Determining Field of View in Outdoors Augmented Reality Applications

Vlasios Kasapakis[1,2(✉)] and Damianos Gavalas[1,2]

[1] Department of Cultural Technology and Communication,
University of the Aegean, Mytilene, Greece
{v.kasapakis,dgavalas}@aegean.gr
[2] Computer Technology Institute and Press 'Diophantus' (CTI), Patras, Greece

Abstract. The use of augmented reality (AR) becomes increasingly common in location based application development. A situation often encountered in AR applications is the -partial or full- occlusion of virtual objects by physical artifacts; if not appropriately handled, the visualization of occluded objects often misleads users' perception. This paper presents a Geolocative Raycasting technique aiming at assisting developers of outdoors augmented reality applications into generating a realistic field of view for the users by integrating real time building recognition, so as to address the occlusion problem.

1 Introduction

Augmented reality (AR) requires only a limited amount of the user's field of view to be rendered with computer-generated graphics with the major part of the user's view covered by the physical world [2]. The allowance of users to view the physical world provides them a better sense of where they are and what is around them. Nevertheless, cases often occur that a physical object occludes a virtual object; like when surrounding buildings exist and are highly likely to occlude a point of interest. Then, the overlaying of the augmented image may cause confusion to users' perception. This incorrect display contributes to misconceptions and wrong pursuance of tasks amongst users [1, 3]. The problem of occlusion in AR can be observed in a variety of location-based applications. TripAdvisor[1] is a popular mobile travel application which provides reviews of travel-related content. Recently, TripAdvisor added an AR projection mode for points of interest (POIs), superimposing AR markers upon the smartphone's camera views. A similar technique is followed in mTrip[2], another popular, commercial mobile tourism route planner. The occlusion problem is also common in pervasive games utilizing AR, affecting the players' immersion when virtual characters are not hidden when located behind surrounding buildings [4].

In classic video games, the visibility of virtual objects is estimated utilizing the raycasting technique. Raycasting is the act of casting imaginary light beams (rays) from a source location (typically the point of view of the character or object controlled by the

[1] www.tripadvisor.com.
[2] www.mtrip.com.

Kameas et al. (Eds.): AmI 2015, LNCS 9425, pp. 344–348, 2015.
DOI: 10.1007/978-3-319-26005-1_23

player) and recording the objects hit by the rays. Herein, we extend this idea in outdoors AR applications wherein, unlike video games, the virtual space is integrated with the physical one, is not pre-registered and occlusion is typically caused by surrounding buildings. In particular, we introduce a *Geolocative Raycasting* technique that allows augmented reality application developers to detect buildings or custom-generated obstacles in location-based and AR game environments, thereby reliably resolving the object occlusion issue.

2 Preparing the Building Data and Performing Raycasting

In order to perform geolocative raycasting, the information about the location of buildings surrounding the user should be available. In our approach the building data is yield from the Overpass Turbo API[3], where the latitude and longitude points of every building polygon are utilized to generate a list of polygons[4] and LatLngBounds[5] (i.e. rectangular bounding boxes utilized to approximate the coordinates of the building's center). The building polygons are drawn on the OSM map[6]. Next, the accelerometer and magnetometer sensors[7] of the user's Android smartphone are enabled to extract the azimuth from the rotation matrix of the device[8], determining the device's orientation (taking into account the device inclination and remapping the axis when needed). The device's bearing is calculated utilizing the azimuth measurement. An extract from our raycasting algorithm implementation is listed in Fig. 1 below.

```
for(int degree=-5; degree<5; degree=degree+1){
    for (int i=1;i<27;i++){

double distance =   0.0000006 *i ;
double bearing = Math.toRadians((int) Math.round(Math.toDegrees(azimut)-90)+degree);
double userlat = Math.toRadians(myloc.getLatitude());
double userlon = Math.toRadians(myloc.getLongitude());

double newlat = Math.asin(Math.sin(userlat)*Math.cos(distance)
       + Math.cos(userlat)*Math.sin(distance)*Math.cos(bearing) );

double a = Math.atan2(Math.sin(bearing)*Math.sin(distance)*Math.cos(userlat),
       Math.cos(distance)-Math.sin(userlat)*Math.sin(newlat));

double newlon = userlon+ a;
newlon = (newlon+ 3*Math.PI) % (2*Math.PI) - Math.PI;

    }
}
```

Perform raycasting for different angles
Number of ray steps
Add distance to next ray step
Calculate bearing and get user location via GPS
Calculate ray step latitude
Calculate ray step longitude

Fig. 1. Extract from the raycasting algorithm implementation.

[3] http://overpass-turbo.eu/.
[4] https://github.com/sromku/.
[5] developer.android.com/reference/com/google/android/gms/maps/model/LatLngBounds.html.
[6] https://code.google.com/p/osmdroid/.
[7] developer.android.com/guide/topics/sensors/sensors_overview.html.
[8] http://developer.android.com/reference/android/hardware/SensorManager.html.

The raycasting algorithm utilized in our work[9] generates virtual locations along a straight line (26 points, each positioned ~3.8 m further from the previous one, resulting in a 100 m ray) towards the user's facing direction, until one of the ray steps (i.e. virtual location) is found to lie inside a polygon (building) of the above mentioned polygon list. Upon detecting such event, it is realized that the ray has been blocked by a building; hence generating further ray steps along that line is unnecessary. Since a single ray is insufficient to accurately estimate the user's field of view, the above detailed process is executed every second degree (note that in the implementation of the raycasting is performed for every one degree of the field of view), in a range of −5 to +5°, considering the current bearing of the device as central direction (10 raycasts in total, resulting into a 10° degrees angle field of view). The above described method is illustrated in Fig. 2a, where 10 raycasts determine the 10° users' field of view in an area featuring buildings stored in the OSM database (red-colored dots denote points invisible from the device's current location).

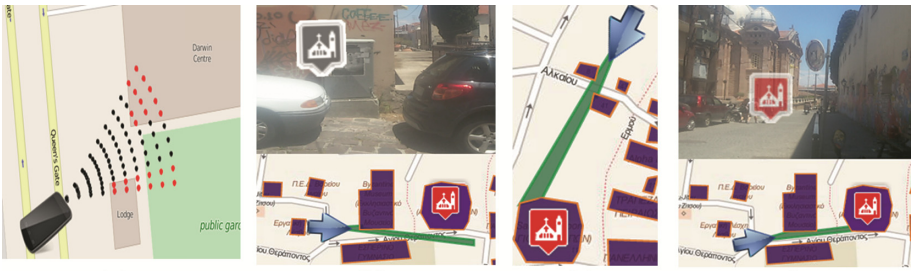

Fig. 2. (a) multi-angle raycasting generating users' field of view; (b) POI outside the users' field of view; (c) field of view representation; (d) POI partially inside the users' field of view.

In order to validate the raycasting approach presented in this work, a simple mobile tourist AR application has been developed as a case-study utilizing OSM and BeyondAR[10] framework. The application included a POI church building which was represented by a marker on OSM maps and an augmented reality marker in BeyondAR framework. When the building polygon of the POI is out of the user's field of view, a grey-colored AR marker is used to denote the location of the church. (Figure 2b) When the ray steps hit the POI building, the point of impact of the blocked ray is saved in an array; upon the completion of the raycasting process, those impingement points are utilized to draw a polygon on the OSM map, providing a visual representation of the users' field of view (Fig. 2c). Finally when the POI is inside the users' field of view the church icon turns from grey to red, informing the user that she has eye constant with it (Fig. 2d). Also the total number of the rays which hit the building were utilized to adjust

[9] The formula for calculating a virtual point in front of the user utilizing her current locations' latitude and longitude, along with her device direction may be found at http://www.movable-type.co.uk/scripts/latlong.html.

[10] http://beyondar.com/.

the augmented reality marker transparency, visualizing this way the percentage of the field of view of the user where the POI was included[11].

A factor largely affecting the performance of raycasting is the number of buildings examined (among those returned from the Overpass Turbo API). To limit that number we have applied a distance threshold (representing the ray's reach) around the user's location. The distance is calculated from the user's current location to the center of every building (i.e. the center of the LatLngBounds bounding box). Nearby buildings are re-calculated upon every change on the user's position. The application of a distance threshold slightly longer than the length of the ray ensured that the corners of buildings whose centers are slightly further from the ray's reach are also detected. In order to evaluate the sufficient preface of the raycasting method presented in this work for real time building recognition a full performance test has been conducted.

The test space (see Fig. 3 below) has been set in the center of Athens (Greece), as the OSM database contains a large number of registered buildings in that area. The size of the test area has been set to 707 m^2, adjusting the ray to the same settings as presented above. Updates of the nearby buildings list have been triggered every 2 s by applying a distance threshold of 120 meters. The device was constantly rotated throughout the test (approximately 25 rotations in a 60 s testing session). The total number of buildings within the 120 m radius was 266 with a mean of 43 buildings taken into account by every ray cast. A total number of 457 raycastings were executed, with a mean of 7.6 raycastings per second and an execution mean of 131.2 ms per raycast, providing sufficient evidence that the presented technique would be sufficient for location-based AR applications with real time performance requirements.

Fig. 3. Athens test area.

3 Conclusion

In this article a geolocative raycasting technique has been proposed to detect buildings in real-time, aiming to help future developers into addressing the occlusion problem which is common in location-based AR applications.

[11] A video presenting the performance of the test application can be found at https://youtu.be/zCoI0RW1CcI.

A prototype location-based AR tourist guide application has been used as a case study to showcase the validity of our approach. The performance evaluation of the above application revealed its efficiency which makes it appropriate for relevant location-based outdoors AR applications, wherein marked POIs are commonly occluded by surrounding buildings.

References

1. Shah, M.M., Arshad, H., Sulaiman, R.: Occlusion in augmented reality. In: 2012 8th International Conference on Information Science and Digital Content Technology (ICIDT), pp. 372–378. IEEE (2012)
2. Thomas, B.H.: A survey of visual, mixed, and augmented reality gaming. ACM Comput. Entertainment **10**, 1–33 (2012)
3. Tian, Y., Long, Y., Xia, D., Yao, H., Zhang, J.: Handling occlusions in augmented reality based on 3D reconstruction method. Neurocomputing **156**, 96–104 (2015)
4. Wetzel, W., Blum, L., McCall, R., Oppermann, L., Broeke, T.S., Szalavári, Z.: Final Prototype of TimeWarp application, IPCity (2009)