, allowing virtual AR 3D models to be occluded by real world buildings.

The research question of the study, and the motivation for developing the proposed solution, were:

> What are the results of occluding virtual 3D objects by real world buildings using geospatial data in an outdoor urban Augmented Reality environment?

2.3 Concepts and technologies

2.3.1 Tracking techniques. Tracking is the technologic term for the AR device's pose and field of view is estimated in relation to the real world [24]. AR is often divided into three categories based on the type of tracking technology being used [24]. The first category is *vision-based tracking*, which uses computer vision on the video stream from the camera on the device to estimate its pose in relation to real world objects [24]. The first vision-based tracking methods were tracking markers such as QR-codes. Then came tracking of images and the environment with 3D models, and finally *SLAM* which estimates the pose and position of the device where no model of the environment is available [5]. The second category is called *sensor-based tracking* which uses magnetic, acoustic, inertial, optical and/or mechanical sensors to determine the device's pose and FOV [24]. In mobile AR these sensors are typical GPS, magnetometer and gyroscope [24]. Sensor-based tracking is more suitable for outdoor AR since it does not require any type of marker tracking which makes it more versatile [25]. However, sensor-based tracking for mobile AR has issues with inaccuracy of the sensor which introduces noise to the device pose estimation function [25]. The third and last category is called *hybrid-based tracking* which is a combination of the two previous [24].

2.3.2 Geospatial data and OpenStreetMap. Geospatial data is data regarding the Earth's features [26]. One provider of open geospatial data is the crowd-sourced initiative *OpenStreetMap* (OSM) [27] and their data can be retrieved through their API, called *Overpass API* [28].

2.3.3 Argon.js. Argon.js is a *Software Development Kit* (SDK) for AR created by MacIntyre et al. [29]. The purpose of the SDK is to give developers a tool to create applications using common web technologies such as WebGL, HTML, CSS and Javascript with support of vision-based and sensor-based tracking. It is accompanied with the Argon browser, which is a native mobile web browser application with support for AR developed with Argon.js. The Argon browser was created due to the problem where different earlier AR all had their own mobile application, making it impossible to use several of them in parallel on the same device. The Argon browser creates a common ecosystem for different kinds of AR making it possible to use them simultaneously and thereby solving this problem [29].

3 METHOD

For the purpose of this study an experimental AR application was developed using sensor-based tracking. The application visualized a future building, that had not been built, in Stockholm, Sweden. The application included the developed model-based occlusion handler, and the models were created from geospatial data of buildings from OSM. A model-based approach was used since it should function on devices without depth sensors.

The final goals for the proposed occlusion handler were:

- Converting geospatial data of buildings from OpenStreetMap (OSM) to 3D models.
- Use the models from the conversion to create a model-based occlusion handler of buildings in a dynamic outdoor urban AR environment.
- Functioning on regular non depth-sensing smartphones equipped with a camera, Internet connection, GPS, gyroscope and magnetometer.
- The goals above will be achieved on the cost of not creating pixel-perfect occlusion.

This application was evaluated with a performance test and a user test. The performance test included execution time measurements of the creation of 3D models out of the geospatial response from Overpass API. The execution time for updating the position of the occlusion models for each render frame were also measured. Lastly a frame rate (FPS) analysis was made to see if the proposed occlusion handler affected the overall performance on the WebGL scene by comparing it to usage without the occlusion handler activated.

A user study were then conducted with 13 participants in the ages 22-31. They were instructed to use the developed AR application freely, first with the occlusion handler deactivated and then activated. However, before the test started they were given information about the purpose of the study in order to know what to evaluate. After using the prototype they were given a questionnaire consisting of 13 statements with Likert scale [30] type of options ranging from "Strongly disagree" to "Strongly agree". The answers were then scored in the analysis from 1 to 5 with 1 equal to "Strongly disagree" and 5 equal to "Strongly agree". The purpose of the user study was to evaluate if the develop occlusion handler was efficient enough to give the users a sense of perception and depth, i.e. did the occlusion handler reach the goal of occlusion handlers.

The proposed occlusion handling function was not evaluated with a root-mean-square error or similar. This decision was made because pixel-perfect occlusion were never a goal of the developed solution. Instead the visual effect of the occlusion handler, i.e. how it looked on the screen on the device, is presented with screenshots.

All tests were conducted on an Apple iPhone 5S with 1 GB RAM, Dual-core 1.3 GHz Cyclone CPU and PowerVR G6430 GPU.

4 RESULT

This section covers the result of this study. It starts off by listing the technologies used in the study's experimental application. After that, a description of the algorithm used for the occlusion handler will be presented. Finally are the results of the occlusion handler presented in three different topics, first the occlusion result (i.e. the visual effect of the occlusion handler), then the performance test and last the result from the user study.

4.1 Application structure

The entire application was developed to be used within the Argon browser app, so the AR features (the sensor-based tracking) was made with the Argon.js SDK. Since the application was made for the Argon browser, the application was developed using web technologies, i.e. WebGL, HTML and JavaScript. Three.js, a 3D computer graphics library for WebGL [31], was used for the 3D rendering.

4.2 Occlusion algorithm

Step one in the algorithm of the occlusion handler is to query geospatial data of buildings from Overpass API. The query is made in the Overpass query language called *Overpass QL* (see listing 1). Before querying the API, the algorithm needs a radius r and geolocation L as input. r and L determines a bounding circle on the earth's surface with r as the circle's radius and L as the circle's center. All buildings within this bounding circle will be fetched from Overpass API.

Listing 1: *Example query to Overpass API, given in Overpass QL, with r set to 100 and L set to (59.329553,18.079065).*

```
[out:json];
(
  way["building"](around:100,59.329553,18.079065);
  relation["building"](around:100,59.329553,18.079065);
);
out body;
>;
out skel qt;
```

The response from Overpass API with geospatial data of the buildings is then encoded in the *OSM JSON* format (see listing 2).

Listing 2: *Example response from Overpass API of the National Museum of Fine Arts in Stockholm, Sweden, given in OSM JSON.*

```
{
  "type": "way",
  "id": 24968329,
  "nodes": [
    1406078451,
    1863238948,
    2554104722,
    ...
  ],
  "tags": {
    "building": "museum",
    "building:colour": "brown",
    "building:levels": "3",
    "description": "Sweden's leading museum for art and
        design.",
    "name": "Nationalmuseum",
    "name:en": "National Museum of Fine Arts",
    "opening_hours": "Tu-We, Fr-Su 10:00-17:00; Th
        10:00-20:00",
    "roof:colour": "MediumAquamarine",
    "source": "yahoo_imagery",
    "tourism": "museum",
    "website": "http://www.nationalmuseum.se",
    "wheelchair": "yes",
    "wikipedia": "sv:Nationalmuseum"
  }
```

Figure 3: *The nodes, or corners, from Overpass API making the ground floor polygon of the National Museum of Fine Arts in Stockholm.*

```
}
```

As seen in listing 2 a great variation of data is listed in the response from OSM. The algorithm in this study utilizes that each building has references to multiple so called nodes which are the data representing a geolocation (see listing 3). In general, a node is a specific geolocation represented by a latitude and a longitude value.

Listing 3: *Node that is referenced by the National Museum of Fine Arts in Stockholm given in OSM JSON.*

```
{
  "type": "node",
  "id": 1863238950,
  "lat": 59.3285374,
  "lon": 18.0785740
}
```

Nodes referenced by a building specifically represents the geolocation of a corner on the building. The National Museum of Fine Arts has for example 34 node references, making the polygon seen in fig. 3.

The next step in the algorithm is to transform the OSM JSON data to *GeoJSON* [32]. GeoJSON is an alternate format to OSM JSON that makes it easier to traverse and extract desired data in the next steps of the algorithm. This step is not necessary for the occlusion handler to function, but it was included to make the development of the next steps easier.

As mentioned earlier, the occlusion handler uses a model-based method, therefore is the next step to create 3D models out of the geospatial data fetched in previous steps. As seen in fig. 3, the nodes referenced by each building creates a polygon representing the ground floor. The next step in the algorithm is to elevate this polygon along the y-axis to create a 3D model (see fig. 4). The height of this elevation is defined by the `building:levels` tag, representing how many levels the building has, that is included in

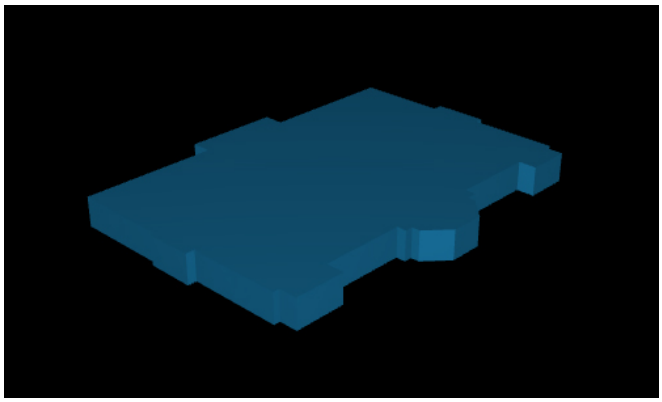

Figure 4: *3D model of the National Museum of Fine Arts in Stockholm created by using geospatial data from OSM.*

the Overpass API result (see listing 2). The level amount are then multiplied by 3 to decide the height in meters. Also if the building has a `roof:shape` (not included in listing 2) tag that is not equal to `flat`, meaning that the real building has a roof that is not flat, the level amount is increased with one unit. However, since OSM do not have level data for all buildings the algorithm falls back and sets the levels to the integer 3, i.e. three is the default number of levels in the algorithm. The creation of the model mesh is done by using the *ExtrudeGeometry* function in Three.js [33].

Before creating the models the algorithm waits until the device has initialized its GPS and the Argon.js environment has retrieved the geolocation of the device. All previous steps is done upon launch of the application. The algorithm awaits the geographical position of the device to be initialized before creating the models because this makes the rotation of the created models accurate and aligned with the real world buildings. The reason for this is that Argon.js always positions the device in origo in the WebGL scene.

The next step is to, in each render loop, accurately position the created occlusion models in the Argon.js environment in respect to the device so it aligns with their corresponding real buildings. Since the application uses sensor-based tracking, each building are positioned at the same geographical position as the first node (corner) that the building is referencing in its geospatial data. Each building also has its local origo around this node and, as mentioned earlier, it is already correctly rotated resulting in the entire model aligning with its real building.

The last step of the algorithm is to make all the occlusion models transparent and use them for Z-buffering, just like many previous attempts. This was made by setting the `colorWrite` property on the Three.js *Material* object to false, and setting the `renderOrder` on the Three.js *Mesh* object of the occlusion models to an integer value lower than the virtual building [34, 35].

The source code of the proposed solution is available at: https://github.com/johankasperi/argon-osm-occlusion

4.3 Visual effect

The visual effect of the developed occlusion handler can be seen in fig. 5. Fig. 5a shows how the virtual building in the application looked when the occlusion handler was deactivated. The building

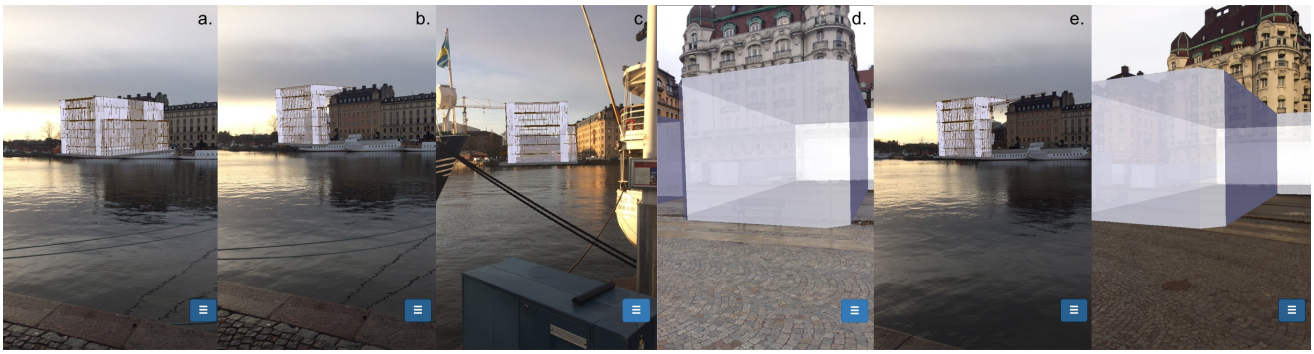

Figure 5: *Visual effect of the developed occlusion handler. a) Occlusion deactivated. b) Occlusion activated. c) Occlusion activated but no real buildings in line-of-sight. d) Occlusion 3D model aligned with real world building. e) Occlusion activated but position of virtual building incorrect. f) Occlusion 3D model not aligned with real building*

appears in front of a real building even though this building is supposed to be in the line-of-sight between the user and the virtual building. Compare this to fig. 5b where the occlusion handler is activated, the virtual building's vertices is pushed down the depth buffer by the transparent occlusion models, resulting in the virtual building being occluded along the edge of the building that is in the line-of-sight. Fig. 5c shows the visual effect when the occlusion handler is activated but there is no real building in the line-of-sight, resulting in no occlusion on the virtual building.

Fig. 5d shows a opaque occlusion model of a building in Stockholm, Sweden. The occlusion model aligns well with its real building, but as you can see the occlusion model is not as detailed as its building. Fig. 5d also shows that the height of the occlusion model does not match the real building. This particular building did not have any level data stored by OSM, making the occlusion algorithm fallback to creating the model with a height of 9 meter. Also occlusion models for buildings with level data stored by OSM did not match the height of their real buildings sometimes. Causes of this was the algorithms general assumption that one level equaled 3 meter and that the height of the roof was not included in the OSM data.

The application used a sensor-based tracking method, relying on sensor such as GPS, gyroscope and magnetometer to position the virtual content in the real world. Fig. 5e and fig. 5f shows the issues emerging using a sensor-based method. The sensors of the device in this study was from time to time inaccurate, which resulted in the virtual building as well as the occlusion models being positioned wrong. For example, fig. 5e shows the virtual building being occluded by the occlusion model, but since both the virtual building and the occlusion models has incorrect positions due to the sensors, the edge of the occlusion does not align with the real building at all.

4.4 Performance

All of the performance tests were conducted with four different input sets. Each set had a different bounding circle radius (r) as input parameters to the algorithm. These different radiuses retrieved different amount of buildings from the Overpass API due to the larger area of buildings to be included, resulting in different impact on the performance of the algorithm. The geolocation of the circle (L) were always set to a position in central Stockholm, Sweden (see tab. 1).

Table 1: *Radius (r) and location (L) input parameters to the occlusion algorithm and retrieved amount of buildings used for all of the performance tests*

Input set	Radius (r)	Location (L)	No. of buildings
1	100 m	59.3474572, 18.0737809	6
2	200 m	59.3474572, 18.0737809	35
3	500 m	59.3474572, 18.0737809	159
4	1000 m	59.3474572, 18.0737809	439

The first performance test was the average execution time measurement for creating the occlusion 3D models out of the geospatial data from Overpass API (see fig. 6). The algorithm ran for 100 iterations for each input set. The 100 iterations made the sample size for measuring the average execution time for each input set. This sample size made the result 95 % confident (using a t-distribution) and the confidence interval magnitude about 10 % of the average execution time for each input set (see fig. 6). For example, the confidence interval for input set 1 were 1.25 ms. The results show that as the number of buildings included in the algorithm increases, the execution time for creating all of the occlusion models increases (see fig. 6).

The average execution time for updating the position of the occlusion models in each render loop, due to update of the device position or orientation are presented in fig. 7. This average was measured during 5 minutes of regular usage of the prototype for each input set (see tab. 1). The results show that as the number of real world buildings included in the occlusion increases, the average execution time for updating all of the occlusion models also increases (see fig. 7).

The last performance test conducted were the FPS analysis. This test were also measured during 5 minutes of regular usage of the app for each input set. However, this test were first conducted with the occlusion handling function deactivated to get an comparison value. The results show that the occlusion handling function had

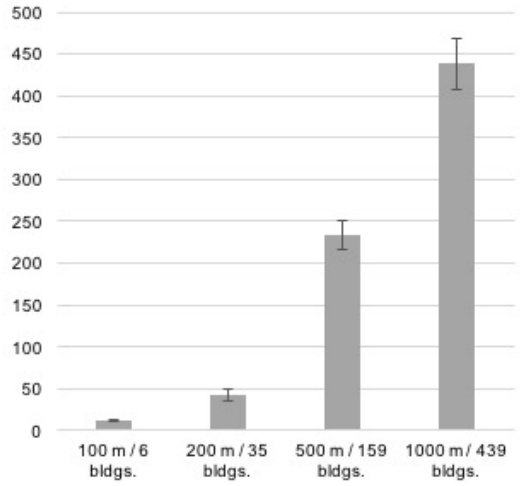

Figure 6: *Average execution time (ms) for creating models out of geospatial data for the four different input sets to the algorithm.*

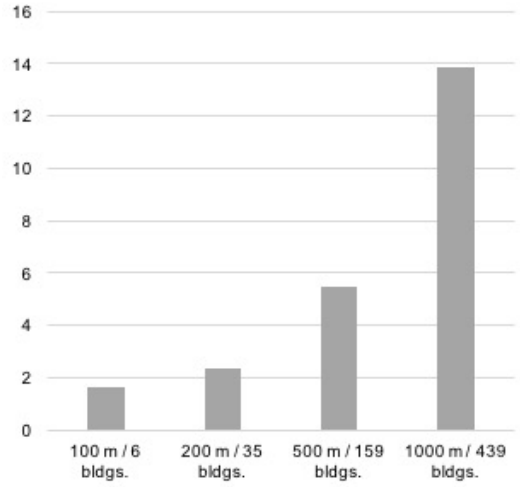

Figure 7: *Average execution time (ms) for updating the position of the occlusion models in each render loop.*

no impact on the prototype's FPS, no matter how many real world buildings that were included in the occlusion. The average FPS was 30 for input set 1-5 as well as for when the occlusion handling was deactivated.

4.5 User study

The 13 participants in the user study had an average age of 27.3 and median of 27. 10 of them were engineering students and the other 3 had non-engineering jobs. 6 of the participants answered "Strongly agre" or "Agree", 4 "Neutral" and 3 "Strongly disagree" or "Disagree" when asked if they have a lot experience from AR. The average on this statement was 3.23 and median 3. When asked if they had any experience from AR and occlusion handling neither

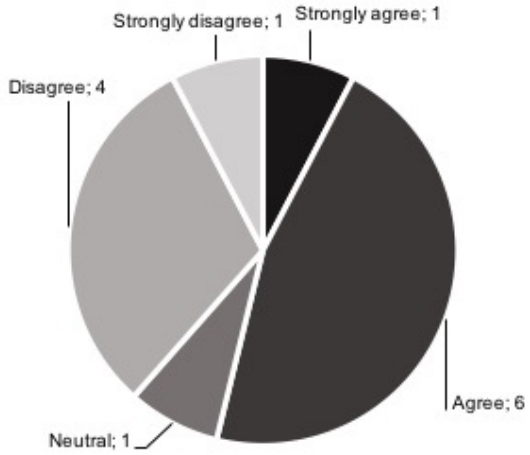

Figure 8: *Answers to the statement "The tracking affected my experience of the prototype".*

answered "Strongly agree" or "agree", 3 answered "neutral" and the rest answered "Strongly disagree" or "Disagree". This statement got an average of 2.38 and median of 2. The participants were then asked about their previous knowledge about the future building visualized in the prototype, and its placement in Stockholm. 3 participants agreed that they had knowledge about the position of this building but the majority strongly disagreed to this statement. The average on this statement was 2.15 and median 2.

The participants were then given a statement about the prototype in general, not focusing on the occlusion handling function. On the statement "My general impression of the prototype was good" 10 of the participants answered "Strongly agree" or "Agree", 2 answered "Neutral" and 1 "Disagree". This resulted in an average of 3.85 and median of 4.

The last statement before the statements about the occlusion handler was about the tracking. Since the sensor-based tracking had proved to be inaccurate sometimes the participants were asked if the tracking affected their experience of the application. The majority answered "Agree" on this statement (see fig. 8) and the average was 3.15 and median 4.

On the statement "Occlusion handling is important in applications with location based Augmented Reality" 6 participants answered "Agree" and 7 participants answered "Strongly agree". When asked if the occlusion handling in this prototype was satisfying, the majority agreed to this statement (see fig. 9) and the statement got an average of 3.69 and median of 4.

11 participants answered that they agreed that the prototype was better when the occlusion handler was activated compared to having it deactivated (see fig. 10). This statement got an average of 4.23 and median of 5.

The next topic in the user study regarded depth perception, one of the main goals for occlusion handling functions in AR. All participants except 1 agreed or strongly agreed that they got a better depth perception due to the occlusion handling function (see fig. 11), giving it an average of 4.38 and median of 4. They were also

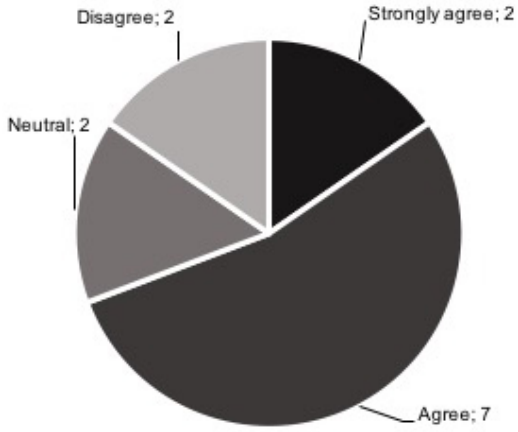

Figure 9: *Answers to the statement "The occlusion handling in this prototype was satisfying".*

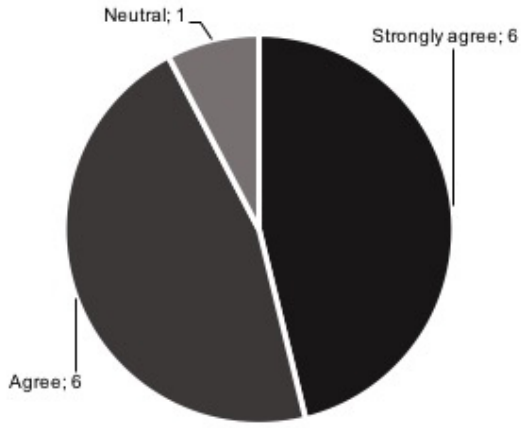

Figure 11: *Answers to the statement "The occlusion handling in this prototype gave me a better depth perception".*

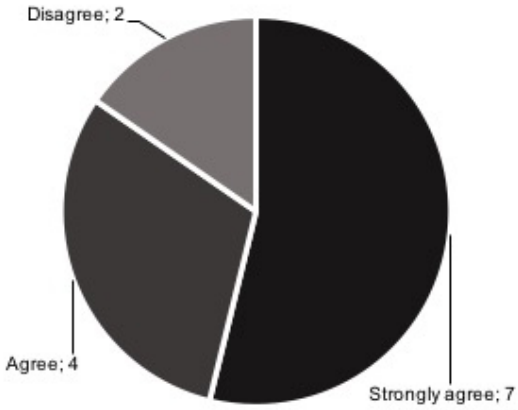

Figure 10: *Answers to the statement "The prototype was better with the occlusion handling activated compared to having it deactivated".*

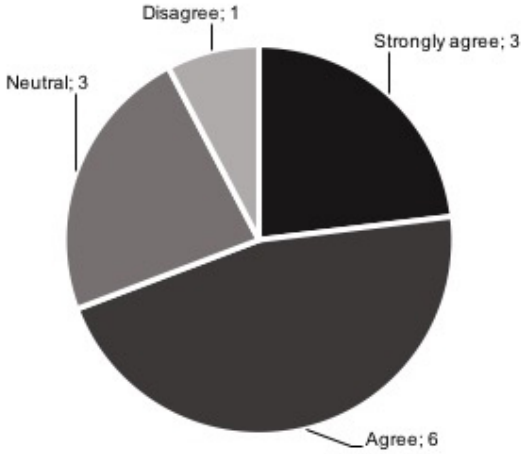

Figure 12: *Answers to the statement "The occlusion handling in this prototype gave me an experience that the virtual content was mixed/interacted with the real buildings".*

asked if they got a better depth perception with the occlusion handling activated compared to having it deactivated. This statement got 7 "Strongly agree" and 6 "Agree" answers, giving it an average of 4.54 and median of 5.

The last topic of the user study concerned the users experience of whether the virtual building was more mixed and interactive with the real buildings or not, which was another main goal for occlusion handlers. The participants were asked to answer the statement "The occlusion handling in this prototype gave me an experience that the virtual content was mixed/interacted with the real buildings". 9 of them agreed to this statement, 3 answered "Neutral" and 1 disagreed (see fig. 12). The average was 3.85 and mean 4. They also answered the statement "The occlusion handling in this prototype gave me a better experience that the virtual content was mixed/interacted with the real buildings compared to having it deactivated", in order to compare this topic to when they had the occlusion handler deactivated. 9 participants agreed and 4 participants strongly agreed to this statement, resulting in an average of 4.31 and median of 4.

5 DISCUSSION

In this section will the results of the study be discussed and analyzed in order to answer the research question. The results are also compared with previous similar attempts and the section finishes with proposing further research within the area. The purpose of this study was to develop and evaluate a dynamic occlusion handling function for outdoor urban AR that didn't require any special hardware more than a smartphone or a predefined static environment. These goals would be reached on the cost of not achieving

pixel perfect occlusion and that only real world buildings would be included.

The evaluation of the developed occlusion handler would then answer the research question:

> What are the results of occluding virtual 3D objects by real world buildings using geospatial data in an outdoor urban Augmented Reality environment?

Looking at the visual effect result (see fig. 5b) one can see that the virtual building are occluded along the roof and corner of the real world building. The same results were achieved from several different angles with different buildings in the line of sight between the device and the virtual building. Also the virtual building were fully visible when there was no real building in the line of sight (see fig. 5c), and not visible at all when the entire virtual building was behind a real building. Given these results the study's occlusion handler is functioning as expected, it simulates the effect of how it would have been if the virtual building was a real building placed at the same location. However, it produces some visual artefacts, for example the occlusion of the virtual building does not align with the edge of the real building perfectly. This means that the proposed solution is not handling occlusion in AR realistically. But the goal of the proposed solution was never to create pixel-perfect occlusion, it was to create an occlusion handler that was efficient enough. Meaning that it would fulfill the goals of occlusion handlers, to give the users a sense of depth, size and distance and the feeling that the virtual and real objects was mixed and interacted [9]. Looking at the answers from the user study of the questions regarding this it might seem so. 12 of 13 participants in the user study thought that their depth perception got better (see fig. 11 and 9) agreed that the virtual content felt more mixed with the real buildings (see fig. 12) when trying the prototype. But further user studies with more participants from a more heterogenous selection group is needed to confirm if the proposed occlusion handling function reached the goals of occlusion handlers.

The method used for occlusion handling in this study had two main issues, which also caused the visual artefacts. The first issue was that the geospatial data from OSM could not be transformed to detailed occlusion models. One of these details missing are those regarding the roof of the real world building. Looking at fig. 5d you can see that the roof of the occlusion model is completely flat, which was the case with all occlusion models. Obviously real world buildings have much more detailed roofs with slants, chimneys etc. OSM has data about these type of details for some buildings [36] that was not implemented in the solution of this study. The main reason for this was that few buildings in Stockholm had any roof shape data and that this data is not a reflection of the reality. However, future studies could explore using the roof shape data from OSM. Compared to the roof, the occlusion behind corners and walls of real buildings was better. The cause for this was that data of the nodes, or corners, from OSM was more accurate and aligned with the corners of the real buildings.

Another detail that was missing in the geospatial data was an accurate height of the real buildings. First of all, the `building:levels` data, that was used for setting the height of the occlusion models, was missing on a majority of the real buildings tested in this study. So for all of these buildings the algorithm used the default level amount (see 4.2). This resulted in that all of the occlusion models for buildings that lacked level data had a height mismatch to their real building. Secondly, using the arbitrary height of 3 meters per level also caused a height mismatch since the height per level for a building is fairly unique.

Since the building height and roof details was troublesome and the building corners and walls were accurate the proposed solution could be altered when taking these findings into consideration. This alternative algorithm could create the occlusion models with infinite height instead of trying to match the height with the real buildings. Obviously would all virtual content positioned above buildings be occluded, but that alternative might be better than trying to incorrectly, due to incorrect height or roof data, occlude virtual content along the roof of the real buildings. This alternative would utilize the strength of the current state of the OSM database, the nodes or corners, and skip the weaknesses, the building height and roof details.

The second issue of the prototype causing a lot of the visual artefacts was the tracking method. As mentioned earlier, the prototype used sensor-based tracking for placing the virtual content in the real world. The results show that this was a mistake, a lot of the time were the virtual building and the occlusion models incorrectly positioned (see fig 5e and 5f). Previous studies has proven that it is important for the occlusion models to align with the real objects and since the tracking in this study struggled with that, all results have been affected. For example, the occlusion of the building in fig. 5b might align with the edge of the building better if the positioning of the content was more precise. This issue was also confirmed by the user study, 7 participants answered that the sensor-based tracking affected their experience (see fig. 8. Even though this study was not focusing on or evaluating any tracking technology it is obvious that an alternative tracking technology should have been used.

Comparing the proposed solution to previous studies the visual effects have some differences. All previous studies have tried to achieve pixel-perfect occlusion, with either a depth- or model-based method, and many of them have somewhat succeeded. But all of these have required either special hardware or a predefined static environment, making it impossible to easily implement on AR applications for smartphones. As mentioned earlier, this study has done the opposite. The visual effect of the occlusion is far from perfect, but it can be implemented on a regular non depth-sensing smartphone. The results show that the main goal of an occlusion handler does not need pixel-perfect occlusion. It could instead be simplicity. Even though it had some errors, the occlusion handler was appreciated by the participants of the user study (see fig. 9 and 10) and as mentioned earlier it could achieve the ultimate occlusion handler goals. Also the visual artefacts due to poor occlusion models could be compared with the depth sensing noise experienced in previous studies. The proposed solution was also light in computation, the impact on the frame rate of the WebGL renderer was insignificant and instantiating all the occlusion models upon application launch were fast (see fig. 6). Even though it is difficult to compare the computation power needed for the proposed solution to previous studies, since previous studies has been more advanced, this at least means that the solution's negative aspect of simulating

occlusion poorly is countered with being lightweight. But it is important to note that the proposed solution only included buildings in the occlusion. Previous studies, especially the depth-based, have included all arbitrary objects in the current AR environment. This was also mentioned by the participants of the user study, several of them asked if the virtual building was occluded behind cars, signs and similar.

As mentioned in the theory section, the challenge for occlusion handlers is to construct the geometric model of the occluding real objects in real-time as well as estimate the pose and position of the constructed objects as close as possible to the actual real objects. The same challenge exists for all type of functions that tries to simulate any physical interaction between the real objects and the virtual content, such as shadowing or collisions. It needs to know the state and form of the current environment. This is also what the solution in this study does, it uses geospatial data to recreate the models of the buildings in the current environment. This means that it is possible to use the same approach for all the other physical simulations.

That the results show that using geospatial data, without perfect reconstruction of the environment, can create efficient enough physical simulations is positive for the future development of AR. Mainly because smartphones are currently the main driver for making AR available for everybody, making it important to find alternate methods that can run on them. The approach of using geospatial data could be one of these alternatives. However, all these results might only be applicable until AR hardware such as Microsoft Hololens and Google Tango has taken over a larger part of the market. Both of them, and probably many not yet invented AR devices, are equipped with depth sensors. And since depth sensors constructs the current environment in real-time very accurately, all of the alternative methods for non depth sensing devices might become obsolete. But until then this study shows that geospatial data could play a role in making the AR field closer to its ultimate goal:

> *"The ultimate goal will be to generate virtual objects that are so realistic that they are virtually indistinguishable from the real environment"* [1]

However, due to methodological reasons, some of the conclusions in this study cannot be confirmed. The two main issues, that have been described above, are the selection of tracking method and the selection group of the user study participants. If a more accurate tracking method would have been used the results might have been different, since previous studies have confirmed the importance of perfect alignment of the occlusion models. The background of the user study participants also makes it difficult to confirm if the proposed solution reaches the goal for occlusion handlers. A majority of them had previous AR experience (see 4.5) and had a computer science background, so they probably had more knowledge about the challenges and implications of the AR medium in general. Different results may have been gathered if the user study's selection group was more heterogenous.

Future studies within the area of this study should first of all try to fix the issues with the method described above in order to confirm the conclusions. First, this means using the same geospatial approach but with an alternative tracking method. Probably an hybrid-based tracking method which utilize the strengths of the smartphone's sensors and the vision through its camera. Secondly, further studies should also conduct a more thorough user study with more participants with more diverse backgrounds. There are also some improvements that could be made to the proposed occlusion algorithm which invites for interesting future studies. One of them is to include the roof shape data from OSM in order to create more detailed occlusion models. Another is to alter the algorithm to create occlusion models with infinite height, which as described earlier utilizes the strength of the current state of the OSM database. Other interesting future studies could be to use the same geospatial approach but for other physical simulations, for example shadowing or collisions, and see if similar conclusions as in this study can be drawn. Lastly it would be interesting to explore using different types of geospatial data, from perhaps other sources, to be able to one day fully reconstruct the current environment of all real objects in the scene with the geospatial data approach.

6 CONCLUSION

This study has explored using geospatial data to construct the geometric model of buildings in a dynamic outdoor environment. This model have been used to simulate occlusion between virtual content and real buildings in AR. The goal has been to create and evaluate a simple but yet efficient enough occlusion effect that can run on a regular smartphone in any outdoor environment. The conclusions are that occlusion handlers using geospatial data can be efficient enough, since the function made the virtual content partially or fully occluded when a real building was in the line-of-sight. However, the study also shows that the geospatial data from OSM does not support creation of models that have the same level of details as the real buildings. For example features such as chimneys or slanted roofs was poorly represented by OSM and therefore not included in the occlusion. It was also found that the geospatial data from OSM does not provide accurate height data, if any, about the real buildings which resulted in a notable height differences between the occlusion models and the real buildings.

The study has also shown that occlusion handlers for AR do not need to be pixel-perfect, which have been mostly researched in previous studies, to fulfill its ultimate goal. A majority of the participants in the conducted user study thought that their AR experience improved when the proposed solution was activated. This could mean that simpler methods to create the geometric model of the current AR environment could be used for mobile AR until depth-sensing devices gets more widely used. But any definite conclusions about this cannot be drawn due to issues with the sensor-based tracking. It produced a significant noise in the positioning of the occlusion models and the virtual content which affected the user experience.

7 ACKNOWLEDGEMENT

I would like to thank my supervisor from KTH, Malin Picha Edwardsson, for the help and support while conducting this study. Also, thanks to Bonnier News and Blair MacIntyre for assisting me during the development stage of the prototype. Finally, a huge thanks to my partner in crime, Daniel Lindström.

REFERENCES

[1] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[2] Dean Takahashi. The walabotdiy lets remodelers see through walls like superman. http://venturebeat.com/2016/08/16/the-walabotdiy-lets-remodelers-see-through-walls-like-superman/, 2016. Accessed: 2017-01-31.

[3] Paul Ridden. Ikea catalog uses augmented reality to give a virtual preview of furniture in a room. http://newatlas.com/ikea-augmented-reality-catalog-app/28703/, 2013. Accessed: 2017-01-27.

[4] Thomas P Caudell and David W Mizell. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, volume 2, pages 659–669. IEEE, 1992.

[5] Ronald T Azuma. The most important challenge facing augmented reality. *PRESENCE: Teleoperators and Virtual Environments*, (00), 2016.

[6] Tobias Höllerer and Steve Feiner. Mobile augmented reality. *Telegeoinformatics: Location-Based Computing and Services. Taylor and Francis Books Ltd., London, UK*, 21, 2004.

[7] David E Breen, Ross T Whitaker, Eric Rose, and Mihran Tuceryan. Interactive occlusion and automatic object placement for augmented reality. In *Computer Graphics Forum*, volume 15, pages 11–22. Wiley Online Library, 1996.

[8] P Fortin and Patrick Hebert. Handling occlusions in real-time augmented reality: dealing with movable real and virtual objects. In *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, pages 54–54. IEEE, 2006.

[9] Manisah Mohd Shah, Haslina Arshad, and Riza Sulaiman. Occlusion in augmented reality. In *Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on*, volume 2, pages 372–378. IEEE, 2012.

[10] Chao Du, Yen-Lin Chen, Mao Ye, and Liu Ren. Edge snapping-based depth enhancement for dynamic occlusion handling in augmented reality. In *Mixed and Augmented Reality (ISMAR), 2016 IEEE International Symposium on*, pages 54–62. IEEE, 2016.

[11] Anton Fuhrmann, Gerd Hesina, François Faure, and Michael Gervautz. Occlusion in collaborative augmented environments. *Computers & Graphics*, 23(6):809–819, 1999.

[12] Kiem Ching Ong, Hung Chuan Teh, and Tiow Seng Tan. Resolving occlusion in image sequence made easy. *The Visual Computer*, 14(4):153–165, 1998.

[13] Vincent Lepetit and M-O Berger. A semi-automatic method for resolving occlusion in augmented reality. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 225–230. IEEE, 2000.

[14] Yuan Tian, Tao Guan, and Cheng Wang. Real-time occlusion handling in augmented reality based on an object tracking approach. *Sensors*, 10(4):2885–2900, 2010.

[15] Yuan Tian, Yan Long, Dan Xia, Huang Yao, and Jincheng Zhang. Handling occlusions in augmented reality based on 3d reconstruction method. *Neurocomputing*, 156:96–104, 2015.

[16] Matthias M Wloka and Brian G Anderson. Resolving occlusion in augmented reality. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 5–12. ACM, 1995.

[17] Jochen Schmidt, Heinrich Niemann, and Sebastian Vogt. Dense disparity maps in real-time with an application to augmented reality. In *Applications of Computer Vision, 2002.(WACV 2002). Proceedings. Sixth IEEE Workshop on*, pages 225–230. IEEE, 2002.

[18] Jiejie Zhu, Zhigeng Pan, Chao Sun, and Wenzhi Chen. Handling occlusions in video-based augmented reality using depth information. *Computer Animation and Virtual Worlds*, 21(5):509–521, 2010.

[19] Jan Fischer, Benjamin Huhle, and Andreas Schilling. Using time-of-flight range data for occlusion handling in augmented reality. In *IPT/EGVE*, pages 109–116, 2007.

[20] Amir H Behzadan and Vineet R Kamat. Scalable algorithm for resolving incorrect occlusion in dynamic augmented reality engineering environments. *Computer-Aided Civil and Infrastructure Engineering*, 25(1):3–19, 2010.

[21] Dan Seifert. The first phone with google tango is a 6.4-inch monster from lenovo. http://www.theverge.com/2016/6/9/11882514/lenovo-phab-2-plus-pro-tango-google-specs-announce-smartphone, 2016. Accessed: 2017-01-26.

[22] Adi Robertson. Anyone (with $3,000) can now buy microsoft hololens. http://www.theverge.com/2016/8/2/12358554/microsoft-hololens-augmented-reality-opens-developer-sales, 2016. Accessed: 2017-01-26.

[23] Vlasios Kasapakis and Damianos Gavalas. Determining field of view in outdoors augmented reality applications. In *European Conference on Ambient Intelligence*, pages 344–348. Springer, 2015.

[24] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE Computer Society, 2008.

[25] Arnis Cirulis and Kristaps Brigis Brigmanis. 3d outdoor augmented reality for architecture and urban planning. *Procedia Computer Science*, 25:71–79, 2013.

[26] Open Geospatial Consortium. Faqs - ogc's purpose and structure. http://www.opengeospatial.org/ogc/faq, 2017. Accessed: 2017-01-25.

[27] OpenStreetMap. About openstreetmap. http://wiki.openstreetmap.org/wiki/About_OpenStreetMap, 2016. Accessed: 2017-02-01.

[28] OpenStreetMap. Overpass api. http://wiki.openstreetmap.org/wiki/Overpass_API, 2016. Accessed: 2017-02-01.

[29] Blair MacIntyre, Alex Hill, Hafez Rouzati, Maribeth Gandy, and Brian Davidson. The argon ar web browser and standards-based ar application environment. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 65–74. IEEE, 2011.

[30] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[31] Three.js. Three.js. https://github.com/mrdoob/three.js/, 2017. Accessed: 2017-01-31.

[32] GeoJSON. Geojson. http://geojson.org/, 2016. Accessed: 2016-12-20.

[33] Three.js. Extrudegeometry. https://threejs.org/docs/#Reference/Geometries/ExtrudeGeometry, 2016. Accessed: 2016-12-20.

[34] Three.js. Material. https://threejs.org/docs/api/materials/Material.html, 2016. Accessed: 2016-12-20.

[35] Three.js. Mesh. https://threejs.org/docs/api/objects/Mesh.html, 2016. Accessed: 2016-12-20.

[36] OpenStreetMap. Simple 3d buildings. http://wiki.openstreetmap.org/wiki/Simple_3D_buildings, 2017. Accessed: 2016-12-20.