Manoforia

Using Vuforia Object Recognition and ManoMotion Hand Gesture Recognition for User Guidance

Project Paper for the Augmented Reality Course

University of Applied Sciences Vorarlberg
Informatik – Software and Information Engineering

Submitted to
Andreas Künz, Walter Ritter

Handed in by
Team VR with Glasses (Pascal Grones, Janik Mayr, Patrick Poiger, Matthias Rupp)

Github link: https://github.com/BakaBoing/itb6-ar-project

Dornbirn, 02.05 2019

Abstract

Augmented reality is a great way to enhance user guidance, which can be split into two factors: object recognition and gesture recognition. This paper aims to assess how well these two factors can be combined into one seamless interaction between the real world and the virtual world. In order to evaluate this combination, this paper features the implementation of an AR application that provides instructions to a user by recognizing an object with the help of a camera. The application also dynamically displays information regarding that object next to it inside a container, which can be scrolled through via gestures made by the user as well as other input possibilities on a screen. Although the prototype only scratches the surface, this implementation clearly shows that object recognition and gesture recognition can be combined in a meaningful way. It also enhances the interaction between the real and virtual world. A variety of different sectors might benefit from this insight, from showing detailed instructions next to a piece of complicated machinery right where they are needed, to displaying how a printer in an office works. Gestures eliminate the need for conventional inputs, enabling the usage of user guidance in unconventional situations.

Table of Contents

List of Figures

List of Abbreviations and Symbols

APK	Android Package Kit
AR	Augmented Reality
OR	Object Recognition
UI	User Interface

Introduction

Introduction of Concepts

Augmented Reality (AR) can be defined as a view of the real world that has been augmented with additional information provided by a computer. (Carmigniani et al., 2011)

Object Recognition (OR) refers to the concept of detecting and tracking 3D-objects. In the context of this project, said detection and tracking are achieved with the help of the Vuforia Engine and either the camera of a laptop or a smartphone. (Vuforia Object Recognition, 2020)

The Vuforia Engine is a widely used platform for Augmented Reality (AR) development. It offers functionality like object and image recognition for a variety of platforms like Unity or Android. For this project, the Vuforia Engine's Object Recognition will be used in Unity. (Vuforia, Overview, 2020)

ManoMotion is an SDK that provides gesture recognition for Android devices by using the device camera. It supports Unity. (ManoMotion, 2020a)

Unity is a real-time 3D development platform that can be used for developing applications for various platforms. It is the main development environment for the Manoforia project. (Unity, 2020)

Motivation and Goals

A need for instructions for everyday items that are needed for work (e.g. how does the office printer work?) exists in nearly every job. Instead of having a current employee teaching every new employee (which takes valuable time) and having to keep current employees updated on changes in equipment like a new printer with new instructions (which is cumbersome and may be overlooked by the employees in question), Manoforia aims to store such information in one place and guide the user dynamically by using an AR approach and displaying necessary information with the help of OR, as well as allowing the user to control the User Interface (UI) using gestures.

In order to fulfil this task, the following goals will have to be met:

- A database holding objects and their respective information and instructions needs to be implemented and filled with data
- OR needs to be implemented in a way that allows a camera (e.g. smartphone, webcam, camera of AR glasses, etc.) to properly recognize said object
- Recognition of an object should lead to the corresponding information being retrieved from the database
- The information needs to be displayed in AR
- The user needs to be able to perform necessary actions (e.g. scroll through longer instructions)
- Said necessary actions should be performable by using gestures with the help of Gesture Recognition
- In case Gesture Recognition does not work or is not desired by the user, alternative ways of performing actions should be implemented

To achieve these goals, the current state of the art in regards to AR and OR needs to be understood. To this end, a literature study was conducted.

Literature Study

Syberfeldt et al. conducted a study in 2015 which also focused on enhancing a human's perception of reality with AR. They used an Oculus Rift system with added webcams and the Unity platform to develop their software. The paper does not go into detail regarding the implementation, and the main focus was on user acceptance of AR over classic paper instructions for the assembly process. (Syberfeldt, Anna et al., 2015) The study provides insight into what an AR-system needs to provide to be accepted by users.

Chen, Chang and Huang developed an AR guidance system for a museum in a study that was published in 2014. The focus of their implementation was on providing a guidance system that did not require any additional input devices (e.g. keyboard, touchscreen). They used markers in combination with a database and a brochure to allow the user to interact with the guidance system with their hands, displaying information matching the selected markers which were retrieved from the database. (Chen, Chang, Huang, 2014) This article proves that an AR guidance system is possible and contains information on a possible approach using markers.

An article published by Purnomo et al in 2018 focuses on the implementation of AR technology in the museum of Sangiran using Vuforia. It more specifically focuses on the markerless 3D recognition offered by Vuforia and the tolerances regarding metrics like detection distance and deviation angle. (Purnomo et al, 2018) Information regarding markerless 3D recognition and at which angles and distances it stops working can be found in this article.

A thesis written by Grahn in 2017 inspects the performance of Vuforia and Unity on Android devices. It also contains concise summaries of the features offered by both Vuforia and Unity as well as the theories behind these features. (Grahn, 2017) With the summarized information it provides, this thesis is a good starting point for those interested in working with Vuforia and Unity in AR.

While there is some literature on using AR for user guidance, the exact setup of Manoforia with Unity, Vuforia and ManoMotion has not been examined in current literature, which is where this paper comes into play.

Methods

Object Scanning

In order for the Vuforia Object Recognition to work, the objects that should be recognized need to be scanned first. Vuforia offers an Object Scanner for exactly this purpose. The Object Scanner is an Android app. In order to install it on an Android device, the Android Package Kit (APK) needs to be downloaded from the Vuforia webpage and installed on the device that will be used for the scanning. Before starting with the scanning, a special “Object Scanning target” is needed, which can be seen in figure 1. (Vuforia Object Scanner, 2020a)

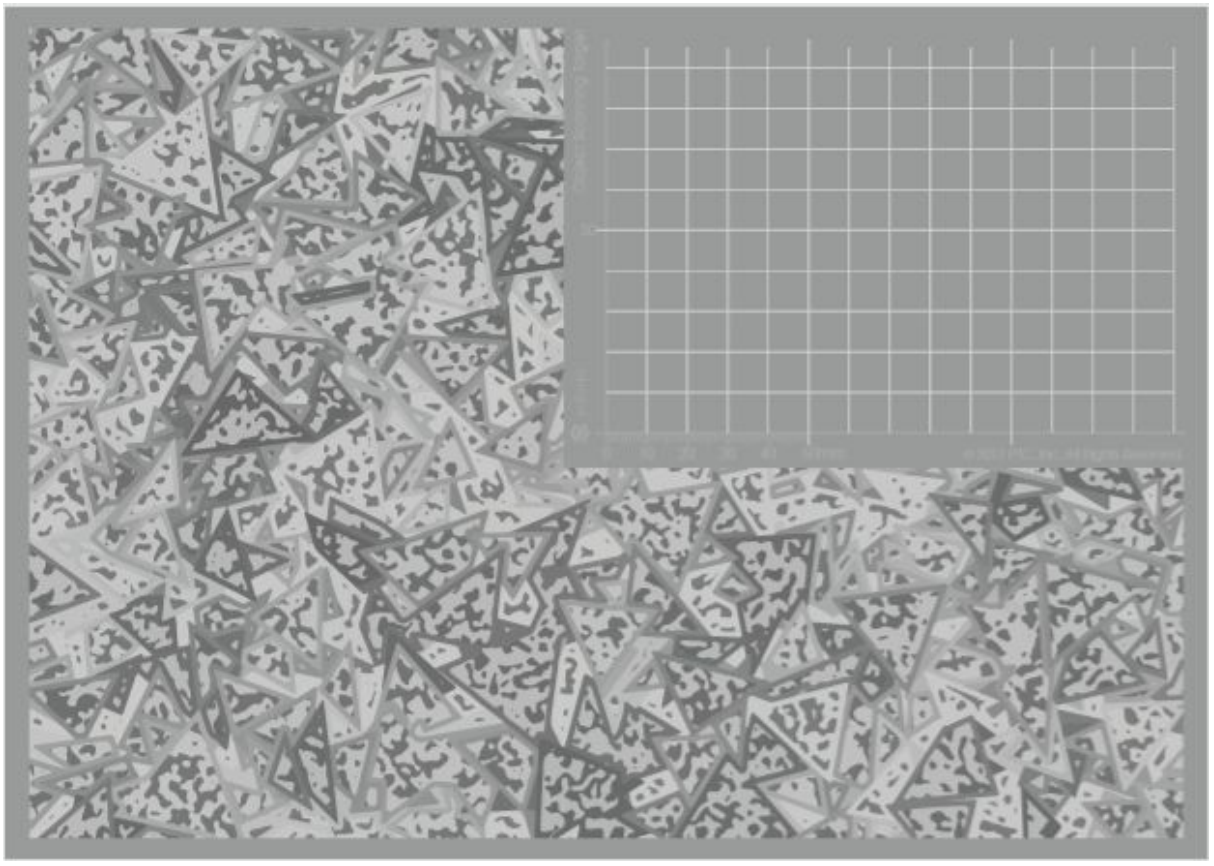

Figure 1: The underlay for object scanning

The region to the upper right is where the object that is to be scanned is placed. The rest of the underlay consists of triangular shapes which overlap. These help the scanner identify which part of the scan can be culled and in what position the object in the upper right region is placed. After printing out the overlay, the object one wants to be scanned is placed on the scanning region (the upper right, without triangular shapes). Then, the Object Scanner

application is started on the Android device and the object is scanned with the help of said device's camera. The more points are captured in the scan, the better the object can be recognized later. Figure 2 shows what such a scan may look like. (Vuforia Object Scanner, 2020b)

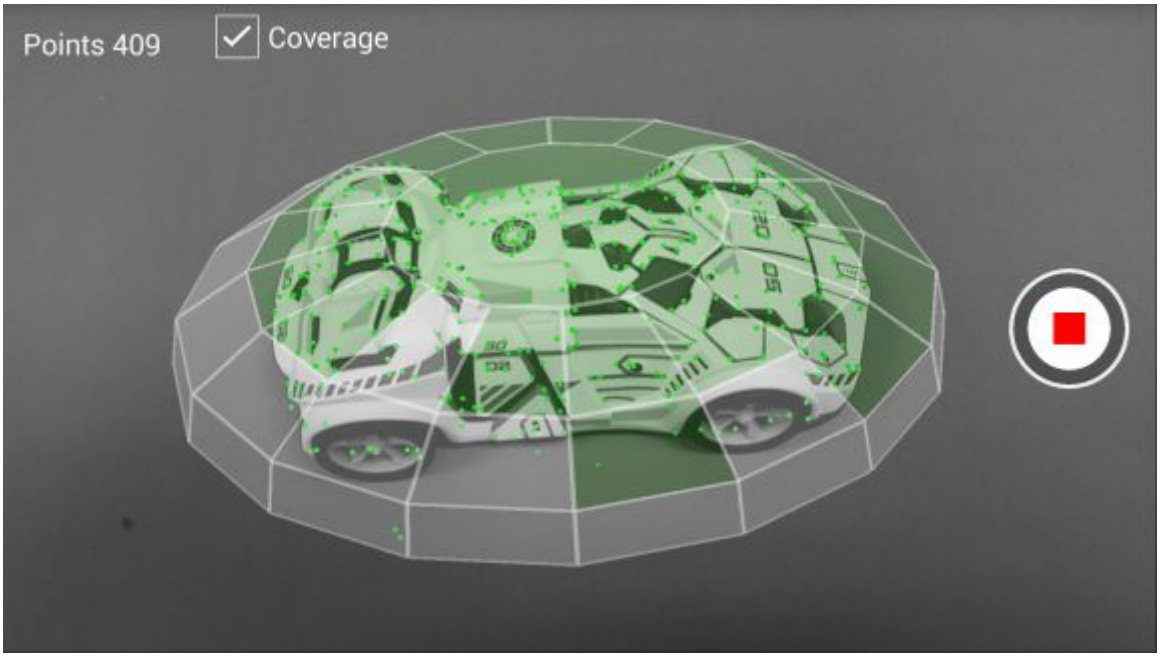

Figure 2: Object scan in progress

The little green dots show the captured points, while the green rectangles represent regions of the object's surface that have been captured. The grey rectangles are recognized surfaces that have not been captured yet. It should be noted that not all surfaces need necessarily be captured for the object to be recognized later. If, for example, only the front of the object needs to be recognized for an application, the sides and the back do not need to be scanned. After enough has been captured, the scan can be finished by tapping the stop button that can be seen on the right. For the sake of everyone being able to have an object that was scanned and can be recognized by the application, a pdf containing a QR-cube was added to the project resources. By printing out and folding said cube, an object that the application recognizes can be produced.

The application will ask you to enter a name for the created object file and save it on the device used for the scan. Said object file can then be shared over the application, on a google drive, for example. In order to add the object file to a target database, one must use the Vuforia Target Manager. It is an online tool that allows you to create databases and add object files as targets to said databases. For Object Recognition, the object file should be added to the database as a 3D-object. After enough targets have been added (to a

maximum of 20), the database can be downloaded. A development platform can be selected when downloading, which will also determine the file format of the database. Choosing Unity will lead to the database being a unity-package, for example. (Vuforia Object Scanner, 2020c).

After scanning objects, the next step is to implement their recognition.

Unity Object Recognition Implementation

For the implementation of our concept, a new Unity project was created. In order to be able to use Vuforia in said project, the first thing that needs to be done is to import the needed package. As of Unity 2019.2 or later (our project uses Unity 2019.2.3f1), all that needs to be done is to import *add-vuforia-package-9-0-12.unitypackage*, which can be downloaded from the Vuforia website. This will take care of the Vuforia import (Vuforia Engine Package Hosting for Unity, 2020).

A new scene was then created. The objects needed for Object Recognition are an AR-Camera and a Game Object containing the trackable objects. The Main Camera that is included by default in any scene created with Unity can be deleted. Instead, the AR Camera supplied by Vuforia should be used. This camera can be found by right-clicking and selecting the Vuforia Engine and then the AR Camera. The second required object, the DynamicTrackables, starts as an Empty Game Object. After renaming this Game Object, a script-Component is added with the ‘Add Component’ button. Said script will be responsible for dynamically loading the objects to track from the object database we created in the Object Scanning chapter. An excerpt from the script can be seen in figure 3:

```
void LoadDataSet()
{
    ObjectTracker objectTracker = TrackerManager.Instance.GetTracker<ObjectTracker>();
    DataSet dataSet = objectTracker.CreateDataSet();
    if (dataSet.Load(dataSetName))
    {
        objectTracker.Stop();  // stop tracker so that we can add new dataset
        Logging if error
        int counter = 0;
        IEnumerable<TrackableBehaviour> tbs = TrackerManager.Instance.GetStateManager().GetTrackableBehaviours();
        foreach (TrackableBehaviour tb in tbs)
        {
            if (tb.name == "New Game Object")
            {
                // change generic name to include trackable name
                tb.gameObject.name = ++counter + ":DynamicImageTarget-" + tb.TrackableName;
                // add additional script components for trackable
                tb.gameObject.AddComponent<TurnOffBehaviour>();
                tb.gameObject.AddComponent<PrinterTrackableEventHandler>();
                if (augmentationObject != null)
                {
                    // instantiate augmentation object and parent to trackable
                    GameObject augmentation = (GameObject)GameObject.Instantiate(augmentationObject);
                    augmentation.tag = Tags.PrinterInfo;
                    augmentation.transform.parent = tb.gameObject.transform;
                }
                elseLog
            }
        }
    }
    elseLog
}
```

Figure 3: Code example for loading Data Sets

The dataset is loaded dynamically from the database we created beforehand, so we do not need to create a new Game Object in Unity every time we add an object to the database. To achieve this, the script creates a new Game Object at runtime for every target in the dataset and appends the necessary Components, which are scripts for handling behaviour when turning off and our custom script for handling Trackable events, PrinterTrackableEventHandler. It also sets the correct Augmentation Object, which is the object used to display the information retrieved from the dataset, called InfoScreen in this project. Which Data Set and augmentation object to use can be specified in Unity, as seen in figure 4. (Vuforia Unity Documentation, 2020a)

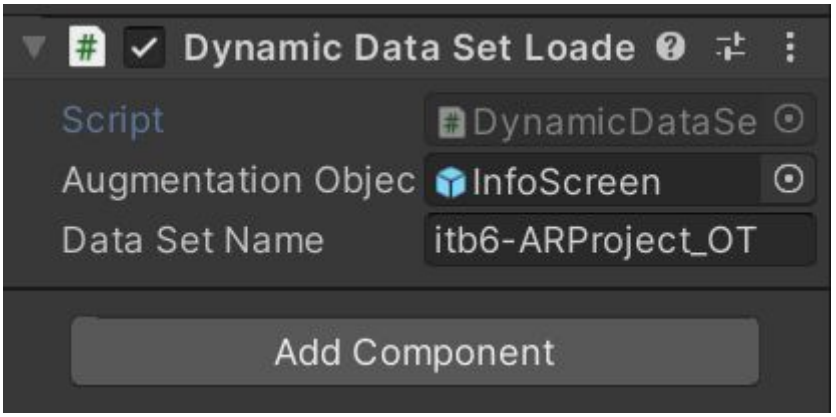

Figure 4: Configuring Data Set and Augmentation Object

For the Data Set, the name of the database appended with "_OT" is added. The name of the Data Set used in the figure is "itb6-ARProject", so the name used to specify the dataset is as seen in the figure. The Augmentation Object is a self-created prefab called InfoScreen.
The PrinterTrackableEventHandler takes care of loading the information into the InfoScreen. The next chapter shows the concrete UI implementation. (Vuforia Unity Documentation, 2020b)

With the help of the DataStoreQuery, the information for the objects in the dataset is retrieved. The script then loops over the children of the InfoScreen, which are the fields for name, paper formats and the instructions in text form, and sets the text according to the retrieved information. The datastore used for this project is a simple script file (DataStoreDummyImpl.cs) that holds the information as strings or enums in an Object, as the storing of the data was not the main focus of this implementation. The store holds:

- The name of the Trackable Object
- The paper format
- The instructions for the Trackable object

With this, the object recognition works and displays the information for objects which were scanned beforehand and had information about them entered into the datastore. If the instructions are too long, there needs to be a way to navigate through the information. For this purpose, a User Interface (UI) for the displayed information is needed.

Unity UI Implementation

Worldspace UI

When Vuforia recognises one of its Trackables, the corresponding InfoScreen Prefab (see figure 5) is displayed in the scene at the predefined position relative to the Trackable in World Space. The spawned Prefab features a canvas exposing the data from the database. It is composed of a Scroll View and a vertical Scrollbar. The Scroll View shows the current part of the content in its Viewport corresponding with the scroll progress and hides the overlapping content. The Content Game Object organises the content in custom categories. Specifically General Information and Instructions. The former category features information about the printer's name, its paper size and an image of the printer. The latter category, Instructions, holds a single text, which can be utilised to give an in-depth explanation about the trackable.

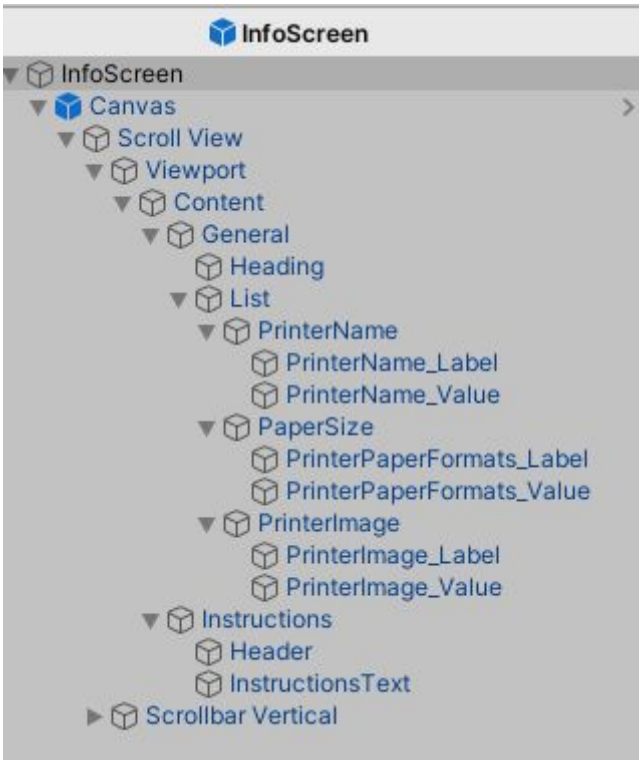

Figure 5: Hierarchy of InfoScreen.prefab

The text fields are filled by the ShowInfoScreen method in the PrinterTrackableEventHandler.cs script as seen in figure 6. In line 38 the data for the current

trackable is queried based on its name. In the next line, all child text fields of the InfoScreen object are extracted. Consequently, the fields are looped (line 41) and filled with their assigned Tag in Unity (line 43). In line 60 until 69 the scrollbar is registered to the ScrollManager, to enable the gesture-controlled scroll and on-screen ScrollButtons.

```
20      private void ShowInfoScreen()
21      {
22          if (_isInfoScreenShowing) return;
23          // Searches for a InfoScreen in the scene by tab and assigns it into _infoScreen.
24          foreach (var component in _trackableBehaviour.gameObject.GetComponentsInChildren<Component>()){...}
35
36          if (_infoScreen == null) return;
37
38          var printerInfo = DataStoreQuery.GetPrinterInfo(_trackableBehaviour.TrackableName);
39          var infoChildren :TMP_Text[]  = _infoScreen.GetComponentsInChildren<TMP_Text>();
40
41          foreach (var info :TMP_Text  in infoChildren)
42          {
43              switch (info.tag)
44              {
45                  case Tags.PrinterName:
46                      info.SetText(printerInfo.Name);
47                      break;
48                  case Tags.PaperFormats:
49                      info.SetText(string.Join( separator: ", ", printerInfo.PaperFormats));
50                      break;
51                  case Tags.InstructionsText:
52                      info.SetText(printerInfo.Instructions);
53                      break;
54              }
55          }
56          /* Manipulate Position Information of _infoScreen
57                   *   ...
58                  */
59
60          // Get Scrollbar and assigning it to ScrollManager
61          if (_infoScrollbar == null)
62          {
63              _infoScrollbar = _trackableBehaviour.gameObject.GetComponentInChildren<Scrollbar>();
64          }
65          if (_infoScrollbar != null)
66          {
67              ScrollManager.AddScrollbar(_infoScrollbar);
68          }
69      }
```

Figure 6: Method ShowInfoScreen in PrinterTrackableEventHandler.cs script

Screen Space - Overlay UI

Scroll Buttons UI:

The first "Screen Space - Overlay" UI shows optional on-screen buttons for scrolling. The buttons have an event trigger component, listening for Pointer Up and Down Events. If an event occurs, the respective scrolling action for the currently displayed Trackable is invoked. The scrolling itself works by manipulating the value of the scroll views scrollbar via its property setter.

ManoMotion Hand Recognition Feedback UI:

The second "Screen Space - Overlay" UI is a stripped-down version of the example Hand Recognition Feedback UI from ManoMotion. It provides feedback about the gesture, which was recognised inside the camera feed. In figure 11, an example of the recognition can be seen.

Unity Gesture Recognition Integration

To allow the user to scroll the InfoScreen with simple hand gestures, they need to be recognised by the application. For this purpose, the ManoMotion gesture recognition framework was integrated into Manoforia by importing the ManoMotion SDK Lite into Unity (ManoMotion, 2020b).

The sample scene provided by the SDK, called “ManoMotion SDK Lite Features”, was used as the basis for the gesture recognition, since it works out of the box and provides recognition for the two gestures we need (for scrolling up and down). Elements that were not needed, like sliders for adjusting the smoothness of the gesture recognition, were removed from the respective prefabs. In the end, the hierarchy seen in figure 7 remained.

Figure 7: The reduced ManoMotion hierarchy

The only remaining objects were the ManomotionManager and the GizmoCanvas, as these two aspects are needed for the basic gesture recognition. The gestures ManoMotion recognizes are click, pick, grab and pointer. For this project, click and pick were used. See figure 8 to see a visualization of the hand positions needed to perform these gestures. (ManoMotion Gestures, 2020)

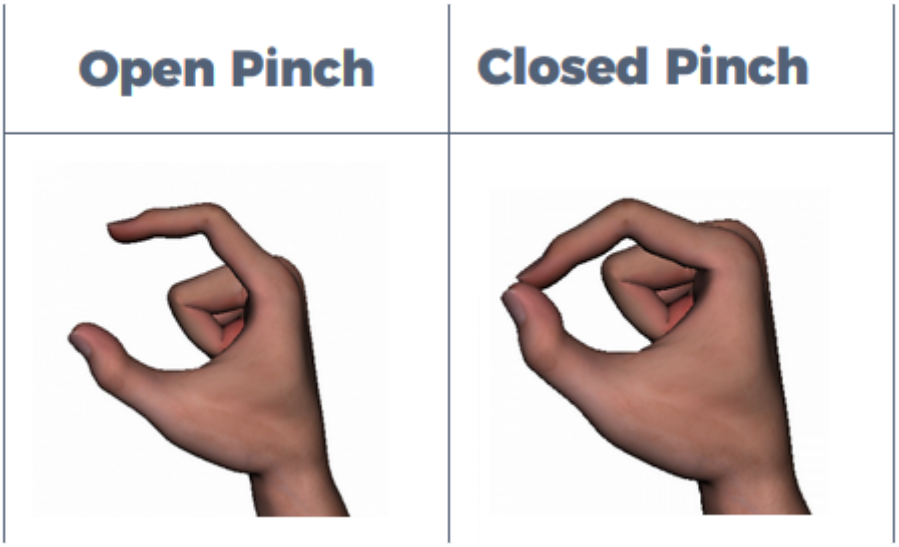

Figure 8: Hand position for starting and ending both pick and click

The Open Pinch is the starting position, the Closed Pinch is the ending position. For the pick gesture, thumb and index finger are pressed together for a longer time. For the click gesture, thumb and index finger are just touched together shortly.

Both object recognition and gesture recognition now work separately. In order to control the UI by using gestures, they need to be combined.

Combining Object Recognition and Gesture Recognition

Vuforia's object recognition and ManoMotion's gesture recognition both use their own camera input to calculate the positions of the overlays and the user's hand. This has to be addressed in order to use both libraries at the same time. The solution for this problem was to use Vuforia's camera since it is impossible to use ManoMotion's camera input inside the UI. Vuforia's input can also be used by ManoMotion without major issues, as it is manipulated before it is processed.

However, this manipulation of the input resulted in several problems while combining the functionalities of both frameworks. Most notably, a memory leak that caused the RAM usage to skyrocket and increase continuously, as shown in figure 9.

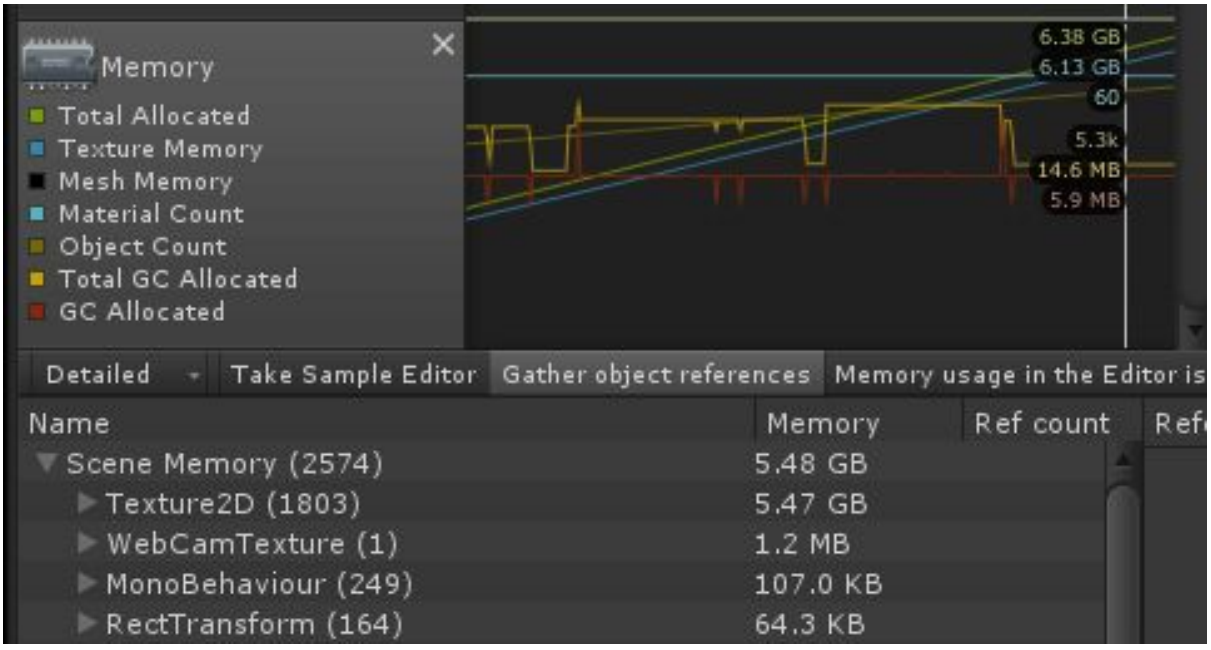

Figure 9: Memory profiler showing memory leak

The steadily rising green and blue lines represent the total memory allocated and the texture memory. Also, the cause of the memory leak can be seen, as the Texture2D uses 5.47 GB of memory after executing for roughly 10 seconds.

The cause for this lies in ManoMotion's implementation for the image processing. Each frame, a new image is created by the camera, which is then converted into a texture and ultimately into a pixel array by a method called GetPixel32. However, ManoMotion never destroys the created texture objects derived from the images, resulting in increased memory

usage over time. In order to mitigate this problem, it was first attempted to call the Destroy method for each texture in order to remove it from memory. This fixed the memory leak, but it also broke the gesture recognition. Consequently, a different solution had to be found. As the root of the problem was the creation of texture objects after every frame, a static texture was created at the start of the program, which was then used by the image processing. This can be seen in figure 10 below. This way, only one object is created, which is then overridden every frame by using the CopyBufferToTexture method. (ManoMotion Documentation, 2020)

```
private static Texture2D _tempTexture;

Event function  new *
void Start()
{
    _tempTexture = new Texture2D( width: 1,  height: 1);
}

Frequently called  5 usages  Matthias Rupp +1
public static Color32[] GetPixels32()
{
    ActualImage?.CopyBufferToTexture(_tempTexture);
    return _tempTexture.GetPixels32();
}
```

Figure 10: Mitigating the memory leak by creating only one texture and overriding it

Results

Some real-life objects were scanned with the Vuforia Object Scanner and turned into a database with the Vuforia Target Manager. This database was added to the prototype implementation. Said prototype implementation can recognise the scanned objects and displays the information that was saved for them in an UI with a scrollbar. The user can scroll through longer instructions by directly utilizing the scrollbar, pressing keys on the keyboard, pressing the UI-buttons and by making gestures with his or her hand in front of the device's camera. Figure 11 shows this in action.

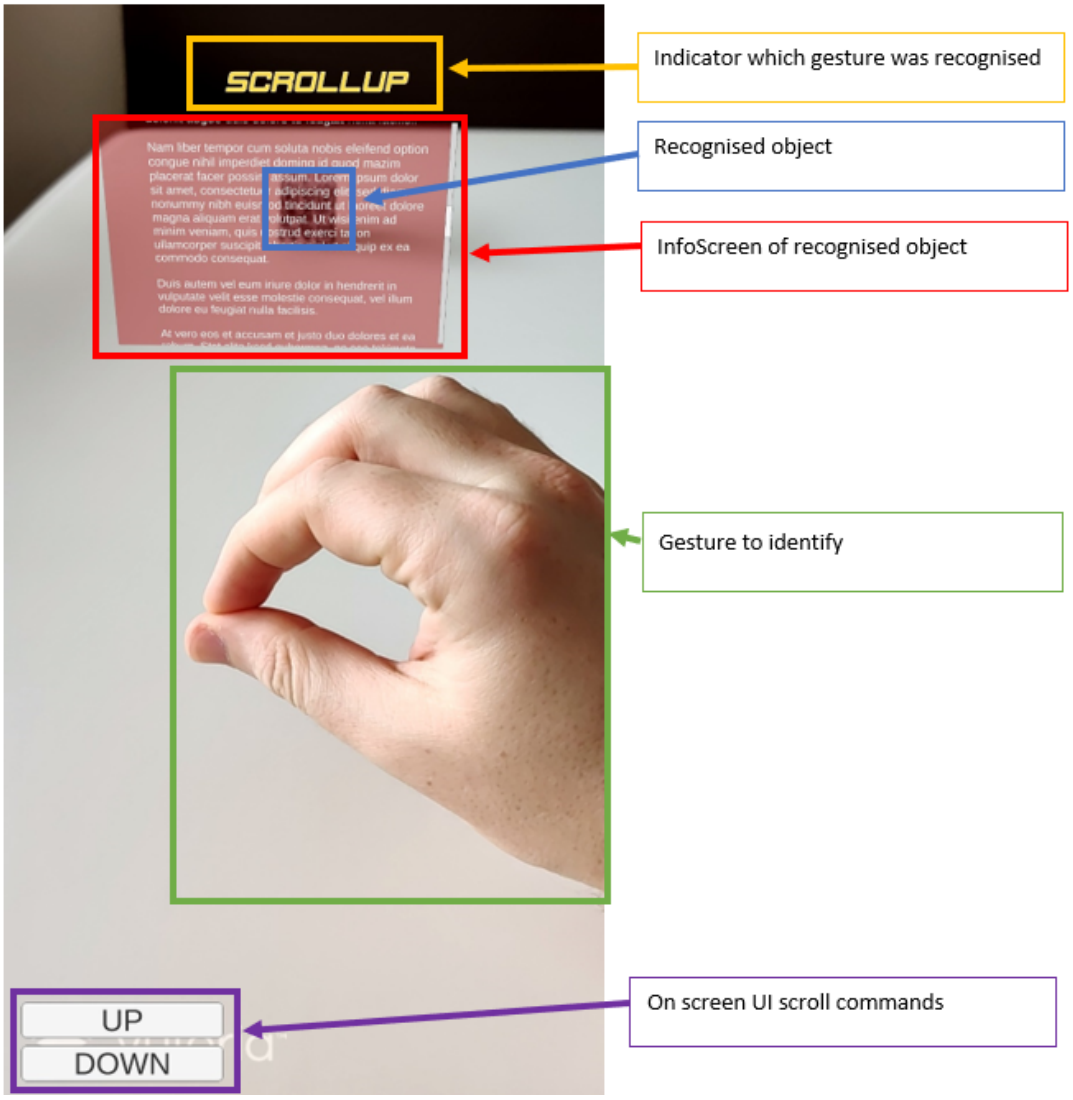

Figure 11: Manoforia in action

The information screen for the QR-cube is displayed and scrollable. The buttons are visible and functional, but the pick gesture is used to scroll up instead.

Figure 12 shows the memory profile of the finished prototype.

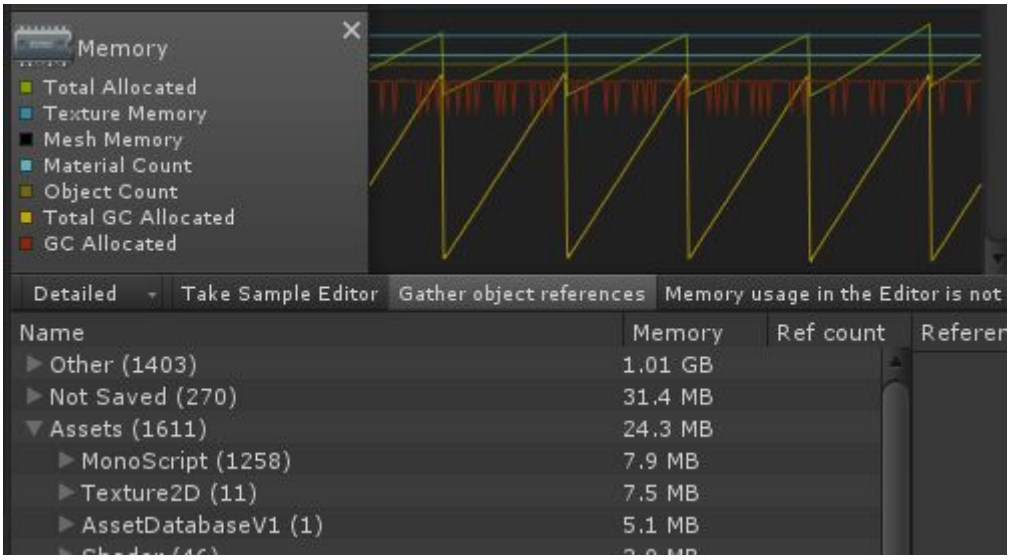

Figure 12: Memory profile after fix

Instead of steadily rising, the green and blue lines for total memory allocated and texture memory are steady. Also, Texture2D now only takes up 7.5 MB after executing for roughly thirty seconds. The memory leak has been fixed.

Vuforia and ManoMotion frameworks were combined to provide this functionality. The result of combining these two frameworks is functional, but unstable.

Discussion and Evaluation

Both the Vuforia Object Recognition and the ManoMotion Gesture Recognition work well when used by themselves. However, while combining both libraries was possible in the end, getting to that point was hard. The fact that both applications need different things as input, with Vuforia using its own camera implementation and ManoMotion requiring an image it retrieves from its own, separate camera, was the first big hurdle. The fix for this problem directly led to the next issue, as the creation of images from Vuforia's camera for ManoMotion resulted in a memory leak. Even after fixing this by using a static image as template for copying the current image from the Vuforia camera, actually using gesture recognition to scroll the scrollbar displayed with the object recognition does sometimes not work properly, as it requires both the user's hand and the object with the respective information displayed to be recorded by the camera at the same time, which can lead to interference when the user accidentally covers the tracked object when trying to use the gesture control, for example. The implementation is also not efficient and limited to Android-devices.

The main result achieved by this paper is this prototype implementation, as well as the findings regarding how to combine the Vuforia and ManoMotion libraries.

Conclusion and Outlook

The work on this project started with objects being scanned with the Vuforia Object Scanner. Then, Object Recognition was implemented in Unity, using the scanned objects as Data Sets. Information to display for recognized objects was saved in an entity created for this specific purpose, and an UI for displaying and navigating the provided information was developed. Gesture Recognition was integrated by itself at first, using a premade example as a base. Both Gesture and Object recognition were then combined in the same scene. The resulting problems with camera overlap and memory leaks were fixed, and a prototype allowing to scroll through information presented by Object Recognition with Gesture Recognition was finished. The following conclusions were reached as a result of the work done:

- Scanning Objects with Vuforia Object Scanner requires an Android device and a camera of at least some quality, but is otherwise easy
- Implementing Object Recognition by itself with Vuforia and Unity is doable and works well
- Displaying the information in a proper UI takes familiarity with Unity, but is possible without undue effort
- Implementing Gesture Recognition with ManoMotion by itself is doable and works, albeit only on Android
- Combining both approaches is not intuitive, requires some extra effort and leads to a result that works, but is neither stable nor efficient
- Controlling the scrollbar in the prototype with gestures is easy
- Gesture Recognition combined with Object Recognition has potential for user guidance, but combining Vuforia and ManoMotion for that goal is hard and error prone

Of the goals stated in the introduction, all of them were fulfilled, with some caveats:

- Scanned Objects are stored in a database with the help of the Vuforia target manager. The information to display for these scanned objects is held in a simple entity in the application
- Object Recognition with Vuforia using the camera of a device was implemented
- The information stored for a scanned object is retrieved from the data-storing entity when an object is recognized

- Said retrieved information is displayed in AR
- Should the information be too much to be displayed at once, the user can use a scrollbar to scroll through the text
- Scrolling up and down can be achieved by performing the gestures associated with either action. Said gesture recognition can be unstable and interfere with the object recognition.
- Additionally, the panel displaying the information can be scrolled directly, with keyboard keys and by using the buttons supplied by the UI.

Further developments in the field of user guidance with AR should definitely keep the approach of combining Object Recognition and Gesture Recognition in mind. However, an important first step would be to either develop a framework that offers both functionalities or find a way to properly integrate two frameworks, as the combination of two frameworks that were not specifically developed to be combined is unnecessarily troublesome and poses some challenges, as laid out in this paper.

References

Carmigniani, Julie et al. (2011): “Augmented reality technologies, systems and applications.” In: Multimedia Tools and Applications, 51 (2011), 1, p. 341–377. Available at: DOI: 10.1007/s11042-010-0660-6

Chen, Chia-Yen; Chang, Bao Rong; Huang, Po-Sen (2014): “Multimedia augmented reality information system for museum guidance.” In: Personal and Ubiquitous Computing, 18 (2014), 2, p. 315–322. Available at: DOI: 10.1007/s00779-013-0647-1

Grahn, Ivar (2017): The Vuforia SDK and Unity3D Game Engine : Evaluating Performance on Android Devices. Available at: URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-139436 (Accessed on: 20 April 2020).

ManoMotion (2020): ManoMotion – Let your hands talk to technology! Available at: URL: https://www.manomotion.com/ (Accessed on: 1 May 2020).

ManoMotion Documentation (2020): Documentation – ManoMotion. Available at: URL: https://www.manomotion.com/documentation/ (Accessed on: 1 May 2020).

ManoMotion Gestures (2020): Supported Gestures – ManoMotion. Available at: URL: https://www.manomotion.com/supported-gestures/ (Accessed on: 2 May 2020).

Purnomo, F A et al. (2018): “Implementation of Augmented Reality Technology in Sangiran Museum with Vuforia.” In: IOP Conference Series: Materials Science and Engineering, 333 (2018), p. 012103. Available at: DOI: 10.1088/1757-899X/333/1/012103

Syberfeldt, Anna et al. (2015): “Visual Assembling Guidance Using Augmented Reality.” In: Procedia Manufacturing, 1 (2015), p. 98–109. Available at: DOI: 10.1016/j.promfg.2015.09.068

Unity (2020): 2D/3D-Software für Animation, Rendering und Simulation | AR/VR-Spiele-Engine | Unity. Available at: URL: https://unity.com/de/products/core-platform (Accessed on: 20 April 2020).

Vuforia Engine Package Hosting for Unity (2020): Vuforia Engine Package Hosting for Unity. Available at: URL: https://library.vuforia.com/content/vuforia-library/en/articles/Solution/vuforia-engine-package-hosting-for-unity.html (Accessed on: 30 April 2020).

Vuforia Object Recognition (2020): Object Recognition. Available at: URL: https://library.vuforia.com/articles/Training/Object-Recognition (Accessed on: 20 April 2020).

Vuforia Object Scanner (2020): Vuforia Object Scanner. Available at: URL: https://library.vuforia.com/articles/Training/Vuforia-Object-Scanner-Users-Guide (Accessed on: 30 April 2020).

Vuforia Overview (2020): Overview. Available at: URL: https://library.vuforia.com/getting-started/overview.html (Accessed on: 20 April 2020).

Vuforia Unity Documentation (2020): Getting Started with Vuforia Engine in Unity. Available at: URL: https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html (Accessed on: 1 May 2020).