Proceedings of the ASME 2019
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2019
August 18-21, 2019, Anaheim, CA, USA

DETC2019-98440

MOBILE BASED REAL-TIME OCCLUSION BETWEEN REAL AND DIGITAL OBJECTS IN AUGMENTED REALITY

Pranav Jain
Computer Science and Engineering Department
The Pennsylvania State University
University Park, PA 16802
Email: pxj5088@psu.edu

Conrad Tucker
Engineering Design, Industrial and Manufacturing Engineering
The Pennsylvania State University
University Park, PA 16802
Email: ctucker4@psu.edu

ABSTRACT

In this paper, a mobile-based augmented reality (AR) method is presented that is capable of accurate occlusion between digital and real-world objects in real-time. AR occlusion is the process of hiding or showing virtual objects behind physical ones. Existing approaches that address occlusion in AR applications typically require the use of markers or depth sensors, coupled with compute machines (e.g., laptop or desktop). Furthermore, real-world environments are cluttered and contain motion artifacts that result in occlusion errors and improperly rendered virtual objects, relative to the real world environment. These occlusion errors can lead users to have an incorrect perception of the environment around them while using an AR application, namely not knowing a real-world object is present. Moving the technology to mobile-based AR environments is necessary to reduce the cost and complexity of these technologies. This paper presents a mobile-based AR method that brings real and virtual objects into a similar coordinate system so that virtual objects do not obscure nearby real-world objects in an AR environment. This method captures and processes visual data in real-time, allowing the method to be used in a variety of non-static environments and scenarios. The results of the case study show that the method has the potential to reduce compute complexity, maintain high frame rates to run in real-time, and maintain occlusion efficacy.

Keywords: Augmented Reality, Computer Vision, Machine Learning, Model Fitting, Occlusion

1. INTRODUCTION

AR applications have become more useful in different industries as the technology behind them becomes increasingly powerful [1]. Mobile devices are commonly used as the basis for AR applications, as they are readily available, durable and easy for industries to implement [1]. However, in many common locations for AR such as homes, offices, and construction sites, the real-world environment is cluttered and dynamic, making the use of AR applications difficult. The device needs to manage a virtual version of the environment, along with any object that is added to, moved, or changed in the environment. Dynamic and complex environments make managing that information computationally expensive. This, along with expensive hardware and inconsistent performance, have limited the growth of Augmented Reality [2].

The random and complex nature of real-world environments make it difficult to accurately use an AR application. The shortcomings of current AR technologies pose a risk to the user by unintentionally placing them in dangerous situations because of the improper rendering of virtual objects

in real spaces [3], [4]. AR occlusion, the process of hiding or showing virtual objects behind physical ones, plays a major role in making these renderings realistic. AR applications often use markers such as QR codes to track the camera's relative position to the virtual objects placed in the environment. However, using images from a standard 2D RGB camera to identify the marker is insufficient in determining how to correctly render the virtual objects that are partially occluded by real-world objects. I.e., the system will visualize either all or none of a virtual object. To improve the occlusion process, AR applications collect depth information to virtualize the environment and calculate what part of the virtual object must be hidden. There are a few methods that AR tools use to collect depth information to virtualize the current real-world environment. The main strategies are, Structured Light, Time of Flight, and Stereoscopic cameras [2].

However, we can take advantage of regular cameras and lateral movement to rebuild a 3D environment using feature points as the point cloud model. Newer mobile device platforms continuously capture images as the device translates laterally to match similar points. This enables an AR application to build a sparse point cloud model that would then be used to define depth fields. An example of such a system is the Simultaneous Localization and Mapping (SLAM) [5]. This reconstruction, while not perfectly real-time, is fast enough that it starts mapping and defining objects as they are placed. The method can also combine data from multiple devices and continuously build on the shared dataset. When coupled with object classification and localization, SLAM has the ability to reduce the processing complexity and enable occlusion on a mobile phone in real-time. This could also allow for faster tracking of physical objects, and occlusion calculations being processed using regular graphics cards [5].

The goal of AR research is to create realistic AR experiences. This requires a sensor that produces a high-resolution depth map with a near infinite range. This paper seeks to remove the need for markers in an environment, remove the need for depth sensors in an environment, reduce the computational complexity of data processing, maintain occlusion efficacy, and maintain the rendering frame rate.

This paper is organized as follows: this section presents an overview and motivation of the work. Section 2 presents the literature review of existing methods for AR depth tracking. Section 3 introduces the method, followed by section 4 that presents a case study that demonstrates the validity of the method. Section 5 presents the results from the case study. Section 6 concludes the paper and discusses possible areas of research expansion.

2. LITERATURE REVIEW

Extensive research exists on how to improve the visual representation of AR environments. The development of toolkits for common AR technologies has led to a rise in its popularity, but also exposed the limitations of existing AR research and methodologies. There has been a rise in accessibility of AR applications in recent years, with trends showing further growth in the industry [1]. The set of papers that are showcased in Table 1 show various implementations and their respective capabilities.

2.1 Depth Sensor Technologies

AR with depth sensors, specifically RGB-D sensors, have been used to collect point clouds directly from the real world. This information is then used to calculate how a projected image should be distorted to be properly shown in a real-world environment in relation to real-world objects. As mentioned in the Introduction section, the main strategies for depth sensing are: Structured Light, Time of Flight and Stereoscopic cameras [2]. Structured Light sensors work by projecting a light pattern (usually infrared) towards a surface that is to be modelled virtually [6]. A camera detects the light that returns and uses the distortions made from the original beam to detect surface contours. 3D laser detection and ranging (LADAR) imaging, and Kinect sensors, employ this approach. A recent iteration of this technology can be found in Face ID by Apple, which uses the structured light approach by projecting 30,000 infrared dots onto a face and detecting contours to build a dense 3D model used for authentication [7]. A major limitation with such approaches is that it has a hard time functioning outside, as the sun outputs high amounts of infrared light, which can distort the sensor readings by adding noise to the environment [8]. There are also challenges with this approach when dealing with transparent and semi-transparent surfaces, as infrared typically goes right through them, making them invisible to detection. Structured light also has issues with highly reflective surfaces as double reflections and inter-reflections can cause the stripe pattern to be overlaid with unwanted light, hereby reducing the fidelity of detection [8].

Time of Flight sensors work by emitting rapid pulses of Infrared light that are reflected by objects within its view [9]. The time it takes for that light to get back to the image sensor gives it depth information at each pixel. There exist RF modulated light sources with phase detectors that send out modulated light and measure the phase shift of the carrier with the phase detector [9]. There are also range-gated images that use a shutter to limit the light sent and received, thereby associating the amount of light received to the distance the pulse had to travel.

TABLE 1: SUMMARY OF EXISTING WORK

	Features	Marker Based Tracking	Representation of Virtual Objects	RGB-D Data Usage	Real-Time Occlusion	Primitive Models from RGB-D Data	Model Permanence	Markerless Tracking	General Device/Camera Functionality
Authors	*ARToolkit (2007)*	■	■						
	Wilson and Benko (2017)	■	■	■					
	Tian et al. (2015)	■	■	■					
	Young and Smith (2016)	■	■	■	■				
	Lesniak and Tucker(2018)	■	■	■	■	■	■		
	Jain and Tucker (2019)	■	■	■	■	■	■	■	■

Unfortunately, many of the existing methods are range limited to detect objects at a range of 5-10 meters away [9]. Some advances make it possible to do longer distances (60m) at the cost of significantly higher power requirements. Furthermore, time of flight technologies such as structured light technologies, have issues with working outside, as they need to provide a large enough dynamic range to differentiate itself from the light that comes from the sun. The illumination power of the sun is 1050 watts per square meter and varies after passing through filters [9]. This increases the power requirements needed to make such technologies work without interference [9]. Time of flight methods also have issues with highly reflective surfaces, as it increases the chance that the modulated light returns along multiple paths, thereby making the measured distance larger than the real distance.

Stereo cameras work by simulating human vision by measuring the displacement between pixels of two cameras a fixed distance apart [10]. This can be implemented with simple pinhole cameras. These technologies are more robust in outdoor conditions and consume less power than other methods. However, these technologies are also range-limited to 3-4 meters on devices such as mobile devices.

In summary, depth sensors are limited by their computational requirements, range, and low resolution of point cloud models to be used for effective occlusion [2]:

Poor range (<5m): The size and power limitations of mobile devices restrict the range of IR and Stereo depth sensors.

Low resolution: Smaller objects in the scene are difficult to discern in the point cloud, making it more challenging to achieve reliable occlusion surfaces.

Slow mesh reconstruction: Current methods of generating meshes from point clouds are too slow for real-time occlusion on these devices.

2.2 Augmented Reality Environments

Wilson and Benko proposed advancements in AR research that seek to address issues of occlusion [11], [12]. Their work serves as a basis for retrieving data captured in the real world and transforming it into a projection surface using markers so that virtual objects can react to the world using the reconstructed world as a basis (see Table 1). The work by Tian et al. incorporates the RGB-D data directly into the virtual environment [13] as seen in Table 1. This approach works to occlude the virtual object with the data but requires that the RGB-D data be collected beforehand and is preprocessed into a surface reconstruction. Issues with this method occur however if the real-world environment changes between the initial capture of RGB-D data and the final use in an AR application. This would render it unusable in scenarios such as workshops, industrial spaces, or a consumer's home where objects and individuals are dynamic. A dynamic environment must be expected for AR for effective use outside a lab setting.

Young and Smith [14] developed a method to use RGB-D data to get users to interact with the virtual objects in an AR environment by capturing skeletal data from the Kinect sensor to create a virtual human that could interact with the virtual objects in a multitude of ways. They were also able to correctly occlude the virtual objects based on the locations of real-world objects. They did this by replacing Z-buffer values in the rendering process with depth data captured by the Kinect sensor. The issue with this approach is that the sensor is stationary and has to remain in view of the marker used for tracking. Since the occlusions are replaced with every new frame of data that is captured, the system loses knowledge of the previous RGB-D data that was captured and their virtual representations of real-world objects. Hence, Table 1 shows that they were partially able to perform real-time occlusion. This issue limits their method to applications wherein the objects are in direct view of the sensor at the time the frame that it is rendered and prevents natural interactions between the real-world objects and virtual objects.

A method developed by Lesniak and Tucker captured RGB data and depth data for each pixel, and then processed them into a colored point cloud model [15]. Using that point cloud data, the authors were able to find surfaces and detect primitive models of objects in the scene and continue to track them as they moved off the scene. These functionalities have been described as primitive models and model permanence in Table 1. The more structured information from the data enabled their method to occlude virtual objects behind real-world objects, as those real-world objects now had associated virtual objects with shaders attached to them. However, their method was also constrained by the use of depth sensors that were required for achieving AR occlusion. Since the proposed method is most closely related to the method presented by Lesniak and Tucker [15], subsequent literature review sections will compare and contrast their method against what is being proposed in this work.

2.3 Sensors for AR Data Acquisition

The first layer of the AR process is the sensors as seen in Figure 1. The method presented by Lesniak and Tucker used a Time-of-Flight sensor: the Kinect for Windows v2 [15], that enabled the collection of RGB data as well as depth data.

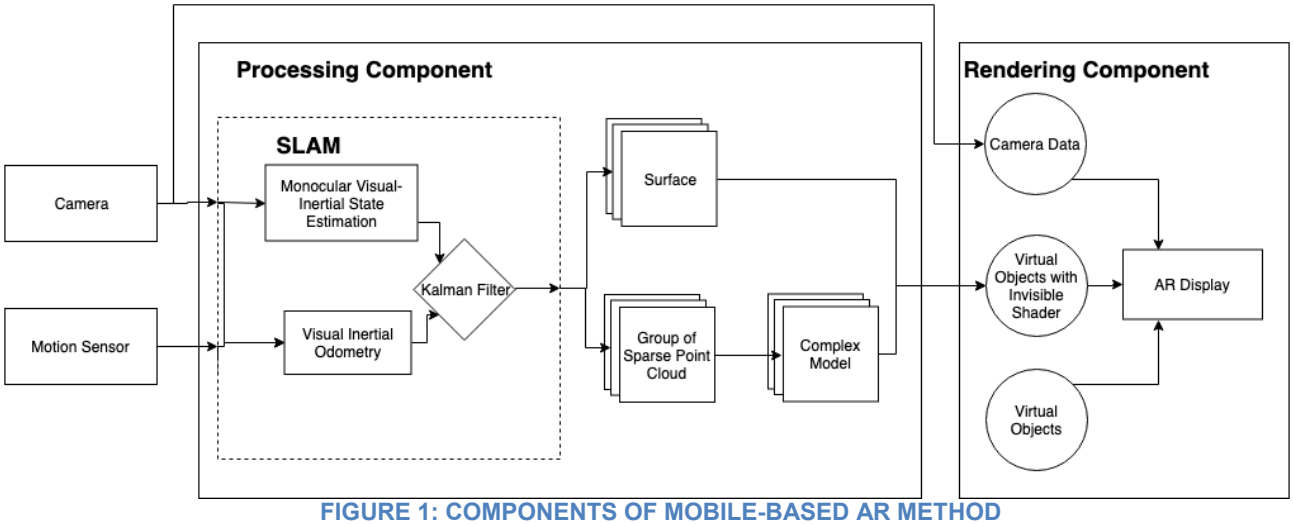

FIGURE 1: COMPONENTS OF MOBILE-BASED AR METHOD

Lesniak and Tucker's method receive RGB data as well as depth data, which supplies the method with RGB-D information [15]. Next, they transform the RGB-D data into a full-featured, colored point cloud model. As the method collects the full point cloud information, they were able to stream the entire environment to a remote system, and store it for future reference [15].

2.4 Model Identification and Localization

Lesniak and Tucker's method takes the fully-featured colored point cloud data and uses that to detect objects using a RANSAC fitting model. Their models are based on Schnabel and Xu's work [16], [17] which uses primitive models to form/process the occlusion as mentioned in Table 1. Once a set of points are found that fit a primitive model, the 3D model is manipulated in order to maximize the overlap over the points that are supposed to be covered. The fitting strategy and algorithm that was used in their experiment is similar to Stricker et al. [18].

2.5 Visualization

Lesniak and Tucker's method to visualize the environment and occlusion process uses a fully-featured colored point cloud model [15]. They use different virtual reality headsets to walk around the points in 3D space. Since their method processes their virtual objects in 3D space, they need no further processing to integrate the real and virtual environment together. This also enables them to stream the virtual environment remotely and have another user walk in the full virtual environment.

2.6 Augmented Reality without Depth Sensors

To help further the growth of AR technologies, and broaden access, different groups have released libraries that provide developers with access to advanced foundational computer vision technologies such as SLAM. Technologies have been developed that optimize device resources in order to attain accurate information while keeping the frame rate high (30-60 fps).

SLAM is a powerful tool that uses only a standard RGB cameras while still detecting depth. The technology requires significant camera movement to maintain continuous tracking of objects in the environment [5]. Recently this method has been implemented by using a process called Visual Inertial Odometry [19], [20]. This technology allows AR applications to retrieve information from the motion sensors to understand the translation amount that the camera has done. Using those observations, feature points will be found on each frame, associated with each other, and depth calculated using the parallax effect.

3. METHOD

The authors of this paper present a method that captures RGB data and uses machine learning (ML) to detect, classify and localize objects in an environment without using markers as described in Table 1. This allows us to approximate the location and position of occluding objects for an AR application, hereby reducing processing complexity. The approximation of the object's position and orientation also enables the placement of virtual models, and reduces the processing complexity of the occlusion process. This is furthered by using fully built models stored either on-device, or on a server to improve the efficacy of the occlusion while minimizing the number of physical objects needed to track. The processing component uses the phone camera to capture live video of the environment to find feature points. An overview of this component can be seen on the left side of Figure 1. RGB data is also converted to meshes and distributed to users in the environment.

After capturing feature points, data is passed to a prebuilt ML model that classifies and tracks the virtual object. Rendering, and occlusion are brought together to create the AR display, as seen on the right side of Figure 1. Initializing the Virtual Environment and Data Capture and Processing along with Algorithm 1, 2, and 3 describe the process in the first component. Figure 1, which is an extension of the work by Lesniak and Tucker's [15], will be discussed next.

3.1 Mobile Sensors and Data Acquired

The first layer of the AR process are the sensors used to collect real-world information, as seen in Figure 1. The sensor uses a standard high-definition RGB camera. For the case study presented in this paper, the authors utilize an iPhone XS, X, and iPad Pro (2018) that have a 12 MP wide-angle camera with $f/1.8$ aperture [21]. This camera will take information in at a rate of 30-60 frames per second and avoids the need to calculate depth [9]. Motion data is also collected from the device's motion sensing hardware to aid and calibrate motion translation of the camera [22]. The information that is collected from these sensors is used to calculate pose and localization of feature points, in contrast to Lesniak and Tucker's method where the depth data is provided by the Kinect depth sensor itself [15].

The RGB data that is received from the mobile device sensor is processed using SLAM as detailed in Figure 1, which generates a sparse point cloud that is used to localize objects and planes in the environment. The system avoids the need for a full point cloud to function. However, if the goal is to stream the entire environment to a remote system, as done in Lesniak and Tucker [15], it would be necessary to collect a fully-featured point cloud model.

3.2 Model Identification and Localization

To identify and localize an object in the environment, a 3D scan is necessary to form a sparse point cloud model. The orientation of the object is also necessary because if the real world object is tilted or rotated, the digital rendition of that object must also match. Capturing the orientation of the scanned object is necessary to transform the digital object to fully match the real world object. In the case study section, a scan of a generic laptop was used to evaluate this process. This scan is given in the file format of .arobject which contains the sparse point cloud now stored as spatial points, as well as the orientation data. These models are stored in a gallery of models that are used as a reference for occluding objects. Figure 5 visualizes this in greater detail.

To detect an object in the scene to occlude, the RANSAC algorithm is employed in order to align the points [16], [17]. Next, an ORB initializer/re-initializer is employed in order to maintain tracking between lost/dropped frames, followed by a pencil filter to make the frame-by-frame tracking more robust, as described in Stricker et al. [18].

Algorithm 1 Model Identification and Localization

```
1: Initialize camera;
2: Load gallery G of scan data;
3: Receive stream queue D from camera;
4: Initialize empty list p;
5: Initialize Variable S for sparse point cloud;
6: for each frame k in D do
7:    Run RANSAC on k and p
8:    Store output of previous statement to t
9:    Compare t and p for points that match
10:   Store comparison matches to variable M
11:   Run pencil filter on S and M
12:   Store output to variable I
13:   for each model in G do
14:      Run RANSAC on I and model
15:   end for
16:   Add comparison matches to S
17: end for
```

As described from Algorithm 1, from a stream of RGB data, **D**, with n frames, for each frame k, we receive a sparse point cloud **S**. Using **S**, we would perform RANSAC on each frame using the object scans in the gallery as a reference. As we move from frame k to k+1, the previous frame will be used to form feature points of the object in the real environment. These computed points are then matched with the previous sparse point cloud **S** to form a new point cloud **S'**. **S** will initially be empty but as more frames are captured, more points are captured and stored. Using a pencil filter, we will use **S** and **S'** to remove outliers more effectively and limit the scope of inliers which will then be reprocessed by Schnabel's RANSAC algorithm that then approximates a fit of a model that matches the real world object [16].

When an object has matched, we can drop an anchor and continuously track it by preforming the steps above. As seen in Figure 2 below, we can overlay items on the object and manipulate the virtual objects based on the orientation of the physical objects.

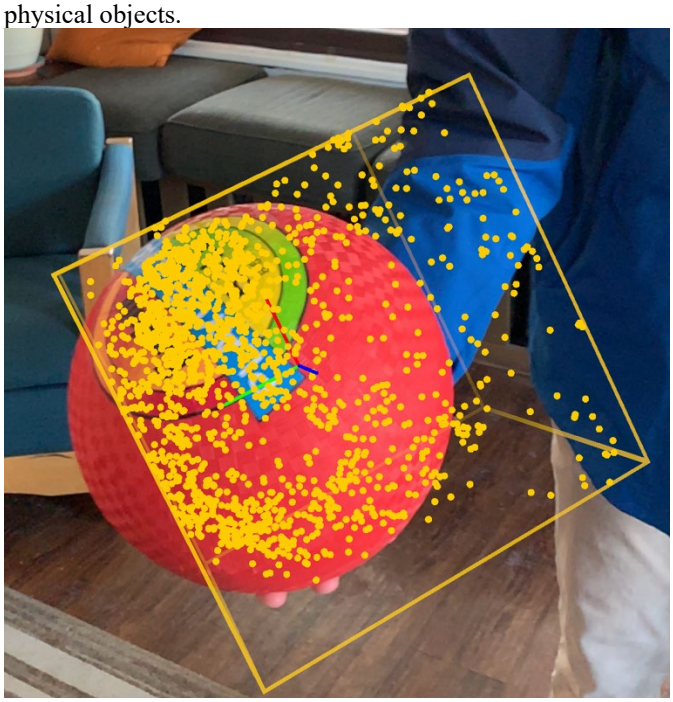

FIGURE 2: VISUALIZATION OF THE DETECTION AND TRACKING OF A RED BALL

3.3 Model Generation

Once a physical object has been identified, the virtual object that represents it will be overlaid. The virtual objects are stored either in a gallery on the device or on a server and retrieved remotely. Once the 3D model has been identified, the AR application only needs to align the center-point to the center of the anchor placed for the physical object. The orientation of the 3D model also needs to be aligned with the orientation of the anchor. Once the object has been placed in the anchor's location, the object's position will update as the tracking anchor moves. This minimizes computational complexity for generated model updates as anchor updates can be based on just a couple of feature points. Compare this to a fully-featured point cloud which could require analysis on thousands of points [13].

Finding the 3D model from a gallery will also be less computationally expensive as the scanned gallery model provides an effective key for different model groups. As long as a discrete 3D model for every scan exists, it is possible to use the classification as a hash key to store and find the exact 3D model needed to occlude the physical object. The virtual objects inserted into the AR environment could also be used for collision detection to enable virtual objects to seemingly have a 1-way interaction with the real environment.

3.4 Scene Visualization

Now that the physical object is represented by a virtual object, it is possible to visualize the entire scene on the mobile device screen. Since the AR application does not have a point cloud model with enough detail to visualize the environment, it uses the RGB data from the mobile camera as depicted on Figure 1. This, coupled with the 3D localization information of all the objects in the scene, provide enough information to overlay them on top of the RGB data from the mobile camera without depth information.

3.5 Initializing the Virtual Environment

To start the visualization of the virtual world, a world origin is placed in the environment to use as an origin position for the virtual objects and a reference point for nearby devices. Instead of collecting a sample of RGB-D data as in [15], the device starts detecting feature points in the processing component to discover surfaces and positions. That information in reference to the world origin, will establish a map of the environment. Using a trained image recognition model and the relative position between the mobile device and world origin, a new sparse point cloud is calculated. This is constructed by the feature points between the different frames and the mobile camera translation. It then uses the stored object scans for occlusion. This process is further described in Algorithm 2.

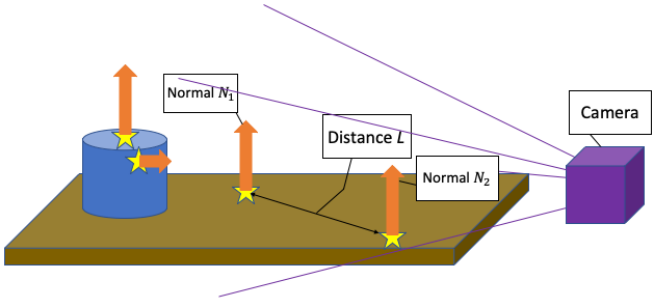

FIGURE 3: FLAT SURFACE DETECTION

These points are then grouped using a multi-stage process as described in Algorithm 2. The first pass of this process detects flat surfaces. Figure 3 shows a simple collection of points and how a surface would be detected. In this figure, the feature points are represented by the dots, while the flat areas are planes detected in the scene. During the first step of this process, each point is compared to its nearest neighboring point. The distance between each point and the angle vector between the points, determines if it is classified as a surface. If the distance is less than the threshold determined *a priori* and the angle on one of their axis is less than a predetermined angle threshold, then the surface points are associated in the same surface group. Once all of the points have been considered in a view, planar equations that define the surface for each group are generated by aggregating the normal vectors and point locations.

Algorithm 2 Initializing the Virtual Environment

Input:
C ⇒ RGB image data
I ⇒ Inertial Motion Data

Output:
W ⇒ World Origin location and orientation as vector tuple
S ⇒ Set of Feature points as vector[]
P ⇒ Set of Surface locations as anchor[] (struct of vector[])

Initialize camera;
Initialize inertial motion sensor;
Initialize SLAM processing;
Receive **C** and **I** from sensors;
Run SLAM on **C** and **I**;
Get set of feature points **S** from SLAM function;
Identify closest feature point **F**;
Receive current orientation vector from camera as **O**;
Assign world origin **W** to tuple of vectors **F** and **O**;
Group feature points **S**, together based on proximity, orientation, similarity;
Identify feature point groups that match flat surfaces as **P**;
Initialize Object detection model, **D**
for *points* in **S** not included in **P** **do**
  Group feature points *points*, into grouped set **G**;
  for each *group* in **G** **do**
    Fit *group* to distinct 3D model, **M**
    if **M** is not **NULL** **then**
      Normalize *group* points to 3D area
      Assign object anchor **A**, the 3D position vector of the normalized *group* points in view
      Load 3D model **M** at anchor position **A**;
    end if
  end for
end for

3.6 Data Capture and Processing

Each new RGB frame of data that is captured by the mobile device sensor is processed by SLAM to find feature points as seen in Figure 1. This allows us to build a generated sparse point cloud without having to use depth data as seen in [15]. The sparse point cloud is then used to determine if surfaces and complex models could be formed in the environment. The processing section differentiates if a surface or model needs to be moved or if new models and/or surfaces need to be added. Any new data not fit to a model or surface will be rendered as a feature point. They will be saved to continue tracking the location of other virtual objects in real space and to be used by the next frames as a reference point for future models/surfaces.

Algorithm 3 Data Processing

```
1:  Identify feature point groups that match flat surfaces as P
2:  Initialize Object detection model, D
3:  for points in S not included in P do
4:    Group feature points points, into grouped set G
5:    for each group in G do
6:      Fit group to distinct 3D model, M
7:      if M is not NULL then
8:        Normalize group points to 3D area
9:        Assign object anchor A, the 3D position vector of the normalized
          group points in view
10:       Load 3D model M at anchor position A
11:     end if
12:   end for
13: end for
```

If a set of points are not associated with a surface at initialization, those points will be processed to check for complex models that could be used for occlusion as seen in Algorithm 3. Each group of points from the previous step is analyzed for scanned 3D models that we have modeled for: computers, cups, bottles, etc. The fitting is done by using a RANSAC algorithm similar to Schnabel and Xu's work [16]. For each detected group, a new thread is made to determine what model and transformation on the model are best for the points in the group. This is done by identifying which model most closely resembles the group from the set of feature points. The RANSAC method is used to determine which model to use and what transformation on that model best fits as is shown in Algorithm 3.

All of the processing is done on a separate chip and run in parallel. For each detected group a new thread is started to match the models. The threads for each group run in parallel to reduce the time it takes to find fitted models for all groups when compared to testing the groups sequentially.

3.7 Augmented Reality Occlusion

The rendering component in the proposed method handles combining the virtual objects, the RGB image of the real environment, and the surfaces and models found in previous sections. Since the surfaces and models are generated from the RGB data that is captured by the sensor, this allows the real-world objects to correctly occlude the virtual objects in the AR component.

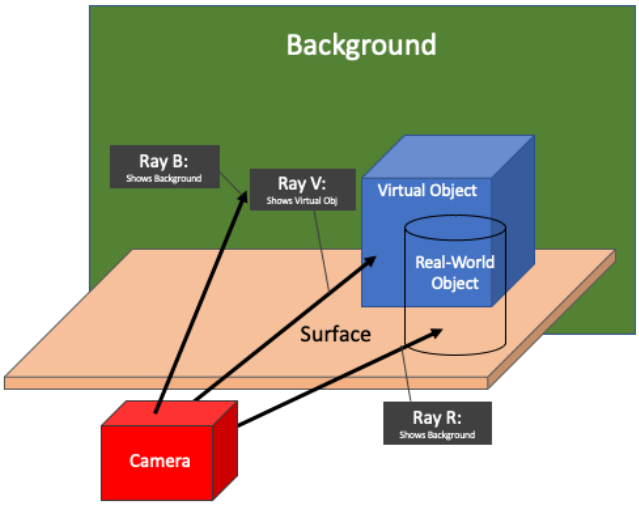

FIGURE 4: RAYCAST OPTIONS

During the rendering process, rays are cast into the virtual environment for each pixel in the display. The result of this ray-cast will result in three possible outcomes depicted in Figure 4. The first outcome, depicted by Ray **B** in the figure, is that the ray contacts the background. If this is the case, the rendering engine renders the normal RGB camera data. The second outcome, depicted by Ray **V** in Figure 4, is that the ray contacts a virtual object. In this case, the rendering engine renders the part of the virtual object that the ray hit. The third outcome, depicted by Ray **R** in Figure 4, is that the ray contacts one of the surfaces or models generated for real-world objects. For the pixel corresponding to this ray, the rendering engine renders the normal RGB camera data. The rendering engine differentiates between the virtual objects created when the AR component was developed and the surfaces and models from Initializing the Virtual Environment or Data Capture and Processing because of a unique invisible shader that is placed on the surfaces and models.

4. APPLICATION

The hardware used in the case study consisted of an iPhone XS, iPhone X, and iPad Pro (2018). Each of these devices runs with 3 distinct chips (CPU, GPU and Neural Core) [23], [24]. These devices contain accelerometer and gyroscope that allows us to gather information to calculate the pose of the objects in view [19]. The physical location is in the D.A.T.A. Lab workspace at Penn State, University Park Campus.

4.1 Initializing the Virtual Environment

At the beginning of the case study, the application and scans are loaded onto the devices, see Figure 2. A laptop is placed on a table to represent the occluding object. From there, the application is loaded on the devices, which is moved laterally to collect data. This process collects similar feature points and

forms a sparse point cloud of the environment, bringing it into the virtual environment as seen in Figure 5 below. Next, virtual objects are placed and surfaces and models are detected.

FIGURE 5: SPARSE POINT CLOUD FROM .AROBJECT

4.2 Data Capture and Processing

After the initialization of the virtual environment, different sparse points can be clustered to form either surface planes or complex 3D models to occlude. If neither work, the algorithm will continue to track the point to maintain depth understanding and mobile camera location in 3D space. The method will also maintain the list of active feature points by checking if old feature points were valid, invalid, or moved, as objects can continuously move throughout the scene [25]. Objects that are no longer a part of the static background are removed or updated to a new location.

4.3 Augmented Reality Occlusion

The rendering component was constructed in Swift ARKit [23]. The data received by the TCP connection was used to load the complex object that closely represented the detected real-world objects. Data was also sent on UDP between multiple devices to share the world coordinate system and share the experience between multiple devices. The results of the case study are provided and discussed in section 5.

5. RESULTS AND DISCUSSION

The goal of this method is to remove the need for markers in an environment, remove the need for depth sensors in an environment, reduce the computational complexity of the method, maintain the occlusion efficacy of the method, and maintain a high frame rate of the rendering process. This paper has presented that markers are not necessary with this method. It has also presented that depth sensors to capture RGB-D data are no longer needed with this method. To validate that the method is still able to demonstrate effective real-time occlusion on mobile devices, three key metrics were measured: Computational Complexity, Frame Rate, and Occlusion Efficacy.

5.1 Computational Complexity

The computational complexity metric seeks to determine whether the method is able to perform the occlusion process without the need of laptops/desktops or expensive hardware to support the process. The goal is to show reduced computational requirements for this method compared to the benchmark, the method by Lesniak and Tucker [15].

As mentioned in the Application section of this paper, the AR application was able to run on an iPhone XS, an iPhone X, and an iPad Pro using this method. The method that Lesniak and Tucker described ran on a MSI GT72VR laptop with an Intel Core i7 CPU and nVidia GTX 1060 GPU [15] and would be unable to run on the mobile devices used in this work due to the computational requirements of both the depth sensor and visualization systems used in [15].

5.2 Frame Rate

This metric is key in validating the real-time effectiveness of the method. Typically graphic based applications that run above 30 frames per second (fps) are considered real-time as human perception of each frame difference is minimized before that rate. To measure this, rendering debug information was captured and visualized at the bottom of the screen, as seen in Figure 6. While running, the lowest frame rate observed was 30 frames per second. Furthermore, the application ran at 60 frames per second (upper bound) over 50% of the time. This result demonstrates the runtime capacity and real-time effectiveness of the method.

5.3 Occlusion Efficacy

The effective real-time occlusion between real and digital objects has been shown in Figure 6. The tracking of real objects as they move is consistent, and the virtual objects that are attached to them move at comparable speeds (see Figure 6). The surfaces and models also provide a foundation for continued research into functionality and visualization that can further aid users to provide even more realistic AR displays. This is discussed in more detail in the Conclusion section of this work (Section 6).

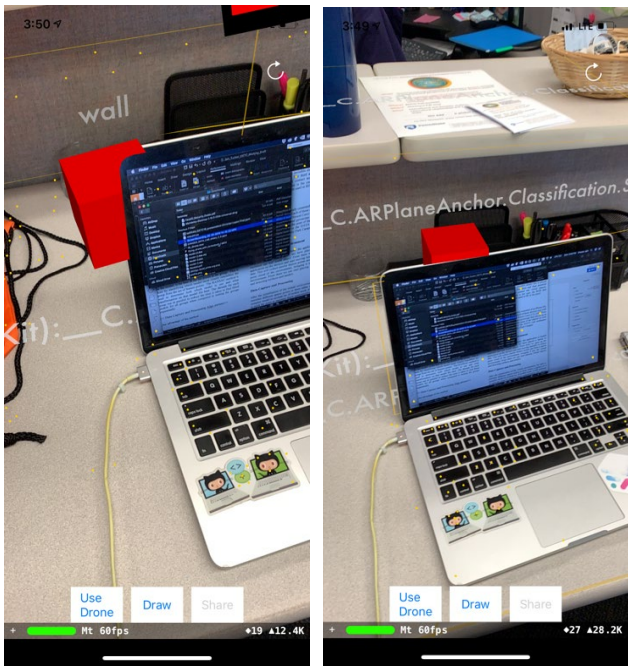

FIGURE 6: RESULTS OF THE OCCLUSION PROCESS IN IMAGES

5.4 Other Findings and Work

One of the most important pieces of technology implemented in this method was used to detect and track real world objects using standard RGB mobile cameras. This technology enabled an AR application to classify and localize objects in the scene which was then used to place an invisible mesh of that object. However it was found that the tracking technology had the potential to be used in more innovative ways as well.

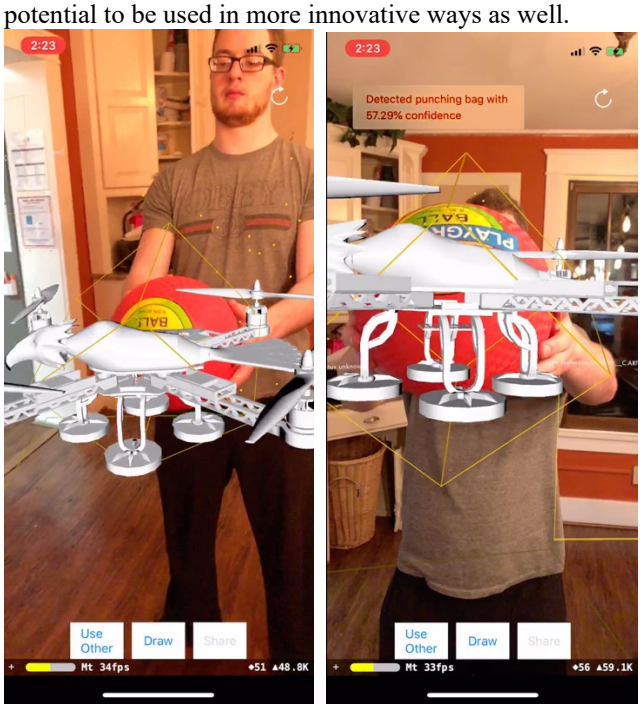

FIGURE 7: VIRTUAL DRONE FOLLOWING REAL BALL

As seen in Figure 7, the same method that was used to detect real world objects to occlude with, has also been shown to track objects for a virtual drone to follow. Each object would need a classification added to the detection model. For this scenario, a ball was classified and localized in 3D space. A virtual drone that was then placed in the scene, followed the tracked ball wherever it moved. Future work aims to have the drone be a real drone and have commands sent to it from a mobile device running this application. With the mobile device detecting the real ball, it is possible to get the real drone to find and go to this ball with no cameras attached to the drone.

6. CONCLUSION

The method proposed by the authors allows for an AR application to occlude digital objects behind real objects on mobile devices. A mobile camera was moved around to process the real world environment and render components of this method in real-time. While moving around the environment, the method detects, classifies, and tracks real-world objects to provide virtual representations of them in the AR environment. Doing this allows it to remove the need of markers or RGB-D sensors that add cost and setup complexity. The use of mobile devices such as the iPhone XS and iPhone X [23], and their standard RGB mobile camera to collect data, gives the method flexibility to run on mobile devices, compared to [15] that is primarily desktop based due to its computer requirements.

Further studies need to be performed in order to determine whether the scanning and detection models can scale effectively to be used in a more generic environment. Further tests need to also be made on non-mobile device cameras to see if the technology can be repeated on any platform of choice. The authors acknowledge that there are many potential improvements and optimizations for the proposed method. This paper should serve as a basis for further research into this topic, with possible extensions listed below:

- Further optimization on model fitting algorithm
- Including multiple platforms to run same method to show cross-platform capability
- Communicating with multiple mobile cameras to speed up the occlusion process
- Human subjects trials to determine the usability of the proposed method, compared to similar systems

7. ACKNOWLEDGEMENT

This work is in part supported by the National Science Foundation under Grant Number 1650527. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors of this work would like to acknowledge:

1) Penn State D.A.T.A. Lab for support and resources

2) Penn State Industrial and Manufacturing Engineering Department for providing workspace
3) Penn State Engineering Design Department for providing a space to perform the case study

REFERENCES

[1] Statista, "Forecast augmented (AR) and virtual reality (VR) market size worldwide from 2016 to 2021," *https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/*, 2018. .

[2] N. Mathew, "Why is Occlusion in Augmented Reality So Hard?," *Hackernoon*, 2018. [Online]. Available: https://hackernoon.com/why-is-occlusion-in-augmented-reality-so-hard-7bc8041607f9. [Accessed: 12-Nov-2018].

[3] S. Kim, M. A. Nussbaum, and J. L. Gabbard, "Augmented Reality 'Smart Glasses' in the Workplace: Industry Perspectives and Challenges for Worker Safety and Health," *IIE Trans. Occup. Ergon. Hum. Factors*, 2016.

[4] K. Lebeck, T. Kohno, and F. Roesner, "How to Safely Augment Reality : Challenges and Directions," *Proc. 17th Int. Work. Mob. Comput. Syst. Appl.*, 2016.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceeding Eighteenth national conference on Artificial intelligence*, 2002.

[6] D. Fofi, T. Sliwa, and Y. Voisin, "A comparative survey on invisible structured light," in *Machine Vision Applications in Industrial Inspection XII*, 2004.

[7] Apple, "Face ID Security," 2017.

[8] M. Gupta, A. Agrawal, A. Veeraraghavan, and S. G. Narasimhan, "Structured light 3D scanning in the presence of global illumination," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011.

[9] L. Li, "Time-of-Flight Camera–An Introduction," *Texas Instruments - Tech. White Pap.*, 2014.

[10] Y. Sumi, Y. Kawai, T. Yoshimi, and F. Tomita, "3D object recognition in cluttered environments by segment-based stereo vision," *Int. J. Comput. Vis.*, 2002.

[11] A. D. Wilson and H. Benko, "Projected Augmented Reality with the RoomAlive Toolkit," 2016.

[12] A. D. Wilson and H. Benko, "Holograms without Headsets: Projected Augmented Reality with the RoomAlive Toolkit," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, 2017.

[13] Y. Tian, Y. Long, D. Xia, H. Yao, and J. Zhang, "Handling occlusions in augmented reality based on 3D reconstruction method," *Neurocomputing*, 2015.

[14] T. C. Young and S. Smith, "An interactive augmented reality furniture customization system," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[15] K. Lesniak and C. S. Tucker, "Real-Time Occlusion Between Real and Digital Objects in Augmented Reality," 2018.

[16] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Comput. Graph. Forum*, 2007.

[17] Y. Xu, S. Tuttas, L. Hoegner, and U. Stilla, "Geometric Primitive Extraction from Point Clouds of Construction Sites Using VGS," *IEEE Geosci. Remote Sens. Lett.*, 2017.

[18] D. Stricker, M. Schneider, J. Rambach, O. Artemenko, and A. Pagani, "6DoF Object Tracking based on 3D Scans for Augmented Reality Remote Live Support," *Computers*, 2018.

[19] M. K. Paul, K. Wu, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, "A comparative analysis of tightly-coupled monocular, binocular, and stereo VINS," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[20] J. L. Schönberger, M. Pollefeys, A. Geiger, and T. Sattler, "Semantic Visual Localization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[21] Apple Inc., "ARKit: Apple Developer Documentation," *Apple Developer*, 2018. .

[22] Apple Inc., "Understanding World Tracking in ARKit," 2018. .

[23] Apple, "A12 Bionic," *Apple*, 2018. [Online]. Available: https://www.apple.com/iphone-xs/a12-bionic/.

[24] D. Banerjee, "A Microarchitectural Study on Apple's A11 Bionic Processor," 2018.

[25] K. Lesniak and C. S. Tucker, "Dynamic Rendering of Remote Indoor Environments Using Real-Time Point Cloud Data," *J. Comput. Inf. Sci. Eng.*, 2018.