Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2019) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC)
August 19-21, 2019, Halifax, Canada

The AREA Algorithm Framework Enabling Location-based Mobile Augmented Reality Applications

Marc Schickler[a,*], Manfred Reichert[a], Philip Geiger[a], Micha Weilbach[a], Rüdiger Pryss[a]

[a] *Ulm University, Institute of Databases and Information Systems, James-Franck-Ring 1, Ulm, 89081, Germany*

Abstract

The dramatically increased computational capabilities of mobile devices have leveraged the opportunities for mobile application engineers. Respective scenarios, in which these opportunities can be exploited, emerge almost per day. In this context, mobile augmented reality applications play an important role in many business scenarios. In the automotive domain, they are mainly used to provide car customers with new experiences. For example, customers can use their own mobile device to experience the interior of a car by moving the mobile device around. The device's camera then detects interior parts and shows additional information to the customer within the camera view. Although the computational capabilities have been increased, the realization of such mobile augmented reality applications is still a complex endeavor. In particular, the different mobile operating systems and their peculiarities must be carefully considered. In the AREA (Augmented Reality Engine Application) project, a powerful kernel was realized that enables location-based mobile augmented reality applications. This kernel, in turn, mainly focuses on robustness and performance. In addition, it provides a flexible architecture that fosters the development of individual location-based mobile augmented reality applications. As many aspects have to be considered to implement individual applications based on top of AREA, this paper provides the first comprehensive overview of the entire algorithm framework. Moreover, a recently realized algorithm and new features will be presented. To demonstrate the applicability of the kernel, its features are applied in the context of various mobile applications. As the major lesson learned, powerful mobile augmented reality applications can be efficiently run on present mobile operating systems and be effectively realized by engineers using AREA. We consider such mobile frameworks as being crucial to provide more generic concepts that are able to abstract from the peculiarities of the underlying mobile operating system and to support mobile application developers more properly.

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

* Corresponding author. Tel.: +49-731-50-24-230 ; fax: +49-731-50-24-134.
E-mail address: marc.schickler@uni-ulm.de

1. Introduction

Mobile augmented reality applications become more and more useful for many existing IT solutions used in everyday life [29, 16]. However, the development of this application type in a robust and reliable manner is still challenging. Most importantly, the peculiarities of the different mobile operating systems must be properly addressed, including their frequent update cycles. In addition, mobile device vendors continuously enhance the technical capabilities of their products (e.g., by adding new sensors). As these enhancements often enable new applications or ease the implementation of existing applications, it is compelling for the mobile application developers to learn more about any changes very quickly. Therefore, generic concepts are highly welcome to abstract from the peculiarities of the underlying mobile operating system in the best possible way. In the *AREA* (Augmented Reality Engine Application) project, a kernel was developed that enables location-based mobile augmented reality applications. *AREA* considers the aforementioned aspects and is based on four fundamental pillars. First, the kernel shall provide the same features on Android and iOS. Second, the kernel shall enable robust and powerful mobile applications. Third, it shall be easily possible to integrate new features into the kernel. Fourth, mobile application developers shall be enabled to easily create their own location-based mobile augmented reality applications on top of *AREA*. The latter aspect is enabled through a modular design of the *AREA* kernel. Note that in-depth information to the modular design principles of *AREA* can be found in [24, 25]. In addition, the latter works, as well as the works shown in [30, 7, 6, 26], discuss in what way *AREA* deals with the peculiarities of the different mobile operating systems. This work, in turn, contributes with respect to three other aspects. It is the first work that provides a comprehensive overview of all algorithms developed in *AREA*. As the second contribution, it discusses a new feature that was recently realized on top of the existing algorithms. Third, *AREA* will be compared to ARKit. The latter was recently released by Apple and pursues the same goal as *AREA*: Mobile application developers shall be enabled to easily create their own applications on top of ARKit. Therefore, we compare the functions of ARKit with *AREA*.

The remainder of this paper is organized as follows. Section 2 presents an overview of the algorithm framework, while Section 3 discusses a feature that enables an optimization of tracks and areas. In Section 4, the comparison of *AREA* with ARKit is presented, and in Section 5, the practical use of *AREA* and its features based on the presented results is summarized. Section 6 discusses related work, whereas Section 7 concludes the paper with a summary and an outlook.

2. AREA Algorithm Framework

First of all, we give a brief introduction into the concept of AREA [6, 25]. AREA relates users holding their smartphone to the objects (i.e., Points of Interest (POIs), tracks, areas) detected in the camera view. Thereby, AREA is based on five aspects. First, a virtual 3D world is used to relate the user's position to the position of the objects. 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 characteristics of the supported mobile operating systems are covered to enable the virtual 3D world. Fifth, the physical camera of the mobile device is adjusted to the virtual 3D camera, based on the assessment of sensor data. To enable these five aspects, an overview of the algorithms that were developed for *AREA* will be provided (see Fig. 1). In a first version of *AREA* [25], a Points of Interest (POI) algorithm was developed that showed already considerable results (see Fig. 1, *POI Algorithm v1*). More detailed information on this algorithm can be found in [30, 7, 6]. However, as the computational capabilities of smart mobile devices have been continuously increased – and the practical requirements of the real-life projects as well –, a new POI algorithm became necessary (see Fig. 1, *POI Algorithm v2*). All presentations in this work are based on this currently used POI algorithm. Note that the technical aspects of the *POI Algorithm v2* can be found in [24, 25]. Furthermore, in Fig. 1, all references are shown, in which the respective information of the algorithms (e.g., their listings) and backgrounds can be found. As this work provides the first comprehensive overview of all algorithms developed on top of the POI algorithms, the following list summarizes the important aspects as well as new features shown in this work:

- The calculations to position POIs correctly are based on a multitude of other calculations (i.e., mainly the sensor fusion) and design decisions (i.e., mainly whether or not using external libraries). In this context, the correct

positioning of POIs constitutes a major challenge. On the other, the provision of a good performance is also very important. When considering preciseness and performance on different mobile operating systems in the same way, the overall technical endeavor is even more challenging. For example, on Android, for the sensor fusion, an additional 3D rotation matrix algorithm [25] became necessary to enable the same user experience as on iOS (see Fig. 1, *only on Android*).

- On top of the POI algorithms, two additional algorithms were implemented. **The first algorithm is able to handle clusters**. Clusters, in turn, are overlapping POIs, which are difficult to interact with. How clusters are handled is presented in [25]. **The second algorithm is able to calculate tracks and areas**. The demand for this feature emerged while using *AREA* in practice. For the development of the algorithm that is able to handle tracks and areas, note that OpenGL libraries were used in addition. The decision was made with respect to the differences of the two mobile operating systems. In this context, we wanted to combine the (1) best of the already proven *AREA* developments and the (2) existing features of OpenGL. More specifically, the proven sensor fusion and POI algorithms of *AREA* should be further used as they revealed comparable results on both mobile operating systems. In addition, as the OpenGL libraries are decoupled from the sensor fusion with respect to general functions required for track and area handling, it was efficient to additionally use these features of OpenGL for *AREA*. Therefore, we also used OpenGL libraries for track and area handling. In-depth information to this algorithm can be found in [26].
- In practice, even when considering the existing powerful mobile device capabilities, the handling of tracks and areas is challenging with respect to the resource perspective. When handling a huge number of tracks or areas, or a combination of both, present mobile devices reveal their limits. Therefore, we implemented a new algorithm, which deals with performance issues while displaying many tracks and areas at the same time. How this algorithm works will be presented in Section 3).
- Except for the new algorithm that deals with the performance while displaying many tracks and areas, all other algorithms were evaluated in experiments (i.e., compared to other mobile augmented reality applications). As can be found in [25, 26], *AREA* competes well with other smart mobile applications providing the same features. However, in future experiments, this will be evaluated again and again. Moreover, the new algorithm for track and area handling must be evaluated in a separate experiment.
- Since Apple recently released its ARkit [2], a comparison of *AREA* with ARkit is a must in future experiments.

3. Track and Area Reduction Algorithm

In real-life projects[1] for which *AREA* is used for, the displaying of many tracks and areas with the algorithm shown in [26] revealed performance issues. Therefore, we implemented a feature on top of this algorithm in order to cope with the demanding scenarios. Here, due to the lack of space, we only show the optimization for Android and tracks; i.e., areas and the implementation on iOS are performed in the same way. First of all, we explain necessary background information on the performance indicators. In general, a track is displayed by the use of bars. Between each bar, a distance of 1m (m=meter) is used. Having this in mind, a track of 1km (km=kilometer) requires roughly 501 bars. Each bar, in turn, is represented by two triangles. Each vertex of a triangle has 3 coordinates (x, y, and z) and a RGBA value (i.e., r, g, b, and a components). The three coordinates as well as the 4 RGBA components require 4 bytes. Having these values in mind, reconsider the track of 1km. This track would require $501 \cdot 2 \cdot 3 \cdot (3+4) \cdot 4 = 84168$ *bytes* to store it. If many tracks or areas shall be displayed, this affects the performance based on the required data to be stored and calculated. To increase the performance in the case that many tracks have to be displayed (or/and areas), the general idea is to manage a **detail level** for tracks (and areas). Based on this detail level, tracks and areas can be displayed in different resolutions. The notion of the resolution and how it is calculated are shown in the following.

First of all, we present required preliminary calculations. As a first step, consider a list *checkpoints* containing n vectors (x, y, and z). Each vector n in *checkpoints* represents on point on the track that shall be displayed. Furthermore, the *checkpoints* list stores the values in an ordered manner according to a track. In addition, three further lists are

[1] see http://www.liveguide.de for all mobile applications that use AREA

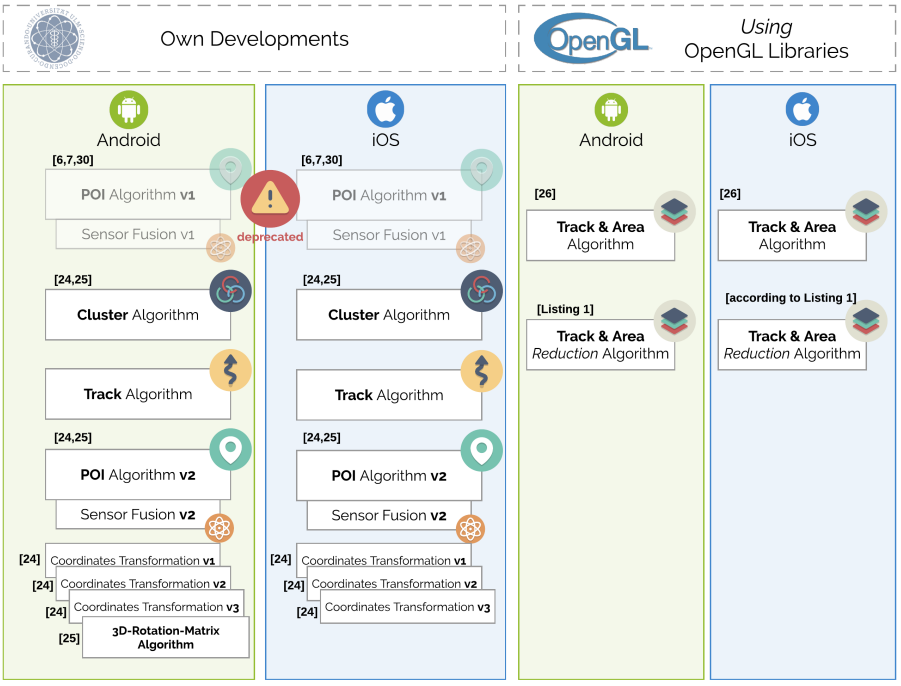

Fig. 1. AREA Algorithm Framework

managed: *degreesY*, *degreesXZ*, and *pairs*. Each of these lists stores $n-2$ values. More specifically, the values of the three lists store the following:

- *degreeY*: stores the angle of a track point B that lies between points A and C. More precisely, it stores the height difference between A and C, based on point B.
- *degreeXZ*: stores the angle of a track point B that lies between points A and C. More precisely, it stores the cardinal points between A and C, based on point B.
- *pairs*: stores the indexes of points $A, B, and\ C$.

To determine *degreesY* and *degreesXZ*, the following calculations are applied:

- *degreesY*: To calculate the angle for a point B, the two points B' and B'' are calculated. Thereby, B' contains (xb, ya, zb) and B'' contains (xb, yc, zb). Following this, B' holds the y-value of the point A and B'' the y-value of the point C. Based on this, the two rectangular triangles $A-B'-B$ and $B-B''-C$ can be created. Finally, the sum of triangles between $A-B'-B$ and $B-B''-C$ results in one entry *degreesY*.
- *degreesXZ*: To calculate all values for *degreesXZ*, the vectors AB and AC are calculated.

Based on these shown lists, the size of a track can be decreased with respect to so-called detail levels. A detail level reduces tracks (and areas) to x track points. Reduction means that the originally defined amount of track points n is reduced to x. The reduction, in turn, is calculated as follows:

- The lists *degreesY*, *degreesXZ*, and *pairs* are calculated for a track that shall be minimized.
- Then, within a loop, all values in *degreesY* and *degreesXZ* are evaluated whether they will be in the list for x. If x points have been identified, the loop will be quit. The next steps show how the evaluation is performed.
- First, a variable *steps* is initialized with 1 (meaning 1 degree). Thereby, *steps* is a threshold that must be exceeded when calculating $|180-degreesY[i]|+|180-degreesXZ[i]|$ for each entry in the lists *degreesY* and *degreesXZ*. If the calculation exceeds the value of *steps*, the entry will be used for x.

- Visually speaking: The more the triangle between two points approaches 180 degrees, the more it approaches a straight line. Consequently, the elimination of such a point can be visually accepted.
- If no value can be found for within a loop run that can be eliminated, then *steps* is increased to 2 (and so on).
- If a value can be found at index i of the lists *degreesY* and *degreesXZ*, then they are recalculated as follows: $degreesY[i-1]$, $degreesY[i+1]$, $degreesXZ[i-1]$, $degreesXZ[i+1]$, $pairs[i-1]$, and $pairs[i+1]$ are newly calculated and the entries for the index i are removed.
- If the initial lists *degreesY*, *degreesXZ*, and *pairs* are decreased to x points, the algorithm is finished.
- The list of x points is then displayed using the algorithm shown in [26]. All other relevant calculations for a further understanding can be found in [6, 25].

The implementation of the algorithm to reduce the number of track points is shown in Listing 1. Note that the listing only shows the Android version (the iOS version works accordingly).

Listing 1. Track Reduction Algorithm (Android version)

```
int[] complexitySteps = {50, 40, 30, 20, 10, 5};
int[] distanceSteps = {100, 200, 300, 400, 500};
int[][] complexityLvl = new int[7][];

abstractTrail(){
   complexityLvl[0] = new int[totalPath.size()];
   for(int i = 0; i < totalPath.size()-1; i++){
      complexityLvl[0][i] = i;
   }

   float[] orientationDegree = new float[totalPath.size()-2];
   float[] horizontalDegree = new float[totalPath.size()-2];
   int[][] pairs = new int[totalPath.size()-2][2];

   * fill orientationDegree, horizontalDegree, pairs
   * orientationDegree[i] and horizontalDegree stores the degree
   * between points (i-1),(i) and (i+1)
   * pairs[i][] stores point (i-1) and (i+1)

   int actualCount = orientationDegree.length;
   int lastAbstraction = 0;
   float actualAbstraction = 1;

   for(int i = 0; i < orientationDegree.length; i++){
      for(int j = 0; j < complexitySteps.length; j++){
         if(actualCount <= complexitySteps[j] && complexityLvl[j+1] == null)
            * fill complexityLvl[j+1]
      }

      float check0 = min(abs(180 - orientationDegree[i]), abs(180 + orientationDegree[i]));
      float check1 = min(abs(180 - horizontalDegree[i]), abs(180 + horizontalDegree[i]));
      if(check0 and check1 < actualAbstraction){
         orientationDegree[i] = 999;
         horizontalDegree[i] = 999;

         * recalculate pairs[i-1], pair[i+1]
         * recalculate orientationDegree[i-1] and [i+1]
         * recalculate horizontalDegree[i-1] and [i+1]

         lastAbstraction = i;
         actualCount--;
         if(i == orientationDegree.length){ i = 0; }
      }
   }

   //complexity[7][]-array is managed as follows:
   //complexityLvl[0] contains the indexes to all points [i.e., -> 0, 1, 2, ..., totalPath.size()-1]
   //complexityLvl[1] contains indexes from maximally 50 points,
   //complexityLvl[2] contains indexes from maximally 40 points,
   //complexityLvl[3] contains indexes from maximally 30 points,
   //..
   //complexityLvl[6} contains indexes from maximally 5 points.

   //which complexityLvl then is actually used depends on the distance from the user to
   //the nearest point of the bounding box of the track
   //the bounding box must be calcuted beforehand while creating the list totalPath
   //the array distanceSteps stores the distances to decide which complexityLvl is actually used.
}
```

3.1. Track Reduction in Practice

AREA manages **seven** different detail levels. To be more specific, the tracks (and areas) are displayed using these levels depending on the distance a user has to them. The detail levels are distinguished by the number of track points to which the track is reduced to by Algorithm 1. The levels, in turn, are managed as follows:

- Level 0: No reduction
- Level 1: Tracks within 50m of a user's position (i.e., using 50 track points)

- Level 2: Tracks within 100m of a user's position (i.e., using 40 track points)
- Level 3: Tracks within 200m of a user's position (i.e., using 30 track points)
- Level 4: Tracks within 300m of a user's position (i.e., using 20 track points)
- Level 5: Tracks within 400m of a user's position (i.e., using 10 track points)
- Level 6: Tracks beyond 400m of a user's position (i.e., using 5 track points)

Which level is actually used is determined during run time based on the position changes of a user. The algorithm is used in practice in real-life applications that can be found at [5]). Currently, we conduct experiments (as shown in [25]) to obtain quantitative results on the actual performance increase.

4. AREA and ARKit

Since Apple recently released its ARkit [2], this section shall enable researchers to directly compare the features and function of the iOS version of *AREA* with ARkit. In general, ARkit was developed by Apple with the goal to provide a multitude of mobile augmented reality applications. Currently, all features developed in *AREA* pursue the goal to enable location-based mobile augmented reality applications. To be more precise, the location is based on GPS coordinates and therefore *AREA* mainly aims at outdoor location-based augmented reality applications. In ARkit, also many other features like face tracking are provided. Consequently, ARkit aims at a broader perspective on mobile augmented reality applications. However, from the technical perspective, it might be of interest to directly compare the functions of the iOS version of *AREA* with ARkit. In ARkit, the following chain of classes must be used to enable a location-based mobile augmented reality experience: $ARSession- > ARFrame- > ARCamera$. Thereby, $ARSession$ must be used to handle the sensor fusion, while $ARFrame$ and $ARCamera$ must be used to handle the positioning of POIs. Regarding $ARSession$, compared to *AREA*, a developer must manually add GPS data to the sensor fusion. With ARkit, compared to *AREA*, a developer is relieved from directly reading data from the device's motion sensing hardware. Another interesting comparison is related to *ARCamera* of ARKit. By using *ARCamera*, the correct positioning of POIs can be realized. Therefore, the relevant components of *ARCamera* [1] can be compared to the iOS version of *AREA* as follows:

- ARKIT $func\ transform$ with *AREA* $func\ normalizedRotationMatrix$
- ARKIT $func\ projectionMatrix$ with *AREA* $func\ redrawAreaView$
- ARKIT $func\ projectPoint$ with *AREA* $func\ positioningPOI$

Currently, we conduct performance experiments to compare ARKit with *AREA*. In general, the provisioning of ARKit emphasizes that mobile augmented reality has become an important mobile application type. In line with ARKit, the application of *AREA* in practice revealed that features enabling mobile augmented applications beyond location-based outdoor scenarios are promising. Therefore, we work on new features like, for example, the recognition of objects in *AREA*. Furthermore, we currently compare *AREA* with the AR SDK from Google for Android, which is called ARCore [9].

5. Discussion

Currently, *AREA* is used in various scenarios in everyday life [5]. Three aspects are particularly important for this frequent usage. First, the algorithm framework shown in this work (see Fig. 1) was bundled into the *AREA* kernel, including its modular architecture [25, 26]. Based on this, the development of business applications on top of *AREA* becomes easily possible. Second, *AREA* reveals a good user experience with respect to robustness and performance. Experiments conducted with *AREA* [25, 26] confirm that it is competitive to mobile applications that provide the same or a similar feature set at the time of conducting the experiment. Third, *AREA* provides the same feature set on Android and iOS. The ability to cope with the peculiarities of the different mobile operating systems, while providing the same features on all of these mobile systems, is highly welcome in practice. However, to keep pace with the frequent updates of the underlying mobile operating systems on one hand and to continuously implement new features that emerge in practice on the other, is still a very challenging endeavor. Therefore, insights into frameworks and operating principles

as shown in this work are of utmost importance. Finally, in a future experiment, *AREA* must show its performance compared to ARkit and ARCore.

6. Related Work

Previous research related to the development of a location-based augmented reality application in non-mobile environments is described in [15]. In turn, [13] uses smart mobile devices for developing an augmented reality system. The augmented reality application described in [17] allows sharing media data and other information in a real-world environment and enables users to interact with this data through augmented reality. However, none of these approaches share insights regarding the development of location-based augmented reality on smart mobile devices as *AREA* does. Regarding tracks in mobile augmented reality, only little work can be found. For example, the approaches [32, 18, 11] present tracks as key feature of (mobile) augmented reality applications. However, algorithms for implementing track handling are not presented. In addition, no performance issues related to track algorithms are discussed. Moreover, only little work exists, which deals with the engineering of mobile augmented reality systems in general. As an exception, [10] validates existing augmented reality browsers. Moreover, [14] discusses various types of location-based augmented reality scenarios. More precisely, issues that have to be particularly considered for a specific scenario are discussed in more detail. However, engineering issues of mobile applications are not considered. In [33], an authoring tool for mobile augmented reality applications, which is based on marker detection, is proposed. In turn, [23] presents an approach for indoor location-based mobile augmented reality. Furthermore, [28] gives an overview of various aspects of mobile augmented reality for indoor scenarios. Another scenario for mobile augmented reality is presented in [19]. The authors use mobile augmented reality for image retrieval. However, [33, 23, 28, 19] do not address engineering aspects of location-based mobile applications. In [4], an approach supporting pedestrians with location-based mobile augmented reality is presented. Finally, [3] deals with a client and server framework enabling location-based applications. Presently, new scenarios emerge, in which mobile augmented reality is investigated. Recent related works can be found that provide overviews (e.g., [29]). Other related works deal with particular scenarios. In [22, 20, 31], many examples related to education are discussed and evaluated. Other scenarios constitute tourism [12], crime management [21], or gaming [27]. Finally, also in medicine, mobile augmented reality becomes increasingly important [8]. Many more recent works could be mentioned. However, how to realize these applications from scratch is not covered by these works. Altogether, neither software vendors nor research projects provide many in-depth insights into the engineering of a location-based mobile augmented reality kernel and the development of a feature set as provided in this work.

7. Summary and Outlook

This paper provided insights into the development of the *AREA* framework. To be more precise, a comprehensive overview of the algorithms to enable location-based mobile augmented reality applications were presented. Regarding the entire *AREA* project, this is the first work that provides a comprehensive overview of all implemented *AREA* algorithms. In general, the development of mobile applications is demanding when considering the peculiarities of the different mobile operating systems. To cope with this heterogeneity, *AREA* provides the same functionality for business applications that are developed based on top of it. This is enabled by enclosing all features in the *AREA* kernel. Application developers can use it like the ARKit from Apple or the SDK ARCore from Google to easily create their own location-based mobile augmented reality applications. This paper also presented a new algorithm that was realized on top of the track and area algorithm, which provides a better performance if a huge number of tracks or areas shall be displayed at the same time. It was also presented that *AREA* has been evaluated in experiments [25, 26]. These experiments have shown that *AREA* reveals considerable performance results compared to other mobile augmented reality applications providing a similar feature set. Further, it was reported that currently conducted experiments investigate how *AREA* competes with ARKit and ARCore. Another experiment investigates the new feature that was developed on top of the track and area algorithm. Altogether, mobile augmented reality applications support many new scenarios. However, powerful solutions that can be easily used across the different mobile operating systems in the same way are still rare.

References

[1] Apple, 2019a. Arcamera. https://developer.apple.com/documentation/arkit/arcamera. [Online; accessed 30-April-2019].

[2] Apple, 2019b. Arkit. https://developer.apple.com/arkit/. [Online; accessed 30-April-2019].

[3] Capece, N., Agatiello, R., Erra, U., 2016. A client-server framework for the design of geo-location based augmented reality applications, in: 20th Int'l Conf on Information Visualisation, IEEE. pp. 130–135.

[4] Chung, J., Pagnini, F., Langer, E., 2016. Mindful navigation for pedestrians: Improving engagement with augmented reality. Technology in Society 45, 29–33.

[5] CMCityMedia, 2019. Liveguide. http://www.stadtsindwir.de/data/referenzen.php. [Online; accessed 30-April-2019].

[6] Geiger, P., Pryss, R., Schickler, M., Reichert, M., 2013. Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices. Technical Report UIB-2013-09. University of Ulm.

[7] Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M., 2014. Location-based mobile augmented reality applications: Challenges, examples, lessons learned, in: 10th Int'l Conf on Web Information Systems and Technologies, pp. 383–394.

[8] Ghandorh, H., Mackenzie, J., Eagleson, R., de Ribaupierre, S., 2017. Development of augmented reality training simulator systems for neurosurgery using model-driven software engineering, in: 30th Canadian Conference on Electrical and Computer Engineering, IEEE. pp. 1–6.

[9] Google, 2019. Arcore. https://developers.google.com/ar/. [Online; accessed 30-April-2019].

[10] Grubert, J., Langlotz, T., Grasset, R., 2011. Augmented reality browser survey. Technical Report. Graz University of Technology.

[11] Hollerer, T., 2004. User interfaces for mobile augmented reality systems. Ph.D. thesis. Columbia University.

[12] Jung, T., Lee, H., Chung, N., et al., 2018. Cross-Cultural Differences in Adopting Mobile Augmented Reality at Cultural Heritage Tourism Sites. International Journal of Contemporary Hospitality Management 30.

[13] Kähäri, M., Murphy, D., 2006. Mara: Sensor based augmented reality system for mobile imaging device, in: 5th IEEE and ACM Int'l Symp on Mixed and Augmented Reality.

[14] Kim, W., Kerle, N., Gerke, M., 2016. Mobile augmented reality in support of building damage and safety assessment. Natural Hazards and Earth System Sciences 16, 287.

[15] Kooper, R., MacIntyre, B., 2003. Browsing the real-world wide web: Maintaining awareness of virtual information in an ar information space. Int'l Journal of Human-Computer Interaction 16, 425–446.

[16] Korinth, M., et al., 2020. Design and Evaluation of a Virtual Reality-Based Car Configuration Concept, in: Advances in Computer Vision, Springer. pp. 169–189.

[17] Lee, R., Kitayama, D., Kwon, Y., Sumiya, K., 2009. 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. p. 7.

[18] Lee, T., Hollerer, T., 2008. Hybrid feature tracking and user interaction for markerless augmented reality, in: Virtual Reality Conference, IEEE. pp. 145–152.

[19] Lee, Y., Rhee, S., 2015. Efficient photo image retrieval system based on combination of smart sensing and visual descriptor. Intelligent Automation & Soft Computing 21, 39–50.

[20] Leighton, L., Crompton, H., 2017. Augmented Reality in K-12 Education, in: Mobile Technologies and Augmented Reality in Open Education. IGI Global, pp. 281–290.

[21] Liao, T., Yang, H., Lee, S., Xu, K., Feng, P., Bennett, S., 2017. Augmented Criminality–How Mobile Augmented Reality Crime Overlays Affect People's Sense of Place. AoIR Selected Papers of Internet Research 6.

[22] Liou, H., Yang, S., Chen, S., Tarng, W., 2017. The influences of the 2D image-based augmented reality and virtual reality on student learning. Journal of Educational Technology & Society 20, 110–121.

[23] Paucher, R., Turk, M., 2010. Location-based augmented reality on mobile phones, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, IEEE. pp. 9–16.

[24] Pryss, R., Geiger, P., Schickler, M., Schobel, J., Reichert, M., 2016. Advanced algorithms for location-based smart mobile augmented reality applications. Procedia Computer Science 94, 97–104.

[25] Pryss, R., Geiger, P., Schickler, M., Schobel, J., Reichert, M., 2017a. The AREA Framework for Location-Based Smart Mobile Augmented Reality Applications. International Journal of Ubiquitous Systems and Pervasive Networks 9, 13–21.

[26] Pryss, R., Schickler, M., Schobel, J., Weilbach, M., Geiger, P., Reichert, M., 2017b. Enabling Tracks in Location-Based Smart Mobile Augmented Reality Applications. Procedia Computer Science 110, 207–214.

[27] Rauschnabel, P., Rossmann, A., tom Dieck, M., 2017. An adoption framework for mobile augmented reality games: The case of Pokémon Go. Computers in Human Behavior 76, 276–286.

[28] Reitmayr, G., Schmalstieg, D., 2003. Location based applications for mobile augmented reality, in: Proc of the Fourth Australasian user interface conference on User interfaces, Australian Computer Society, Inc.. pp. 65–73.

[29] Sánchez-Acevedo, M., Sabino-Moxo, B., Márquez-Domínguez, J., 2017. Mobile Augmented Reality. Mobile Platforms, Design, and Apps for Social Commerce , 153.

[30] Schickler, M., Pryss, R., Schobel, J., Reichert, M., 2015. An engine enabling location-based mobile augmented reality applications, in: 10th Int'l Conf on Web Information Systems and Technologies (Revised Selected Papers). Springer. number 226 in LNBIP, pp. 363–378.

[31] Tosun, N., 2017. Augmented Reality Implementations, Requirements, and Limitations in the Flipped-Learning Approach, in: Mobile Technologies and Augmented Reality in Open Education. IGI Global, pp. 262–280.

[32] Vlahakis, V., Karigiannis, J., Tsotros, M., Ioannidis, N., Stricker, D., 2002. Personalized augmented reality touring of archaeological sites with wearable and mobile computers, in: Sixth International Symposium on Wearable Computers, IEEE. pp. 15–22.

[33] Yang, Y., et al., 2016. Mobile augmented reality authoring tool, in: 10th Int'l Conf on Semantic Computing, IEEE. pp. 358–361.