

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/333039077>

Capabilities of ARCore and ARKit Platforms for AR/VR Applications

Chapter · January 2020

DOI: 10.1007/978-3-030-19501-4_36

CITATIONS

26

READS

10,241

2 authors, including:



[Marek Sławomir Woda](#)


Wrocław University of Science and Technology

47 PUBLICATIONS 109 CITATIONS

[SEE PROFILE](#)



Capabilities of ARCore and ARKit Platforms for AR/VR Applications

Paweł Nowacki and Marek Woda^(✉) 

Department of Computer Engineering, Wrocław University of Technology,
Janiszewskiego 11-17, 50-372 Wrocław, Poland
nowackipawel@yahoo.com, marek.woda@pwr.edu.pl

Abstract. In this paper ARCore and ARkit capabilities were scrutinized and compared. Authors established comparison criteria for both platforms, developed test applications and ran comparison tests. Obtained results can be a help in choosing the right framework to speed up prototyping and development of modern AR/VR applications. This work consists of a comprehensive comparison of these new frameworks in the following respects: general performance (CPU/memory use), mapping of planes on various surface types, influence of light and movement on mapping quality etc.

Keywords: Comparison · ARKit · ARCore · Augmented reality · Virtual reality

1 Introduction

With the development of computer technology, new methods of user interaction with the computer and data presentation appear. Such methods include, among others, Augmented Reality (AR) and virtual reality (VR). Both methods of data presentation differ mainly in the ratio of computer-generated images to the real world. In a nutshell, AR [5] is when the real world is enriched with data generated by the computer, i.e. there are connections between the real and virtual world. VR [9] occurs when computer generated data completely obscures the real world.

Until recently, these methods required efficient computers and specialized equipment (GPUs) [12], however, thanks to the rapid development of mobile technology, millions of users have gained access to powerful mobile devices [10] capable to deal with AR and VR in almost real-time. In 2017, Apple and Google - two giants of the mobile industry introduced two competitive application programming interfaces supporting the creation of augmented reality applications for mobile devices: *ARKit* (September 19, 2017) and *ARCore* (stable release March 1st, 2018), giving users of iOS and Android devices new opportunities to create immersive applications and games.

According to a forecast [11], VR and AR markets are expected increase almost tenfold to reach the size over 209 billion USD in 2021, from a mere 27 billion in 2018. Many programmers are looking for methods and tools to simplify a complex process of building realistic AR and VR applications. There are several frameworks available that facilitate rapid prototyping and development of AR/VR apps. Many portals [1, 6, 7, 13]

recommend different solutions, amongst the most popular ones are: *ARCore*, *ARKit*, *ARToolkit*, *Kudan*, MAXST Wikitude. Besides the fact that *ARCore* and *ARKit* are free of charge, offer plethora of features which were previously only available only in commercial (paid) version of competitive SDKs. *ARCore* and *ARKit* though new ones, constantly evolving, they already caught attention of market (Fig. 1).

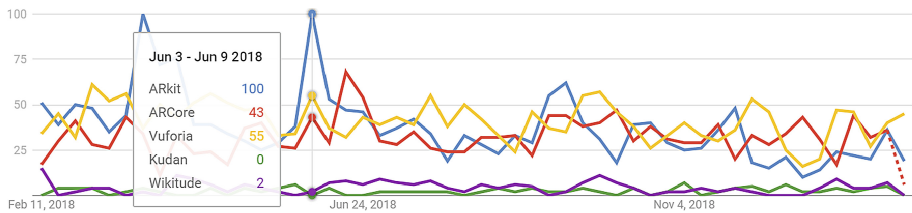


Fig. 1. Worldwide interest (Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. A score of 0 means there was not enough data for this term.) – popular AR/VR frameworks (source trends.google.com)

Even though there are several publications available [2, 8, 9, 14, 15] related to use of newcomers on AR/VR market, it is hard to draw any conclusion which one a worthy competitor is, in comparison to existing commercial solutions, not saying about differences between themselves. Authors in this paper wanted to address the question how good (or bad) new frameworks are, and what are the differences in between them.

2 Comparison Criteria and Test Methodology

To be able to compare capabilities of both frameworks there were developed two applications (iOS and Android) that were taking advantage of given features. Both apps provided following functionalities: detection flat surfaces, imaging of detected flat surfaces/special points, positioning any number of 3D objects on stage, choosing scene lighting based on the actual lighting conditions, support for shadows cast generation by virtual objects, measure the distance between 2 points, saving information about detected planes (and special points) to a file, measurement and display of frames per second during operation, measurement and save information on app startup time. Thanks to the above-mentioned features, it was possible to create a base for tests of both frameworks, and finally presentation of strengths and weaknesses. Apps were tested on several devices (Table 1).

During the evaluation of both frameworks following comparison criteria were established:

- General performance
 - Time to load models, initialize, run application and camera
 - CPU load and memory usage during use of app (with 100 and w/o models)
- “Understanding” the surroundings

Table 1. Test devices – comparison sheet

Device	Processor	Cores	RAM [GB]
<i>ARKit</i>			
iPhone X	Apple A11 Bionic	6	3
iPhone 8	Apple A11 Bionic	6	2
iPhone 7plus	Apple 10 Fusion	4	2
iPhone 7plus	Apple 9	2	2
iPad Pro (2017)	Apple A10X Fusion	6	4
iPad (5th Gen)	Apple A9	2	2
<i>ARCore</i>			
Google Pixel 2XL	Qualcomm Snapdragon 835	8	4
Google Pixel	Qualcomm Snapdragon 821	4	4
Google Nexus 6P	Qualcomm Snapdragon 810	8	4
Google Nexus 5X	Qualcomm Snapdragon 808	6	2
Samsung Galaxy S9	Samsung Exynos 9810	8	4
Samsung Galaxy S8+	Samsung Exynos 8895	8	4
Samsung Galaxy S7	Samsung Exynos 8890	8	4

One of the key elements of augmented reality is the “understanding” by the application of the surrounding space. One of the components is surface detection. Well-mapped surfaces allow for realistic placement of virtual objects in real space. The quality of the surface detection itself translates to the quality of the entire application. If the accuracy is too low virtual models will hang in the air, move, or appear in places where it would be impossible in the real world. Such phenomena spoil the general impression and the comfort of using the application. In addition to accuracy, it is also important time that the application needs to detect surfaces, because too long detection time would weaken the functionality of such applications and again comfort of use. Both frameworks offer detection of flat horizontal surfaces (floors, countertops) and vertical surfaces (walls).

- Percentage of plane detected in relation to the total plane area
- Number of incorrectly detected planes (including false positives)
- Time needed to detect and map a plane [14]
- Accuracy of the scale selection¹
- Work in various lighting conditions
 - Scene display (quality) based on the actual lighting conditions
 - Impact of brightness and color of light on virtual objects [daylight, incandescent light (100 W, 60 W bulb), 3 W LED bulb]

¹ Measurement of a distance between 2 points on a mapped plane vs. real distance.

- Work in unpropitious conditions
 - Impact of a shape on plane mapping [flat surface, single-colored, devoid of pattern and texture (wall, uniform top), flat surface with a pattern (wooden table top), surface with unevenness (grass, uneven scrub), shiny surface (water)]
 - Influence of low-light
 - Work in motion – during rapid movement of a device (number of detected characteristic points, time to detect a plane)

3 Tests Results

Measurements of CPU load and memory usage were performed for two application states. The first state is the state at startup, when no plane has been detected yet and there are no objects on the stage. The second state is when the application has already detected the plane and 100 test objects have been placed on the stage. *Android Profiler* tool, which is part of Android Studio since version 3.0, was used to measure the CPU load and use the memory of the ARCore test application. The test parameters for the ARKit application were measured using *Instruments* tool that is part of the Xcode programming environment.

The purpose of first test was to examine the impact of test applications on device resources. For this purpose, the CPU load (Fig. 2) and memory usage (Fig. 3) were measured. The results were checked for applications that do not have any models on the stage and applications with the stage on which there are 100 test objects.

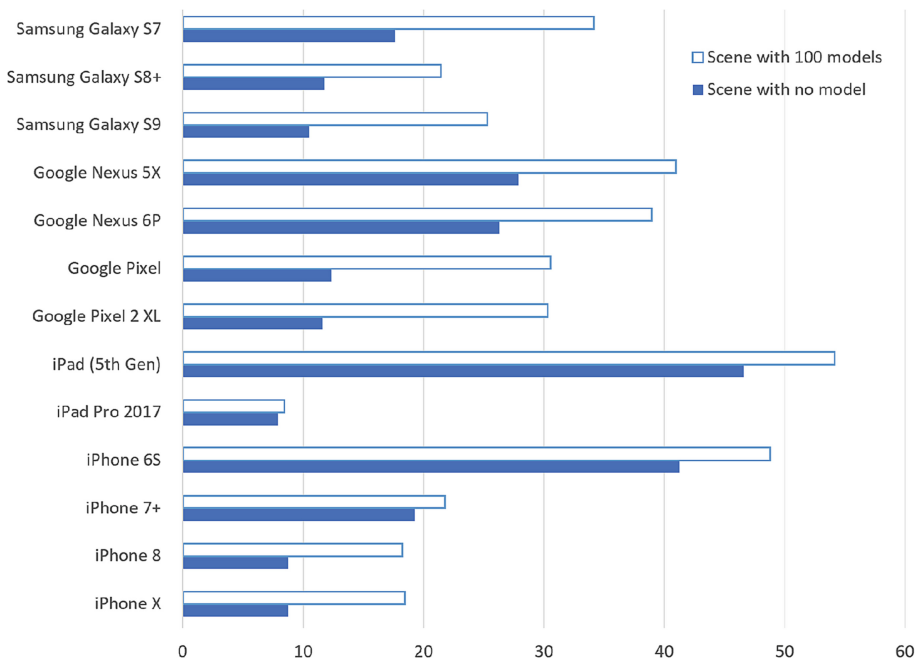


Fig. 2. Avg. total CPU usage [%]

For Android devices, memory usage was almost $3\times$ higher than for iOS. Differences can arise from many factors, such as library requirements, nature of OS (its memory management). Keep in mind that increased memory usage for Android devices should not be a problem, as most of the tested devices had more memory available than iOS devices.

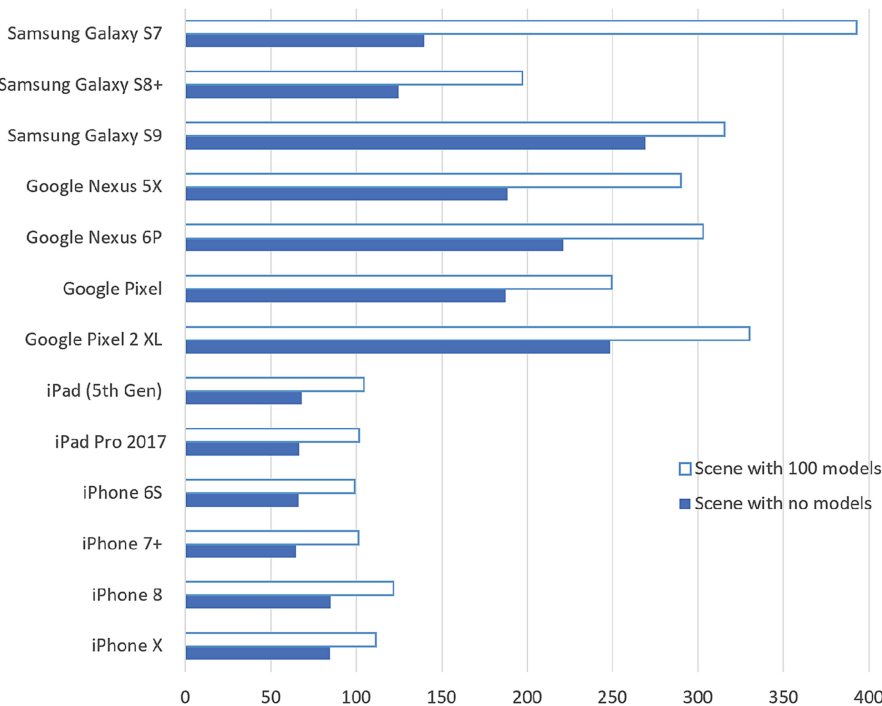


Fig. 3. Avg. memory use [MB]

Plane Detection. The tests were carried out in conditions enabling the best performance of both platforms (daylight, planes with a different surface pattern). It should be mentioned that the tests tried to achieve a reasonable compromise between the accuracy of the measurements and the time of surface mapping, because this reflects the best real use of both platforms. Therefore, during the surface mapping, each try didn't last longer than 25 s. The measurements were carried out using the best available devices supporting the discussed platforms (Google Pixel 2XL and iPhone 8+).

Accuracy of Plane Detection. The measured parameter was the percentage coverage of the test plane by the mapped surface expressed as a percentage. 30 measurements were taken for each platform based on an attempt to map a test surface measuring 90 cm × 60 cm. The tests showed the tendency of the ARKit platform to cover a smaller area of the tested plane. In addition, taking the standard deviation value, the results obtained for the Apple framework were more diverse than in the case of ARCore. Greater diversity means that the result of the operation is more difficult to predict, sometimes

the application will be able to cover the surface to a satisfactory degree (over 90%) and sometimes not. The Google Framework was characterized by a much greater repeatability, 2/3 of all measurements were in the range of 93.54–96.60%, which is a good result (Table 2).

Table 2. Accuracy of plane detection

Platform	Average coverage [%]	Standard deviation
ARCore	94,833	2,731
ARKit	90,712	4,725

Percentage of Incorrectly Detected Planes. The exact measurement of the surface area of the detected plane is an important factor in determining the accuracy of the detected plane. To determine this parameter, a script was prepared calculating the area of each detected plane based on a model consisting of a triangle grid, which is the basis of each plane. In addition, to make the location of a given plane in space, screenshots were made showing the detected plane plotted on the actual object. An additional advantage of the screenshot is the ability to determine the extent to which the virtual flat surface coincides with the real one. Thus, the examined aspect is the amount of the actual area covered by the virtual expressed in percent, as well as the amount of virtual surface detected, which does not coincide with the real object (false positive).

The test consisted in approaching the device with cameras at various angles on a mapped object, in this case a table, to cover as much as possible, the whole surface of the table top. Next, a screen shot was made with device devices aimed perpendicular to the table top to easily calculate the percentage coverage. The screen capture itself was combined with a function that calculated the surface area of the virtual plane, which was then saved to a file for further analysis. The collected data was processed, and then based on them were calculated percentage coverage values according to the formula:

$$S_{pk} = \frac{100 P_a}{P_t}$$

where S_{pk} is the amount of coverage of the surface to be tested by the detected area expressed in %, P_a is the area of the correctly detected plane, and P_t is the surface area of the test plane - in this case the table.

The size of incorrectly determined planes (false positives) was calculated based on the formula:

$$S_{fp} = \frac{100 P_b}{P_a + P_b}$$

where S_{fp} is the part of the detected surface that has been incorrectly determined expressed in %, P_a is the area of the correctly detected plane, and P_b is the area of the incorrectly determined surface (the one that does not coincide with the real object).

To calculate the surface area of the abovementioned planes, the ImageJ tool was used based on a screenshot. ARCore performed noticeably worse, in more than half of measurements, detected plane exceeded the real object surface. During tests, Apple devices only sporadically drawn incorrectly detected surfaces (5 times exceeding 3.2% of the surface) (Table 3).

Table 3. Percentage of incorrectly determined plane area

Platform	Incorrectly determined planes [%]	Standard deviation
ARCore	5,518	5,970
ARKit	1,468	2,809

Time to Detect the First Plane. ARKit was faster to than the ARCore in this test. Apple framework achieved much more repeatable results, which also confirms the superiority of ARKit in this test (Table 4).

Table 4. Time to detect the first plane

Platform	Average time [s]	Standard deviation
ARCore	5,604	1,314
ARKit	4,809	1,101

Time to Map the Surface. Another test was to check how much time it takes to cover the 90×60 cm test plane (in at least 80%) (Table 5).

Table 5. Time to map the surface

Platform	Average time [s]	Standard deviation
ARCore	12,978	2,965
ARKit	13,665	2,211

ARCore despite the worse result in the previous test, now is ahead of ARKit. Although the detection of the first plane in the case of Android takes more time, then for larger areas better results in terms of quality coverage are achieved. It was observed that plane coverage elements were greater than in the case of ARKit, which translates into shorter mapping times.

Table 6. Influence of surface type on mapping

Surface (Fig. 5)	Mapped?		Time [s]	
	ARCore	ARKit	ARCore	ARKit
A. Single-color wall	No	Partially	-	39,249
B. Furniture board (vertical)	Yes	Yes	46,765	5,529
C. Upholstery fabric (green)	Yes	Yes	5,337	4,447
D. Poster board	No	No	-	-
E. Upholstery fabric (black)	Yes	Yes	8,295	10,571
F. White canvas	Yes	Yes	6,203	4,857
G. Wooden table top	Yes	Yes	4,924	5,057
H. Worktop	Yes	Yes	4,879	5,173

Influence of Surface Type on Mapping Quality. Tests were divided into two parts. In the first part, the applications were tested against various surfaces inside a room. The second part consisted in checking the possibility of detecting planes on various type of terrain outside (very good lighting conditions). Google Pixel 2XL and iPhone X devices were used for tests. ARCore demonstrated performed significantly worse during the mapping of vertical planes. This was manifested by a much longer detection time of the plane and a smaller number of detected characteristic points (case with a vertical furniture board). In addition, ARKit in case of surfaces whose pattern has low contrast (e.g. a uniform wall, white canvas) increases the contrast to detect more specific points.

Table 7. Influence of terrain type on mapping

Surface (Fig. 5)	Mapped?		Time [s]	
	ARCore	ARKit	ARCore	ARKit
A. Low grass (~ 5 cm)	Yes	Yes	4,791	5,006
B. Med grass (~ 15 cm)	Yes	Yes	5,038	5,576
C. High grass (~ 40 cm)	Yes	Partially	5,292	6,686
D. High scrub (~ 70 cm)	Yes	No	18,620	-
E. Thick bushes (~ 50 cm)	Yes	Yes	6,201	5,276
F. Sheet of water	No	No	-	-
G. Concrete	Yes	Yes	5,832	7,273

The worst result was obtained during technical paper mapping, where the number of detected points was so low that none of the platforms was able to identify the plane. Partial mapping of a single-color wall by the Apple framework was possible only after the device was moved closer (approx. 20 cm) to the tested surface (Fig. 4).

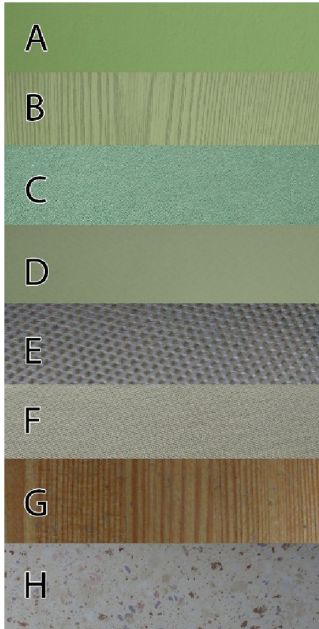


Fig. 4. Internal test surface (see Table 6)



Fig. 5. External test surface (see Table 7)

Outdoor tests indicate the advantage of Google tools. ARCore was able to detect plane where ARKit wasn't (high scrub) or covered a much larger area (high grass). Surface mapping using the ARCore took less time in all cases except thick bushes. In addition, faster mapping of large areas by Android system was observed (with each movement, increase of covered plane was larger), and greater range (mapped space was larger than in case of ARKit). A complete surprise was the detection of the surface in high scrub by ARCore. Surprisingly, the mapped plane was equal to ground level (SIC!), and not the plane defined by the upper part of the bushes, as it was the case with thick bushes. Sheet of water mapping in both cases was disappointing, but ARCore was able to detect much more characteristic points.

Work in Low Light Conditions. The parameters examined were the detection time of the first plane and the number of specific points detected. The results presented were averaged based on 10 attempts made for each setting. The devices used were Google Pixel 2XL and iPhone 8+. Together with the reduction of lighting power, the detection time of the planes increased. This can be seen in both ARCore and ARKit. The performance of ARKit is significantly reduced in lower light conditions. This is manifested by slower or completely unsuccessful surface detection. In situations where ARCore was still able to detect the plane, ARKit failed. Better results obtained by ARCore may be related to the fact that the application increases the image contrast, thanks to which it is much better at dealing with surface detection in low light conditions (Table 8).

Table 8. Influence of light conditions on mapping

Light Source (Fig. 5)	Mapped?		Time [s]	
	ARCore	ARKit	ARCore	ARKit
Daylight (high noon)	Yes	Yes	4,965	5,637
Incandescent (100 W bulb)	Yes	Yes	6,564	8,849
Incandescent (60 W bulb)	Yes	No	27,620	–
Incandescent (3 W LED bulb)	Yes	No	–	–

Work in Motion. Since the position of the device in space is calculated in real time, rapid sudden movements of the device in many directions may cause the application to run out of balance. An additional factor is blurred image caused by rapid movements. The tests (repeated 50 times) assumed making attempts to map a surface, with three exemplary models, during random, sudden movements (however repeatable) of devices. Both devices were connected, and tests were made in simultaneously. ARCore performed worse, losing 14 times (vs. only 5 in case of ARKit) the location of mapped plane, however both were able to recover the detected plane just after 2 s after the movement stopped. Accidental covering of camera lens (ARCore) results in losing the location of detected plane, and in the case of ARKit, a plane remains visible on the screen, but it is a bit shifted. Reopening lens results in gaining focus much faster on devices with ARCore (>0,5 s.) whereas in ARKit focus time is about 5 s.

It should be mentioned that it is very difficult to cause abnormal imaging. The movements were so violent that it deviates significantly from the normal use cases. The purpose of the test was however to determine which platform better handle rapid movements. ARKit can react faster to unwanted disturbances thanks to having possibility of handling twice as many frames per second (iOS and camera specific). In addition, ARKit largely relies on the device's accelerometer data, so that even after covering camera lens completely, the models and planes remain on the screen (Table 9).

Table 9. Comparison results ARCore vs. ARKit

Field of comparison	ARCore	ARKit
Detection of first plane		<i>Faster than ARCore</i>
Mapping a plane	<i>Faster to map larger surfaces</i>	
Plane detection	Tendency to incorrectly detect planes by going beyond the boundaries of mapped surfaces	
Estimation of lighting conditions		Slightly higher sensitivity to changes in lighting
Memory usage ^a [MB]	Greater (187-392)	Smaller (84-121)

(continued)

Table 9. (continued)

Field of comparison	ARCore	ARKit
Frames per second	Limited to 30	<i>Always 60 FPS</i>
Performance	Performance drops on older devices (Google Nexus 6P and Google Nexus 5X)	Smooth operation on all tested devices
Mapping of uneven/complex planes	Better suitable for outside mapping (good in mapping of complex planes like high grass)	Better suitable for inside mapping (vertical planes, not overly complex)
Work in low light conditions	<i>More resistant to interference due to insufficient lighting</i>	
Work in motion		<i>More resistant to interference due to fast, variable movements of the device</i>
Documentation for application in UNITY	<i>Fully documented</i>	Partially documented
Working examples	A few, simple ones	<i>Many, sophisticated examples documenting application of framework</i>
Shadow casting	Requires own implementation	<i>Built-in</i>
Software requirements	Android 7 and ARCore libs installed on a device	iOS 11 or higher
Other	Sharing of characteristic points for multiplayer games	In iPhone X (and newer) - <i>TrueDepth</i> feature allowing face-mapping

^aTest application size ARCore – 36,85 MB, ARKit 52.8 MB.

4 Conclusions

It should be noted that some of the advantages of ARKit arise from the fact that it runs on more powerful devices. For example, at this point it is not possible to run the ARCore application in 60 frames per second mode due to hardware limitations. The Apple framework thanks to this feature can respond better to rapid camera movements. Another example is faster application startup, or faster detection of the first plane, which is probably caused by faster and better optimized devices.

Both platforms differ in the approach to delivering the required library to devices on which AR applications run. In the case of ARCore it is necessary to install an additional library, which can be downloaded from the Google Play store, while ARKit is delivered together with the iOS version 11 or higher. These approaches are conditioned by the current state of the mobile devices market. Most Android devices do not have the latest version OS installed, so delivering ARCore together with subsequent versions of the operating system would significantly limit the development of this platform,

because it would reach a small group of users. In the case of ARKit, most devices have the latest version of the operating system, so delivering subsequent library updates together with the system seems to be a sensible approach.

Both platforms have their strengths and weaknesses, but it is difficult to find technology that would be objectively better. Both platforms are better suited to the situation. In the final analysis, the choice of technology should be conditioned by the current state of the market and your own preferences.

Because the technologies discussed are relatively new (both presented in the second half of 2017), there are many areas where improvements can be done. Both are new enough to sometimes give the impression of unfinished. It should be borne in mind that platforms are actively developed, by increasing functionality and fixing issues, so the impression of an unfinished product can quickly pass away. While making this comparison, Google announced support for shared characteristic points enabling multi-player interaction [4] on one stage, then Apple a month later announced similar functionality in new ARKit2 [3]. Such behavior indicates high competitiveness of platforms and prompt reaction to appearance of competitive features.

In authors opinion following are areas for improvement.

- estimation of lighting color on the stage
- estimation of the angle from which shadows fall to improve the realism of the objects displayed on the stage
- possibility of mapping the face based only on image from a camera
- improved plane detection accuracy, elimination of occasional fault lines on a surface
- (ARKit only) low light performance conditions - allowing longer exposure time by e.g. contrast increase or temporarily limiting camera frame rate
- (ARCore only) – better plane detection on less diverse and/or vertical surfaces

References

1. 6 Best Augmented Reality SDKs and Frameworks, Themindstudios. <https://themindstudios.com/blog/5-best-augmented-reality-sdks-and-frameworks/>. Accessed 01 Feb 2019
2. ARCore Resources <https://developers.google.com/ar/discover/>. Accessed 28 Feb 2019
3. ARKit 2. <https://www.apple.com/newsroom/2018/06/apple-unveils-arkit-2/>. Accessed 01 Feb 2019
4. Gosalia, A.: Experience augmented reality together with new updates to ARCore. <https://www.blog.google/products/arcore/experience-augmented-reality-together-new-updates-arcore/>
5. Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., MacIntyre, B.: Recent advances in augmented reality. Naval Research Lab Washington DC (2001)
6. Best AR SDK kits. <https://thinkmobiles.com/blog/best-ar-sdk-review/>. Accessed 01 Feb 2019
7. Best AR SDK for development for iOS and Android in 2019. ThinkMobiles. <https://thinkmobiles.com/blog/best-ar-sdk-review/>. Accessed 01 Feb 2019
8. Bergquist, R., Stenbeck, N.: Using Augmented Reality to Measure Vertical Surfaces. Bachelor thesis, Linköping University (2018)

9. Duan, G., Han, M., Zhao, W., Dong, T., Xu, T.: Augmented reality technology and its game application research. In: 2018 3rd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2018). Atlantis Press, May 2018
10. Halpern, M., Zhu, Y., Reddi, V.J.: Mobile CPU's rise to power: quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 64–76. IEEE, March 2016
11. Forecast AR and VR market size worldwide (2016-1022). <https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/>. Accessed 01 Feb 2019
12. Willemsen, P., Jaros, W., McGregor, C., Downs, E., Berndt, M., Passofaro, A.: Memory task performance across augmented and virtual reality. In: 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp. 723–724. IEEE, March 2018
13. Six Top Tools to Build Augmented Reality Mobile Apps InfoQ. <https://www.infoq.com/articles/augmented-reality-best-skds>. Accessed 01 Feb 2019
14. Wang, W.: Plane detection. In: Beginning ARKit for iPhone and iPad, pp. 261–297. Apress, Berkeley (2018)
15. Zhang, W., Han, B., Hui, P.: Latency mobile augmented reality with flexible tracking. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 829–831. ACM, October 2018