TP 5 - Desktop VR: Head tracking and asymmetric frustum with OpenCVSharp and Unity

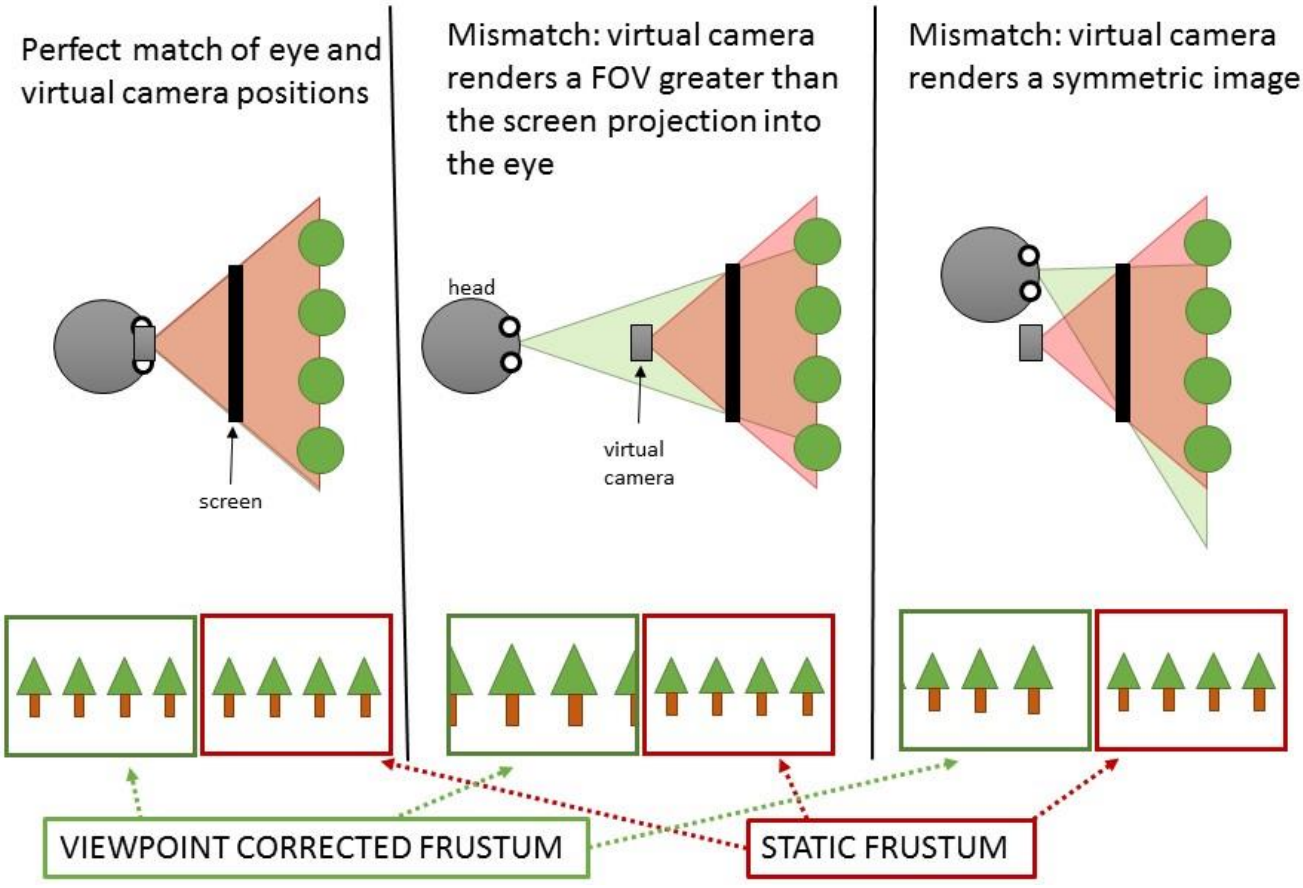
This tutorial has been printed from http://henriquedebarba.com/index.php/2016/04/03/21/, use that website if possible as copy-pasting code might be difficult depending on your PDF reader.

Introduction

This TP requires Windows and UNITY 5.

In this TP we will implement head tracking using OpenCVSharp (a C# wrapper for openCV), as well as projection correction that compensates for the dynamic change of the viewpoint relative to the screen. Start by watching this video: https://youtu.be/Jd3-eiid-Uw

We don't always notice, but what is rendered in a screen when watching movies or playing a game is always wrong to some degree. Most often, a big field of view (FOV) is compressed into a smaller angular size of our retina, causing rendered objects to seem smaller than their real counterparts (the opposite happens in a movie theater). Below we illustrate this mismatch:



If we look at the projected images in the center, the rendered object becomes bigger in the screen as the user gets further from it. This may sound counter-intuitive, but this happens because the screen now falls under a smaller portion of the FOV of the user. In practical terms, the objects are projected into a smaller portion of the user's FOV, even though they are represented in a bigger portion of the screen. In the image to the right we see a distinct mismatch. In this case the viewpoint of the user is not centered relative to the screen, and the lines connecting the viewpoint with the screen have different length. Using a regular camera will render portions of the scene that should be occluded, and omit parts of the scene that should be visible.

To correct for these mismatches we should modify the camera frustum as follows:

- Dynamically change the FOV so that the virtual camera and viewer head are matching.
- Dynamically change the shape of the frustum (deforming the usual pyramidal shape into an asymmetric shape).

The limitations of such corrections are:

- A normal screen can only support one user with corrected viewport, the image will look incorrect to all other viewers.
- Prior knowledge of the screen size and camera position relative to the center of the screen are required.
- We need to track the users' head position.

Material

We will first download and load the packages containing the assets for this tutorial (textures/materials/scripts/scene)

You may find the TP5_HeadTracking.unitypackage at any of the following links:
https://drive.google.com/file/d/0BwA8OpKLTKR2NzZBc0hnVFVQUUU/view?usp=sharing
https://www.dropbox.com/s/dkqe60ta0vfnxzw/TP5_HeadTracking.unitypackage?dl=0

Now open Unity 5, create a new project named VR_TP5 and load the TP package (Assets -> Import Package -> Custom Package …)

Make sure that all les are marked.

You will also need a calibration for the features we are going to track in this tutorial, download the le haarcascade_frontalface_alt.xml from:
https://drive.google.com/file/d/0BwA8OpKLTKR2RGJqY3BwTEJWVHM/view?usp=sharing
and place it in the project folder (i.e., in the same folder where the Assets, Library … folders are located).

Head Tracking

In this section we will learn how to use OpenCVSharp (OpenCV stands for Open source Computer Vision, Sharp stands for a C# wrapper of OpenCV) to track the head position of a user.

It is subdivided in the following parts:

1. Preparation.

2. Gather webcam feed, render it as a textured plane, and convert to the openCV format cvMat.
3. Create a Unity camera and t the textured plane from 2 on its Field of View (FOV).
4. Track the face of the user using the Haar Cascade Classifier openCV algorithm.
5. Transform the coordinates system from 4 in order to overlay the textured plane with the head position.
6. Transform the coordinates system from 4 in order to present the head position in 3D.
7. Improve the tracking reliability of item 6 with a smoothing algorithm.

1 Preparation

Open the scene HeadTracking (by double-clicking it in the Project tab), and start by creating a new C# le named HeadTracking.cs (Project tab -> Create -> C# Script). Now create a new camera by selecting GameObject -> Camera, set the new camera position to 0 in the Inspector, then rename the camera to unityWebcam (it will serve as a virtual representation of the actual webcam)

Finally, drag and drop the newly created HeadTracking.cs le to the unityWebcam game object.

2 Gather webcam feed, render it as a textured plane, and convert to the openCV format cvMat

Open the le HeadTracking.cs and replace its content with the code below:

```
using UnityEngine; using
System.Collections; using
OpenCvSharp;
// Parallel is used to speed up the for loop when converting a 2D texture to CvMat using Uk.Org.Adcock.Parallel;

public class HeadTracking : MonoBehaviour {


   // webcam video
   public Webcam wc = new Webcam();      // opencv
video
   public OCVCam ocv = new OCVCam ();

   // Initialization
   void Start () {
      // init webcam capture and render to plane
      if (wc.InitWebcam ()) {
         // init openCV image converter
         ocv.InitOCVCam (wc);


      } else {
         Debug.LogError ("[WebcamOpenCV.cs] no camera device has been found! " +                "Certify that a camera is
connected and its drivers " +
                  "are working.");
         return;
      }
   }

   // Update is called once per frame
   void Update () {
      ocv.UpdateOCVMat ();
   }
}
```

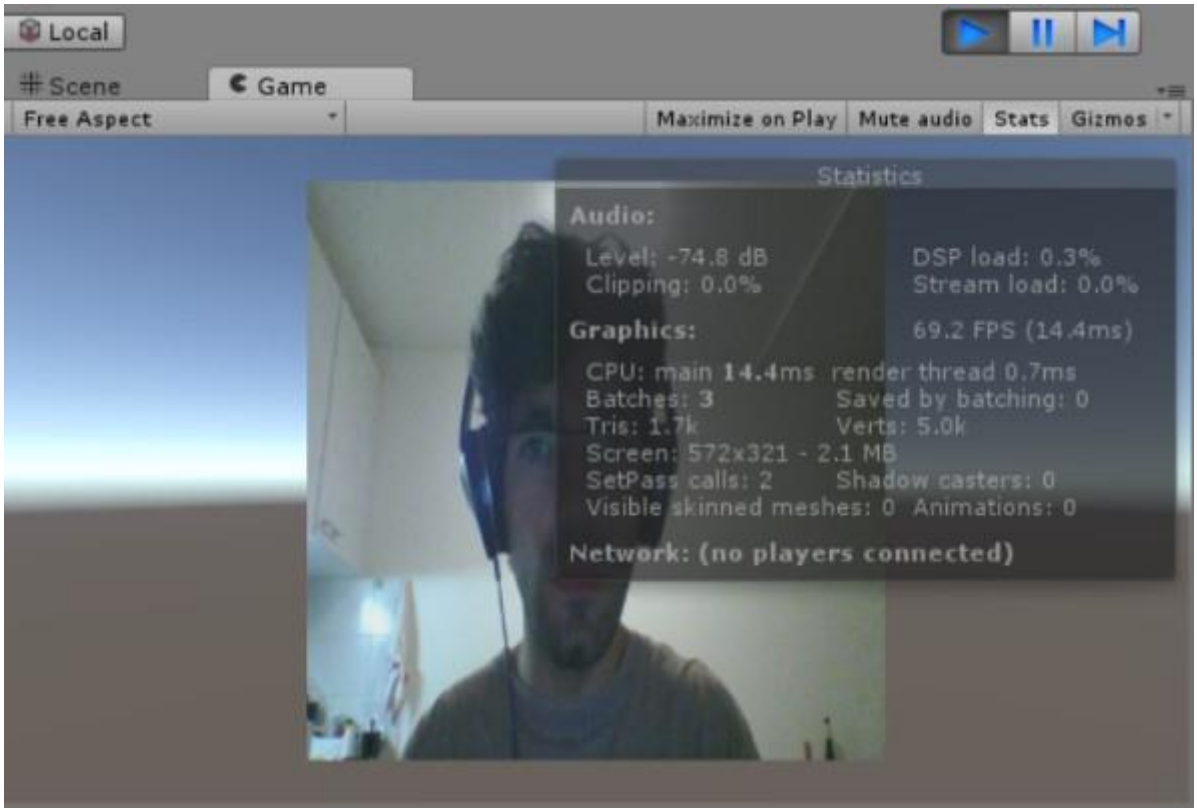
The class Webcam encapsulates what is required for gathering the webcam feed and rendering it to a plane in the Unity scene. The class OCVCam encapsulates the code for converting a Unity 2D texture into a cvMat, which is the image format used by openCV. You may optionally open the les where these classes are declared in order to nd details on how this is done.

Save the HeadTracking.cs le and go back to the editor. Select the unityCamera in the hierarchy tab. If you look in the inspector, you will notice that there are a lot of parameters that you can set for the component HeadTracking.cs. These are variables from the Webcam and OCVCam classes exposed to our HeadTracking class:

- Wc (Webcam)
  - Cam Frame Rate: frame rate request to the webcam (-1 for no request)
  - Cam Width: image width request to the webcam (-1 for no request)
  - Cam Height: image height request to the webcam (-1 for no request)
  - Cam FOV: you must set the vertical FOV of the webcam. Most webcams have a 40~45 degrees FOV
  - Flip Up Down Axis: flip image vertically?
  - Flip Left Right Axis: flip image horizontally?
- Ocv (OCVCam)
  - Reduction Factor: factor to divide the webcam feed resolution before converting to the CvMat format (this will signi cantly improve the performance of the program, as this conversion is the main bottleneck of OpenCVSharp)
  - Parallel Conversion: use multiple threads to convert the webcam feed to CvMat format (unless it is causing problems, keep it checked as it signi cantly increases the execution speed)

Ideally you should set Cam Frame Rate to 30, and Cam Width and Cam Height to the smaller supported widescreen resolution (in my case it was 424 x 240). Mind that these parameters are only requests to the camera, if the camera isn't compatible with the settings or doesn't allow these requests the standard camera settings will be used.

In the tab Game open the Stats display. Now hit the PLAY button, there is a new GameObject in the scene named "imgPlane", which is rendering the camera feed:



The image should be rendered as a mirror (e.g., if you move to the left, your image in the object should also move to the left). If this is not the case, use the ip parameters to correct the image.

In the Stats display, check the frame rate (FPS) that this scene is running. If it is bellow 40 FPS, reduce the Cam Width and Cam Height, and increase the Reduction Factor until your frame rate is in the desired range.

3 Create a Unity camera and t the textured plane from 2 on its Field of View (FOV)

Currently we are presenting the webcam feed at an incorrect aspect ratio (1:1), and the imgPlane doesn't span all the unityCamera FOV. The following modi cations to the code address these issues: Include the following variable declarations to the HeadTracking class

```
1    // unity cam to match our physical webcam
2    public Camera camUnity;
3    // distance between camUnity and webcam imgPlane
4    float wcImgPlaneDist = 1.0f;
```

Append the following additional code to the Start function

```
1   void Start () {
2     ...
3
4     // make sure that a camera has been provided,
5     // or that this GameObject has a camera component
6     if (camUnity == null) {
7       Camera thisCam = this.GetComponent<Camera>();
8       if (thisCam != null)
9         camUnity = thisCam;
10      else
11        Debug.LogError ("[WebcamOpenCV.cs] there is no camera component " +                    "in the GameObject
12  where this script is attached");
13    }
14
15
16    // webcam and camUnity must have matching FOV
17    camUnity.fieldOfView = wc.camFOV;
18    FitPlaneIntoFOV (wc.imgPlane);    }
```

And add the following function to the end of the HeadTracking class

```
1   void FitPlaneIntoFOV (Transform wcImgPlane){
2     wcImgPlane.parent = camUnity.transform;
3
4     // set plane position and orientation facing camUnity
5     wcImgPlane.rotation = Quaternion.LookRotation(camUnity.transform.forward,
6                                 camUnity.transform.up);       wcImgPlane.position = camUnity.transform.position +
7               wcImgPlaneDist * camUnity.transform.forward;
8
9
10    // Fit the imgPlane into the unity camera FOV
11    // compute vertical imgPlane size from FOV angle
12    float vScale = AngularSize.GetSize (wcImgPlaneDist, camUnity.fieldOfView);
13    // set the scale
14    Vector3 wcPlaneScale = wcImgPlane.localScale;
15    float ratioWH = ((float)wc.camWidth / (float)wc.camHeight);
16    wcPlaneScale.x = ratioWH * vScale * wcPlaneScale.x;
17    wcPlaneScale.y = vScale * wcPlaneScale.y;
18    wcImgPlane.localScale = wcPlaneScale;    }
```

Now hit the PLAY button, and the webcam image should have the correct aspect ratio and cover the whole Game tab screen (at least in the vertical axis).

Although not evident at this point, the main reason why we want to make the imgPlane span the whole FOV of the unityCamera is to facilitate the conversion from the coordinate system of the screen to the coordinate system of the world (seen in step 5 and 6). Instead of applying these transformations ourselves, we will use the Camera built in capability for that.

4 Track the face of the user using the Haar Cascade Classifier openCV algorithm.

We will rst create the following variables in the HeadTracking class:

```
1   // scaleFactor defines the spatial resolution for distance from camera
2   [Range (1.01f, 2.5f)]
3   public double scaleFactor = 1.1;    CvHaarClassifierCascade cascade;
4   CvMemStorage storage = new CvMemStorage ();
5
```

scaleFactor will de ne the tracking resolution of the distance from the webcam. The closer to 1, the higher the resolution. Mind that a low value will signi cantly affect the performance of the tracking algorithm.

Append the following code to the Start function. It is used to load the face tracking calibration.

```
1   void Start () { ...
2
3
4      // load the calibration for the Haar classifier cascade
5      //    docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html                cascade    =
CvHaarClassifierCascade.FromFile("./haarcascade_fr" +
6                                  "ontalface_alt.xml");    }
7
```

Replace the Update function with

```
1   void Update () {
2      ocv.UpdateOCVMat ();
3      Vector3 cvHeadPos = new Vector3 ();
4      if (HaarClassCascade (ref cvHeadPos))
5         Debug.Log (cvHeadPos);    }
6
```

Finally, copy the following function to the end of the HeadTracking class. It will track a face in the CvMat, and return a Vector3 containing the approximate position of the center of the eyes in the x and y coordinates, and the approximate diameter of the face (all in the cvMat coordinate system)

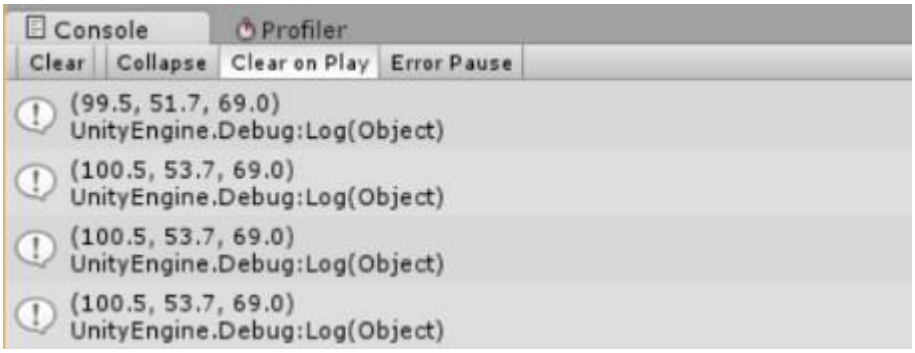
```
bool HaarClassCascade (ref Vector3 cvTrackedPos){
   CvMemStorage storage = new CvMemStorage ();
   storage.Clear();

   // define minimum head size to 1/10 of the img width
   int minSize = ocv.cvMat.Width / 10;
   // run the Haar detector algorithm
   // docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html
   CvSeq<CvAvgComp> faces =
      Cv.HaarDetectObjects (ocv.cvMat, cascade, storage, scaleFactor, 2,
                  0|HaarDetectionType.ScaleImage,
                  new CvSize(minSize, minSize));

   // if faces have been found ....       if (faces.Total > 0){
      // rectangle defining face 1
      CvRect r = faces [0].Value.Rect;
      // approx. eye center for x,y coordinates
      cvTrackedPos.x = r.X + r.Width  * 0.5f;
      cvTrackedPos.y = r.Y + r.Height * 0.3f;
      // approx. the face diameter based on the rectangle size
      cvTrackedPos.z = (r.Width + r.Height) * 0.5f;

      return true;   // YES, we found a face!
   }
   else
      return false;  // no faces in this frame    }
```

Now hit PLAY. As you move your head in front of the webcam, the Console tab should be printing the tracked position in the cvMat coordinate system.



If at this point your FPS is bellow 25, you should consider reducing the resolution of the image (e.g. reductionFactor, Cam Width and Cam Height) as well as the scaleFactor of the Haar classi er (to a bigger value).

5 Transform the coordinates system from $_4$ in order to overlay the textured plane with the head position

We rst implement a function to convert from the cvMat coordinate system to the screen coordinate system in the HeadTracking class. Mind about the difference in the y axis between these coordinate systems. The cvMat image has the x=y=0 point at the top left corner, with +y pointing down, while the screen has its x=y=0 in the bottom left corner, with +y pointing up.
We use Height – pos.y in order to compensate for this axis inversion.

```
public Vector3 CvMat2ScreenCoord (Vector3 cvPos){
   // rescale x,y position from cvMat coordinates to screen coordinates
   cvPos.x = ((float) camUnity.pixelWidth / (float) ocv.cvMat.Width) * cvPos.x;
   // swap the y coordinate origin and +y direction
   cvPos.y = ((float) camUnity.pixelHeight / (float) ocv.cvMat.Height) * (ocv.cvMat.Height - cvPos.y);


   return cvPos;
}
```

Add the following variables to the HeadTracking class

```
// approx. size of a face
float knownFaceSize = 0.15f;
// object to be controlled with head position
public Transform controlledTr;
```

Append the following to the end of the Start function

```
void Start () {       ...
   // scale controlled object to match face size
   controlledTr.localScale = knownFaceSize * Vector3.one;
 }

```

Replace the Update function with the following

```
void Update () {
   ocv.UpdateOCVMat ();
   TrackHeadOverImgPlane ();
   //TrackHead3D ();
   //TrackHead3DSmooth ();    }

```

Finally, paste the tracking function bellow to the HeadTracking class.

```
 void TrackHeadOverImgPlane (){
    Vector3 cvHeadPos = new Vector3();
    // find head position in the cvMat coordinates
    if (HaarClassCascade (ref cvHeadPos)) {


       // Set position of the controlledTr object
       // x,y cvMat coordinates to screen coordinates
       Vector3 cTrPos = CvMat2ScreenCoord (cvHeadPos);
       // z is fixed at the wcImgPlane depth
       cTrPos.z = wcImgPlaneDist;
       // x,y scree coordinates to world coordinates
       cTrPos = camUnity.ScreenToWorldPoint (cTrPos);
       controlledTr.position = cTrPos;



       // Scale controlledTr to fit the tracked head size
       // tracked head size proportional to cvMat image
       float relHeadSize = cvHeadPos.z/ocv.cvMat.Height;
       // scale x,y proportional to the plane used for render
       Vector3 cTrScale = controlledTr.localScale;
       cTrScale.x = relHeadSize * wc.imgPlane.localScale.y;
       cTrScale.y = cTrScale.x;
       controlledTr.localScale = cTrScale;


    }
 }
```

This function converts x,y coordinates from the cvMat coordinate system to the world coordinate system. Notice that we use the command camUnity.ScreenToWorldPoint to convert from the screen position to the world position, which is only possible due to the fact that the imgPlane perfectly spans the unityCamera FOV.

The obtained position is used to set the controledTr object position. We also use the head diameter to proportionally rescale this object.

Save the script. Create a new GameObject in the shape of a sphere (Game Object -> 3D Object -> Sphere), and assign it as the controlledTr in the Inspector of the unityCamera object.

Hit Play, you should see a flattened sphere overlaying your face in the Game tab

6 Transform the coordinates system from $_4$ in order to present the head position in $_{3D}$

Similarly to step 5, we now want to track the position in the z axis too. That is, instead of setting the size of the trackedTr we want to use the scale information to approximate the face distance from the webcam.

We use the difference between the obtained face diameter and the expected face diameter (knownFaceSize variable) in order to estimate the z displacement.

To do so, copy the following function to the HeadTracking class:

```
1   void TrackHead3D () {
2     Vector3 cvHeadPos = new Vector3 ();
3
4
5     // find head position in the cvMat coordinates
6     if (HaarClassCascade (ref cvHeadPos)) {
7
8       // Set position of the controlledTr object
9       // compute the face angular size
10      float faceAng = AngularSize.GetAngSize (wcImgPlaneDist, knownFaceSize);
11      // compute the face angle to camera angle ratio
12      float faceHeightRatio = faceAng / camUnity.fieldOfView;
13      // approximate face distance in world coordinates
14      cvHeadPos.z = ((faceHeightRatio * (float) ocv.cvMat.Height ) /
15              (float) cvHeadPos.z) * wcImgPlaneDist;
16      // x,y cvMat coordinates to screen coordinates
17      cvHeadPos = CvMat2ScreenCoord (cvHeadPos);
18      // x,y screen coordinates to world coordinates
19      cvHeadPos = camUnity.ScreenToWorldPoint (cvHeadPos);
20
21      controlledTr.position =  cvHeadPos;
22    }
  }
```

In the Update function, comment the call to TrackHeadOverImgPlane () and uncomment the call to TrackHead3D (). Now hit PLAY.

It is hard to perceive how the controlledTr navigates in the 3D space, to make it clearer you can activate the SplitscreenDebugView game object, which will render 3 orthogonal viewpoints of the scene (plus a perspective viewpoint).

7 Improve the tracking reliability of item $_6$ with a smoothing algorithm.

As you might have noticed, the tracking is very unreliable and the sphere is very shaky. We address this issue using a double exponential smoothing

(http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc434.htm) The double exponential smoothing can greatly stabilize the tracked position with a good response time. We do not use its prediction capability for the missing data cases (i.e. frames when no face could be tracked).

First declare the following variables in the HeadTracking class

```
Vector3 priorPos = new Vector3();
public V3DoubleExpSmoothing posSmoothPred = new V3DoubleExpSmoothing ();`
```

Then we add the TrackHead3DSmooth function to the HeadTracking class:

```
void TrackHead3DSmooth () {
   Vector3 cvHeadPos = new Vector3 ();
   if (HaarClassCascade (ref cvHeadPos)) {

      float faceAng = AngularSize.GetAngSize (wcImgPlaneDist, knownFaceSize);
      float faceHeightRatio = faceAng / camUnity.fieldOfView;
      cvHeadPos.z = ((faceHeightRatio * (float) ocv.cvMat.Height ) /
               (float) cvHeadPos.z) * wcImgPlaneDist;
      cvHeadPos = CvMat2ScreenCoord (cvHeadPos);
      cvHeadPos = camUnity.ScreenToWorldPoint (cvHeadPos);

      // the tracking is noisy, thus we only consider the new reading if it
      // lands less than .4 meters away from the last smoothed position
      if ((cvHeadPos - priorPos).magnitude < 0.4f)
         priorPos = cvHeadPos;
   }

   // update the smoothing / prediction model
   posSmoothPred.UpdateModel (priorPos);
   // update the position of unity object
   controlledTr.position =  posSmoothPred.StepPredict();    }
```

In the Update function, comment the call to TrackHead3D () and uncomment the call to TrackHead3DSmooth (). Now hit PLAY.

A noticeable improvement in the quality of tracking should be evident, with an added tracking latency as a drawback. You can re ne the smoothing alpha parameter in order to control the quality/latency tradeoff.

Now that our head tracking pipeline is ready, and you have re ned the webcam/opencv/smoothing parameters, create a prefab of the unityCamera object (drag and drop it from the Hierarchy tab to the Project tab)

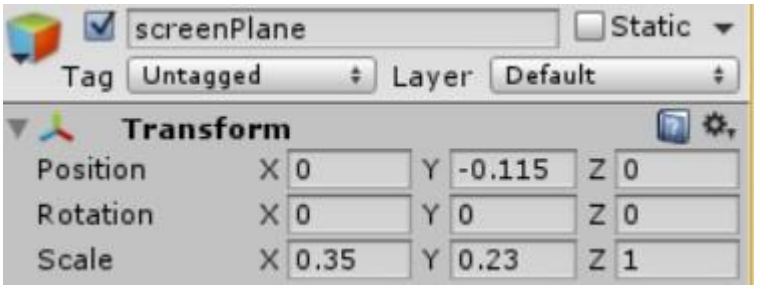
Asymmetric Camera Frustum

Now we will use our HeadTracking class in order to de ne the position of a camera, and deform the camera frustum in order to replicate the effect discussed in the introduction section.

We will test it in two different scenes, the FishtankVR and the WindowVR, which demonstrate distinct conceptual scenarios. Fishtank refers to scenes of very limited dimensions, and the fact that your reach is limited by a glass-like barrier (the computer screen). Window to virtual reality refers to scenes where the computer screen behaves as a window to a vast immaterial world of virtual reality.

1 FishtankVR

Open the FishtankVR scene. Create a new Empty game object gameObject (GameObject -> Create Empty), rename it to screenPlane. Set the scale of the screenPlane x,y scale to the size of your notebook screen (in meters), and set the y position to minus half of the y scale:

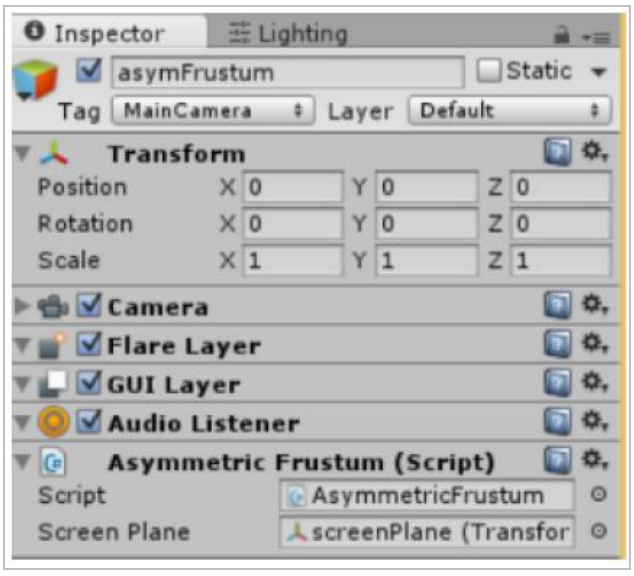


In the gure I use an approximation of my computer screen size.

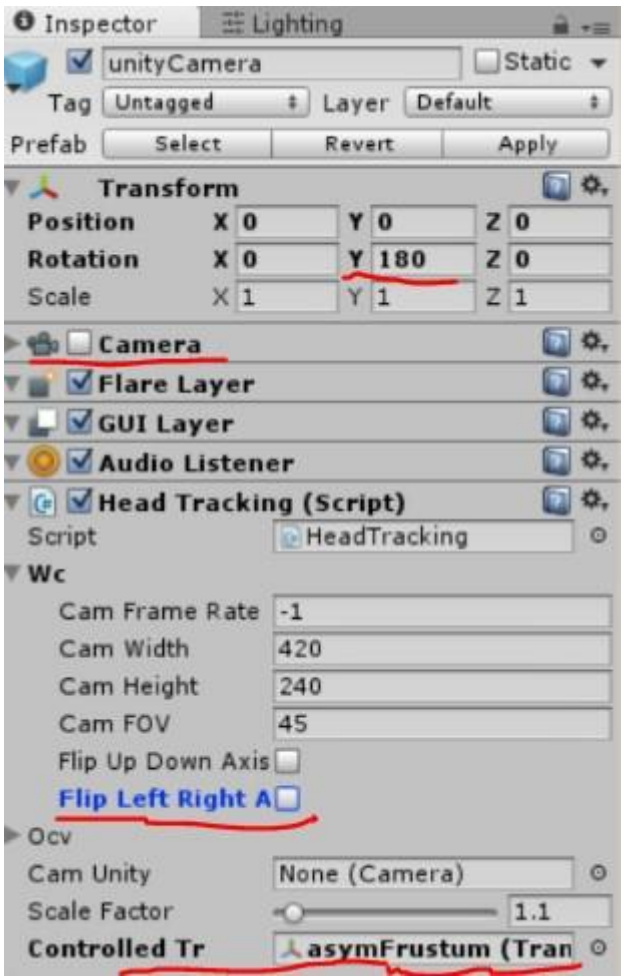
In the FishtankVR object, set the Reference Transfer parameter (from the TransformScene.cs script) to screenPlane (drag and drop the object from the hierarchy tab to the Inspector tab).

The FishtankVR object will be scaled to t the screenPlane size. It will also be translated/rotated to the same position/orientation.

Now create a new camera (GameObject -> Camera), rename it to asymFrustum. Attach the script AsymmetricFrustum.cs to it, and set the parameter Screen Plane with the screenPlane object (drag and drop).



Instantiate your unityCamera.prefab (drag and drop from the Project tab to the Hierarchy tab). Set the position of the new unityCamera to 0, and rotate it by 180 degrees in the y axis. Next disable the Camera component of the unityCamera object, and set the controlled Tr parameter to the asymFrustum object (drag and drop). Finally, you will have to invert your option Flip Left Right Axis in the HeadTracking.cs component.



Save the scene (ctrl + s) and hit PLAY. The asymCamera should be moving according to your head position, and the frustum of this camera should be deformed so that its boundaries are passing through the screenPlane limits. The dynamically changing frustum allows for motion parallax, which in turn gives relevant cues about the relative depth of the objects in the scene.

2 WindowVR

Open the WindowVR scene. Repeat the steps given for 1-FishtankVR, the only difference is that the TransformScene.cs script is located in the windowFrame object (before it was located in the FishtankVR object).