AFFECTIVE COMPUTING AND INTERACTION IN VR

Corso Realtà Virtuale 2023/2024

susanna.brambilla@unimi.it

WITH UNITY V2022.3.5

GAZE AND FACE TRACKING

GAZE TRACKING

ONLY on the **Quest Pro**, Gaze tracking is available

It refers to the detection of **eye movement**, which is used to drive an eye transform on an avatar as the user looks around

Eye Tracking for Movement SDK for Unity: Unity | Oculus Developers

OVRGAZE

The OVREyeGaze component will update the GameObject’s position and orientation based on the real human’s eye movement

There are 3 types of tracking mode:

- **World Space** tracking mode will convert the eye pose from tracking space to world space
- **Head Space** tracking mode will convert the eye pose from tracking space to local space which is relative to the VR camera rig
- **Tracking Space** is raw pose information from VR tracking space.

FACE TRACKING

Creating a character with **face tracking** consists of

- creating the character to be represented with **blendshapes** that represent the facial expressions of your character
- adding the scripts to the character containing the blendshapes to read the API and map the expressions detected into the character blendshape

The Face Tracking API converts the facial movements detected by the headset sensors into activations of expression as weights between 0 and 1

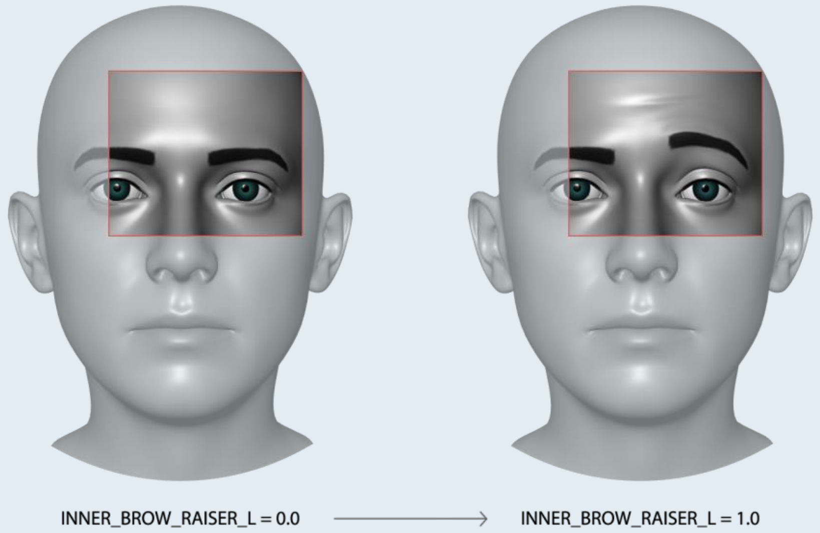
Up to 63 expressions detected by the sensors, based on the **Facial Action Coding System (FACS)**

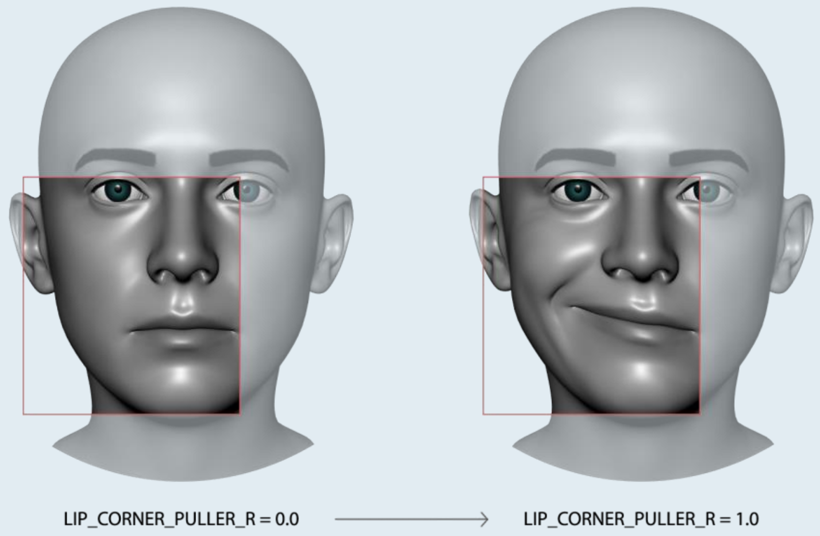
Face Tracking for Movement SDK for Unity: Unity | Oculus Developers

OVRFACEEXPRESSIONS

Some of the face-related scripts are distributed in the sample and not in Meta XR All-in-One SDK. As such, it is necessary to download the samples Oculus Samples GitHub repo to have access to these scripts.

The OVR Face Expressions component queries and updates face expression data every frame

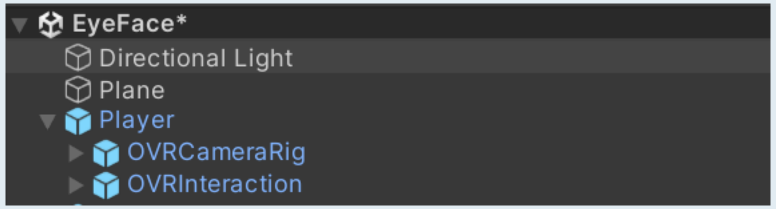




INITIALIZATION 1/2

1. Download the ‘GazeFace_demo’ project from github

You will find a EyeFace scene without camera and with a plane, drag the Player prefab you find in the EyeFace_demo folder and drop it in the Hierarchy



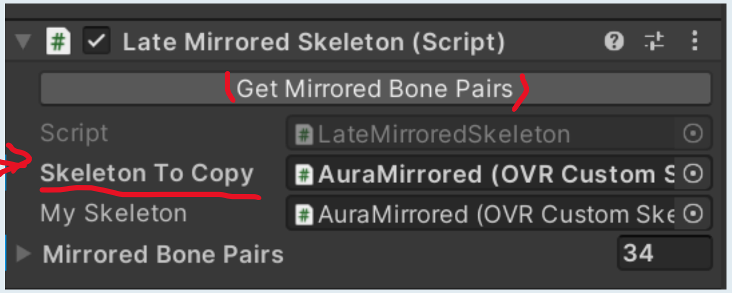
EyeFace*
Directional Light
Plane
Player
OVRCameraRig
OVRInteraction

In the FaceEye_demo folder folder you can find three other prefabs taken from Oculus Samples:

- AuraFirstPerson: the character with blendshapes
- AuraMirrored: the avatar which will mirror the character movements
- MirrorTransformation: to mirror the character, so that the user can see their facial emotions and gaze animate in real time

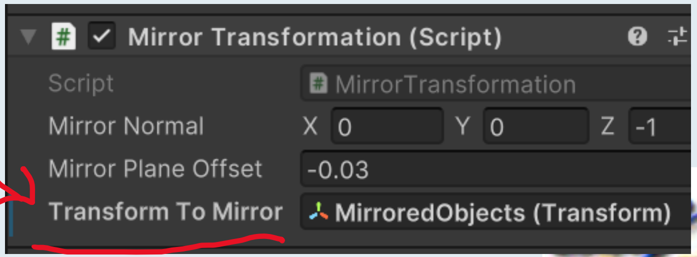
INITIALIZATION 2/2

2. Add the AuraFirstPerson prefab in the Hierarchy
3. Create a new Empty GO by clicking the right mouse button in the Hierarchy and call it MirroredObjects
   - Add under the MirroredObjects the AuraMirrored prefab
   - Drag and drop the AuraFirstPerson object in the Skeleton To Copy filed of Late Mirrored Skeleton script of AuraMirrored and click on Get Mirrored Bone Pairs button:



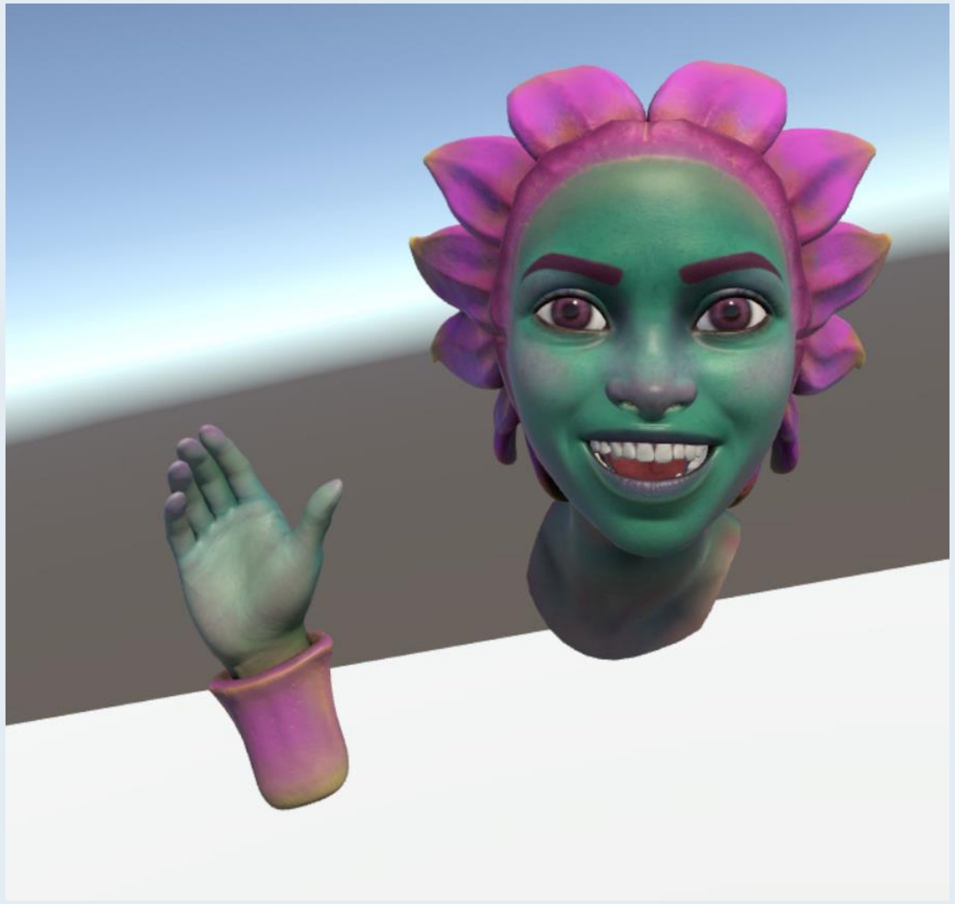
Script	LateMirroredSkeleton
Skeleton To Copy	AuraMirrored (OVR Custom S
My Skeleton	AuraMirrored (OVR Custom Ske
Mirrored Bone Pairs	34

4. Add the MirrorTransformation prefab to the Hierarchy
   - Drag and drop the MirroredObjects GO in the Transform To Mirror field of Mirror Transformation
   - If necessary, reset the position of AuraMirrored object



Script	MirrorTransformation
Mirror Normal	X 0 Y 0 Z -1
Mirror Plane Offset	-0.03
Transform To Mirror	MirroredObjects (Transform)

PLAY



Press play… you will see that AuraMirrored will copy your facial expressions and gaze movements

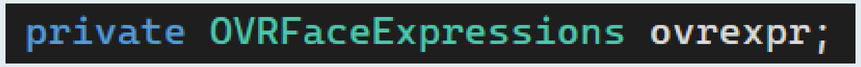
EYEFACE_TRACKER SCRIPT

Open the EyeFace_tracker script

This script is in charge of getting facial and gaze movements

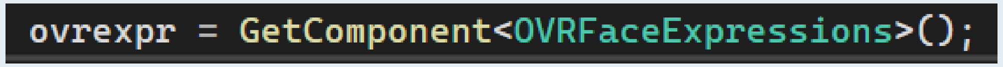
The script need to be attached to an object with a OVRFaceExpressions script on it, in order to acquire all the data

We need a variable to store the information from OVRFaceExpressions:



```
private OVRFaceExpressions ovrexpr;
```

and get it in Start():



```
ovrexpr = GetComponent<OVRFaceExpressions>();
```

COLLECT EYE DATA

In order to get Eye-related data, we need a reference to the eyes:

```
public GameObject leftEye, rightEye;
```

We can collect eye name and position using:

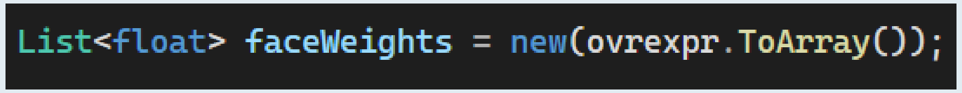
```
private void CollectEyeTransform(GameObject eye)
{
    OVREyeGaze gaze = eye.GetComponent<OVREyeGaze>();
    string label = "Eye" + gaze.Eye.ToString();
    Vector3 eyePos = eye.transform.position;

}
```

Call this function in FixedUpdate() for both eyes

```
CollectEyeTransform(rightEye);
CollectEyeTransform(leftEye);
```

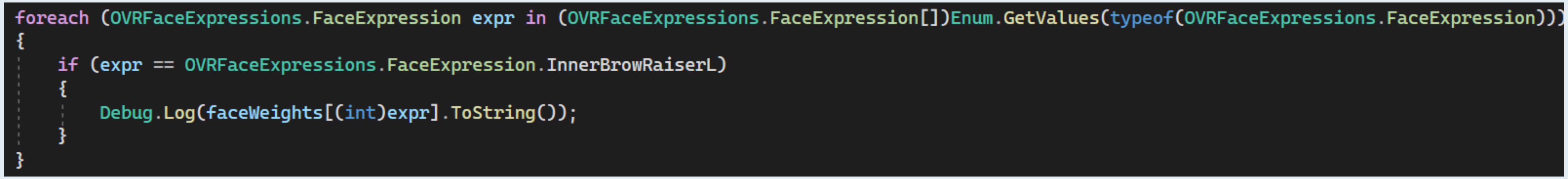
COLLECT FACIAL DATA

In order to get face-related data we first need a list of float to store all the blendshapes weights:



```
List<float> faceWeights = new(ovrexpr.ToArray());
```

If we want to find a single blendshape, e.g. the inner brow raiser left:



```
foreach (OVRFaceExpressions.FaceExpression expr in (OVRFaceExpressions.FaceExpression[])Enum.GetValues(typeof(OVRFaceExpressions.FaceExpression)))
{
    if (expr == OVRFaceExpressions.FaceExpression.InnerBrowRaiserL)
    {
        Debug.Log(faceWeights[(int)expr].ToString());
    }
}
```

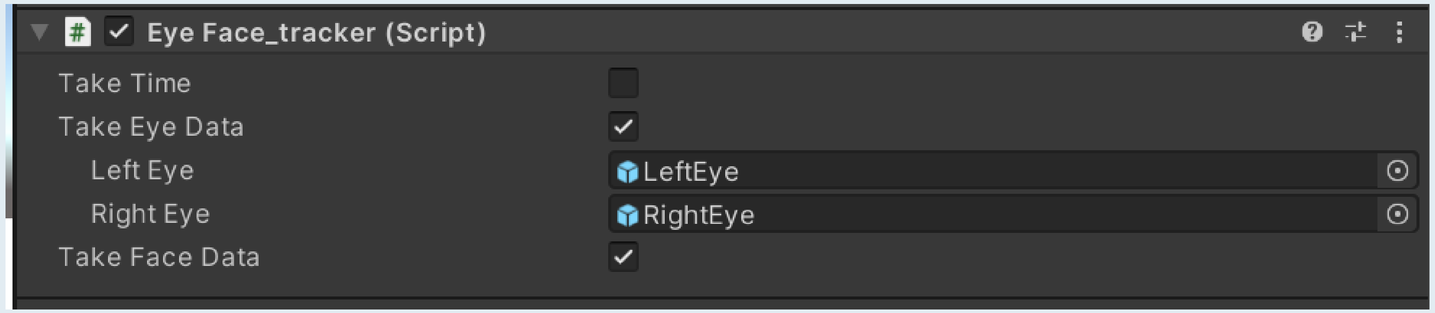
VISUALIZE DATA 1/3

1. In the editor, create an empty GO and call it DataTracker
2. Attach the script ‘EyeFace_tracker’ to the DataTracker object
3. Check the Take Eye Data and Take Face Data fields
4. Create two Empy GOs as children of DataTracker and call them Left/RightEye

VISUALIZE DATA 2/3

1. Attach the OVR Eye Gaze script to the Left/RightEye Gos
    1. Select the Eye field for each eye
    2. Change the Tracking Mode to World Space

2. Add the Left/RightEye to Left Eye and Right Eye fields of the EyeFace_tracker script

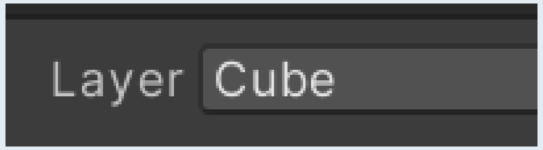


Eye Face_tracker (Script)

Take Time	☐
Take Eye Data	✓
Left Eye	LeftEye
Right Eye	RightEye
Take Face Data	✓

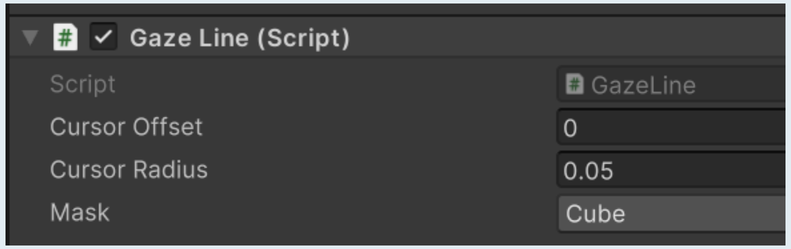
VISUALIZE DATA 3/3

5. Create some Cubes and, in top right part of the Inspector, assign them a new Layer called ‘Cube’



Layer Cube

6. Open the GizmoModule folder and drag and drop the GimoModule prefab in the Hierachy
7. Attack the GazeLine script you can find in the GizmoModule folder on the Left/RightEye GOs
   - Change Cursor Radius to 0.05
   - Change Mask to Cube

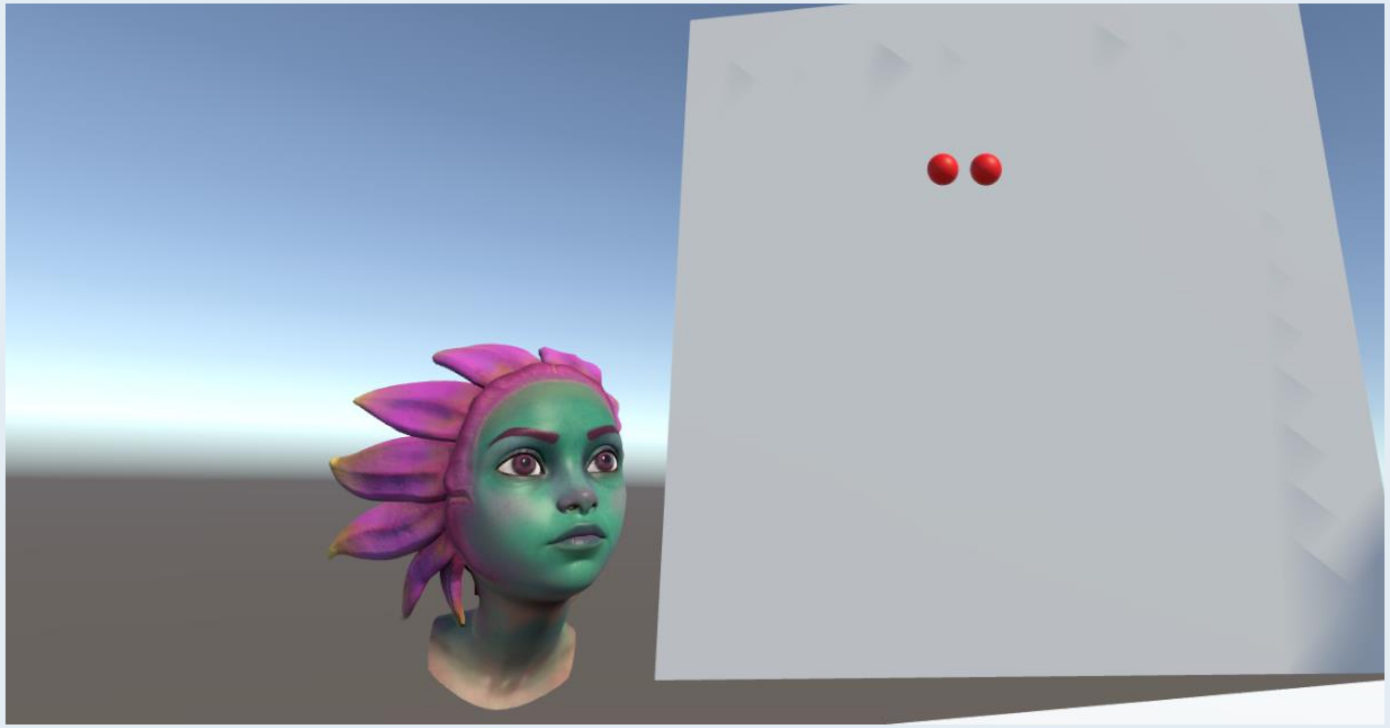


Gaze Line (Script)

Script	GazeLine
Cursor Offset	0
Cursor Radius	0.05
Mask	Cube

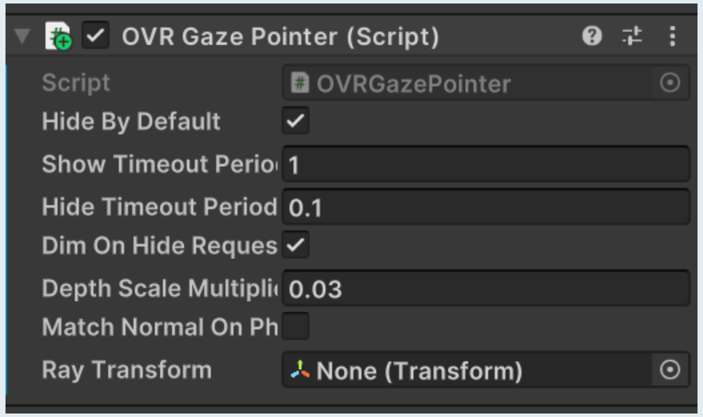
RESULT

…press Play and see the result!



GAZE POINTER

If you want to control element in the scene, e.g., UI elements, you should use the OVR Gaze Pointer script:



OVR Gaze Pointer (Script)

Script	OVRGazePointer
Hide By Default	✓
Show Timeout Perio	1
Hide Timeout Period	0.1
Dim On Hide Reques	✓
Depth Scale Multipli	0.03
Match Normal On Ph	
Ray Transform	None (Transform)

Assign the Center Eye Anchor to the Ray Transform field