Exploring Multi-Hand Gestures as Shortcuts in VR

John Ho
Georgia Institute of Technology
jho49@gatech.edu

Yi Li
Georgia Institute of Technology
yli813@gatech.edu

Implementation

Description

As mentioned prior, our project is to explore whether or not multi-gestures could be a potential shortcut for user interfaces. We wanted to experiment on an example such as a 3D paint program similar to Tilt Brush [1] or APainter [2]. In these programs, toggling an option on from a 2D menu floating in front of one controller can prove to be quite a tedious task, especially when you need to position your other controller to act as a mouse pointer. Then, to turn the option off or go into a different mode (create, erase, etc.) you usually have to redo the process again. If multi-hand gestures were a shortcut similar to keyboard shortcuts, we can foresee them being handy shortcuts for such paint tools.

Environment

The environment we have developed on will be utilizing the Oculus Quests obtained from class. Due to the nature of the Oculus Hand Tracking SDK, we have setup a Unity 2019 3.8f1 project written in C# that is built on the Android module since the Oculus Quest environment is a mobile platform [3]. We have taken example scenes to recreate a hand tracking scene without the need of controllers. At the time of this writing, Oculus Quest does not support development of both controllers and hands at the same time. This will result in Unity crashes if done so [4].

Furthermore, we wanted to enhance our developer environment by allowing us to live test our hand tracking movements in real time. Prior to this, we were building an apk and pushing it onto our Oculus Quests everytime. This resulted in too long of a wait time between deployments. We first discovered a third party library called UnityQuestRemoteHandTracking [5], however upon further research we discovered that Oculus had recently implemented this functionality in v13 onwards of the Unity Oculus Integration Package [6]. We upgraded appropriately to the latest version, v15 of the time of this writing. In addition, we discovered both of us had compatible link cables without the need of purchasing a separate one. We used the link cables to emulate a traditional Oculus Rift S for Unity in order for us to enable this live debugging feature with hand tracking.

Scene Setup

We have created a Unity scene called example.scene under Assets/Scene in order to develop our prototype. The scene has all the proper Oculus Integration package setup. We have created custom state and button scripts that attach to the scene. In the scene we have placed a couple cube objects such that the user can interact with at the start.

Custom Gestures

We have decided that pinching gestures would be the easiest to implement and differentiate from. Our decision was influenced by the fact that we mainly wanted to create interesting UI interactions, not particularly creating new custom gestures from scratch. The pinching gestures could be generated by combining boolean values and defined as any individual finger touching the thumb. Thankfully, the SDK provided us with enough functionality to distinguish between which individual finger was touching the thumb at any time. This effectively allowed us eight different pinch gestures, four for each hand.

We decided that the best way to enter and exit different states was with two handed gestures. Pinching with specific fingers on both hands would be a very particular multi-gesture command. This would help us get rid of false positives of someone accidentally entering a state with only one handed gestures. We have decided for the purposes of usability that our multi-gestures would be the same pinching gestures on both hands. Both index fingers or pinky fingers touching the thumb would be examples. By using the exact pinch gesture on both hands, this would also help reduce confusion. An example of a double index finger pinch gesture is shown below [7]:

Pinch Modes

Our implementation includes four main pinch modes:

1. Double index finger pinch: Rotate/Scale
2. Double middle finger pinch: Create/Erase
3. Double ring finger pinch: Copy/Paste
4. Double pinky finger pinch: Color

The default mode you start off in is color mode. To enter in different modes you simply do the appropriate duo gesture. Note that there is no floating 2D or 3D menu present to enter these modes. The purpose of this project is to explore how we can purely use hand gestures as shortcuts in place. However, there is a UI text field that visibly tells you which mode you are currently in.

Note that every pinch mode will have controller-like rays pointing outwards from your hands. This is a common adaptation we implemented from Microsoft's MixedReality toolkit [8]. In place of controllers, these rays will help with object detection and selection in Unity.

Rotate/Scale Mode

In this mode you can interact with cubes placed in the scene to rotate or scale them. To do a rotate shortcut command, with either hand ray intersecting a cube, do an index finger pinch and hold it while slowly rotating your hand. When you let go of the index finger pinch, the cube stops rotating. The rotation is linearly based on the position of the hand.

To do a scale up shortcut command, with either hand ray intersecting a cube, do a middle finger pinch and hold it to scale up. To do a scale down shortcut command, do a ring finger pinch and hold it to scale down. When you let go of either finger pinches, the cube stops scaling respectively. The scale will be linearly based on how much you hold down the gestures.

It is important to note that you can perform and hold the pinching gesture with either hand. You can rotate or scale the cube object entirely using one hard, or alternatively, use your left hand ray to select the cube, then rotate or scale with your right hand doing the gesture. We made the decision to give the user both options as they might be left-handed, right-handed or ambidextrous.

Create/Erase Mode

In this mode you are granted the ability to create cubes with your left hand and erase with your right hand. To do a create cube command, use your left hand to do an index finger pinch gesture. A cube will be created in front of your hand direction. You can intersect cubes within each other.

To do an erase cube command, use your right hand to do a middle finger pinch gesture. If there is a cube intersecting your right hand ray, it will erase it.

We decided on giving two different functionalities in this mode to test the ease of both hands. We understand sometimes when we draw or create in these paint programs, we want to quickly erase if we've misplaced an object.

Copy/Paste Mode

In this mode you are granted the ability to copy an object that is intersecting with one hand ray, and paste the copied object with the opposite hand ray . To do a copy cube command, simply use one of your hands rays to intersect a cube, that cube will be the one that is copied.

To do a paste cube command, use your opposite hand to do an index finger pinch gesture. This will paste the copied cube in front of the opposite hand.

This mode was heavily inspired by the classic keyboard shortcut of copy and pasting. We recognized that current VR and AR paint programs did not support this functionality, so we sought an opportunity to implement it as it is a common shortcut that people use. This mode will allow the user to quickly select objects to copy and paste using both hands.

Color Mode

The default mode you start off in is color mode. In this mode, you are granted the ability to change the color of a cube that is intersecting with either hand ray. To do a color change command, do pinch gestures with different fingers on either hand. Each pinch gesture will correlate to a different color mapped below:

- Left index finger pinch: Red
- Left middle finger pinch: Green
- Left ring finger pinch: Blue
- Left pinky finger pinch: Yellow
- Right index finger pinch: Pink
- Right middle finger pinch: Teal

- Right ring finger pinch: Purple
- Right pinky finger pinch: Orange

We wanted to experiment with all eight different gestures in one mode and thought changing color was the most appropriate way to do so. This will allow the user to quickly change the color of the object with the eight main colors of the rainbow. This does not have the accuracy or diversity of a color wheel, but could be useful for applications with a limited range of color options or other toggles.

Video Demo

Here are links to our video demos below. You will find short demos for each of the modes described above. Then we have put together a combined gesture demo utilizing all the gestures in sequence.

https://drive.google.com/drive/folders/1Wh1YHZmp-W6IZQmeInrB7hMXyl0u7rsY?usp=sharing

Github Repo

Here is the link to our class github repo where the cost is all hosted from. It has been built with the proper Unity .gitignore file and all unwanted folders have been removed. You will also find an Android apk file in the builds folder that you can sideload onto the Oculus Quest:

https://github.com/3dui-class/milestone-2-implementation-ho-li

REFERENCES

[1] https://www.tiltbrush.com/

[2] https://aframe.io/a-painter/

[3] https://developer.oculus.com/documentation/unity/book-unity-gsg/

[4] https://forums.oculusvr.com/developer/discussion/86322/v13-editor-crashes-on-play-after-upgrading-oculus-integration

[5] https://github.com/handzlikchris/Unity.QuestRemoteHandTracking

[6] https://developer.oculus.com/blog/oculus-developer-release-notes-v13/?locale=en_US

[7] https://www.123rf.com/photo_42590975_zen-businessman-in-yoga-pose-on-white-background.html

[8] https://github.com/microsoft/MixedRealityToolkit-Unity/blob/mrtk_development/Documentation/Input/Pointers.md