HoloLens Project: Building the Framework

by

IERIMONTI, Dario

18502970

A thesis submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science (Honours)

in

Computer Science

at

Hong Kong Baptist University

December 2018

Declaration

I hereby declare that all the work done in this Final Year Project is of my independent effort. I also certify that I have never submitted the idea and product of this Final Year Project for academic or employment credits.

_____

IERIMONTI, Dario

Date: _____

Hong Kong Baptist University

Computer Science Department

We hereby recommend that the Final Year Project submitted by IERIMONTI, Dario entitled “HoloLens Project” be accepted in partial fulfillment of the requirements for the degree of Bachelor of Science (Honours) in Computer Science.

Supervisor: Dr. CHOY, Martin Man Ting	Observer: Prof. XU, Jianliang
Date:	Date:

ACKNOWLEDGEMENT

I would like to thank Dr. Alfredo Milani and the University of Perugia for the great opportunity they gave me. I would also like to thank Dr. H.W. Tam who is in charge of my exchange programme. Last but not least I would like to thank Dr. Choy, Martin Man Ting, who gave me the opportunity to work with modern tools and fulfill my creativity and Prof. Xu, Jianliang for being the observer in my project.

TABLE OF CONTENTS

HoloLens Project:
Building the Framework

by

IERIMONTI, Dario

Computer Science Department

Hong Kong Baptist University

ABSTRACT

Augmented Reality is the mix of digital objects with the real physical world. It has started as a fictional concept but now it is becoming a reality, thanks to the new technologies, such as powerful smartphones and the new AR glasses, like **Microsoft HoloLens**. These glasses are very powerful standalone devices that make use of room perception, hand gestures and holograms to give the best AR experience. The problem is that the Development Kits are still in beta version, and they have bad documentation. The first part of this project's aim is to create an easy and fast development framework for **Microsoft HoloLens** and **Unity3D** that is suitable for general purpose applications, from gaming to productivity.

Introduction

Working with Microsoft Hololens

Microsoft Hololens is a powerful standalone device where you can create Mixed Reality Apps, that combines the real world with the virtual holographic world.

There are 5 core building blocks for HoloLens applications:

1. **Gaze**

   This is the simplest form of input in Microsoft HoloLens. It tells what object the user wants to interact with. It uses the position and orientation of the user's head to determine their gaze vector. It can be visualized as a ray that starts from the user's eyes and intersect with holograms and the spatial mapping mesh.

2. **Gestures**

   With gestures the user can perform actions and interact with holograms. It does not provide the precise hand position. It can be visualized as a mouse with you one button, where you can press, hold and release.

3. **Voice**

   Actions can be performed via voice commands as well. It gives a bigger range of possible actions than gestures, because it is possible to register as many

keywords as wanted. It is a good practice not to use reserved keywords like “Hey Cortana” or “Select”.

4. **Spatial Mapping**

   Spatial mapping is used to build a mesh of the area surrounding the user. That mesh can be used to place holograms in real world surface.

5. **Spatial Sound**

   In VR/AR applications, 3D sound is extremely important to gain user’s attention and make him feel immersed. If the application make good use of 3D sound, the user will be engaged and will feel reality in the virtual world.

Microsoft HoloLens and Unity3D

Using Unity3D is the fastest way to build a Microsoft HoloLens applications. With a basic understanding of the Editor, developers can easily build any kind of application for the HoloLens. But developers must keep in mind that this device has a mobile-class GPU, so applications need a lot of optimization to be light and still look good. Writing good and optimized shaders, using just a little memory for textures and maps and not overloading the CPU are some of the good practices to follow when developing. Microsoft offers a set of tools, the **MRTK** (Mixed Reality ToolKit) , available on GitHub, that help developers build applications. It contains advanced tools like Spatial Mapping and Spatial Understanding, and examples on how to use them.

System Overview

Purpose

Even though Microsoft MRTK is a powerful toolkit, sometimes it can be hard for most developers to get things working. There are still many problems with the HoloLens. One of them is that the ToolKit and the HoloLens itself are still in beta version, so they contains a lot of errors and sometimes they crash. There is a lack of documentation too. Most of the time developers will find themselves analyzing examples and other people's code to understand how to use the toolkits. This is very slow and frustrating, and probably developers will misuse the toolkits, ending up with errors that are hard to debug and unexpected behaviours. Another one is that the device is not so diffused, so the developing community is still little. There is a lack of forums and discussions online, so developers that works with HoloLens at the moment usually have to figure out things on their own. For these reasons, the aim of the first part of this project is to create an easy to use framework that all developers, even beginners, can use to build simple HoloLens applications fast.

Utility Classes

Some classes are useful when developers wants to build a solid, easily maintainable, application. An example of this is the implementation of the Singleton programming pattern. Singleton are widely used, as they come in handy for managers, helpers, or any kind of single instance object that doesn't have to be static.

Singleton Monobehaviour

A very useful class to implement the Singleton pattern in Unity3D's `Monobehaviour` objects. A class that inherits `SingletonMonobehaviour` will have a property called Instance, that is of the same type of the class, and contains a reference to the instance of that class. It also give the option to maintain the object when changing scene (`DontDestroyOnLoad`).

```
using UnityEngine;

namespace Thesis
{
    public class SingletonMonobehaviour<T> : MonoBehaviour
    where T : SingletonMonobehaviour<T>
    {
        private static T instance = default(T);
        public static T Instance { get { return instance; } }

        public bool dontDestroyOnLoad = false;

        private void Awake()
        {
            instance = this as T;
            if (dontDestroyOnLoad) DontDestroyOnLoad(instance.gameObject);
        }
    }
}
```

HoloInputController

Overview

The `HoloInputController` is a simple tool that developers can use to manage **GGV** (Gaze, Gesture, Voice) easily in their application. The idea behind it it's simple: this class exposes events of different of different nature, and developers can register and deregister actions to these events from anywhere in the code (this class inherits `SingletonMonobehaviour` so it can be accessed via `HoloInputController.Instance`).

Events

The Events that this class exposes are:

- `public event System.Action<InteractionSourcePressedEventArgs> InteractionSourcePressed;`

  Called when the user perform the press action.

- `public event System.Action<InteractionSourceReleasedEventArgs> InteractionSourceReleased;`

  Called when the user perform the release action.

- `public event System.Action<InteractionSourceDetectedEventArgs> InteractionSourceDetected;`

  Called when the the application detects the hand.

- `public event System.Action<InteractionSourceLostEventArgs> InteractionSourceLost;`

  Called when the the application loses the hand.

- **public event** System.Action<InteractionSourceUpdatedEventArgs> InteractionSourceUpdated;

  Called when the the application updates the hand state.

- **public event** System.Action<TargetAcquiredArgs> OnTargetAcquired;

  Called when the the users looks at an Interactable object.

- **public event** System.Action<TargetLostArgs> OnTargetLost;

  Called when the the users looks away from an Interactable object.

- **public event** System.Action<NoTargetArgs> OnNoTarget;

  Called when the the gaze is updated with no target.

- **public event** System.Action<RaycastHit> OnHit;

  Called when gaze intersect a physical object. It gives information about the hit.

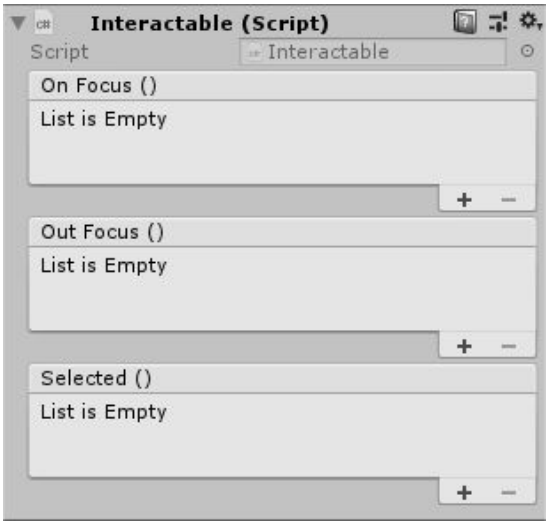
Gaze

Gaze is handled using Unity3D's `Physics.Raycast.` It draws a ray from the users head. This ray follows the position and orientation of the user's head. If this ray hits an object of type `Interactable,` then it acquires it as the new target. If it has a target, and loses it, then the target will be set to `null`.

```
void UpdateGaze()
{
     RaycastHit hitInfo;
     if (Physics.Raycast(Camera.main.transform.position,
         Camera.main.transform.forward, out hitInfo, 20.0f,
         Physics.DefaultRaycastLayers))
     {
          var o = hitInfo.collider.gameObject.GetComponent<Interactable>();
          if (o != null)
          {
               if (current != o)
               {
                   current = o;
                   if (current.onFocus != null) current.onFocus.Invoke();
                   if (OnTargetAcquired != null)
                        OnTargetAcquired(new TargetAcquiredArgs()
                        {
                             target = current.gameObject, hitInfo = hitInfo
                        });
                    }
                }
                if (OnHit != null) OnHit(hitInfo);
            }
            else if (current)
            {
                if (current.outFocus != null) current.outFocus.Invoke();
                if (OnTargetLost != null)
                     OnTargetLost(new TargetLostArgs()
                     {
                          target = current.gameObject
                     });
                current = null;
            }
            else
            {
                if (OnNoTarget != null) OnNoTarget(new NoTargetArgs() { });
            }
        }
 }
```

Interactable

An `Interactable` is a simple object that the users can interact with. It uses 3 `UnityEvents,` events that can be serialized in the inspector and make it easy for developer to register and deregister actions for that particular object.



The 3 events are `OnFocus()`, called when this `Interactable` in acquired as new target, `OutFocus()`, called when this `Interactable` is no longer the target, and `Selected()`, called when the user perform one of the interact actions (gestures or voice).

```
namespace Thesis
{
    public class Interactable : MonoBehaviour
    {
        public UnityEvent onFocus;
        public UnityEvent outFocus;
        public UnityEvent selected;
    }
}
```

Keyword Recognizer

The `HoloInputController` stores a `Dictionary<string, System.Action>` of the registered keywords with the corresponding action. It uses this dictionary to initialize the `KeywordRecognizer.`

```
private KeywordRecognizer keywordRecognizer;

private Dictionary<string, System.Action> keywords =
        new Dictionary<string, System.Action>();
```

Developers can use the public method `AddKeyword(string keyword, System.Action action)` or the corresponding `AddKeywords(Dictionary<string, System.Action> kws)` to add a single keyword or multiple keywords at once. The keyword “Select” is reserved for the system, as it performs the action `SelectInteractable()`, and can’t be overwritten.

Cursor

With the `HoloInputController` it is easy to implement features like a cursor. A cursor is an object that indicates the current gaze vector. It allows the user to understand the exact point that they are looking at. It is a fundamental part in almost every HoloLens application, so it is recommended that it should be always

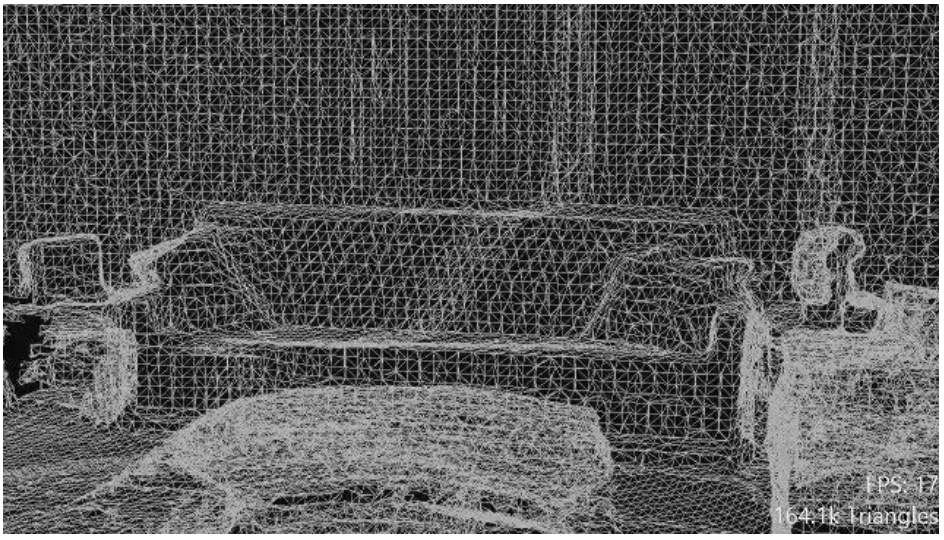
present (in applications that use Gaze for input). Without it the user will feel lost. It should provide a continuous feedback to the user about what they a looking at. To implement a cursor with `HoloInputController` the developer can register actions to the input controller events to handle the various states of the cursor. The cursor is placed at the end of the gaze vector, but when the gaze vector intersects a physical object, the cursor will be place on the surface of that object.

```
namespace Thesis
{
    public class Cursor : SingletonMonobehaviour<Cursor>
    {
        void Start()
        {
            HoloInputController.Instance.OnHit += Instance_OnHit;
            HoloInputController.Instance.OnTargetLost += Instance_OnTargetLost;
            HoloInputController.Instance.OnNoTarget += Instance_OnNoTarget;
        }

        private void Instance_OnTargetLost(TargetLostArgs obj)
        {
            transform.position = Camera.main.transform.position +
Camera.main.transform.forward * 20.0f;
        }

        private void Instance_OnNoTarget(NoTargetArgs obj)
        {
            transform.position = Camera.main.transform.position +
Camera.main.transform.forward * 20.0f;
        }

        private void Instance_OnHit(RaycastHit hit)
        {
            transform.position = hit.point;
            transform.forward = hit.normal;
        }

        void OnDestroy()
        {
            HoloInputController.Instance.OnHit -= Instance_OnHit;
            HoloInputController.Instance.OnTargetLost -= Instance_OnTargetLost;
            HoloInputController.Instance.OnNoTarget -= Instance_OnNoTarget;
        }
    }
}
```

Room Setup

Spatial Mapping

Spatial Mapping provides a mesh that represent the area surrounding the user. The mesh approximates the real world around the user. Developers can use informations from that mesh to build mixed reality applications that looks real. Mixing holograms with the real world make them look real in the eyes of the user, as they will not be seen as holograms anymore, but as part of out world.



The most common use cases for Spatial Mapping are:

- **Placement**

  Developers can place object on real world surfaces, like walls, tables, floors, and so on.

- **Occlusion**

  When holograms are occluded by real world object, the overall experience will be better and more realistic.

- **Physics**

  If an object falls, it can bounce on the real world ground, making it look real.

- **Navigation**

  If a character in a game walks on the floor, jumps on tables and sit on real chair, the line between virtual and reality will get thinner.

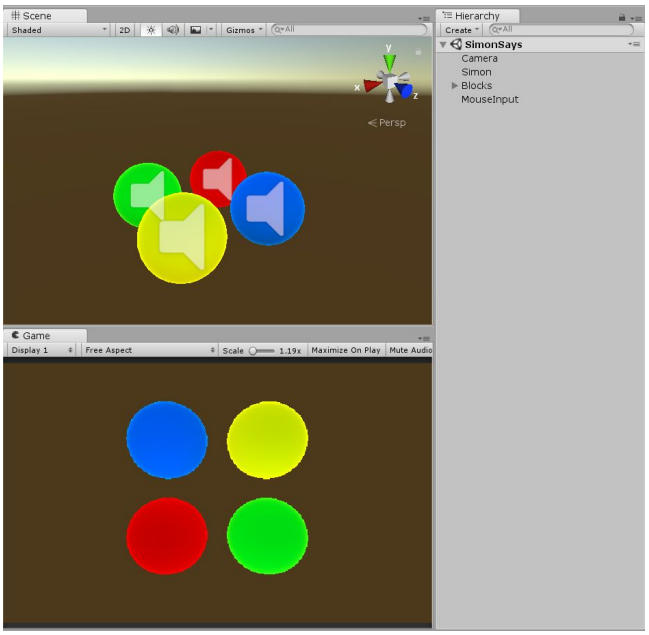
Spatial Understanding

For some applications, a better understanding of the user environment is required. When developers want to build applications that procedurally place objects in the real world, a higher level of awareness of the environment could come in handy.
A Spatial Understanding implementation is provided in Microsoft MRTK. But using it can be hard. This project tries to create an easier to use implementation that wraps the Microsoft one.

Usage

Making a game compatible with Hololens: Simon Says

This framework can be used to make many games compatible with Microsoft HoloLens. A basic example is turning a really simple game like Simon Says into a HoloLens application.



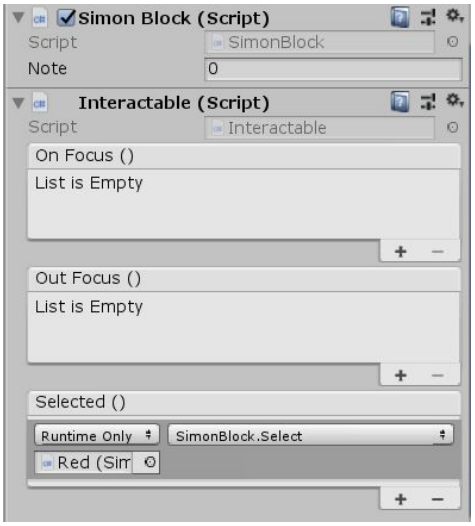
This is a basic implementation of the game Simon Says: the sphere will glow and emit sound in a sequence, and the player has to repeat the exact same sequence.

The game part has two Components: **Simon** and **SimonBlock**.

At the moment the Input is handled by the component MouseInput.

```
RaycastHit hitInfo = new RaycastHit();
bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
                           out hitInfo);
if (hit)
{
     var s = hitInfo.collider.gameObject.GetComponent<SimonBlock>();
     if (s) s.Select();
}
```

It performs a raycast using the mouse position, and calls the method `Select` on the SimonBlock, if it finds one. With the framework it is possible to replace this implementation, and without writing any code. The first step is to remove the **MouseInput** component. Then we add the **HoloInputController** prefab to the scene. At this point we can add the **Interactable** component to the SimonBlock. The UnityEvents of the Interactable component are serialize, so we will have a list of function to call when the SimonBlock will be selected by the cursor. Now we can assign the Select method to the Selected event of the Interactable Component.



Now the input is handled by the HoloInputController. Now all we have to do is to delete the existing **Camera** component and replace it with the **HoloCamera** prefab.

At this point the game is fully compatible with Microsoft Hololens.

Spatial Mapping

In this case the Framework is used to build a simple app to let the user throw a ball in the room. It makes use of Spatial Mapping to make the ball collide with the real world. The Button is an extension of the `Interactable` component. The result is a ball rolling on objects and surfaces of the room. This is a simple example on how you can easily and rapidly build prototypes with this Framework.

Keywords Recognition

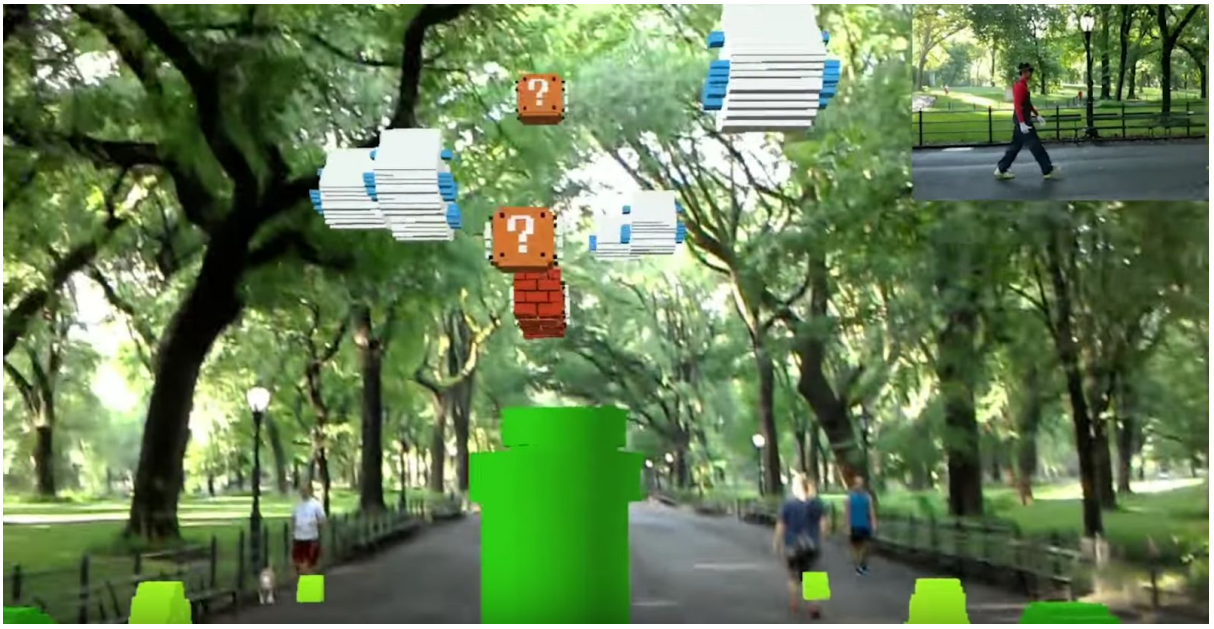
Using the keyword recognition system is very easy in the framework: Once you have the **HoloInputController** component into the scene you can access it from any script and register actions to its keyword recognizer system. This can be done using the method **AddKeyword** (or the equivalent **AddKeywords**, that is optimized to insert more keywords at once, due to the fact that every time **AddKeyword** is called, the keyword recognizer system has to be re-initialized, and calling **AddKeywords** will re-initialize it just once no matter how many keywords are inserted).

```
void Start ()
{
    Thesis.HoloInputController.Instance.AddKeywords(new Dictionary<string, System.Action>
    {
        { "red", () => { r.PlaySound();  } },
        { "blue", () => { g.PlaySound(); } },
        { "green", () => { b.PlaySound(); } },
    });
}
```

Game Ideas

Real Life Mario

A nice game for HoloLens could be a real life 3D platform. This could shape the world around the player to make him feel like a real life Super Mario.



Dodging Game

Another idea could be that the player has to avoid objects that comes towards him. It could be a dodgeball game, or anything similar. The player can use the world around him to hide and maybe use objects to defend himself.

This games can be really healthy and lead people to a more active videogame session than just a passive button clicking.