3D-Paint

Authors

Yuval Shildan

Ran Mansoor

Shlomit Sibony

Supervisores

Boaz Sterenfeld

Yaron Honen

Table of content

Introduction

Project Idea

We have created Virtual Reality 3D Paint using Unity engine with C# scripting.
The equipment we used includes Manus-VR gloves and HTC Vive headset and trackers.

Since the Manus VR gloves is a new product we wanted to explore its capabilities and create an easy to use 3D-paint platform.

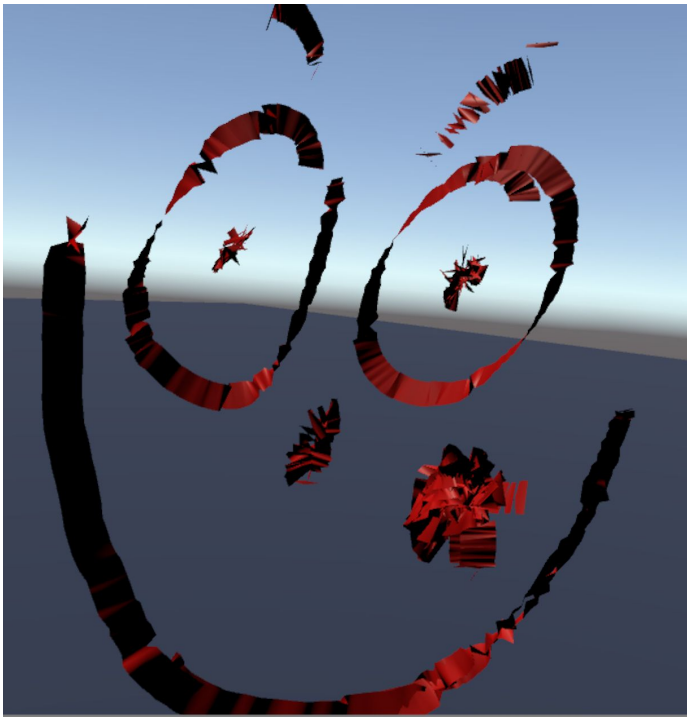
Project Setup

1. Connect Manus-VR gloves with the computer
2. Download Manus-VR test and calibration tools, calibrate your gloves - https://developer.manus-vr.com/
3. Connect the HTC set to the computer and pair the trackers using Vive-dashboard application
    a. Calibrate the headset if needed
4. Launch the 3D-Paint application

Application User Guide – Point & Paint

1. To paint - just point with your index finger and move your hand.
2. To stop painting – stop pointing
3. To grab a painting – make a fist and touch it
4. To release the grabbed painting – stop making a fist

You can use your both hands to either paint or grab simultaneously.



Technologies Description

Platforms

Unity is a cross-platform game engine. It is compatible with any Virtual Reality products, including the HTC-Vive and the Manus-VR gloves.

We used Visual Studio IDE to write C# code, which is Unity's primary API programing language.

HTC Vive is a virtual reality headset which uses "room scale" tracking technology, allowing the user to move in 3D space and use motion-tracked handheld controllers to interact with the environment.

We used the HTC Vive Trackers in order to track the user's hands and present it in the scene.

Manus VR is a virtual reality glove input device. it tracks the hand movement in real time and uses the captured data to faithfully reproduced the movement in virtual reality.
The gloves can be easily connected to unity and the HTC equipments

VRTK, stands for Virtual Reality Tool Kit, is a Unity plugin which is used to build virtual reality solutions. It supports diverse VR plugins including SteamVR.

Useful Links

Manus-VR developer - https://developer.manus-vr.com

Download SteamVR - https://store.steampowered.com/about/setup

VRTK - https://assetstore.unity.com/packages/tools/vrtk-virtual-reality-toolkit-vr-toolkit-64131

Implementation Description

Interaction with Manus-VR gloves

Since the gloves are a pretty new product, its documentation and open source Q&A can't be easily found. However, Manus-VR provides us with its SDK implementation written in C#, thus we dived into the given code and explored some specific features.

First, we went over the execution flow of how the real time data on the hands state is being collected. We discovered that Manus keeps the data in the following structure:

1. each finger is defined as the two joints that constituent it.
2. each joint gets a value from [0,1] which 0 means closed and 1 means open
3. all these data is stored in the object "manus_hand_t" in the array "raw.finger_sensor"
    a. 0, 1 refer the pinky
    b. 2, 3 refer the ring finger
    c. 4, 5 refer the middle finger
    d. 6, 7 refer the index
4. each hand has 4 states
    a. Fist
    b. Small
    c. Tiny
    d. Open
5. the state is calculated as an average of all its fingers close value

In the following sub chapters we will discuss about how we used the SDK for painting and grabbing objects.

Painting

As described before, Manus-VR SDK maintains real time data about all fingers joints.
We were searching the right combination of all fingers joints in order to start painting. In general we aimed it to be a kind of a pointing - neither too flat, since it will probably harden the user, nor too soft so that paintings couldn't be painted accidently.

After a lots of experiments we decided that the hand value when starting to paint should be the following:

1. the two joints values of the index should be smaller than 0.1, 0.2 respectively (out of 1)
2. all other fingers joints values should be greater than 0.3 (out of 1)

Those values can be changed in "CalculateIsPointing" function that is located in "Assets/ManusVR/Scripts/HandData.cs" file.

Grabbing objects

As described before, there are 4 possible values for the state of the hand - Fist, Small, Tiny, Open. by applying those values on our grabbing method we realized that the intuitive way (not too tough to perform but still not a common movement) would be the Fist value.

On each painting we added a collision event, which in case of a collision with the hand object would do the following:

1. if the hand state is Fist
    a. set the painting to be the hierarchical son of the hand object
    b. mark the painting as “grabbed”
2. return

This technique of setting the hand as the parent of the painting makes the movement of the object smooth and easy to maintain.

Releasing the painting is performed inside the update event. In each frame we do the following:

1. if the painting is marked as “grabbed”
    a. if the hand state is not Fist
        i. set the painting’s parent as null
        ii. mark the painting as “released”
2. return

These both procedures are easy to maintain and are very intuitive to the user.

Until now we answered the question when should we start and stop painting. In the next chapter we’ll discuss about how the painting is being build.

Building the Painting - Mesh

Unity provides us with several ways for implementing the painting feature.
We first tried to implement it using line renderer component, the result were surprisingly good; the painting looked well, we could control the color and width. However, the main problem with that technique was that no game object was created, thus we couldn’t move the painting around the scene.

Our second idea was making a game object in each frame of the scene, making a combination of all points and treat them like one painting. Unsurprisingly, that didn’t look very well. also, making so many point caused performance issues.

Finally, we have decided to implement the painting feature using Mesh.
According to Unity’s documentation, *a mesh consists of triangles arranged in 3D space to create the impression of a solid object. A triangle is defined by its three corner points or vertices*.
More practically, *In the Mesh class, the vertices are all stored in a single array and each triangle is specified using three integers that correspond to indices of the vertex array.*

We’ve created the Paint class with the following structure:

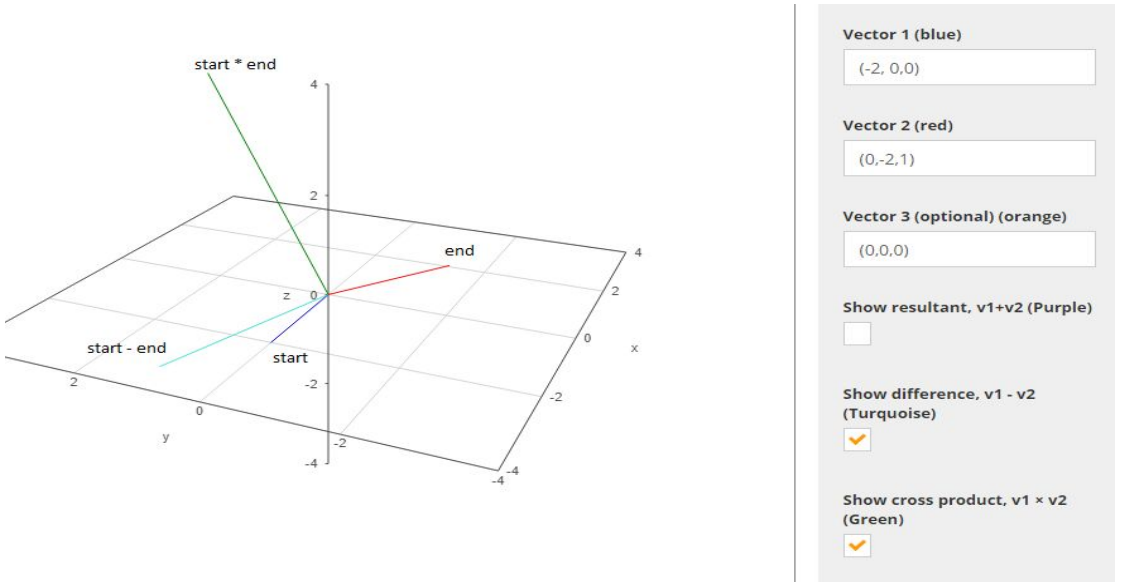
1. it saves the previous 3d vector that has been added to the mesh.
2. it has one public method AddPoint which gets a 3D point

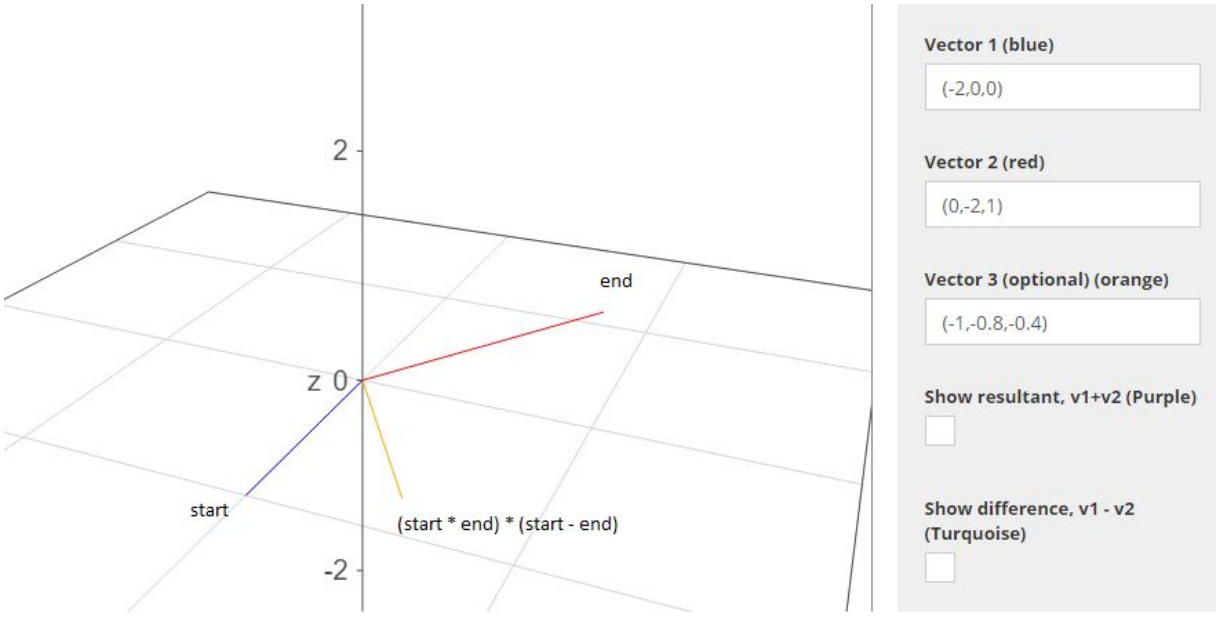
Note: In each call, the algorithm adds the previous point into the object's mesh, thus, at the first call of the algorithm it only stores the given point and no mesh is created.

In each frame we call AddPoint with the finger's position of the hand that is currently painting. The calculation inside this function is the following:

1. calculate the quad that created by the starting point, current point and a custom width
    a. let the starting point be s and the current point be e.
    b. let the normal s * e be n.
    c. let l be n * (s - e).
    d. the two new edges that calculated are:
        i. s
        ii. s + l * w, where w is the custom width.

Here is a scheme that explains the steps





Here the orange vector (in our case l) is the direction in which the second point should be placed

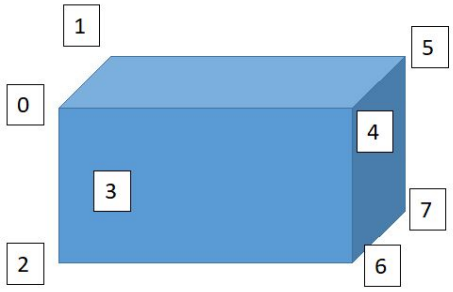
In practice, the two new points which the previous algorithm returns are:

1. s + (w/2)*l
2. s - (w/2)*l

We found that method more intuitive since the painting is being built symmetrically to the finger's position.

Looking deeper in our code, you'll find the building of the mesh respectively to Unity's Mesh class. We resize the vertices and triangle arrays to fit the new size of the mesh, and added the following:

1. vertices
    a. add four slots which are two replications of the new two points as described before each replication refers to the front/back side of the mesh respectively.
2. triangle
    a. let the vertices array be as the following scheme
    (0 - 3 are the previous quad, 4 - 7 are squad we just calculated)



    b. the triangles we added to the front-facing are
        i. 0, 2, 4
        ii. 2, 6, 4
    c. and respectively the back-facing triangles
        i. 5, 3, 1
        ii. 5, 7, 3

Though building a mesh is not easy to implement, it gives you as a developer a full control of the newly created object; starting from setting its size and color, up to alter its transform.

Proposed Next Steps

Ability to save painting and print/share them

Online painting with friends/co-workers

Additional colors of paintings

Bibliography

http://www.wolframalpha.com/widgets/view.jsp - Vectors Painting

https://docs.unity3d.com/Manual/AnatomyofaMesh.html

https://docs.unity3d.com/Manual/class-LineRenderer.html