Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016)

Advanced Algorithms for Location-Based Smart Mobile Augmented Reality Applications

Rüdiger Pryss[a,*], Philip Geiger[a], Marc Schickler[a], Johannes Schobel[a], Manfred Reichert[a]

[a] *Ulm University, Institute of Databases and Information Systems, James-Franck-Ring, Ulm, 89081, Germany*

Abstract

During the last years, the computational capabilities of smart mobile devices have been continuously improved by hardware vendors, raising new opportunities for mobile application engineers. Mobile augmented reality is one scenario demonstrating that smart mobile applications are becoming increasingly mature. In the AREA (Augmented Reality Engine Application) project, we developed a kernel that enables such location-based mobile augmented reality applications. On top of the kernel, mobile application developers can easily realize their individual applications. The kernel, in turn, focuses on robustness and high performance. In addition, it provides a flexible architecture that fosters the development of individual location-based mobile augmented reality applications. In the first stage of the project, the LocationView concept was developed as the core for realizing the kernel algorithms. This LocationView concept has proven its usefulness in the context of various applications, running on iOS, Android, or Windows Phone. Due to the further evolution of computational capabilities on one hand and emerging demands of location-based mobile applications on the other, we developed a new kernel concept. In particular, the new kernel allows for handling points of interests (POI) clusters or enables the use of tracks. These changes required new concepts presented in this paper. To demonstrate the applicability of our kernel, we apply it in the context of various mobile applications. As a result, mobile augmented reality applications could be run on present mobile operating systems and be effectively realized by engineers utilizing our approach. We regard such applications as a good example for using mobile computational capabilities efficiently in order to support mobile users in everyday life more properly.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Mobile Augmented Reality; Location-based Algorithms; Mobile Application Engineering; Augmented Reality

1. Introduction

The proliferation of smart mobile devices on one hand and their computational capabilities on the other have enabled new kinds of mobile applications. So-called millenials, people born after 1980, pose demanding requirements with respect to the use of mobile technology in everyday life. Amongst others, they want to be assisted by mobile technology in their leisure time. For example, when walking around in Rome with its numerous ancient spots, the

* Corresponding author. Tel.: +49-731-5024136 ; fax: +49-731-5024134.
E-mail address: ruediger.pryss@uni-ulm.de

1877-0509
Peer-review under responsibility of the Conference Program Chairs.

smart mobile device shall provide related information about these spots in an intuitive way. In such a scenario, location-based mobile augmented reality is useful, e.g., if a user is located in front of the St. Peter's Basilica, holding his smart mobile device with its camera switched on towards the Basilica, the camera view shall provide additional information (e.g., worship time).

Our developed AREA kernel supports these scenarios. More precisely, AREA detects predefined *points of interest* (POIs) within the camera view of a smart mobile device, positions them correctly, and provides additional information on the detected POIs. This additional information, in turn, is interactively provided to the user. For this purpose, the user touches on the detected POIs and then obtains further information. Three technical issues were crucial when developing AREA. First, POIs must be correctly displayed even if the device is hold obliquely. Depending on the attitude of the device, the POIs then may have to be rotated with a certain angle and moved relatively to the rotation. Second, the approach to display POIs correctly must be provided efficiently to the user. To be more precise, even if multiple POIs are detected, the kernel shall display them without any delay. Third, the POI concept shall be integrated with common mobile operating systems (i.e., iOS, Android, and Windows Phone). To tackle these challenges, the *LocationView* concept was developed. Additionally, an architecture was designed, which shall enable the quick development of location-based mobile augmented applications on top of the kernel [1,2,3].

The AREA project was started four years ago. Already one year after the first kernel version had been finished, it was integrated with various mobile applications. During these projects, three issues emerged that are not properly covered by the AREA kernel so far. First, the changed characteristics of the used mobile operating systems need to be addressed. Second, the continuously growing number of POIs must be handled more efficiently. Third, new features were demanded. Table 1 summarizes the evolution of AREA from its first to its second version.

Table 1. AREA Versions

	AREA Version 1 (AREA)	**AREA Version 2 (AREAv2)**
Android App	√	√
iOS App	√	√
Windows Phone App	√	*under development*
POI Algorithm (all mobile OS)	*LocationView Algorithm* [1,2,3]	*RenderingPipeline Algorithm*
Clustering Algorithm (all mobile OS)	*X*	*AREACluster Algorithm*
POI Coordinate System (all mobile OS)	*GPS*	$GPS, ECEF, ENU, Virtual3D$
Position Change Sensors (all mobile OS)	$SensorFusion(Compass, Accelerometer)$	$SensorFusion(Compass, Gyroscope, Accelerometer)$
Architecture (all mobile OS)	$Version\ 1$	$Version\ 2$
Overall Sensor Management Android	*Own approach*	*Own approach*
Overall Sensor Management iOS	*Own approach*	$Built-in\ functions\ OS$
Overall Sensor Management Windows Phone	*Own approach*	*under development*

ENU=East-North-Up Coordinate System, ECEF=Earth-Centered Earth-Fixed Coordinate System, GPS=Global Positioning System

The changed characteristics of mobile operating systems as well as performance issues with many POIs are addressed by the development of a new kernel and architecture called AREA Version 2 (AREAv2) (cf. Table 1, *AREAv2*). Moreover, AREAv2 provides three new features. The first one deals with so-called POI clusters. If a huge number of POIs causes many overlaps on the camera view, it is difficult for users to precisely interact with single POIs inside such cluster. In order to precisely select (i.e., do not touch the wrong POI as they are positioned to close to each other) a single POIs inside a cluster, a new feature was developed. The second feature we developed connects POIs through lines in order to visualize tracks. For example, such a track may be used as the cycle path a user wants to perform in a certain area. The third feature highlights areas (e.g., football fields). From a technical perspective, the added features are demanding if they shall be supported in the same manner on different mobile operating systems.

This work presents fundamental concepts developed in the context of AREA Version 2 (AREAv2). Section 2 presents the architecture of AREAv2. In Section 3, the coordinate system used by AREAv2 is introduced, while Sections 4 and 5 present the algorithms for POI and cluster handling. Section 6 illustrates the use of AREAv2 in practical scenarios. Section 7 discusses related work and Section 8 concludes the paper.

2. Architecture

The architecture used for AREAv2 is shown in Fig. 1. It enhances the one used in the first version of AREA [1,2,3] and comprises **nine** major components (cf. Fig. 1).

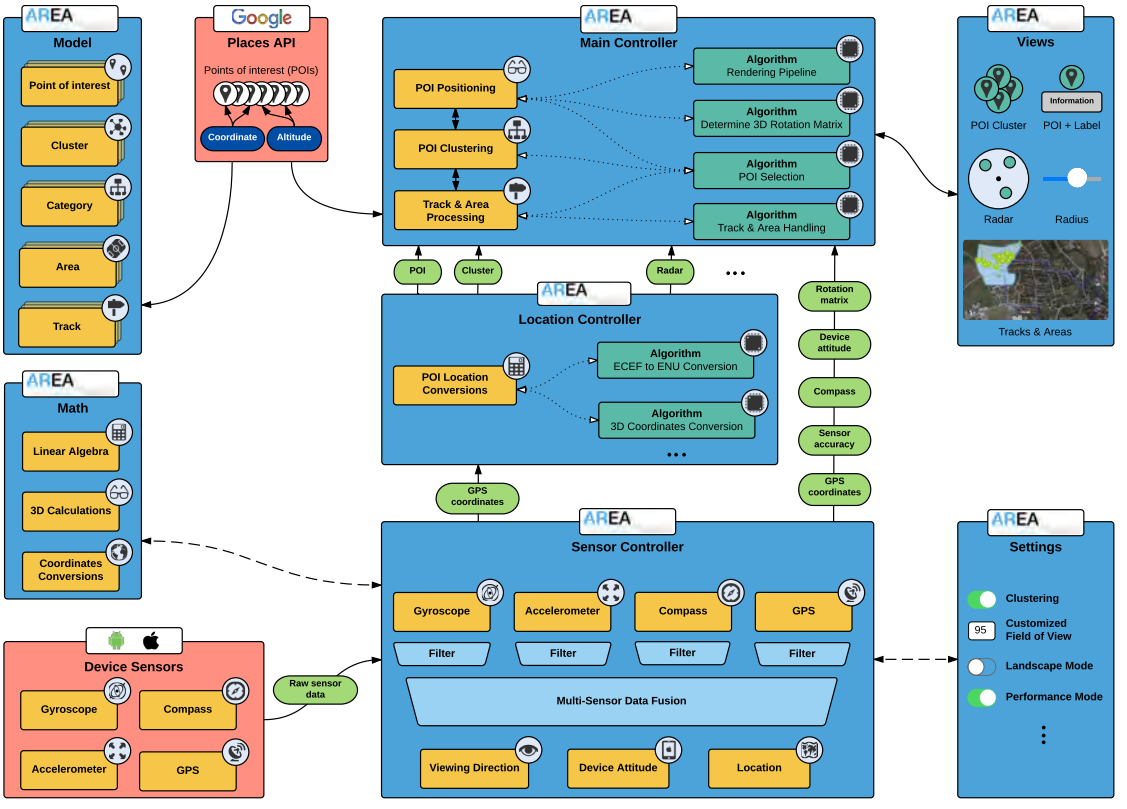

Fig. 1. AREAv2 Architecture

The *Model* component manages POIs, POI categories, POI clusters, POI tracks, and POI areas. Developers can use this component to integrate application-specific POI categories and change the visualization of all provided POI features. The *Places API* component, in turn, allows displaying POIs provided by Google or other remote APIs. Note that this component was integrated for testing huge numbers of POIs or POI clusters more easily. As AREAv2 shall also work without any online connection, the used POIs are locally stored on the smart mobile device. The local database, however, can be synchronized with a remote database. Note that the components to store POIs locally and synchronize them with a remote database are neither presented here nor depicted in Fig. 1. Using this approach, developers are able to test AREAv2 and their individual implementations on top of AREAv2 with huge numbers of POIs even in case the local database is empty. The *Math* component provides functions for the coordinate systems used. Compared to AREA, AREAv2 uses a new sensor fusion approach that provides a more precise positioning of POIs through the *Sensor* component. Therefore, four sensors are considered on all supported mobile operating systems, i.e., gyroscope, compass, accelerometer, and GPS (cf. Fig. 1). The *Location* component provides algorithms for handling the different coordinate systems. Their results, combined with the ones of the *Sensor* component, are used by the *Main* component. The *Main* component realizes algorithms that enable the handling of all POI related features. The *View* component, in turn, enables all required visualizations, i.e., POIs, POI labels, POI clusters, POI tracks, POI areas, a POI radar, and a POI radius. The radar can be used to evaluate if POIs that are currently not displayed can be obtained by changing the facing of the smart mobile device towards another direction. The radius, in turn, can be used to specify the maximum distance a user shall have to the POIs to be displayed. By calculating the distance between the device and the POIs based on coordinate calculations, AREAv2 can determine the POIs located inside the chosen radius and hence the POIs to be displayed on the screen. Finally, the *Settings* component realizes functions that enable users to adjust selected AREAv2 features (cf. Fig. 1).

3. Coordinate System Algorithms

AREAv2 uses a coordinate system that differs from the one used in the first version. AREA solely worked on GPS coordinates. More precisely, the GPS coordinates of the user were calculated through the GPS sensor of his smart mobile device, while the GPS coordinates of the POIs were obtained from the local database. Based on a comparison of user and POI coordinates as well as supporting calculations (e.g., if the device is hold obliquely), the POIs could be correctly displayed in the camera view of a mobile user.

The core idea of AREAv2 is based on five aspects which require the use of a new coordinate system. First, a virtual 3D world is used to relate the user's position to the one of the POIs. Second, the user is located at the origin of this world. Third, instead of the physical camera, a virtual 3D camera is used that operates with the created virtual 3D world. The virtual camera is therefore placed at the origin of this world. Fourth, the different sensor characterictics of the supported mobile operating systems must be properly covered to enable the virtual 3D world. On iOS, sensor data of the gyroscope and the accelerometer are used, whereas on Android sensor data of the gyroscope, the accelerometer and the compass of the mobile device are used to position the virtual 3D camera correctly. Due to lack of space, the different concepts to integrate sensor data in the same way on all mobile operating systems by AREAv2, cannot be presented in detail. Fifth, the physical camera of the mobile device is adjusted to the virtual 3D camera based on the assessment of sensor data.

In order to realize the presented core idea of AREAv2, a complex coordinate system, consisting of three different sub-systems, is required. The first sub-system uses GPS, ECEF (Earth-Centered, Earth-Fixed), and ENU (East, North, Up) coordinates.[1] The second sub-system, in turn, uses a virtual 3D space with the user located at the origin. Finally, the third one uses a virtual 3D camera, with the camera being again located at the origin of the 3D world. Note that the first sub-system (including GPS, ECEF, and ENU coordinates) is a major prerequisite (cf. Fig. 2) for transforming sensor data of the smart mobile device to data that can be used for the virtual 3D world.

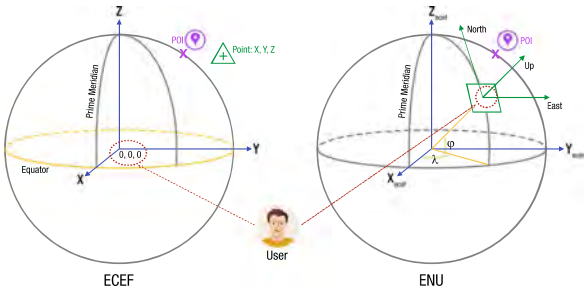

Fig. 2. ECEF and ENU Coordinate Systems

As illustrated in Fig. 2, the user is located at the ECEF origin $(0, 0, 0)$. The POIs, in turn, are located on the surface of the earth, again using ECEF coordinates. To use this metapher for the virtual 3D world, two additional transformations became necessary. As a smart mobile device can only sense GPS coordinates, first of all, the GPS coordinates of the user and POIs need to be transformed into ECEF coordinates (cf. Algorithm 1). Second, as a user cannot be physically located at the origin of the earth, ECEF coordinates are transformed into ENU coordinates (cf. Algorithm 2). ENU coordinates, in turn, enable the required metapher for the virtual 3D world. More precisely, ENU coordinates are transformed into coordinates for the virtual 3D world by a transformation of axes. Finally, in addition to the two transformation algorithms (i.e., Algorithm 1 and 2), Algorithm 3 calculates the distance between a user and the POI based on ENU coordinates.

[1] See https://en.wikipedia.org/wiki/ECEF and https://en.wikipedia.org/wiki/East_north_up

Algorithm 1: Transforming GPS coordinates into ECEF coordinates for user and POI

Data: lat: Latitude of user or POI
lon: Longitude of user or POI
alt: Altitude of user or POI

begin
/* Numeric constant $WGS84\_A$ constitutes the length, in kilometers, of the earth's semi-major axis. */
$N \leftarrow \frac{WGS84\_A}{\sqrt{1.0 - WGS84\_E^2 * sin(radian(lat))^2}}$; $x \leftarrow (N + alt) * cos(rad(lat)) * cos(rad(lon))$;
$y \leftarrow (N + alt) * cos(rad(lat)) * sin(rad(lon))$;
/* Numeric value $WGS84\_E^2$ constitutes the WGS-84 eccentricity squared value. */
$z \leftarrow (N * (1.0 - WGS84\_E^2) + alt) * sin(rad(lat))$;
end

Algorithm 2: Transforming ECEF coordinates into ENU coordinates for a POI

Data: lat, lon: Longitude and latitude of the GPS position of the user
x_p, y_p, z_p: ECEC coordinates of the user
x_r, y_r, z_r: ECEF coordinates of the POI
Result: ENU: East-North-Up (ENU) coordinates of POI

begin
$latCos \leftarrow cos(rad(lat))$; $latSin \leftarrow sin(rad(lat))$; $lonCos \leftarrow cos(rad(lon))$; $lonSin \leftarrow sin(rad(lon))$;
$vector \leftarrow (x_p - x_r, y_p - y_r, z_p - z_r)$;
$e \leftarrow (-lonSin) * vector[0] + lonCos * vector[1]$;
$n \leftarrow (-latSin) * lonCos * vector[x] + (-latSin) * lonSin * vector[1] + latCos * vector[2]$;
$u \leftarrow latCos + lonCos * vector[0] + latCos * lonSin * vector[1] + latSin * vector[2]$;
$ENU \leftarrow (e, n, u)$;
end

Algorithm 3: Calculating distance between user and POI

Data: $user$: User's GPS coordinates
poi: POI GPS coordinates

begin
$user.ECEF \leftarrow fromGpsToEcef(user.GPS)$; /* Transform GPS position of the user to ECEF. */
$poi.ECEF \leftarrow fromGpsToEcef(poi.GPS)$; /* Transform GPS position of the POI to ECEF. */
$poi.ENU \leftarrow fromEcefToEnu(poi.ENU, user.GPS, user.ECEF)$; /* Transform ECEF coordination of the POI to ENU. */
/* $poi.N^2$ constitutes the n component of $poi.ENU$, whereas $poi.E^2$ the e component. */
$poi.distance \leftarrow \sqrt{poi.N^2 + poi.E^2}$; /* Calculate distance between user and POI. */
/* Horizontal course is the angle between vector $\vec{NE}$ (e.g., POI is located at (27,9)) and the north vector (1,0). */
$poi.hCourse \leftarrow vec2angle(poi.EN, [1, 0])$; /* Calculate the horizontal course of the POI. */
/* Vertical course is the attitude difference of GPS attitudes of the user and the POI. */
$poi.vCourse \leftarrow getHeightAngle(poi.GPS, user.GPS)$; /* Calculate the vertical course of the POI. */
end

4. POI Algorithm

Although AREAv2 uses a virtual 3D world for displaying POIs, the direction in which a user holds his smart mobile device must be properly evaluated. For example, if the smart mobile device is hold obliquely, the POI shall be correctly positioned within the virtual 3D world. The correct positioning of POIs is ensured by Algorithm 4. Since Algorithm 4 depends on the algorithms that establish the coordinate system on one hand and is the base for the clustering algorithm on the other, Fig. 3 illustrates its dependencies.

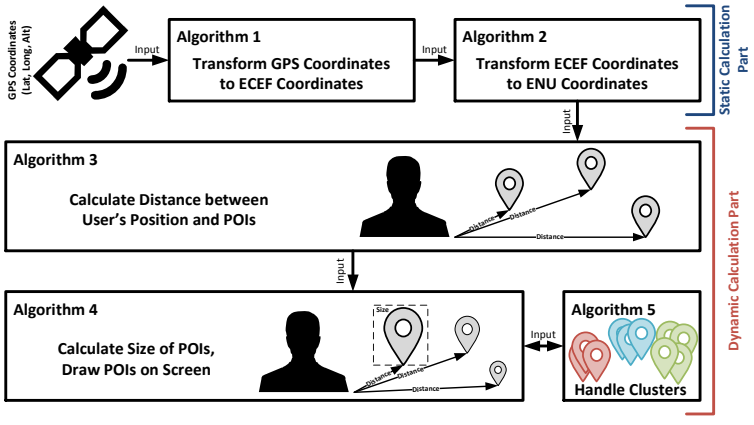

Fig. 3. Algorithm Dependencies

To be more precise, Algorithm 4 uses the following inputs: First, the list of POIs *poiList* (i.e., the ENU coordinates), locally stored on the smart mobile device, is used. Each time a user changes the position of his smart mobile device, all POI ENU coordinates are recalculated. Second, a rotation matrix *rotationMatrix RM* is used that manages relevant sensor data. On iOS, for example, this means the data of the gyroscope and accelerometer are used, whereas on Android the data of the gyroscope and accelerometer plus additional compass data are used. To be more precise, to obtain the device's attitude relative to true north as a rotation matrix, we simply use the *CMMotionManager* API provided by Apple iOS. On Android, however, we were not able to obtain any reliable data by the Android standard API. Hence, we decided to develop our own sensor fusion algorithm to obtain a similar rotation matrix like on iOS. Due to lack of space, the Android algorithm cannot be presented. Third, the *rotationMatrix RM* is used to adjust the virtual camera managed with the matrix *cameraView CM*. This matrix, in turn, is used to decide which POIs are actually shown on the camera view.

Algorithm 4 then works as follows[2]: A view called *areaview* is created and shown to the user. Next, each POI in *poiList* is created as a separate view. These POI views are placed on the *areaview* and are initally marked as invisible. They will be only shown to the user if Algorithm 4 indicates that they shall be visible (cf. Lines 9-15). Note that the entire view structure is precalculated and will not be changed afterwards by Algorithm 4. It makes POIs visible or invisible based on the changes by the position of the user. The position, in turn, is determined through the rotation matrix *rotationMatrix RM* (cf. Lines 2-8). Changes in *rotationMatrix RM* are evaluated up to 60 times per second. Hence, the precalculation of the view structure with respect to performance is indispensable.

Algorithm 4: Rendering pipeline with redraw up to 60 times per second

```
   Data: poiList, rotationMatrix RM, cameraView CM
1  begin
2      P ← CM · RM ;                                   /* Multiply camera matrix with rotation matrix to retrieve rotated camera projection matrix.  */
3      foreach poi ∈ poiList do
4          v⃗ ← [poi.ENU.E, poi.ENU.N, poi.ENU.U, 1] ;   /* Create homogeneous vector out of the POI's ENU coordinate.  */
5          v⃗ ← v⃗ · P ;                                 /* Multiplication of vector with projection matrix to project the position of the POI onto the camera view frustum.  */
6          x ← ( v⃗.x / v⃗.w + 1.0) ∗ 0.5 ;                /* Normalize vector components to 0...1 */
7          y ← ( v⃗.y / v⃗.w + 1.0) ∗ 0.5 ;                /* Normalize vector components to 0...1 */
8          z ← v⃗.z
9          if v⃗.z < −1 then
10             transformAndMovePOI(poi, x, y);          /* POI is located in front of the camera.  */
11             poi.visible = true;                      /* Position POI on the screen of the user and make it visible.  */
12         end
13         else
14             poi.visible = false
15         end
16     end
17 end
```

5. Cluster Algorithm

Algorithm 5 sketches the main calculation how POI clusters are handled. Algorithm 5 again uses the *poiList*. In addition, the two parameters *thHor* and *thVer* are used (cf. Algorithm 5) to evaluate all POIs in *poiList*. Note that they are manually specified by the mobile user and are used as follows: POIs that are inside an area spanned by *thHor* on the horizontal course and *thVer* on the vertical course (i.e., in the ENU coordinate system), will be regarded as POIs in the same cluster (cf. Algorithm 3). Finally, Fig. 4 illustrates how cluster handling (i.e., the screens marked with *activated*) is presented to the user. Note that the realized features for track and area handling are not presented due to space limitations.

6. AREAv2 in Practice

Table 2 illustrates examples of mobile applications that were developed on top of AREAv2. As can be seen, it is used for various scenarios in everyday life (cf. Table 2). Considering the huge number of realized mobile applications,

[2] Note that parts of the algorithm concept can be related to perspective transformation and clipping in the context of rendering pipeline in 3D computer graphics.

Algorithm 5: Handling Clusters

```
Data: poiList: List of surrounding pois of the user; thHor: Horizontal threshold; thVer: Vertical threshold
Result: clusteredPoisList: List of clusters and single POIs
1  begin
2      clusteredPoisList ← [];
3      while poiList not empty do
4          refPoi ← poiList[0]; poisToCluster ← []; poisToCluster.append(refPoi);
5          foreach poi ∈ poiList do
6              if refPoi ≠ poi then
7                  Δh ← 0
8                  if refPoi.hCourse ≤ poi.hCourse then
9                      Δh ← refPoi.hCourse − poi.hCourse;
10                 else
11                     Δh ← poi.hCourse − refPoi.hCourse;
12                 end
13                 if Δh ≤ −180 then
14                     Δh ← (Δh + 360) mod 360;
15                 else
16                     Δh ← |Δh|;
17                 end
18                 Δv ← |refPoi.vCourse − poi.vCourse|;
19                 if Δh ≤ thHor AND Δv ≤ thVer then
20                     poisToCluster.append(poi);
21                 end
22             end
23         end
24         if poisToCluster not empty then
25             clusterPoi ← Cluster(refPoi);
26             foreach poi ∈ poisToCluster do
27                 if poi ≠ refPoi then
28                     clusterPoi.addToCluster(poi); poiList.remove(poi);
29                 end
30             end
31             clusteredPoiList.append(clusterPoi);
32         else
33             clusteredPoiList.append(refPoi); poiList.remove(refPoi);
34         end
35     end
36 end
```

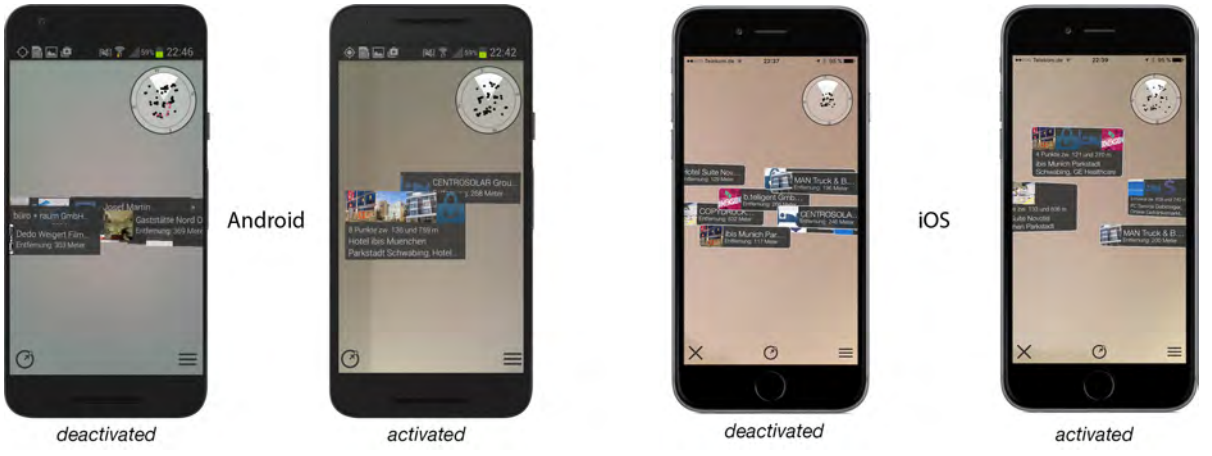

Fig. 4. Cluster Algorithm in Practice

AREAv2 has revealed a high practicability. The numbers of POIs integrated with the mobile applications vary among the different scenarios, but in all scenarios AREAv2 revealed same user experience.

7. Related Work

Previous research related to the development of a location-based augmented reality application in non-mobile environments is described in[4,5], in turn, uses smart mobile devices to develop an augmented reality system. Another application using augmented reality is described in[6]. Its purpose is to share media data and other information in a real-world environment and to allow users to interact with this data through augmented reality. However, none of

Table 2. AREA in Practice

Mobile applications using AREAv2	Categories	iOS	Android	#POIs	Clustering
Abfallinfo HOK	I	√	√	190	√
Altenahr	C	√	√	964	√
Goldpartner	F	√	√	205	√
Hinterzarten	C	√	√	297	√
Liveguide Muswiese	E	√	√	97	√
Renningen	C	√	√	1048	√

C=City Guide, E=Event Guide, F=Finance Guide, I=Infrastructure Guide, see http://www.liveguide.de for all mobile applications

these approaches share insights into the development of location-based augmented reality on smart mobile devices as in AREAv2. Only little work can be found dealing with the engineering of mobile augmented reality systems in general. As an exception,[7] validates existing augmented reality browsers. Moreover,[8] discusses various types of location-based augmented reality scenarios with respect to issues that have to be particularly considered for a specific scenario. However, the engineering issues of mobile applications is not considered.[9] proposes an authoring tool for mobile augmented reality applications based on marker detection. However,[9] discusses no engineering aspects. In[10] an approach is presented to support pedestrians with location-based mobile augmented reality. Altogether, neither software vendors nor research approaches provide insights into the way a location-based mobile augmented reality kernel can be developed.

8. Summary and Outlook

This paper gives insights into the development of the framework of an augmented reality kernel for smart mobile devices. It presents that mobile augmented reality is a complex endeavour that must be continuously improved. As a particular project experience, the concepts had to be evolved in order to keep pace with frequently changing requirements of mobile operating systems. In this context, also new functions like POI cluster handling are presented. However, the development of mobile applications is demanding with respect to the pecularities of the different mobile operating systems. Therefore, AREAv2 uses a modular architecture that supports mobile applications engineers. This paper further showed that business applications can be implemented using AREAv2. In future, we conduct performance tests that evaluate relevant indicators of AREAv2 systematically. Altogether, mobile augmented reality is one example that shows that mobile applications become more and more mature. On the other, suitable concepts are required towards the trend of mobile killer applications.

References

1. Schickler, M., Pryss, R., Schobel, J., Reichert, M.. An engine enabling location-based mobile augmented reality applications. In: *10th Int'l Conf on Web Information Systems and Technologies (Revised Selected Papers)*; no. 226 in LNBIP. Springer; 2015, p. 363–378.
2. Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M.. Location-based mobile augmented reality applications: Challenges, examples, lessons learned. In: *10th Int'l Conf on Web Information Systems and Technologies*. 2014, p. 383–394.
3. Geiger, P., Pryss, R., Schickler, M., Reichert, M.. Engineering an advanced location-based augmented reality engine for smart mobile devices. Technical Report UIB-2013-09; University of Ulm; 2013.
4. Kooper, R., MacIntyre, B.. Browsing the real-world wide web: Maintaining awareness of virtual information in an ar information space. *Int'l Journal of Human-Computer Interaction* 2003;**16**(3):425–446.
5. Kähäri, M., Murphy, D.. Mara: Sensor based augmented reality system for mobile imaging device. In: *5th IEEE and ACM Int'l Symp on Mixed and Augmented Reality*; vol. 13. 2006, .
6. Lee, R., Kitayama, D., Kwon, Y., Sumiya, K.. Interoperable augmented web browsing for exploring virtual media in real space. In: *Proc of the 2nd Int'l Workshop on Location and the Web*. ACM; 2009, p. 7.
7. Grubert, J., Langlotz, T., Grasset, R.. Augmented reality browser survey. Technical Report; Graz University of Technology; 2011.
8. Kim, W., Kerle, N., Gerke, M.. Mobile augmented reality in support of building damage and safety assessment. *Natural Hazards and Earth System Sciences* 2016;**16**(1):287.
9. Yang, Y., Shim, J., Chae, S., Han, T.. Mobile augmented reality authoring tool. In: *10th IEEE Int'l Conf on Semantic Computing*. IEEE; 2016, p. 358–361.
10. Chung, J., Pagnini, F., Langer, E.. Mindful navigation for pedestrians: Improving engagement with augmented reality. *Technology in Society* 2016;**45**:29–33.