Windows Mixed Reality Experimentation

Index

Quick Links

Repository // https://github.com/lowpolyneko/WMR-EyeTracking
MRTK2 Docs // https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2
Unity Docs // https://docs.unity3d.com/2019.3/Documentation/Manual/index.html

General Overview

The Hololens 2 is a Windows Mixed Reality HMD (head-mounted display) produced by Microsoft for industrial and academic use. This project comprises a set of five demos to study the different interaction schemes of the Hololens 2. Schemes studied include hand-based interaction, head-based eye interaction, iris based eye recognition, and voice interaction. These demonstrations are intended to be used by volunteers and extensive data is collected by the software for statistical analysis.

Hardware and Software Requirements

Software

- Unity 2019.3.29f1 LTS
- Visual Studio 2022
- MRTK Feature Tool
- Git
- (Optional) Hololens 2 Emulator

Hardware

- VR-capable PC
- Hololens 2

Setup

Development Environment

To develop for the Hololens 2, one must be running a Windows 10 or 11 PC. Be sure that developer mode is enabled on your PC before continuing (in Settings > Update & Security > For Developers). To start, install Unity Hub and Visual Studio Community 2022. Microsoft recommends Unity 2019.3.29f1 LTS when working with MRTK-3, so this is the version we will install. Both Visual Studio and Unity require additional workloads. For Unity, ensure that these workflows are enabled:

- Universal Windows Platform (UWP) Build Support
- Windows Build Support (IL2CPP)

Visual Studio requires several workloads to properly cross-compile to the Hololens 2.

- .NET desktop development
- Desktop development with C++
- Universal Windows Platform (UWP) development
    - *Additionally, enable:*
    - Windows 10 SDK version 10.0.19041.0 or 10.0.18362.0, or Windows 11 SDK
    - USB Device Connectivity
    - C++ (v142) Universal Windows Platform tools
- Game development with Unity

Installing these should take a while, as it amounts to roughly 30 GiB.

Base Unity Project

Setting up a template Unity project is extremely simple. Ensure that you have the MRTK Feature Tool executable on your system. Firstly, create an empty project with the 3D Core template. Then, using the MRTK Feature Tool, select the newly created project and enable the following features:

- Mixed Reality Toolkit Foundation
- Mixed Reality OpenXR plugin

Finally, follow this instructional in the Microsoft Docs to properly configure the Unity project to build for the Hololens 2.

Implementation of Eye Tracking

With MRTK, eye tracking is implemented via the `GazeProvider` component within Unity's `InputSystem`. Gaze information is globally accessed via the `GazeTarget`, `GazeDirection`, and `GazeOrigin` member variables among others. By using the `GazeDirection` Vector3, it is possible to calculate the total distance traveled by repeatedly sampling over time and summing the absolute displacement of each vector component. Additionally, the angular speed of the both eyes can be calculated by dividing over the `deltaTime` between frames. Both values are particularly useful for studying eye movement and how it pertains to strain or other factors in VR/AR. The existence of `GazeTarget` provides a trivial method of retrieving the last selected GameObject, which proves useful for providing the user feedback upon gaze selection of an object.

Eye tracking can be switched between head-based or iris-based through a toggle in the main `MixedRealityToolkit` configuration profile (`MixedRealityToolkitConfigurationProfile -> Input -> Pointers -> Gaze Settings -> Use Eye Tracking Data`). Note that new configuration profiles need to be created in order for these settings to not apply globally.

Gesture Recognition

MRTK gesture recognition is also provided via globally accessible singleton classes. Both hand tracking and hand poses are accessible via the `HandJointService` and `HandPoseUtils` classes respectively. Within this project, gesture recognition has been abstracted via a static `GestureRecognition` class that is used by a `GestureHandler`. This handler provides easy in-editor definitions of callbacks for pinch and grabbing on to any `MonoBehaviour`. Invoking public methods within other GameObjects is extremely simple with this abstraction. The `EyeInteractionHandler` script was written explicitly for this purpose to manage selection of intractable cubes globally and maintain selection state. Included is an option for placement offset as a public editor variable.

Voice Recognition

Detection and keyword recognition is handled solely by MRTK's built in `SpeechInputHandler`. In the global MRTK `MixedRealityToolkitConfigurationProfile`, custom speech phrases may be added and intercepted by the handler script. In practice, this involves instantiating a global GameObject which handles speech recognition. For the voice interaction demos, a supplementary `EyeInteractionHandler` script is attached to the global GameObject which handles pinch events and tracks the currently selected interactable cube in the scene. Events in the `EyeInteractionHandler` are linked to the `SpeechInputHandler` via the `UnityEvent` addition menu in the editor. In practice, much of the

`EyeInteractionHandler` script is reused in both gesture and voice recognition. Additionally, a custom written `SpeechHandler`, was done for simple single keyword recognition. However, functionality is currently broken for this script.

Logging

For statistical analysis, logging is done by the aptly named `Logger` script. This collects all applicable information of the interactable cubes along with global eye positioning data and timestamps this information while calculating some additional values. These include the total distance traveled by the iris and angular speed. This data is saved into a `.csv` file with a unique name describing the name of the demo and time the experiment was started. The `.csv` allows for easy importing and visualization of the data through external software such as Excel. Information on the Logger's values are listed below.

Header Name	**Description**
Time	Timestamp when information was recorded.
Gaze Angular Speed	Calculated angular speed of the head's rotation (or iris if eye tracking was enabled). Calculated by taking the distance traveled and dividing by the deltatime between frames.
Distance Traveled	Distance moved (rotated) by the head or iris (if eye tracking enabled). Calculated by solving for the rotation displacement as a vector.
Total Movement	Cumulative sum of the absolute value of distance traveled over each frame. Represents the total displacement of movement of the head/iris.
Gaze Origin	Position in world space of the gaze raycast. In practice, this is the position of the camera in world space.
Gaze Target	Name of the current object selected in the scene. Null if nothing is selected.
Gaze Direction	Current rotational vector of the head (or iris if eye tracking is enabled).
Hit Position	Position in world space of the last collision of the gaze raycast.
Head Movement Dir	Head movement direction vector from the `GazeProvider`.

Head Velocity	Head velocity calculation from the `GazeProvider`.
Is object ontop of #	Boolean value representing if an object (typically another interactable cube) is on-top of the given numbered interactable cube.

Menu and Scene Selection

The scene selection menu at program launch was prototyped extremely quickly using MRTK buttons and a global script to attach to the `UnityEvent` interface provided by the buttons. The global script included simple code to load a scene by a given name and a keyword recognition event to attach to a `SpeechInputHandler` for quitting the program.

Miscellaneous

- For the GazePointer to always be enabled, the `PointerUtils.SetGazePointerBehavior` must be called and set to `PointerBehavior.AlwaysOn`.
- Selection and color change of the interacting cubes is handled by the `RemoteInteract` script. The trigger is exposed as a public boolean and can be attached to the `Logger` script to be included in the `.csv` analysis.
- Each interactable cube includes a `GrabbableCube` script to play a pickup/drop sound effect for user feedback. The sound effects are taken from the MRTK sdk assets.
- Included in `Scripts/Test Scripts` are several demo scripts that were used near the beginning of the project's development. A sample scene (named `SampleScene`) is also included as a reminiscence of early prototyping. These include two test models prefabbed `Cube` and `Benchy`. Potentially useful for reference.
- The custom MRTK configuration profiles are stored in `MixedRealityToolkit.Generated/CustomProfiles`. These are important for the proper functionality of each demo (eye vs gaze vs voice).