Dynamic Occlusion of Virtual Objects in an 'Augmented Reality' Environment

David Sandström

Computer Science and Engineering, master's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
This thesis explores a way of increasing the perception of reality within an “Augmented Reality” application by making real objects able to obstruct the view of virtual objects. This mimics how real opaque objects occlude each other and thus making virtual objects behave the same way will improve the user experience of Augmented Reality users. The solution uses Unity as the engine with plugins for ARKit and OpenCV. ARKit provides the Augmented Reality experience and can detect real world flat surfaces on which virtual objects can be placed. OpenCV is used for image processing to detect real world objects which can then be translated into virtual silhouettes within Unity that can interact with, and occlude, the virtual objects. The end result is a system that can handle the occlusion in real time, while allowing both the real and virtual objects to translate and rotate within the scene while still maintaining the occlusion. The big drawback of the solution is that it requires a well defined environment without visual clutter and with even lighting to work as intended. This makes it unsuitable for outdoor usage.
This project was done in the first half of 2018. It is my Master’s Thesis in Computer Science and Engineering at Luleå University of Technology. I would like to thank Staffan Johansson and Johan Hedlund at Neava Technologies AB for supporting me during the project and letting me use their offices in Luleå to work in. Thanks to Prof. Peter Parnes at Luleå University of Technology for being my supervisor during this project.

David Sandström
# CONTENTS

**CHAPTER 1 – INTRODUCTION**

1.1 Background .......................... 1
1.2 Problem Definition .................. 2
1.3 Delimitations ......................... 2
1.4 Related Work ......................... 2

**CHAPTER 2 – THEORY**

2.1 The Occlusion Problem ............... 4
2.2 Method ................................ 5
   2.2.1 Finding a Horizontal Plane & The Distance to the Object 6
   2.2.2 Finding Real Objects in the Scene and Constructing the Silhouette 7
2.3 Design Choices & System Architecture 8

**CHAPTER 3 – RESULTS**

3.1 Experiments .......................... 11
3.2 Evaluation ........................... 11
   3.2.1 Scene 1 .......................... 12
   3.2.2 Scene 2 .......................... 13
   3.2.3 Scene 3 .......................... 14
   3.2.4 Scene 4 .......................... 15
   3.2.5 Scene 5 .......................... 16
   3.2.6 Scene 6 .......................... 17
   3.2.7 Final results ...................... 18

**CHAPTER 4 – DISCUSSION**

4.1 Project Execution ..................... 21
4.2 Ethical Discussion .................... 21
4.3 Future Work .......................... 21
4.4 Reflection on the Project ............. 22
'Augmented Reality' (AR) is an exciting and relatively old field in computer science but which has only recently had a massive increase in popularity. Augmented Reality aims to combine the real world with the virtual world in a way that feels natural and immersive using some type of screen to overlay virtual objects onto the real world which is being captured by a camera in real time. The aim is to have the real and virtual objects within the user’s environment seamlessly merged. For it to be convincing, real and virtual objects have to interact realistically. A common example is putting a virtual cup on a real table which will look and behave like a real cup. Many companies are now investing to try and be at the forefront of the constantly evolving technology. There are a multitude of fields where Augmented Reality is being used, like gaming, education and information sharing. More use cases are being researched and developed every day.

The choice of studying a problem in Augmented Reality was made due to the aforementioned increase in popularity of the field and to gain some experience which will likely be useful in the future.

1.1 Background

This project started based on the idea of wanting to show customers what their future house would look like in its environment before it has begun construction. Being able to place a virtual house on an empty lot in an Augmented Reality setting will give a greater impression than only seeing a picture of the house model. However, this virtual house would not be occluded by real world objects, like trees or other buildings, as it would if it was a real house. Thus the idea came to try and investigate ways to provide dynamic occlusion of virtual objects by real objects in an Augmented Reality scene.
1.2 Problem Definition

The goal of the project is to be able to have real objects dynamically occlude virtual objects in an Augmented Reality scene. Both the real and the virtual objects are allowed to translate and rotate in the scene while still maintaining the occlusion. The virtual object should also be aware of whether it is in front of a real object or behind it, relative to the camera, which will decide whether the occlusion should occur or not. Questions that need to be answered are:

- Does the occlusion work in an idealised environment (eg. a uniform table with few objects)?
- Does the occlusion work in a less controlled environment (eg. outdoors in the street)?
- Does the occlusion work if the real or virtual objects are translated and rotated?
- What factors increase or reduce the performance of the system in regards to the occlusion?

1.3 Delimitations

The amount of frames per second achieved by the application, to generate a good user experience, will not be taken into consideration when designing the system. The goal is not to create an application for widespread use where user experience would be an important variable.

The solution is primarily developed to work in an idealised environment, where the objects and background are well defined. This means an indoors environment with even lighting where the occluding objects do not look too similar to the background. This was decided to limit the scope of the project and keep it simple, reducing the variables needed to be taken into account, as this is also the environment that currently works best for Augmented Reality. The performance in outdoor environments will still be evaluated.

The platform used will be an iPad and there is no guarantee that the system will work on any other similar platform. Developing for many platforms would be an unnecessary waste of time because it is just a proof of concept, not a product.

1.4 Related Work

There have been many approaches to the handling of the occlusion problem within Augmented Reality. They can mostly be separated into two different types: depth-based and model-based.
A model-based approach sometimes uses prior knowledge of real objects which have then been translated into geometric models. In the research done by Breen et al. [1] a model of the 3D scene is already known. Having a 3D model available at all times in an AR application is not feasible and would only work for specific use cases.

To remove the requirement of having a pre-made 3D model of the scene, Ong et al. [2] propose a method of first segmenting the real object silhouettes by hand in selected “key frames”. These are then used to automatically construct approximate 3D computer models which then mask the real objects. These masks are then made to be invisible while still able to occlude the virtual object. This requires user input to set up these masks which is what the approach described in this paper tries to do without.

Trying to construct a virtual 3D object that corresponds to a real object without previously known information about the object is tricky which led to researching a contour-based approach instead. That way it is not necessary to know or detect the 3D shape of the object. One solution that uses a contour-based approach was proposed by M. Berger [3]. In this paper the author analyzes and compares the edge map of the scene and the virtual object from different viewpoints to determine their relative positions. The problem here is that the virtual and real objects would have to be static in the scene and recalculated, using different viewpoints, whenever any of the objects move.

A depth-based approach by Schmidt et al. [4] uses a binocular stereo camera system for computing dense disparity maps. The occlusion of virtual objects with this method is fairly good, but most Augmented Reality devices today have to rely on one camera instead of two which makes this method unusable with them. In the future it may be commonplace to have stereo camera systems for normal users which will make this approach more interesting.

In the paper written by Tian et al. [5] the proposed solution does not use depth information or model reconstruction. Instead they let the user first define the occluding object in the first frame which is then tracked through subsequent frames. The user also defines some pixels corresponding to the background so the algorithm can divide all pixels into either object pixels or background pixels. This takes a lot of user interaction and afterwards the object will always be occluding virtual objects with no regard to whether the virtual object is actually behind of or in front of the real object.

The approach discussed in this report requires no interaction with the user other than moving the camera to find a plane. Afterwards the objects are free to move around on the plane and the algorithm can detect automatically in real-time whether the virtual object is behind or in front of a real object and perform the occlusion accordingly.
To achieve occlusion a method needs to be used to identify the real world occluding objects. This method varies greatly between different solutions and is probably the most important step in the whole algorithm when it comes to determining it’s practicality. This chapter will demonstrate one method and will also talk about the other, smaller steps required to get the whole system functional. The system architecture and the implementation will also be touched upon.

2.1 The Occlusion Problem

As opposed to a purely virtual world, it is significantly more difficult for the user to suspend their disbelief in a virtual world mixed with reality. This is due to virtual objects very easily not ‘looking right’ when held to the standard of real world objects. This is why some special effects in movies can seem so jarring and bad, when in a purely virtual scene they would be well received. The lighting or shadows may be slightly off or maybe the object is seemingly floating in the air - disconnected from the real objects around it. One obvious problem that exists in Augmented Reality is when a virtual object looks to be further away than a real object, yet it still appears in front of the real object, as can be seen in Figure 2.1. This problem is what would be solved with perfect occlusion. Achieving perfect occlusion is extremely difficult, but it does not have to be perfect to be usable in modern applications.

The main goal with occlusion is to preserve the rules of line-of-sight within Augmented Reality scenes. Any virtual objects that are behind real objects should be hidden, partially or fully, behind that real object in a realistic way. This enhances the depth perception of the object within the scene and removes any doubt as to where the virtual object actually is within the scene.
2.2 Method

The method with which this is done in this paper is to selectively prevent parts of the virtual scene from rendering on the screen based on the knowledge of the real scene gained by using computer vision algorithms. This is done in five steps:

1. Finding at least one horizontal plane in the scene.
2. Finding the real world objects situated on a detected horizontal plane.
3. Determining a relative distance between the real object and the camera.
4. Reconstructing a virtual 2D silhouette of the real object.
5. Rendering that silhouette as a transparent mask that occludes any virtual objects behind it.

The most difficult part is finding the distinct real world objects and converting them into a correct virtual silhouette. The difficulty increases dramatically with increased
complexity of the real world scene. A common street scene, such as in Figure 2.2, can contain people, trees, vehicles and other kinds of objects at various distances from the camera. Furthermore the background can consist of different types of buildings, each with their own unique sizes and shapes. Correctly identifying and separating all these entities dynamically and automatically to create realistic occlusion masks would require a lot of computing power, or a very efficient algorithm. Further limiting factors are the poor range of the current Augmented Reality devices (e.g., Apple’s iPad that will be used for this approach), which only go up to about 4 meters, and the somewhat low resolution that makes discerning smaller objects difficult.

![Figure 2.2: A common street scene with many objects of varying shapes, sizes and colours.](image)

### 2.2.1 Finding a Horizontal Plane & The Distance to the Object

Apple’s ARKit has the ability to detect horizontal surfaces, such as tables or floors. This is done by finding features in the image captured by the camera each frame and then tracking those features across frames while simultaneously reading the different motion
sensors of the device to calculate how the camera, and thus the features too, moved between the frames. After a short period of time and assuming the tracking works fine, some planes will be found. These planes can be used to place virtual objects within the application onto real world surfaces that correspond to the planes. These planes also give a notion of depth to the 2D images that are retrieved by the camera, which can be used to determine the distance from the camera to real objects that are situated on one of those planes.

If the silhouette of a real world object has been detected it can be placed correctly in the virtual world under the assumption that the object is situated on a detected horizontal plane (it can not be hovering above the plane, like a ceiling lamp), or an estimated horizontal plane. Then the points where the silhouette touches the plane can be queried to determine the distance to the camera. Whichever distance is the smallest can then be used to set the corresponding silhouette in the virtual world at the correct distance from the camera. Virtual objects in the scene can then clearly determine if they are behind or in front of the silhouette as they have the same distance parameter.

2.2.2 Finding Real Objects in the Scene and Constructing the Silhouette

Segmenting the background and foreground of a scene in a correct and effective way is a difficult problem that could be a whole project in itself. The method used in this project is first converting the image taken from the camera each frame into grayscale. A grayscale image is easier to work with to distinguish objects based on contrasts. The image is then blurred to remove some of the visual noise present in the image before doing a thresholding operation.

A thresholding operation segments the grayscale image into a binary image of just black and white. The threshold value determines the new color of a pixel based on its intensity. If the intensity is lower than the threshold value then the pixel will become black and if the intensity is higher than the threshold value then the pixel will become white. In mathematical terms it can be described as:

\[
g(x, y) = \begin{cases} 
1 & \text{if } f(x, y) \geq T \\
0 & \text{otherwise}
\end{cases}
\]  

(2.1)

Where \( g(x, y) \) is the thresholded version of \( f(x, y) \) at the threshold value \( T \). One way of obtaining the threshold value automatically is using “Otsu’s Method” [6]. The purpose of the thresholding operation is to separate the background and the foreground under the assumption that they are dissimilar in contrast. This will not produce good results in circumstances where the foreground and background are very similar.

For the thresholding to work properly it needs to know whether the background is lighter or darker than the foreground. The algorithm used for this simply calculates where the median of all the light intensity values lie and samples it for a few frames.
If the median consistently lies below a predetermined value it has determined that the background is darker than the foreground. If the median consistently lies above the value it will instead determine that the background is lighter than the foreground. This works on the assumption that the background corresponds to a larger part of the frame than the foreground.

After thresholding the grayscale image into a binary black-and-white image some optional morphological operations can be applied to remove noise and fill in small holes that may appear in the image. This is done by applying “closing” and “opening” operations. The closing operation finds isolated black areas and transform the pixels into white pixels, while the opening operation does finds isolated white areas and turn them black.

To be able to create a virtual silhouette of the object only the contour of the object is necessary to find, as the object is assumed to be fully opaque. Finding the contours in a binary image is simple, since it will be everywhere the black meets the white. Some objects can, after the thresholding, contain holes within the object. These holes are sometimes present in the object itself, in a donut for instance, while other times it is due to incorrect threshold caused by reflected light or an opaque feature with similar colour as the background. Filtering these holes out causes problems in the former case but greatly reduces the problems of the second case, which happens very often.

After the contour is found it will contain a lot of vertices due to the thresholded image having many jagged edges when zoomed in close. To alleviate some computing power when constructing the virtual silhouette the contour will first be approximated to reduce the number of vertices using the Douglas-Peucker algorithm where the epsilon is chosen to be a small percentage of the total arc length of the contour. After the vertices have been reduced a triangulation is made and then the virtual silhouette can be created. A special shader, which determines how the silhouette will be rendered, is then put on the virtual silhouette to make it invisible but still able to occlude other virtual objects. The process is visualized in Figure 2.3 except the virtual silhouette is made black instead of invisible to demonstrate its shape.

2.3 Design Choices & System Architecture

ARKit was chosen as the Augmented Reality platform due to it being one of the biggest platforms available with continuous updates being released even during the course of the project. The company that suggested the project, Neava, also has a lot of experience with ARKit so it was their suggested platform. Another popular platform is Google’s AR Core.

Unity was chosen as the engine to build system on, which has ARKit available as a plugin. Unity provides an easy to use interface and very powerful tools for handling all the virtual objects and creating the necessary user interface. Unreal Engine 4 was another option as it also supports ARKit, but Unity felt more accessible with great
2.3. Design Choices & System Architecture

Figure 2.3: Each stage of one frame being processed to create a virtual silhouette of a real object. First a frame is captured. A threshold operation is then applied. A contour is found around the object. A virtual mesh is created from the contour that corresponds to the object.

documentation both for Unity and the ARKit plugin as well as the experience that the people at Neava already have with Unity.

OpenCV \[12\] is a computer vision software library. It contains algorithms for image processing that can be used to retrieve a lot of different information from a scene. Some type of video or image processing was required for this project and rather than reimplementing common image analysis operations, a plugin for OpenCV in Unity was used to help create the automatic detection of the occluding objects.

Figure 2.4 shows the simple system architecture and also how one frame gets processed. First ARKit analyzes the frame and compares it to previous frames to find planes. Then OpenCV processes the frame to find silhouettes which are used by Unity to create the virtual meshes corresponding to the silhouettes.
Figure 2.4: Simple system architecture and the processing sequence of a single frame in the system.
CHAPTER 3

Results

After the system was implemented testing was done using the questions posed in section 1.2. The testing was done in different environments and with different objects. The results are aggregated at the end of the chapter.

3.1 Experiments

Experiments were conducted both indoors and outdoors. The surfaces used indoors were uniform tables with different colors, both light and dark, on which the real objects were placed. The objects were opaque and either shiny or matte, to determine the effect when lighting is reflected on the object. The objects were of different sizes, shapes and colors. The lighting was varied with either direct light shining on the scene causing it to be unevenly lit or an indirect light that evenly lit the scene.

Outdoors experiments were conducted on a nearby street in direct sunlight and in the shade. The surfaces experimented on were asphalt, pavement, grass etc. Occluding objects used were normal things like cars, trees and other things you can easily find outside.

3.2 Evaluation

The proposed approach was implemented on an Apple iPad (Model 1822) using Unity 2017.4, Apple’s ARKit 1.0 and OpenCV. The criterias looked at when evaluating the system in each experiment were: (1) Is the occlusion working as intended when the camera is static?; (2) Does the virtual object flicker?; (3) Does moving the camera retain the occlusion as intended?; (4) Is the occlusion only active when the virtual object is behind the real object and not in front? (5) Does adding a direct light negatively affect the scene?

The occlusion works as intended when real objects in front of a virtual object occlude
3.2. Evaluation

![Image](image.png)

*Figure 3.1: A virtual object occluded by the real object in front.*

the virtual object and real objects behind the virtual object do not occlude the virtual object. An example of how this looks is shown in Figure 3.1.

Movements of the camera, small and large, between frames may cause the algorithm to switch back and forth between what it detects as occluding objects and what it detects as the background. This can cause a flicker effect on the virtual object.

Moving the camera around a scene may cause the occlusion to fail due to changes in the background and foreground causing the algorithm to sometimes only work from certain angles within the scene.

Moving the virtual object in front of a real object should mean it is no longer occluded and moving it back behind the real object should make it occlude the virtual object again.

Changing the lighting of the scene can have an effect on the occlusion.

3.2.1 Scene 1

The first experiment was performed indoors on a white table with two black objects, as seen in Figure 3.2. The lighting was even and indirect. The occlusion works here as intended. There is no visual flickering and moving the camera around the scene retains the occlusion. Moving the virtual object behind or in front of the real objects still retains the occlusion as intended. Adding a direct light to the scene causes dark shadows on the table that are sometimes misidentified as occluding objects.
3.2. Evaluation

Figure 3.2: Scene 1: Indoors scene showing the effect without the occlusion on the left and with the occlusion in the middle. The rightmost picture shows the effect of the occlusion when adding a direct light to the scene.

3.2.2 Scene 2

The second experiment was performed indoors on a black table with white occluding objects, see Figure 3.3. The results were similar to the experiment in scene 1 except for the fact that adding the direct light had no negative effect on the occlusion. If one of the white objects intersected with the white wall it caused the whole wall to become an occluding object with the same distance from the camera as the intersecting object.
3.2. Evaluation

![Figure 3.3: Scene 2: Indoors scene showing the effect without the occlusion on the left and with the occlusion in the middle. The rightmost picture shows the effect of the occlusion when adding a direct light to the scene.](image)

3.2.3 Scene 3

The third experiment used the same black table as in scene 2 but with different objects that were more gray and green, see Figure 3.4. Due to the similarity between the real objects and the background the occlusion does not work as intended. The virtual object does not flicker. As the occlusion does not work when the scene is static it does not work when the virtual object or the camera is moved. Adding a direct light made the occlusion closer to the intended result as the objects became more distinct.
3.2. Evaluation

Figure 3.4: Scene 3: Indoors scene showing the effect of the occlusion with objects that are less distinct from the background. The right picture has an added direct light affecting the scene.

3.2.4 Scene 4

The fourth experiment was conducted outside near a car, see Figure 3.5. The occlusion did barely work as intended, with a slight issue from tree shadows, when the camera was static at a specific angle. The object was flickering and moving the camera ruined the occlusion. Moving the virtual object in front of the car did not properly stop the car from occluding the virtual object. Adding direct lights had no effect as the sunlight was too strong for it to make a noticeable difference.
3.2. Evaluation

Figure 3.5: Scene 4: Outdoors scene showing some of the problems with the occlusion algorithm.

3.2.5 Scene 5

The fifth experiment was conducted outside near a construction sign, see Figure 3.6. The occlusion almost worked when the camera was static, but sometimes the pavement was erroneously detected as an occluding object and sometimes parts of the sign were detected as part of the background. The object was flickering. Moving the camera did not retain any occlusion, instead it caused many problems with misidentifications of occluding objects. Moving the virtual object in front of the sign did not stop the occlusion as intended. Even in the shade it did not cause any difference to the scene when adding a direct light.
3.2. Evaluation

3.2.6 Scene 6

The sixth experiment was conducted outside near a tree, see Figure 3.7. The occlusion did not work fully when the camera was static. The object flickers as with all other outdoor scenes. Moving the camera did not help. As with the other outdoor scenes adding a direct light had no effect.
3.2.7 Final results

The complete results of all experiments can be seen in Table 3.1. The results demonstrate that the algorithm performs better in the indoor environments compared to the outdoor environments in regards to the criteria that was used for the evaluation.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Static Occlusion</th>
<th>Flickering</th>
<th>Allows Camera Movement</th>
<th>Occlusion Only When Behind Object</th>
<th>Unaffected By Direct Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene 1</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Scene 2</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scene 3</td>
<td>No</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Scene 4</td>
<td>Sometimes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scene 5</td>
<td>Sometimes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scene 6</td>
<td>No</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: Compilation of the results of each experiment with regards to the criteria defined above.
When the conditions are good, as described below, the system works as it was intended to. First a horizontal plane has to be found by ARKit which can then be used to place real objects on. The objects are detected and made into an invisible virtual silhouette that can occlude other virtual objects. Multiple objects can be present in the scene and they will be treated as separate objects unless their silhouettes overlap each other. At that point the system will think they are the same object until they do not overlap anymore. Both the real and virtual objects can be translated and rotated while maintaining the occlusion and this is the main strength of this solution as opposed to, for example, Berger’s solution \[3\] that was mentioned previously.

The system works best in an indoor environment with even lighting across the whole scene and where it is not too dark. Uneven lighting can cause problems with many shadows that may erroneously be recognised as part of objects. It may also cause bright spots on dark surfaces that will be misidentified. The objects need to be well contrasted against the surface for the image processing to recognise that they are different, otherwise they will not be detected at all. Some colours of the objects therefore produce bad results if they are too similar to the colour of the background. Different sized or shaped objects produce no problems. Matte objects produce better result than shiny objects due to reflected light on an object sometimes being mistaken for being a part of the background.

For outdoor use it does not work well. This is due to generally having complex scenes with high amount of visual noise and differences in lighting producing many shadows. This produces a bad thresholded image which can not be used to separate the background and the foreground as seen in Figure \[4.1\]. No experiment outdoors gave a satisfactory result.

The automatic background shade detection is sometimes unreliable and can switch back and forth depending on movement.

The current usage of thresholding has some significant drawbacks. Only the intensity of the individual pixels are considered, not any relationships between them. No guarantee is given that the pixels identified by the process are contiguous. Extraneous pixels can
easily be included when trying to separate objects from the background. Isolated pixels belonging to the object (especially near the boundaries) may also be wrongly omitted. An example of this is if the object is shiny and light gets reflected on the object which can produce erroneous results. These effects also get worse if there is a lot of noise, due to there being a higher likelihood that a pixel’s intensity does not represent the normal intensity of the surrounding pixels. Sometimes shadows of an object might be mistaken for being a part of the object.

The system has a one frame delay, which is most noticeable when the camera is moved quickly. This is due to the calculations being made after the frame has been rendered in Unity. In effect, this causes the image processing to happen on the previous frame, instead of the current frame. This problem is probably fixable with greater access or knowledge into the inner workings of Unity and ARKit.

The morphological operations caused significant loss of frame rate without improving the image processing enough to justify using.

Relating back to the questions in section 1.2, the occlusion does work as intended in an idealised environment where both the real and virtual objects are allowed to move. It does not work well in any natural environment with higher visual complexity. Shadows, reflections and multi-colored objects or backgrounds are all factors that can reduce the effectiveness of the algorithm.

In conclusion, this approach to solving the occlusion problem is not really usable in practice. There are too many cases where the occlusion fails to perform. It is not resilient to visual disturbances which would be necessary for real world application. This causes the virtual object to flicker a lot in certain situations and is an unpleasant experience for the user.
4.1 Project Execution

The beginning of the project was spent researching similar projects made by other people to get some basic ideas and understanding of the problem. A lot of time was then spent learning Unity and its ARKit plugin from scratch to gain as much experience and knowledge as possible for use within the project. Due to the sheer size of the tools it was important be familiar with what can or can not be done using them. As a first step a basic prototype was created where the user had to define the occluding objects themselves. To automate the process of finding the occluding objects there was need for a computer vision library. After some research OpenCV was chosen due to its popularity and because Unity had some OpenCV plugins available for purchase. A lot of time was spent learning OpenCV and what algorithms would be useful for the project as well as trying to integrate it with ARKit. Many different solutions were researched and implemented parallell to each other to try and find the best fit. Eventually the previously described method was chosen.

4.2 Ethical Discussion

There are no obvious ethical concerns with this system. All information that is used by ARKit, such as the camera and motion sensors, have to be granted permission to use by the user. It does not use the internet to send or receive any information. Nothing is saved to the file system, including the video feed. ARKit only tracks features between frames to detect horizontal planes or perform other calculations related to maintaining the Augmented Reality session.

4.3 Future Work

The image processing step can be improved with regards to separating the foreground from the background. Detecting objects in the foreground is an essential part of the system and if that part can be improved then the system can be applied in more complex environments with more accurate occlusion. This is however a big problem that is not easily solved when there is a moving camera which means that the background is not static. Perhaps this would be more feasible with extra equipment, such as a dual camera setup that provides two slightly different views of the same scene each frame. Calculating the differences between the views may give more information about what part of the scene is the background and what is the foreground.

The one frame delay in creation of the occluding meshes could possibly be removed with greater knowledge of the tools used. Perhaps there is a way to sync Unity, ARKit and OpenCV to remove this delay while still maintaining a smooth and responsive user experience.
The distance finding part of the system relies on ARKit and one of its detected planes. This could be made separately from ARKit to allow distance finding even when no plane has been detected. With extra equipment, such as the dual cameras mentioned before or an infrared distance sensor, it should be possible to implement without too much trouble. The algorithm determining whether the background is mostly white or mostly black using the median of the light intensity can likely be improved. If we had more information of which part of the image is the foreground and which is the background then the algorithm could become more consistent.

4.4 Reflection on the Project

Looking back, the project as a whole went well. There were however many bumps that had to be overcome which at some points instilled a lot of doubt as to whether the project would be able to be finished or not. In the early to middle part of the project, before OpenCV was chosen as a tool, there was too much focus on constructing a good manual solution to the problem instead of the automatic solution that had been propositioned in the first place. That cost a lot of time but it was not completely wasted as a lot of experience was gained with both Unity and ARKit that was useful for the remaining part of the project. After OpenCV became a part of the solution the project came back on track and then I just had to focus on the successive problems that appeared as I moved slowly but surely towards my goal.
REFERENCES


