Utilizing cloud storage for realistic augmented reality on mobile

Phuc Tran
Earlham College
801 National Road West, Richmond IN 47374
ptranh14@earlham.edu

ABSTRACT

As for today, augmented reality technologies are shifting towards small devices with a focus on user interaction. As more techniques in rendering AR objects are developed, more computing powers are needed to keep up. Mobile AR technology has all functions built in, in addition to GPS and compass for realistic AR rendering technology. However, mobile devices lack storage and the raw power for 3D rendering of complex objects. The paper discusses the possibility of integrating cloud to fix these problems, and conclude that using cloud for performance is difficult while using cloud for storage is possible. Results show that performance drop when utilizing cloud storage for 3D objects are minimal. As for now, cloud fetched objects are rendered without textures, leading to a reduce in realism compared to local fetched objects. Thus, the next step of the project is implementing textures fetch from cloud DB on top of the 3D object file fetch.

1. INTRODUCTION

Augmented reality (AR) technology has come a long way in its development. As more techniques in rendering AR objects are developed, we need more computing power to keep our performance up. With Moore's law, the specialized hardware has no difficulty rendering augmented reality with extreme realism and high performance. The same cannot be said for mobile AR due to the need for other functions integrated in a mobile device, such as communications, internet access, long-lasting battery, speaker, receiver, etc. In addition, mobile devices need to be compact and lightweight to provide ease of usage for customers, and the processors for mobile devices do not need to be powerful since the basic usage of mobiles are calling, texting and browsing the internet.

Mobile, on the other hand, is very accessible to customers and has many augmented functions built-in such as the accelerometer and camera. Furthermore, mobile devices have GPS and compass which are suited to outdoor realistic AR using the data collected from GPS and compass to render shadows and shading from the sun. With these tools built in a mobile device, AR frameworks can create a realistic reality from these data collected. The only limitation in mobile AR technology is the low performance and the lack of storage space for complex 3D objects.

With that said, the paper proposes integrating a cloud technology to alleviate problems that mobile devices suffer, while utilizing the variety of sensors available to the device for a smooth and realistic AR experience. The paper uses a variety of frameworks and algorithms to render complex objects outdoor with realism in mind. Furthermore, the paper will discusses the results and decisions in utlizing cloud technology for an improvement to mobile AR.

2. CURRENT ADVANCEMENT

2.1 Realistic AR rendering technology

In the field of realistic rendering technology, one object rendering algorithm proposed by Kolivand and Sunar utilizes the GPS location, the interaction between sky color, the sun, the orientation, GPS, date and time to correctly determine how the object looks, where should it be and its shadow.[3]

There are two notable techniques being proposed in the paper. Firstly, sky modeling is the act of modeling the sky using 3D modeling software and mathematical functions. Then, the location of the sun on the dome-like sky is determined by the GPS location and time of day. Secondly, the shadow of the object is determined by Z-Partitioning following Gaussian approach to create semi-soft shadow. [3]

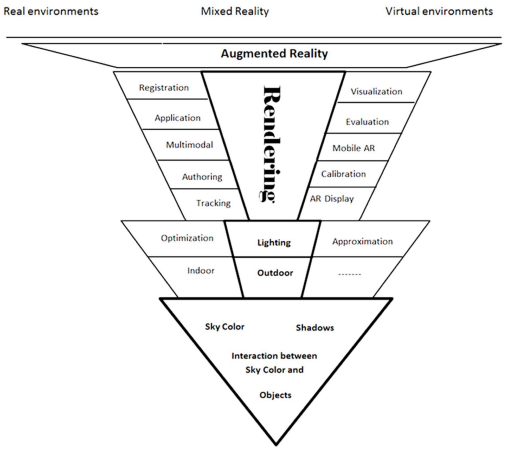

Figure 1: The process of rendering proposed in the paper by Kolivand and Sunar [3]

Another problem in the realistic AR field is the lack of distance between real and virtual objects. Virtual objects usually appear nearer to the camera and this issue is met more in outdoor environments due to long distances and wide areas. To address this issue, the paper suggests adding a parameter of Fog in the spatial partition of the view, makes

the virtual objects appear far from camera and consequently suitable for far distances in outdoor environments. [3]

Figure 2: A scene demonstrating realism of the elephants, rendered on the right scene using the realistic rendering algorithm proposed by Kolivand and Sunar. [3]

2.2 Integrating cloud technology to mobile AR rendering technology

Mobile devices have a weakness of small storage and computing powers and a strength in having connections to networks and location services. Cloud technology utilizes the networks to help alleviate the problem of computing and storage in mobile devices. Augmented reality sometimes requires large global data, it is even more important to integrate cloud technology to AR systems.

To address this technology, a paper by Sundaram et al introduced an application that uses efficiently cloud technology to develop Location Based Augmented Reality application. First and foremost, all the resources that are required or utilized by the application are all stored in the server. Once this is done, a URL will be generated for the AR application. The URL links the app to the server to access information. Then, on the application only the longitude, latitude and the URLs are stored on the database. Whenever the needs to access a point of interest, the URL will be used to access information on the cloud. [6]

This method by the paper works well and is simple enough to be implemented easily. However, it does not address the problem of computing speed by utilizing cloud computing. The only thing it does is utilizing cloud storage for large, global information sets. A proposed improvement to the method is to send the image set to cloud, computing the feature selection algorithm to find a match then send back the output to the device.

Another more advanced method proposed by Bae et al, allows users to access and query 3D information related to real-world physical objects and see it precisely overlaid on top of imagery of the associated physical objects. The method first of all derive a 3D model generated from a set of pre-collected photographs stored on the cloud. Then, the usersâĂŹ images taken by the camera are compared to the 3D model to get the usersâĂŹ orientation and 3D location. Afterwards, the information stored are displayed to the generated 3D model. [1]

Furthermore, the scalability problems are also addressed, such as when the information entities associated with real-world objects increased. With that said, the proposed method is a big improvement to the state-of-the-art cloud technology currently being utilized in mobile AR. [1]

One paper by Shiraz et al in 2014, investigates the overhead of runtime application partitioning of mobile devices for mobile cloud computing. Application partitioning is the offloading algorithms for the distribution of processing load between powerful server nodes and resources constraint mobile devices. The investigation uses a model of mobile cloud computing (MCC) where the mobile devices seamlessly access the services provided by the computational cloud to obtain the resource benefits at low cost through wireless technology. [5]

Application partitioning is very powerful in that it allows adjustments to the amount of work offloaded to the cloud based on how powerful the cloud is and how powerful the devices are. For the environment, the overhead of runtime application partitioning is investigated in simulation and in the real environment. SmartSim is employed for the evaluation of the overhead of runtime application on mobile devices. SmartSim is a simulation tool that models the processing capabilities of mobile devices and uses these to evaluate the overhead of runtime application. [5]

The overhead of runtime application partitioning is further analyzed by benchmarking the application for Android devices in the real mobile cloud computing environment. The experimental setup consists a cloud server node, Wi-Fi network, and a Samsung Galaxy SII mobile device. The mobile accesses the wireless network using a Wi-Fi wireless network connection of type 802.11g, with the available physical layer data rates of 54 Mbps. [5]

Overall, the paper said that the runtime application partitioning of the intensive components of the mobile apps is a resource intensive mechanism. Empirical evidence and analysis indicates that additional resources on mobile devices depends on the instances of runtime application partitioning, and the computational intensity of the application partitions. The paper concludes that the runtime application partitioning of tradition mobile applications results in additional overhead for deployment and management of distributed platform. Thus, it is difficult to employ cloud computing technologies alongside with the application partitioning to maximize computational speed on mobile devices. Furthermore, since there are big limitations on networking for mobile devices, the cloud computing future for mobile is even less appealing. [5]

With that said, the aim for this paper is to utilize cloud technology differently than a performance boost. Aside from performance, cloud technology can be used to allow clients the access to a shared storage for multiple clients without the need for local storage. This can be done due to using cloud technology for storage only decreases the overhead performance without affecting the consistency in rendering through local hardware. For AR, the overhead performance is acceptable since AR requires consistency during rendering the most.

3. PROPOSED SOLUTION

Mobile is the best platform for AR developing due to the accessibility, popularity and the existing sensors on the mobile. Many existing AR framework utilizing all the mobile capabilities to render AR objects successfully.

Mobile devices have all the sensors necessary for AR technology with addition functions for realism. Camera and accelerometers are vital for AR technology. GPS and compass are essential to render realistically AR objects as determining the intensity and direction of the sunlight and the orientation of objects are necessary for realistic rendering.

Previously, mobile devices do not have the capabilities for high-powered realistic AR rendering due to the lack of supported hardware. Furthermore, mobile OS such as Apple iOS has limited access to low level programming, thus making AR development and optimization very difficult. Now, with the new ARKit framework by Apple, AR development and optimization is very easy, making realistic AR rendering more appealing.

However, for complex objects, several problems arises. Notable problems include the lack of raw power for rendering objects and the lack of storage space for storing large 3D objects files. With that said, the paper proposes utilizing and integrating cloud technology to the existing framework to alleviate some of the problems.

3.1 Physically based rendering technology for realistic rendering of objects

The program uses multiple sensors for the realistic rendering of objects. More specifically, a method called Physically Based Rendering (PBR) is used to make objects as realistic as possible. For shadow rendering, deferred method of shadow rendering is used to improve rendering objects with multiple light sources.

Figure 3: An example of PBR on regular computer 3D objects rendering. Using different environment, the object is displayed differently. [7]

The basic concept of PBR is that when you add textures to your object you provide information that includes albedo, roughness and metalness. Albedo is the base color input, commonly known as a diffuse map. It is the material texture without any lighting or shadow information. Roughness describes how rough or smooth the material will be. Rougher surfaces show dimmer reflections and smoother surfaces show brighter specular reflections. Metalness is a rough equivalent to how shiny a material will be. [7]

Utilizing PBR also requires us to have an environment texture to render reflections on the surface of 3D objects with high metalness. In order to fully utilize PBR to render objects realistically, the environment texture needs to be updated frequently to render the reflecting surfaces of the objects. However, updating environment texture is difficult and time-consuming as an image depicting the surroundings of the object is needed.

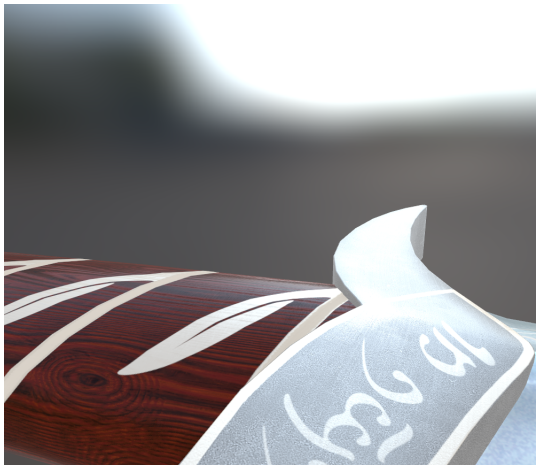

Figure 4: An example of an object reflecting the light from an environment cube map in the background.

With that said, to maximize realism while not forcing users to capture environment texture every few minutes, different default environment textures are stored for different time of the day with different lightings. These environment textures are called cube maps which depicts the six different environment sides of the objects. Using different cube maps results in different lighting and reflections on the object.

However, since the environment of augmented objects change vastly depending on the location of the user, using an accurate and clear cube map every time we render objects is unrealistic. Therefore, blurred, general-purpose cube maps are made to depict different sun position based on time of the day without using any concrete and accurate cube map. This reduces the realism of augmented objects somewhat but also reduce the need to generate cube map every time the user render objects.

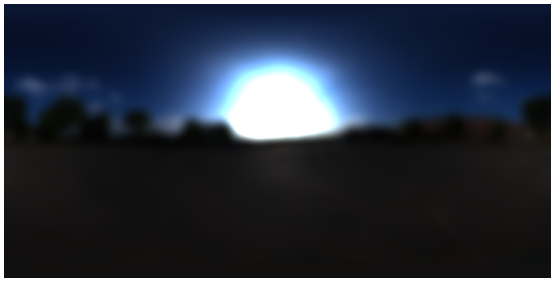

Figure 5: An example of a general-purpose cube map after converting to equirectangular. This shows the sun setting on the west.

Next, we need to estimate as accurately as we can about the sun angle and intensity using the information provided from the GPS and compass. This can be achieved easily by determine the time and location of the mobile device.

3.2 Sunlight direction and shadows

For the sun angle and intensity, the time and location of the mobile devices are needed. Then, an algorithm from an NREL Technical Report, Solar Position Algorithm for Solar

Radiation Applications, is utilized to identify the azimuth angle, elevation angle and altitude angle of the sun. Furthermore, with the time of the day and the day of the year we can calculate the intensity of the sun. [4]

The NREL Technical Report, Solar Position Algorithm for Solar Radiation Applications, describes a procedure for predicting the position of the sun at any point between the year -2000 and 6000. It predicts the position of the sun by predicting two position variable, the solar zenith angle and the solar azimuth angle. The algorithm from the report has uncertainties of 0.0003 degrees for both zenith and azimuth. The algorithm follows the algorithm described by Jean Meeus but simplified focusing on only the sun instead of all the planets. Some changes are also made for ease of usage such as measuring azimuth angle from north and eastward instead of from south and eastward. [4]

Overall, the Solar Position Algorithm is an accurate calculation of the sun position given the time of the day, the date and the position. The algorithm is mainly used to calculate solar azimuth angle and solar zenith angle. However, it is also used to calculate accurate sunset and sunrise time for sun intensity settings. [4]

With these information, it is possible to determine the direction that the sun shines its light towards. With the direction information, we can render directional light source with light intensity derived from the sunrise and sunset time. Furthermore, with the sunrise and sunset time, we can change the environment cube maps to fit the approximate current position of the sun. Using sunrise and sunset time, we can set ambient lighting intensity and directional lighting intensity accordingly to render the object with the most realism. The program that the project utilized is an implementation for Swift from Dr. Jeff Craighead. He implemented the Solar Calculator Algorithm for Swift. [2]

3.3 Plane recognition for objects placement

For plane recognition, the AR framework uses feature selection for detecting planes. First of all, different texture features are detected and selected through the camera input and accelerometer readings. When similarities are recognized, an anchor is created at the coordinate of the similarities.

Using these recorded anchors, the AR framework can render objects effectively by position it relative to these anchors. To detect horizontal planes for object placement, when enough anchors are recorded, a plane object is created similar to a normal object. Afterwards, when more anchors belonging to the same plane are detected, the plane object is expanded to match the span of the anchors.

With these plane objects, the rendered objects are placed right above the planes. To do that, whenever user taps, a hit test is performed on the tap location and the planes existing at that tap location. The closest plane is selected, and the hit location is used to render the object. A visual plane is also rendered whenever a plane is detected to show the clickable area that objects can be rendered on.

3.4 Integrating cloud technology to store objects

Integrating cloud technology to mobile applications has the benefit of allowing the mobile access to the strong performance and data storage that cloud has. However, using cloud technology requires the mobile to have strong and consistent network which the mobile and the cloud cluster can communicate through. As the main network connection mobile has is wifi, the connection is not always consistent and stable.

The consistency problem is even more evident in AR rendering as AR requires continuous and undisrupted rendering of objects on multiple directions and dimensions, if there are stutters or lags, the AR rendering will fail to be useful.

Furthermore, another problem arises which is the runtime partitioning of elastic mobile applications for mobile cloud computing. It is the partitioning of the tasks that mobile and the cloud cluster handles which makes it the most efficient. An investigation by Ahmed shows that it is possible to achieve higher efficiency integrating cloud for performance than just using the mobile hardware performance. However, the cost of calculating the partitioning needed is higher than the cost saved from performing a successful partitioning. [5]

With that said, an elastic partitioning system is necessary because with the inconsistency and variation in speed of mobile network, the partition is different for different runtime and should not be hardcoded. Therefore, using cloud technology to improve speed and performance is currently not appealing and require more research or time.

One thing we can do with the cloud system is allowing mobile to access the shared storage power of cloud system to store and use 3D object files without the need of large local storage space that mobile does not have. Utilizing cloud for storage will also be what the paper aims to do.

To utilize cloud technology, the mobile needs a server to connect itself to. This server will also store all of the 3D objects and textures that the mobile needs to render the 3D objects. When needed, the mobile will fetch the files from the server to the mobile and uses these for rendering.

To create a cloud server, a lightweight SQLite database is used to store the name of the objects and the url to it. For server side processing, we use Python Flask module to receive and process HTTP requests. Using a SQLite database and Flask, we can perform communications between the mobile and the server side. When user downloaded an object, the object is downloaded from the server to the temporary storage on the phone and is then rendered when tap. Since most other information is already accessible on the mobile, rendering realistically using only object file is possible.

In the end, the cloud serves as a powerful shared storage for the project, as using a shared storage provides lots of varied objects for rendering. Furthermore, for large or complex objects, the object description and textures files can be very large. Utilizing cloud server helps reduce the storage at the cost of minimal performance drop. These performance drop will also be unnoticeable due to the need for plane recognition before object rendering. Since plane recognition took longer due to having to find similar features, the object receive procedure will already be finished by the time the plane is recognized.

However, due to the security measures that Apple implore on its products, it is difficult for now to utilize textures downloaded from the cloud server. Because of this reason, for now the cloud will only store 3D object files without any textures. The textures of the object will be white. The method for rendering will remain PBR, thus materials will still reflect the environments. The only difference is that the textures will be white in color with no difference in reflections for all objects.

4. SOFTWARE ARCHITECTURE

Below is the complete software architecture of the project. Overall the project has three main components, a mobile controller, the ARKit framework and the cloud database.

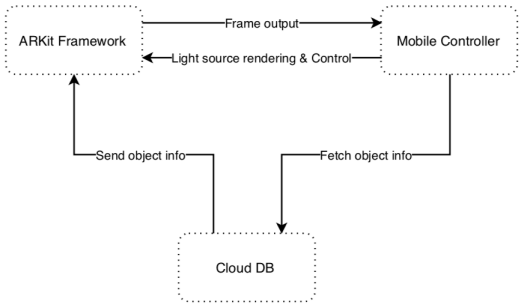

Figure 6: The overall software architecture of the project.

ARKit framework handles rendering object and executing control commands from the mobile controllers. ARKit framework also handles camera, accelerometer and fetching the object info from the Cloud DB. The plane recognition algorithm also resides here and is included inside ARKit Renderer to create frame output. This is where the majority of the computings happens due to the high work load of rendering complex 3D objects and feature selection for anchors and plane recognition.

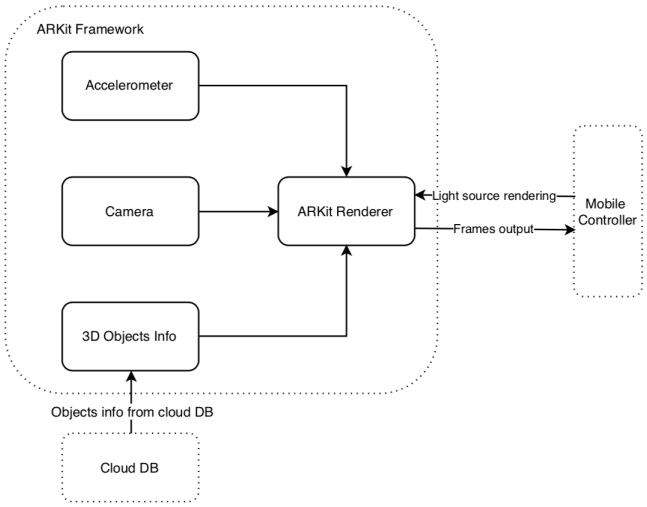

Figure 7: The software architecture of ARKit and its interactions with other components.

The Mobile Controller is the main controller of the project. It handles screen input, output and outside control such as fetch and render. The Mobile Controller also receives and use GPS Coordinates and Time to calculate sun position using Solar Calculator Algorithm and uses that to calculate the sun's coordinate in the AR Coordinate system and finally asks the ARKit Framework to render it. The Mobile Controller also receives frame output from the ARKit Framework and displays it on the screen.

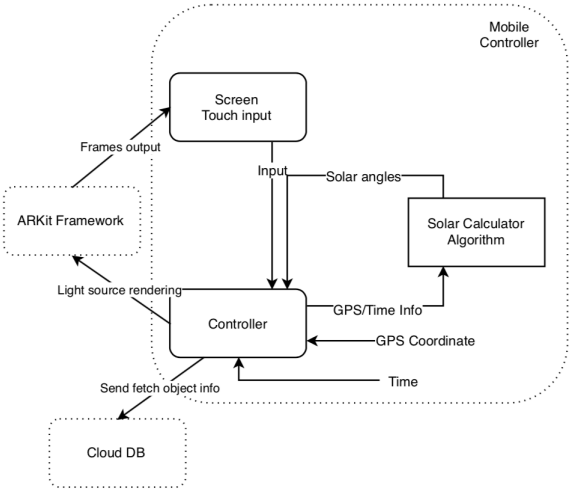

Figure 8: The software architecture of the Mobile Controller and its interactions with other components.

The Cloud DB is the object database of the project. It receives fetch object info and send object info to the framework for rendering. The Cloud DB consists a Python Flask module and a SQLite DB for storing objects name and paths.

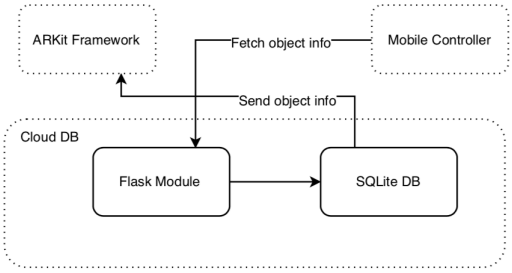

Figure 9: The software architecture of Cloud DB and its interactions with other components.

5. RESULT

The result of the project is a demo for realistic rendering with cloud storage. There are two options for selecting 3D objects, one is the local object database and the other is the cloud object database. The local ones have textures and thus look more realistic due to colors and textures. The database ones do not have textures thus do not have as much realism as the local ones, but in return costs no storage space.

Figure 10: Rendering using local database.

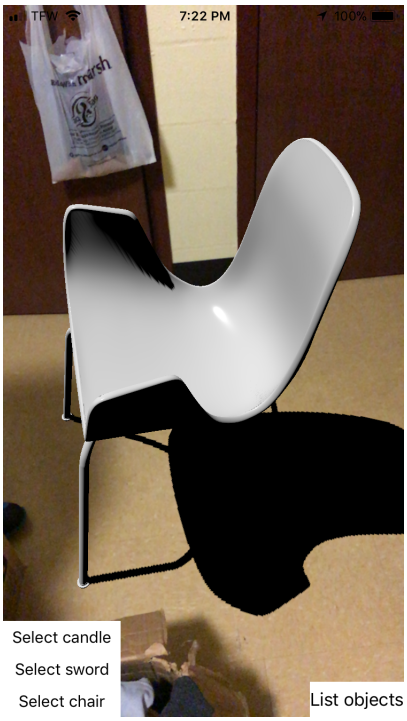

Figure 11: Rendering using cloud database.

For performance drop on using cloud storage, the time it takes to download object files to the local machine is insignificant to the time it take to load it to the AR framework. Therefore, there is little to no difference in performance drop between loading an object from cloud and loading an object from local database.

For plane detection, since it requires anchors and features of real world environment, it is difficult to track the time it takes to detect a plane. However, with slow and steady phone movement, the plane detection can be very accurate.

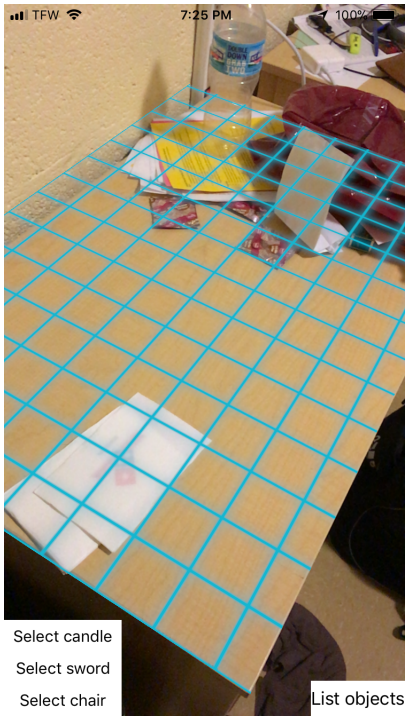

Figure 12: Plane recognition of a table.

Figure 13: Object from cloud DB placed on top of detected surface.

For performance measures, using the same object on cloud and on local database has no noticeable difference except for RAM usage. This is due to local database has textures which are considerable in size, therefore using a lot more RAM than using cloud database. Overall, the CPU utilization are the same for both of the methods, and the RAM usage are higher when using local objects, since local object has textures.

Below is the result of rendering the same object three times for local database and three times for cloud database.

Rendering results local			
	Exp 1	Exp 2	Exp 3
CPU	47%	42%	42%
Memory	63.3 MB	63 MB	62.5 MB
FPS	60 FPS	60 FPS	60 FPS

Rendering results cloud			
	Exp 1	Exp 2	Exp 3
CPU	44%	42%	45%
Memory	47.1 MB	47.1 MB	46.9 MB
FPS	60 FPS	60 FPS	60 FPS

The results show that for CPU Utilization, both cloud and local fetched objects have roughly the same CPU Util, leading to the conclusion that utilizing cloud has little to no impact on CPU Utilization.

Memory usage, however, shows that local fetched object uses much larger memory than cloud fetched objects (roughly 33% increase in memory usage). This is due to when rendering local fetched objects, the ARKit framework has to load the textures files to RAM as well as the 3D object file while for rendering cloud fetched objects the ARKit framework does not have to load textures files. It is the trade of between more realism (from textures) and less RAM usage.

Finally, FPS is the same for all 6 experiments, leading to the conclusion that there is no performance drop on using cloud fetched versus using local objects.

6. FUTURE WORK

There are several weakness to the current model. One biggest weakness lies in the fact that the project could not receive texture information from the model. Therefore, the biggest goal for future work is to allow complete and seamless download of objects from model to textures.

Another weakness to the current model lies in its degrade in performance when loading large objects. Cloud technology utilized currently only alleviate or improve the storage function of the project. It is known that cloud computing can be very useful for augmented reality due to heavy calculations needed for rendering 3D objects. Therefore, another goal for future work is implementing cloud computing as a improvement to the current model.

Of course, as stated by Shiraz et al in 2014 that utilizing cloud computing usually yields little results due to the need for partitioning the work between local and cloud. However, as connection speed improves and as mobile is a platform where connection speed is more important than computing speed, utilizing cloud computing might be worth it. For now, due to time constraint, other functionalities are prioritized over cloud computing. [5]

Another possible expansion of the current project is the ability to convert any 3D object files to Apple's SceneKit file for rendering. Currently, other 3D object data files need to be converted offline to .scn to be able to render successfully in the project. Therefore, a possible next step could be expanding the list of supported files type to .dae, .obj, .blend and a variety of others.

7. CONCLUSION

The process rendering realistic Augmented Reality objects requires lightings, textures and usually further requires lots of storage space due to the amount of textures and complexities. However, utilizing cloud storage will alleviate the storage space issue with little to no impact to performance. The project describes and implements a realistic rendering approach for augmented reality on mobile.

The project uses a simple three component architecture of a controller, the framework and the cloud database, mimicking the modal view controller architecture with slight changes. The result shows that for local stored objects, the performance and realism is as expected. For cloud fetched objects, due to the lack of textures, the realism level could be improved. That leads to the next step of implementing textures storage on cloud database for improving realism for cloud stored objects as well as local objects.

8. ACKNOWLEDGEMENT

I would like to thank Charlie Peck and Xunfei Jiang for the expert advice, support and guidance throughout this project. Furthermore, I would like to thank the 3D modeling experts from TurboSquid for providing open-source high-quality 3D artwork that I can use in my project. I would like to also thank HDRI Hub, a provider of free textures who provided the open-source HDRI environment cube maps that I used in my project.

9. REFERENCES

[1] H. Bae, M. Walker, J. White, Y. Pan, Y. Sun, and M. Golparvar-Fard. Fast and scalable structure-from-motion based localization for high-precision mobile augmented reality systems. 5(1):1–21. OCLC: 6559324201.

[2] J. Craighead. Swift-solar-calculator.

[3] H. Kolivand and M. Sunar. Realistic real-time outdoor rendering in augmented reality. 9(9). OCLC: 5633963447.

[4] I. Reda and A. Andreas. Solar position algorithm for solar radiation applications. Technical report, National Renewable Energy Laboratory, 2008.

[5] M. Shiraz, E. Ahmed, A. Gani, and Q. Han. Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. *The Journal of Supercomputing: An International Journal of High-Performance Computer Design, Analysis, and Use*, 67:84–103, 2014.

[6] V. M. Sundaram, S. K. Vasudevan, A. Ritesh, and C. Santhosh. An innovative app with for location finding with augmented reality using CLOUD. 50:585–589. OCLC: 5824415986.

[7] J. Wilson. Physically-based rendering, and you can too!