Geo-Referenced Occlusion Models for Mixed Reality Applications using the Microsoft HoloLens

Christoph Praschl[a] and Oliver Krauss[b]
Research Group Advanced Information Systems and Technology (AIST), University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria

Keywords: Mixed Reality, Augmented Reality, Geo-Referenced Models, Occlusion, CityJson, CityGML, Microsoft HoloLens.

Abstract: Emergency responders or task forces can benefit from outdoor Mixed Reality (MR) trainings, as they allow more realistic and affordable simulations of real-world emergencies. Utilizing MR devices for outdoor situations requires knowledge of real-world objects in the training area, enabling the realistic immersion of both, the real, as well as the virtual world, based on visual occlusions. Due to spatial limitations of state-of-the-art MR devices recognizing distant real-world items, we present an approach for sharing geo-referenced 3D geometries across multiple devices utilizing the CityJSON format for occlusion purposes in the context of geospatial MR visualization. Our results show that the presented methodology allows accurate conversion of occlusion models to geo-referenced representations based on a quantitative evaluation with an average error according to the vertices' position from 1.30E-06 to 2.79E-04 (sub-millimeter error) using a normalized sum of squared errors metric. In the future, we plan to also incorporate 3D reconstructions from smartphones and drones to increase the number of supported devices for creating geo-referenced occlusion models.

1 INTRODUCTION

The fusion of the real and virtual world is one of the most crucial aspects, in the context of (outdoor) Augmented (AR) and Mixed Reality (MR) applications. Especially, the visual occlusion of virtual objects based on the spatial information about the real world as shown in Figure 1 has a huge impact on users to completely immerse themselves into the virtual world. For this reason, the present work deals with methods for exchanging geo-referenced, spatial information of real world objects between multiple AR and MR devices to create occlusion models.

Due to the constant development in the field of AR and MR like head mounted displays (HMD) as the Microsoft HoloLens 2 (Ungureanu et al., 2020) or the Magic Leap One (Swaminathan, 2019) more and more applications in this area became feasible in the recent years. Especially, when it comes to scopes like guided working or trainings in indoor scenarios many workflows have been digitized as product maintenance and assembly (Kaplan et al., 2020; Gavish et al., 2015; Westerfield et al., 2014; De Crescenzio et al., 2011) or medical trainings (Ingrassia et al., 2020; Vergel et al., 2020; Thøgersen et al., 2020; McKnight et al., 2020). Regardless of that, the utilization of HMDs for MR or AR applications remains largely unaffected for outdoor scenarios. While nowadays most smartphones are equipped with cellular as well as GPS modules and for this reason can be used independently in outdoor scenarios, HMDs as the before mentioned ones don't have these technical features. Thus applications are restricted to local coordinate systems and can't work with global information as geo-referenced objects. In the following, this

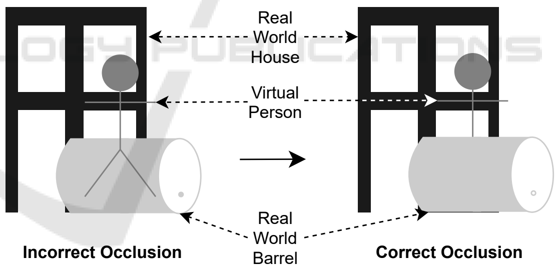

Figure 1: Occlusion of a virtual person based on the spatial information of surrounding real world objects, a barrel and a house. While the barrel is spatially in front of the virtual person, the house is in the background.

[a] https://orcid.org/0000-0002-9711-4818
[b] https://orcid.org/0000-0002-8136-2606

Praschl, C. and Krauss, O.
Geo-Referenced Occlusion Models for Mixed Reality Applications using the Microsoft HoloLens.
DOI: 10.5220/0010775200003124
In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 3: IVAPP*, pages 113-122
ISBN: 978-989-758-555-5; ISSN: 2184-4321

also means that information about for example spatial features can hardly be exchange within a multi-user setting and every device has to create and update its own occlusion model of the surrounding environment.

We already tackled the problem of the missing link to the global position and orientation in a previous publication (Praschl et al., 2020) by extending the Microsoft HoloLens with an external GPS, compass and cellular module, which allow us to synchronize the local coordinate systems of multiple devices within one global system. To the moment, this extension was only used to synchronize the position of multiple users and to trigger events based on these positions in a shared virtual world in the context of outdoor training simulations of emergency response or disaster operations. Based on the results of this previous work, we present advanced methodologies to exchange geo-referenced information about real world objects to create occlusion models as the next step to a spatially unlimited outdoor training system.

2 PROBLEM STATEMENT

While state-of-the-art AR and MR devices as the Microsoft HoloLens 2 are capable of creating occlusion models of their surrounding environment in real-time for indoor applications, those devices face their limitations in outdoor scenarios, because of narrow working distances up to 3.5 meters (Hübner et al., 2020) as shown in Figure 2. Next to the working distances, current head mounted displays in the context of MR are also not capable of associating their local coordinate system with a global position and orientation. Therefore such devices face additional limitations for (outdoor) usages in terms of multi-user purposes, since spatial information can hardly be exchanged or merged. This problem can partially be solved with Microsoft's MR technology by exchanging the room model (Van Schaik, 2017) or using "Spatial Anchors" (Turner and Coulter, 2019). While the first approach works well especially for indoor applications because of many unique and recognizable spatial features, it does not work that well in outdoor scenarios, where such features are often out of range of the limited working area. One advantage of Microsoft's room models is that the underlying mesh can be shared across multiple devices, as a trade-off, it will only be updated within the scan range of the current device. Scans of other participants as well as distant, moving objects will not affect the occlusion model. For this reason, virtual objects may be covered wrongly. The second concept, the so-called "Spatial Anchors", allows associating local, visual features

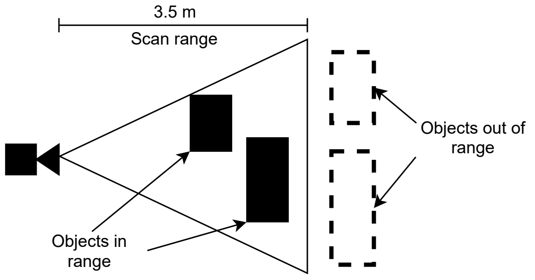

Figure 2: Schematic visualization of the scan range with real world objects in range and such ones that are not included into the environment model.

with a decoupled coordinate system and like this enables linking the relative position of other virtual objects to those anchors. Since "Spatial Anchors" can be synchronized across multiple devices, it also allows exchanging associated child objects. One drawback of this technology is again a spatial limitation of 3 meters to avoid positional errors of referenced virtual objects. In addition to this problem, "Spatial Anchors" are dependent on the associated visual features. This means, that big visual changes of the environment, will disable the anchor and for this the possibility of synchronized objects.

Geometry file formats as OBJ (Chen, 2003) or COLLADA (Khronos Group Inc., 2008) are commonly used to exchange digital 3D models amongst various applications, but are not intended to represent global information. For the purpose of geo-referenced models, other file formats as GML (Portele, 2007), CityGML (Gröger and Plümer, 2012), CityJSON (Ledoux et al., 2019a) or KML (Nolan and Lang, 2014) are available. These file formats are using global coordinate systems such as the `EPSG::4326` or the `WGS::84` system of the Global Positioning System (GPS) (Kaplan and Hegarty, 2017) instead of a three-dimensional Cartesian system as commonly used in 3D engines as Unity (Haas, 2014) or by MR devices. The difference in these systems compared to a Cartesian one is the representation of coordinates based on angular measurements as longitude and latitude, next to the altitude information to map the global position on earth, as shown in Figure 3. Because of that, the utilization of geo-referenced models within a Cartesian application requires a projection of the coordinate system based on e.g. a reference point. Due to the missing link to the global position of devices such as the Microsoft HoloLens 2, this reference point is not available. To overcome this problem, the reference can be statically defined on system start, for example via user input with the risk for errors or dynamically by incorporating an external GPS module.

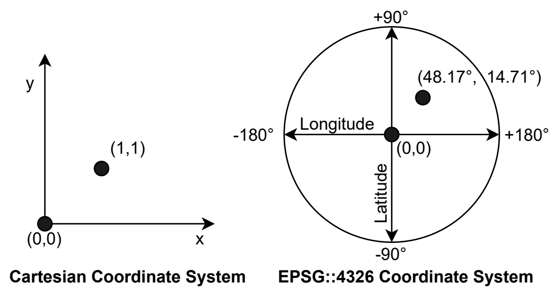

Figure 3: Comparison of a classic Cartesian coordinate system and the `EPSG::4326` system.

3 STATE OF THE ART

The creation of virtual clones of real world objects is strongly connected with the field of 3D reconstruction. Algorithms from the field of Computer Vision and Robotics as the Visual-SLAM algorithm (Taketomi et al., 2017) are well established and allow to map the real world characteristics based on monocular images and videos, but also from depth information in the form of point clouds. While the first type of information can be obtained with monocular RGB cameras, the depth data can be created using specialized depth cameras (Izadi et al., 2011), as well as Time-of-Flight (Nguyen et al., 2018) or LiDAR sensors (Tachella et al., 2019). This heterogeneous information can be utilized by different reconstruction applications, as Bentley's ContextCapture, RealityCapture or Agisoft's Metashape, which allow combining these sources of data and also incorporate additional meta information as GPS positions to create geo-referenced models (Kingsland, 2020).

Next to such on-premise applications, there are also specialized 3D scanning devices (Javaid et al., 2021), but also AR and MR devices that are capable of reconstructing the surrounding environment. While 3D scanners are used to create detailed digital twins of partially distant object surfaces in the context of e.g. 3D printing (Haleem et al., 2020) or land survey (Wu, 2021), those devices sometimes also support geo-referenced scans (Heinz et al., 2015). In contrast to that, object reconstruction methodologies in AR and MR devices are commonly used for less advanced purposes like the determination of comparatively rough occlusion models in the near space and for this don't require such a high level of detail and also don't consider global positions. Due to spatial limitations, such occlusion models are also updated one by one within the available scan range and in the case of the Microsoft HoloLens only for the device itself. Thus, the environment outside the range is completely unknown or may be outdated since a previous scan. To incorporate occlusion models of multiple devices, they have to be exported and imported. When importing such a model, devices such as the Microsoft HoloLens compare the model with the surrounding environment and try to find its position within it and update the local coordinate system based on this knowledge without any global reference. This process is designed to make use of only one single occlusion model, and does not allow supplementing additional geo-referenced information or to adapt individual parts.

4 METHODOLOGY

To overcome the missing link of three-dimensional occlusion models in the context of AR or MR applications, we propose the integration of geo-referenced geometry files in the form of the CityJSON format. This file format is based on the JavaScript Object Notation (JSON) (Bray et al., 2014) and is used for 3D models within a freely definable coordinate system. It is focusing on city models to create light-weight digital representations of e.g. buildings or bridges, but also supports any generic object. Since CityJSON is a subset of the standardized CityGML data model and for this purpose supports bidirectional conversions between both file formats, it can be widely used in multiple applications. In addition to that, CityJSON has less storage requirements as the Extensible Markup Language (XML) (Bray et al., 2000) based CityGML format due to the lower overhead of JSON compared to XML. For this reason, it is more suitable for exchanging information (Ledoux et al., 2019b; Zunke and D'Souza, 2014). Listing 1 shows an excerpt of a geo-referenced object utilizing the `EPSG::4326` coordinate system.

Listing 1: CityJSON sample of a geo-referenced object with a truncated vertice and bounadries list.

```
{
 "type": "CityJSON",
 "version": "1.0",
 "metadata": {
  "referenceSystem": "EPSG::4326",
  "geographicalExtent": [...],
  "presentLoDs": { "1.0": 1 }
 },
 "CityObjects": {
  "Testcube": {
   "type": "GenericCityObject",
   "geometry": [{
     "type": "Solid", "lod": 1,
     "boundaries":[[[[0,1,2]]],...]
```

```
    }]
  }
 },
 "vertices": [[48.3, 14.2, 5], ...]
}
```

The utilization of CityJSON in the context of local, Cartesian coordinate systems as used by most AR and MR devices as the Microsoft HoloLens, requires the projection of the coordinates. Therefore, the proposed methodology requires at least one known reference point in the local coordinate systems with associated global information, as well as the knowledge about the orientation offset α around the applicate axis between both coordinate systems. To make the system more tolerant of input mistakes, we suggest the utilization of a GPS module, that is used to determine the global position of the used device instead of manual inputs. Knowing the global referenced position with a radiant based latitude ϕ_1 and longitude λ_1 allows to calculate the local distance δ relative to the earth radius r with 6378137 meters between the device d and any local point p of an arbitrary mesh, as well as the bearing angle θ. This enables the conversion between the coordinate systems with ϕ_2 and λ_2 as global counterparts of the local system. The opposite conversion requires the Haversine formula to calculate the global distance δ_2 using the interim calculation a and the bearing θ_2 between the geo-referenced device position and any global coordinate as notated in Equation 1 to 7 (Veness, 2019).

$$\delta = \sqrt{(p_x - d_x)^2 + (p_y - d_y)^2} / r \tag{1}$$

$$a = sin^2((\phi_2 - \phi_1)/2) + cos(\phi_1) \cdot cos(\phi_2) \cdot sin^2((\lambda_2 - \lambda_1)/2) \tag{2}$$

$$\delta_2 = r \cdot 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \tag{3}$$

$$\theta = atan2(p_y - d_y, p_x - d_x) + \alpha \tag{4}$$

$$\theta_2 = atan2(sin(\lambda_2 - \lambda_1) \cdot cos(\phi_2), cos(\phi_1) \cdot sin(\phi_2) - sin(\phi_1) \cdot cos(\phi_2) \cdot cos(\lambda_2 - \lambda_1)) - \alpha \tag{5}$$

$$\phi_2 = asin(sin(\phi_1) \cdot cos(\delta) + cos(\phi_1) \cdot sin(\delta) \cdot cos(\theta)) \tag{6}$$

$$\lambda_2 = \lambda_1 + atan2(sin(\theta) \cdot sin(\delta) \cdot cos(\phi_1), cos(\delta) - sin(\phi_1) \cdot sin(\phi_2)) \tag{7}$$

5 IMPLEMENTATION

The implementation of the proposed system is based on a client-server architecture as shown in Figure 4, with a Python (Van Rossum and Drake Jr, 1995) based server for persisting as well as exchanging geo-referenced models using a RESTful (Fielding, 2000) interface and multiple AR/MR clients utilizing these models for occlusion purposes. These clients are created with the Unity Game Engine for the Microsoft HoloLens 2 as target platform. The basic concept of creating geo-referenced models is independent of the target platform in general, but is intended to import and export occlusion models created with Microsoft's Mixed Reality Toolkit (Microsoft, 2021) and for this requires a compatible device.

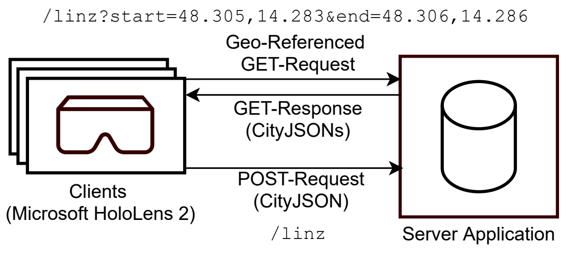
/linz?start=48.305,14.283&end=48.306,14.286
Geo-Referenced GET-Request
GET-Response (CityJSONs)
POST-Request (CityJSON)
/linz
Clients (Microsoft HoloLens 2)
Server Application

Figure 4: The proposed system architecture with multiple clients based on the Microsoft HoloLens 2 and a server application for exchanging geo-referenced models.

5.1 Client

The client application is based on the Unity Engine in the version 2019.4.19f1. It is used to (I) request geo-referenced models from the server application, (II) to use those models for occlusion purposes, and to (III) convert arbitrary `Mesh` objects to CityJSON representations, which are (IV) again forwarded to the server. For this reason, CityJSON's domain model has to be converted to Unity's object domain model and vice versa. The mapping between these domain models is shown in Figure 5, highlighting the related classes and properties of both worlds. Based on this class diagram, the fundamental differences are visible, starting with single floating point precision coordinates in Unity's `Vector3`, compared to the double-precision based representation for geo-referenced coordinates using `Position`. Single-floating point precision is not sufficient for representing coordinates with sub-meter accuracy within longitude, latitude and altitude based systems as `EPSG::4326` or `WGS::84` (Reddy et al., 2000). Note that CityJSON is not limited to these coordinate reference systems, but this simplification is used for the client side conversion process. Among other differences, it has also to be stated that Unity is limited to triangular polygons, represented as a sequence of indices within the `triangles` property of the `Mesh` class, while CityJSON supports any type of polygon using its `Face` representation. This characteristic requires an additional triangulation process, when translating between both worlds. The conversion of the two coordinate representations with `Vector3` and `Position` is shown in Listing 2.

Listing 2: The methods for converting between local and global coordinates in pseudo code based on the methodology presented in Section 4.

```
localDist(px, py, dx, dy){
  x = pow(px - dx)
  y = pow(py - dy)
  return square(x + y) / 6378137
}

globalDist(lat1, lon1, lat2, lon2){
 a = sin2((lat2 - lat1)/2) +
      cos(lat1) * cos(lat2) *
      sin2((lon2 - lon1) / 2)
 return 6378137 * 2 *
         atan2(square(a),
               square(1-a))
}

localBearing(px, py, dx, dy, a){
 return atan2(py - dy, px - dx) + a
}

globalBearing(lat1, lon1, lat2,
   lon2, a){
 return atan2(sin(lon2 - lon1) *
     cos(lat2), cos(lat1) *
     sin(lat2) - sin(lat1) *
     cos(lat2) *
     cos(lon2 - lon1)) - a
}
distantLat(lat1, dist, bearing){
 return asin(sin(lat1)*cos(dist) +
    cos(lat1) * sin(dist) * cos(
    bearing))
}

distantLon(lat1, lon1, lat2, dist,
   bearing){
 return lon1 + atan2(sin(bearing) *
   sin(local_distance) *
   cos(lat1), cos(dist) -
   sin(lat1) * sin(lat2))
}


Vector3 toLocal(Vector3 start,
   Position reference, Position
   toConvert, alpha){
 d = globalDistance(reference.lat,
    reference.lng, toConvert.lat,
    toConvert.lng)
 b = globalBearing(reference.lat,
    reference.lng, toConvert.lat,
    toConvert.lng, alpha)
 return start + b * d
}

Position toGlobal(Vector3 start,
   Position reference, Vector3
   toConvert, alpha){
 d = localDist(start.x, start.y,
    toConvert.x, toConvert.y)
 b = localBearing(start.x, start.y,
     toConvert.x, toConvert.y,
    alpha)
 lat = distantLat(reference.lat, d)
 lon = distantLon(reference.lat,
    reference.lng, lat, d, b)
 return Position(lat, lon,
    toConvert.z - start.z +
    reference.alt)
}
```

The proposed system is dedicated for occlusion models and for this reason doesn't consider textures. Instead of that, Microsoft's `MRTK_Occlusion` material is used. Although, the system can be used for converting any `Mesh` object to a CityJSON representation, it is primarily intended for exchanging geo-referenced environmental models. To access such a model in the context of Microsoft MR devices, the `IMixedRealityDataProviderAccess` service can be utilized to retrieve the mesh of the current spatial environment. This mesh can be converted with the proposed methodology to a CityJSON representation, which in turn can be sent to the server and like that exchanged with other system participants. The `IMixedRealityDataProviderAccess` service allows to access the environment model using one or multiple coherent meshes. These meshes are separated based on a maximal number of vertices and do not consider real world objects, thus parts of an object may be part of the first mesh, while the remaining parts are in another mesh. Due to that, the semantic meaning of a sub-mesh is hardly identifiable, and so CityJSON's `GenericCityObject` type should be preferred in the conversion process, since it does not limit the mesh to a specific semantic depiction. In addition to that, there is also no information about the level of detail (LoD) so the default value of 1 should be used. To extract individual objects from the environment model, we propose a bounding box based approach. Like this, a user can define the region of interest, which allows filtering for suitable faces of the environment's mesh(es) and allows exchanging individual sub-meshes.

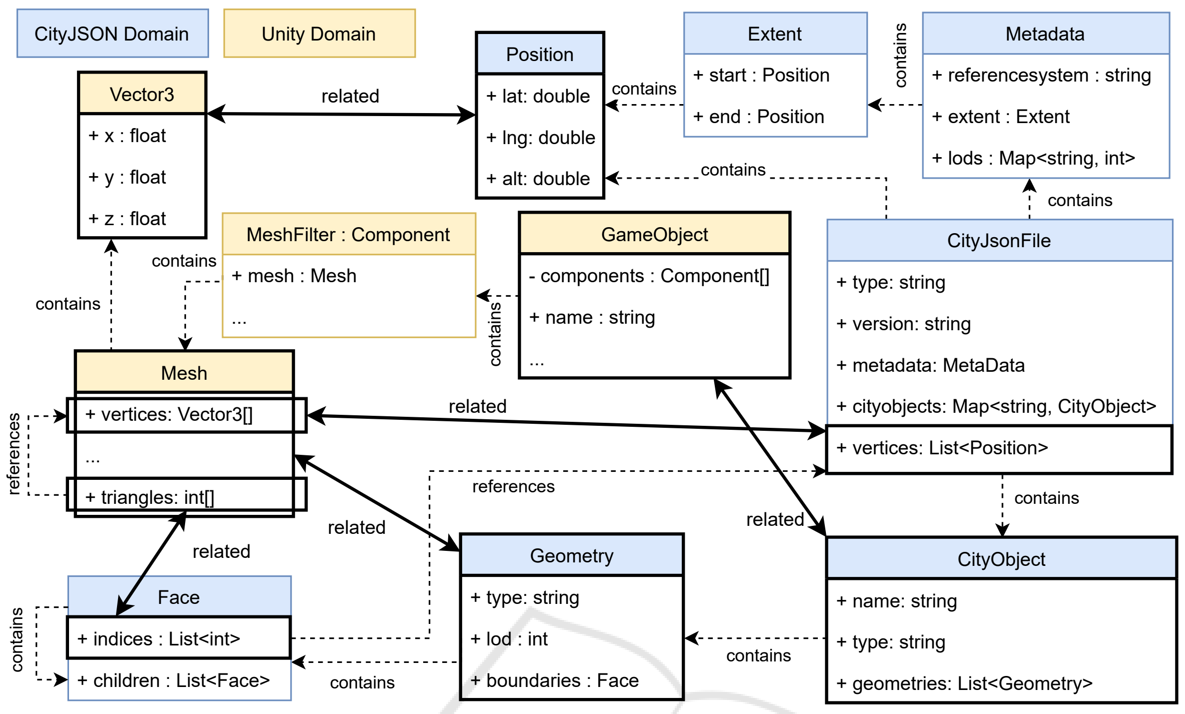

Figure 5: Domain model comparison between CityJSON and Unity in the context of representing object meshes, highlighting the semantically related classes and properties.

5.2 Server

Next to the client application, we propose a server implementation for (I) exchanging and (II) persisting the created CityJSON models. Our reference implementation is based on Python 3.7.9 and utilizes two main frameworks. On the one hand, we are using PyProj (Pyproj4, 2014) in the version 3.1.0 for handling and converting global coordinates, and on the other hand Flask (Pallets, 2010) in the version 1.1.4 to realize the server's RESTful interface. Based on these frameworks, the server handles requests and for this allows to exchange CityJSON model between multiple devices and to manage multiple CityJSON datasets. In the context of accessing models, the server uses two parameters to identify suitable files. First, it uses the dataset name to filter the resulting objects. Since this type of filtering is not always intended, a specialized "all" dataset can be used, which ignores the dataset boundaries. Next to this parameter, a start as well as an end coordinate should be provided to define the region of interest (ROI) in the form of a bounding box. Based on this ROI the server again allows to filter the files based on the geographical extent meta information within the individual CityJSON files. In addition of filtering complete files, the endpoint also offers the possibility of a finer granularity on the level of `CityObjects` and also on `Face` level. These parameters are optional and allow to reduce the amount of transmitted data. While filtering for complete files or objects can be done without any risk, a user has to consider that filtering for faces can dismember the objects and may result in shallow occlusion models.

6 EVALUATION

The proposed methodology is evaluated based on a bidirectional conversion functionality to translate a given Unity `Mesh` object to a `CityJsonFile` object and back to the Unity representation. After the conversion, we are evaluating the error between the input vertices to the corresponding output vertices. This is done decoupled from the actual target MR platform and from the server in order to be able to exclude possible additional external influences.

This quantitative evaluation is done using multiple three-dimensional models in the form of OBJ files and the `EPSG::4326` coordinate (48.30285, 14.28428, 279.807) as reference point at the origin (0,0,0) of the local coordinate system. These models are imported to Unity, converted to a geo-referenced CityJSON representation, and afterwards back-transformed to Unity's `Mesh` representation. Since the conversion process preserves the ordering of individual vertices

Table 1: Table showing the results of the quantitative evaluation with a minimal error of 1.30E-06 and a maximal one of 2.79E-04 based on the normalized sum of squared errors of the corresponding vertices between the conversion's input and output models.

		Min			Max			
Object	Vertices	x	y	z	x	y	z	Error (m)
Cube	8	-0.5	-0.5	-0.5	0.5	0.5	0.5	2.79E-04
Stanford Bunny (Turk and Levoy, 1994)	2503	-0.1	0.0	-0.1	0.1	0.2	0.1	1.30E-06
Utah Teapot (Newell, 1975)	3241	-3.4	0.0	-2.0	3.0	3.2	2.0	3.49E-05
HoloLens Room Model 1	22805	-5.4	-1.9	-6.6	3.0	1.7	5.5	2.89E-05
HoloLens Room Model 2	67173	-4.2	-1.3	-4.5	6.0	1.6	8.8	1.87E-05
Stanford T-Rex (Principia Inc., 1997)	100002	-0.8	-1.3	-2.1	0.8	1.3	2.1	3.90E-06
Stanford Armadillo (Krishnamurthy and Levoy, 1996)	106289	-1.3	-1.1	-1.0	1.3	1.9	0.8	2.48E-06

of the mesh, the distances of every corresponding vertex at index i with x_1i, y_1i and z_1i of the input model and the points of the back transformed mesh with x_2i, y_2i and z_2i can be used for the evaluation based on the Pythagorean theorem as shown in Equation 8. We are using the "sum of squared errors" metric, normalized by the number of vertices, to describe model differences based on the individual point distances, as shown in Equation 9. The results are shown in Table 1 with an error range from 1.30E-06 to 2.79E-04, which can be interpreted as a quadratic offset between the input and output model in meters, i.e. the error is located in a sub millimeter range. This error can be linked to two sources. On the one hand we are using a spheric earth model, when converting between local and global coordinates, with an approximated equatorial circumference of 6378137 meters and ignoring ellipsoidal effects. This is ok for small objects, but the error grows with the spatial expansion of the model. On the other hand, while we are using a double precision for the global coordinates, Unity's local coordinate system is based on a floating point precision. Because of this, the precision is lost, during the conversion between the coordinate systems, which also increases the error.

$$d_i = \sqrt{(x_{2i} - x_{1i})^2 + (y_{2i} - y_{1i})^2 + (z_{2i} - z_{1i})^2} \quad (8)$$

$$error = \frac{\sum_{i=0}^{vertices} \sqrt{d_i^2}}{vertices} \quad (9)$$

To visually highlight the influence of the conversion error the input and the converted models are exemplary compared in Figure 6. The shown model is the Utah Teapot (Newell, 1975) from the previous quantitative evaluation. This example shows, that the minimal conversion errors as described in the quantitative evaluation, are not recognizable by the human eye, when using the system.

7 RELATED WORK

Keil et al. (2021) are utilizing geo-referenced CityJSON models in combination with information from OpenStreetMap to create three-dimensional scenarios for Virtual Reality (VR) applications. To do so, the models are once converted with the computer graph-

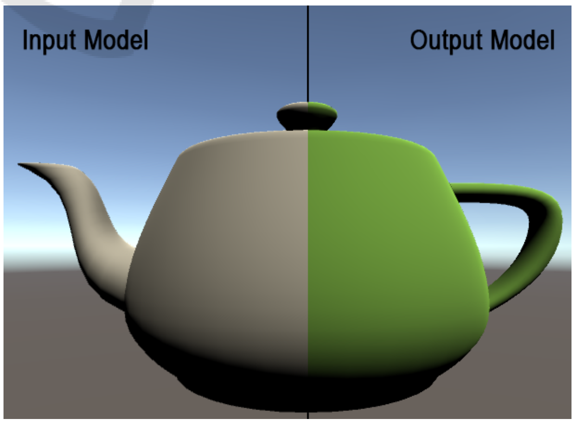

Figure 6: Visual comparison of the Utah Teapot (Newell, 1975) as input mesh model (left side), that is converted to a CityJSON representation and back-transformed to a mesh model (right side), showing that the minimal errors as described in the quantitative evaluation are visually not recognizable.

ics software Blender and then imported into a Unity application. In contrast to our work, the authors are not transferring global coordinates to local ones within their application during run time and don't use this approach for occlusion calculations. In addition to that, they are not creating and sharing environmental models across devices, but only display existing data sets.

Buyukdemircioglu and Kocaman (2020) present a system for visualizing existing urban structures in combination with future planned buildings based on CityGML representations and VR devices. The aim of the system is the virtual exploration of the future city model and like this deviates from our work, where we are focusing on dynamically creating and sharing environmental models in the context of AR applications.

Blut and Blankenbach (2020) describe a smartphone-based system in the context of civil engineering utilizing CityGML models based on a previously published mobile CityGML AR viewer (Blut et al., 2019). The system is intended for geo-referenced on-site visualization of planned buildings and is focusing on visually estimating a user's pose within the given model by comparing real-world objects with their digital representations. Compared to this work, we are not estimating the pose of the user and are using CityGML, respectively CityJSON only as source for object occlusion and information sharing.

Chalumattu et al. (2020) present an approach for sharing location-based, virtual objects in the context of outdoor AR applications. To do so, the authors use a static city model, that is linked with Microsoft's Spatial Anchor system and additional visual markers for Android devices to real world positions. Based on this basic model and the spatial reference information, a user can place additional virtual objects, which can be shared with other participants. The shared objects are occluded based on the city model. In contrast to our work, Chalumattu et al. don't share dynamically created occlusion models of the user's environment, and also don't incorporate world coordinates.

Multiple publications (Kilimann et al., 2019; Capece et al., 2016; Ghadirian and Bishop, 2008) present the utilization of geographic information systems (GIS) in the context of outdoor AR applications. The introduced methodologies are used to place virtual counterparts at given global positions, retrieved from such services. In contrast to our work, the authors don't exchange concrete 3D geometries, but only position and type information and are placing statically pre-defined models based on this input. In addition to that, the authors focus on sharing objects for the visualization of e.g. changes in the local vegetation or infrastructural points of interest as electrical power lines instead of dynamically, created occlusion models. Due to the utilization of GIS, the approaches of the before mentioned publications are dependent on the up-to-dateness of the shared information like torn down or new constructed buildings in urban scenarios.

8 CONCLUSION

The utilization of the CityJSON format for exchanging spatial information in the context of occlusion models shows promise. The results show that dynamically created geo-referenced three-dimensional geometries can be successfully exchanged over multiple AR and MR devices in the context of outdoor scenarios and occlusion models.

As the accuracy of the placed objects' position is dependent on one reference point, it is only as good as the system providing this information. Which means, that sharing occlusion models across multiple devices, will only result in accurately covered objects, when all participants have exact knowledge about their global position. On basis of the server side filtering methods, the system allows sharing occlusion models within a spatial restricted region of interest, leading to a reduced data volume when accessing the information. Like this, state-of-the-art MR devices such as the Microsoft HoloLens 2 can exchange up-to-date environmental information, also with limited network bandwidths.

9 OUTLOOK

Due to the promising results of sharing geo-referenced occlusion models across multiple Microsoft HoloLens 2 devices, we plan to also incorporate models from different sources as e.g. camera based 3D reconstructions from smartphones or drones. This extension will allow us increasing the size of the available dataset, as well as supported devices, with the aim of improving the occlusion possibilities in outdoor scenarios. Especially, in the area of task force training, this step will allow us creating dynamic training scenarios by integrating not fixed positioned objects such as boxes or barrels. In addition, we also plan to partially reduce the dependency to the global reference position by correcting positional errors based on the comparison of visual features and the used models.

ACKNOWLEDGMENT

Our thanks to the Austrian Research Promotion Agency FFG for facilitating the project *MRCC* with the Small Scale Project funding program (program number: 883742), with research budget provided by the Federal Republic of Austria. The authors also thank their project partners of the Austrian Mixed Reality company Realsim®, especially Thomas Peterseil, Erik Thiele and Mario Voithofer, for the close cooperation and their efforts.

REFERENCES

Blut, C., Blut, T., and Blankenbach, J. (2019). Citygml goes mobile: application of large 3d citygml models on smartphones. *International Journal of Digital Earth*, 12(1):25–42.

Bray, T. et al. (2014). The javascript object notation (json) data interchange format.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., and Cowan, J. (2000). Extensible markup language (xml) 1.0.

Capece, N., Agatiello, R., and Erra, U. (2016). A client-server framework for the design of geo-location based augmented reality applications. In *2016 20th International Conference Information Visualisation (IV)*, pages 130–135.

Chen, J. X. (2003). 3d file formats. In *Guide to Graphics Software Tools*, pages 127–136. Springer New York, New York, NY.

De Crescenzio, F., Fantini, M., Persiani, F., Di Stefano, L., Azzari, P., and Salti, S. (2011). Augmented reality for aircraft maintenance training and operations support. *IEEE Computer Graphics and Applications*, 31(1):96–101.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

Gavish, N., Gutiérrez, T., Webel, S., Rodríguez, J., Peveri, M., Bockholt, U., and Tecchia, F. (2015). Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive Learning Environments*, 23(6):778–798.

Ghadirian, P. and Bishop, I. D. (2008). Integration of augmented reality and gis: A new approach to realistic landscape visualisation. *Landscape and Urban Planning*, 86(3):226–232.

Gröger, G. and Plümer, L. (2012). Citygml–interoperable semantic 3d city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12–33.

Haas, J. (2014). A history of the unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE*.

Haleem, A., Javaid, M., Goyal, A., and Khanam, T. (2020). Redesign of car body by reverse engineering technique using steinbichler 3d scanner and projet 3d printer. *Journal of Industrial Integration and Management*, page 2050007.

Heinz, E., Eling, C., Wieland, M., Klingbeil, L., and Kuhlmann, H. (2015). Development, calibration and evaluation of a portable and direct georeferenced laser scanning system for kinematic 3d mapping. *Journal of Applied Geodesy*, 9(4):227–243.

Hübner, P., Clintworth, K., Liu, Q., Weinmann, M., and Wursthorn, S. (2020). Evaluation of hololens tracking and depth sensing for indoor mapping applications. *Sensors*, 20(4):1021.

Ingrassia, P. L., Mormando, G., Giudici, E., Strada, F., Carfagna, F., Lamberti, F., and Bottino, A. (2020). Augmented reality learning environment for basic life support and defibrillation training: Usability study. *Journal of Medical Internet Research*, 22(5):e14910.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al. (2011). Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568.

Javaid, M., Haleem, A., Pratap Singh, R., and Suman, R. (2021). Industrial perspectives of 3d scanning: Features, roles and it's analytical applications. *Sensors International*, 2:100114.

Kaplan, A. D., Cruit, J., Endsley, M., Beers, S. M., Sawyer, B. D., and Hancock, P. (2020). The effects of virtual reality, augmented reality, and mixed reality as training enhancement methods: a meta-analysis. *Human factors*, page 0018720820904229.

Kaplan, E. D. and Hegarty, C. (2017). *Understanding GPS/GNSS: principles and applications*. Artech house.

Khronos Group Inc. (2008). Collada overview - the khronos group inc. https://www.khronos.org/collada/. (Accessed on 07/22/2021).

Kilimann, J.-E., Heitkamp, D., and Lensing, P. (2019). An augmented reality application for mobile visualization of gis-referenced landscape planning projects. In *The 17th International Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 1–5.

Kingsland, K. (2020). Comparative analysis of digital photogrammetry software for cultural heritage. *Digital Applications in Archaeology and Cultural Heritage*, 18:e00157.

Krishnamurthy, V. and Levoy, M. (1996). Fitting smooth surfaces to dense polygon meshes. In Fujii, J., editor, *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*, pages 313–324. ACM.

Ledoux, H., Ohori, K. A., Kumar, K., Dukai, B., Labetski, A., and Vitalis, S. (2019a). Cityjson: A compact and easy-to-use encoding of the citygml data model. *Open Geospatial Data, Software and Standards*, 4(1):1–12.

Ledoux, H., Ohori, K. A., Kumar, K., Dukai, B., Labetski, A., and Vitalis, S. (2019b). Datasets — cityjson. https://www.cityjson.org/datasets/. (Accessed on 08/03/2021).

McKnight, R. R., Pean, C. A., Buck, J. S., Hwang, J. S., Hsu, J. R., and Pierrie, S. N. (2020). Virtual reality and augmented reality—translating surgical training into surgical technique. *Current Reviews in Musculoskeletal Medicine*, pages 1–12.

Microsoft (2021). microsoft/mixedrealitytoolkit-unity: Mixed reality toolkit (mrtk) provides a set of components and features to accelerate cross-platform mr app development in unity. https://github.com/microsoft/MixedRealityToolkit-Unity. (Accessed on 08/03/2021).

Newell, M. (1975). Utah teapot. https://graphics.stanford.edu/courses/cs148-10-summer/as3/code/as3/teapot.obj. (Accessed on 08/05/2021).

Nguyen, T.-N., Huynh, H.-H., and Meunier, J. (2018). 3d reconstruction with time-of-flight depth camera and multiple mirrors. *IEEE Access*, 6:38106–38114.

Nolan, D. and Lang, D. T. (2014). Keyhole markup language. In *XML and Web Technologies for Data Sciences with R*, pages 581–618. Springer.

Pallets (2010). pallets/flask: The python micro framework for building web applications. https://github.com/pallets/flask/. (Accessed on 08/06/2021).

Portele, C. (2007). Opengis® geography markup language (gml) encoding standard. version 3.2. 1. *OGC Standards*.

Praschl, C., Krauss, O., and Zwettler, G. A. (2020). Enabling outdoor mr capabilities for head mounted displays: a case study. *International Journal of Simulation and Process Modelling*, 15(6):512–523.

Principia Inc. (1997). Stanford t-rex. https://www.prinmath.com/csci5229/OBJ/index .html. (Accessed on 08/05/2021).

Pyproj4 (2014). pyproj4/pyproj: Python interface to proj (cartographic projections and coordinate transformations library). https://github.com/pyproj4/pyproj. (Accessed on 08/06/2021).

Reddy, M., Iverson, L., and Leclerc, Y. G. (2000). Under the hood of geovrml 1.0. In *Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML)*, pages 23–28.

Swaminathan, A. (2019). Perception at magic leap. *Georgia Tech Seminars*.

Tachella, J., Altmann, Y., Mellado, N., McCarthy, A., Tobin, R., Buller, G. S., Tourneret, J.-Y., and McLaughlin, S. (2019). Real-time 3d reconstruction from single-photon lidar data using plug-and-play point cloud denoisers. *Nature communications*, 10(1):1–6.

Taketomi, T., Uchiyama, H., and Ikeda, S. (2017). Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11.

Thøgersen, M., Andoh, J., Milde, C., Graven-Nielsen, T., Flor, H., and Petrini, L. (2020). Individualized augmented reality training reduces phantom pain and cortical reorganization in amputees: A proof of concept study. *The Journal of Pain*, 21(11):1257–1269.

Turk, G. and Levoy, M. (1994). Stanford bunny. https://graphics.stanford.edu/ mdfisher/Data/Meshes/bunny.obj. (Accessed on 08/05/2021).

Turner, A. and Coulter, D. (2019). Spatial anchors - mixed reality — microsoft docs. https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-anchors. (Accessed on 07/21/2021).

Ungureanu, D., Bogo, F., Galliani, S., Sama, P., Duan, X., Meekhof, C., Stühmer, J., Cashman, T. J., Tekin, B., Schönberger, J. L., et al. (2020). Hololens 2 research mode as a tool for computer vision research. *arXiv preprint arXiv:2008.11239*.

Van Rossum, G. and Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Van Schaik, J. (2017). Using a hololens scanned room inside your hololens app - dotnetbyexample - the next generation. https://localjoost.github.io/using-hololens-scanned-room-inside-your/. (Accessed on 07/22/2021).

Veness, C. (2019). Calculate distance and bearing between two latitude/longitude points using haversine formula in javascript. https://www.movable-type.co.uk/scripts/latlong.html. (Accessed on 08/03/2021).

Vergel, R. S., Tena, P. M., Yrurzum, S. C., and Cruz-Neira, C. (2020). A comparative evaluation of a virtual reality table and a hololens-based augmented reality system for anatomy training. *IEEE Transactions on Human-Machine Systems*, 50(4):337–348.

Westerfield, G., Mitrovic, A., and Billinghurst, M. (2014). Intelligent augmented reality training for motherboard assembly. *International Journal of Artificial Intelligence in Education*, 25(1):157–172.

Wu, Y. (2021). Application research of terrain mapping based on riegl 3d scanning system. In *IOP Conference Series: Earth and Environmental Science*, volume 719, page 042058. IOP Publishing.

Zunke, S. and D'Souza, V. (2014). Json vs xml: A comparative performance analysis of data exchange formats. *IJCSN International Journal of Computer Science and Network*, 3(4):257–261.