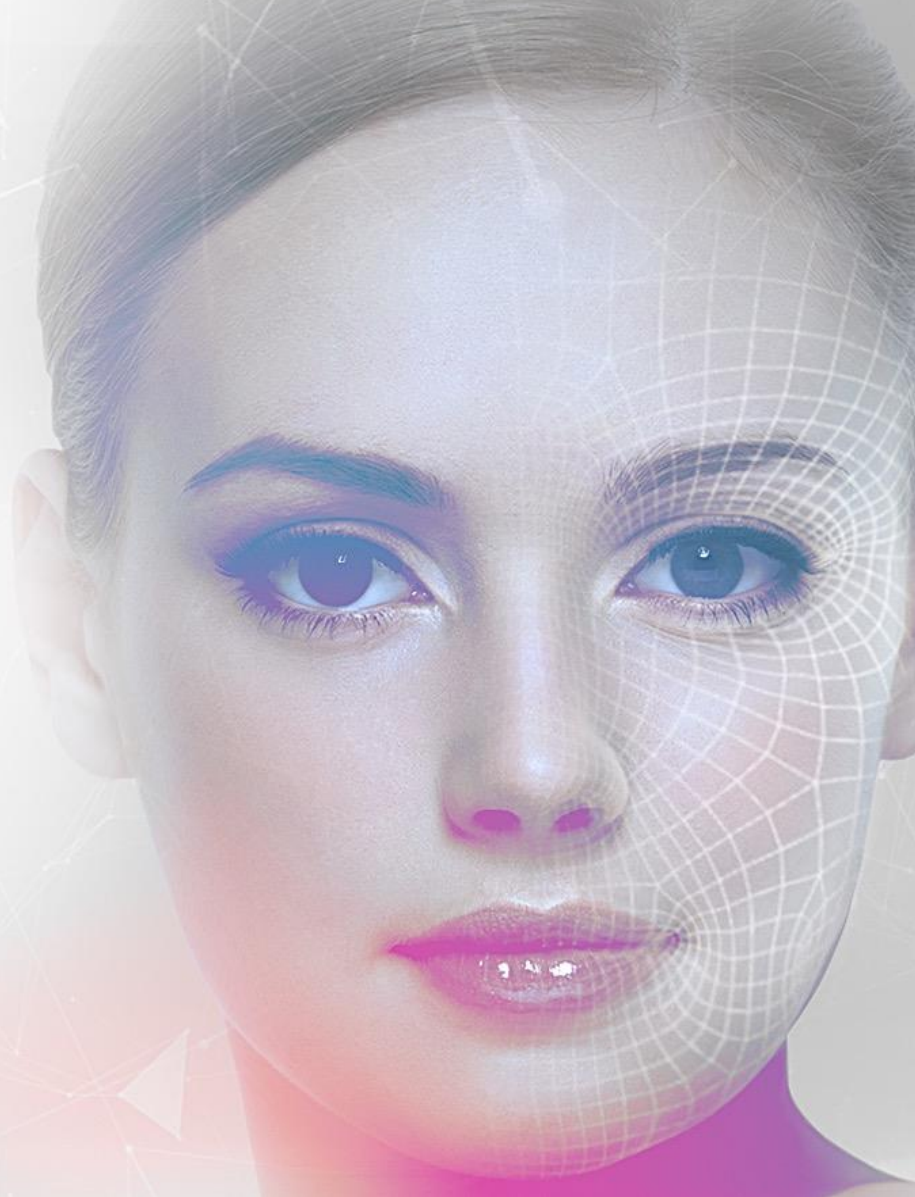**Single Face Tracker for Unity Plugin v1.3.20 User Manual - Windows, macOS, iOS and Android Builds**

Document Version 1.3.20

8 August 2018

ULSee

Contact Us:

Mainland China

Telephone: +86-159-2113-6467

Email: kevin@ulsee.com

Taiwan

3F, No. 28, Lane 128, Jingye 1st Road, Zhongshan District, Taipei, Taiwan, 10462

Tel: +886-2-8502-2798

Email: sales@ulsee.com

Document Revision History

This table describes the changes to this document:

Date	Version	Added by	Notes
2018/08/08	1.3.20	Sunil Surender	Updated document for Single Face Tracker for Unity Plugin v1.3.20 User Manual - Windows, macOS, iOS and Android Builds
2018/06/11	1.3.19	Sunil Surender	Updated document for Single Face Tracker for Unity Plugin v1.3.19 User Manual - Windows, macOS, iOS and Android Builds
2018/03/20	1.3.17	Sunil Surender	Updated document for Single Face Tracker for Unity Plugin v1.3.17 User Manual - Windows, macOS, iOS and Android Builds
2018/01/16	1.3.15	Sunil Surender	Updated document for Single Face Tracker for Unity Plugin v1.3.15 User Manual - Windows, macOS, iOS and Android Builds
2017/10/24	1.3.14	Sunil Surender	Updated document for Single Face Tracker for Unity Plugin v1.3.14 User Manual - Windows, macOS, iOS and Android Builds
2017/05/22	1.3.13	Sunil Surender	New document to create Single Face Tracker for Unity Plugin v1.3.13 User Manual - Windows, macOS, iOS and Android Builds

Table of Contents

Overview

This document is a step-by-step guide to assist Unity game developers in using ULSee's Single Face Tracker plugin sample code for Windows, macOS, iOS and Android application development. These instructions are designed to help these developers with integration of the Face Tracker plugin quickly and effectively.

Face Tracker Features and Capabilities

- Cross-platform plugin for Windows, macOS, iOS and Android

- APIs for addition of 2D objects to face images

- APIs for addition of 3D objects to face images

- API Scripts for – Face Mask implementation, Frames per Second (FPS) Counter and Face Tracker implementation on Windows (32-bit and 64-bit), macOS, iOS and Android

- Materials (incl. shaders and textures)

- Support for Auto Graphics API and Metal API for graphical performance-boost

- Synchronizing of camera input with tracker output (macOS, Windows and iOS only)

- Support for Landscape Mode on iOS and Android devices

- Support for physical/virtual camera rotation in desktop apps

- Change camera setting

- Enable camera flash light and getting status of camera flash light

- Pause or resume camera feed

- Advanced techniques used to detect and track 30 facial discriminant landmarks on a face image in real-time, including critical facial features, like jawlines, under sub-optimal conditions and challenging factors such as over-exposed backgrounds, rapid head movement and various skin tones

- The technology works with standard cameras found in smartphones, tablets, laptops and webcams, therefore, 3D cameras and depth sensors are not required

- Angle and position determination in 3D space with face images from 2D camera

- Changes in head position do not affect automatic detection and tracking of crucial facial landmarks

- Dynamic presentation of facial points to find outline of face and to establish the facial features

- Occluded facial features can be accurately predicted with each facial point possessing a separate confidence value

- Calculation of each face point's confidence value is done separately and intelligent assessment of this data is used to correct facial points continuously

- Estimation of gaze direction at different head positions, such as, pitch, roll and yaw

- Synchronizing of camera input with tracker output (macOS, Windows and iOS only)

- Support for Landscape Mode

- Changing camera setting

- Enabling camera flash light

- Getting status of camera flash light

- Pause/Resume camera feed

Functionality

ULSee's face-tracking technology uses advanced techniques to detect and track a total of *66* facial discriminant landmarks on a face image in real-time, including critical facial features, like jawlines, under sub-optimal conditions and challenging factors, such as occlusion, poor lighting, over-exposed backgrounds, rapid head movement and various skin tones. The technology works with standard cameras found in smartphones, tablets, laptops, webcams and other personal electronic devices. 3D cameras and depth sensors are not required.

The Face Tracker can receive face images from a camera input and detect the face landmarks on the image. These landmarks are then tracked in the frames to follow. This tracking is performed with the help of a 3D facial mask model in the background, that assists with accurate positioning of the detected landmarks. If there is no face in the image, the Face Tracker does not detect the face landmarks.

Figure 1 highlights the positioning of *30* face landmarks provided with this basic version of plugin for Unity. *Figure 2* defines our guideline for different head positions – pitch, roll and yaw. Landmarks 0-4 are for jawline shape detailing. Landmarks 5-6 and 7-8 are for right eyebrow and left eyebrow shape detailing respectively. Landmarks 14-17 and 18-21 are for right eye and left eye shape detailing respectively. Landmark 9 is for nose bridge point tracking and landmarks 10-

13 are for nostrils and nose septum shape detailing. Landmarks 22-29 are for detailing the shape of the upper and lower lips.
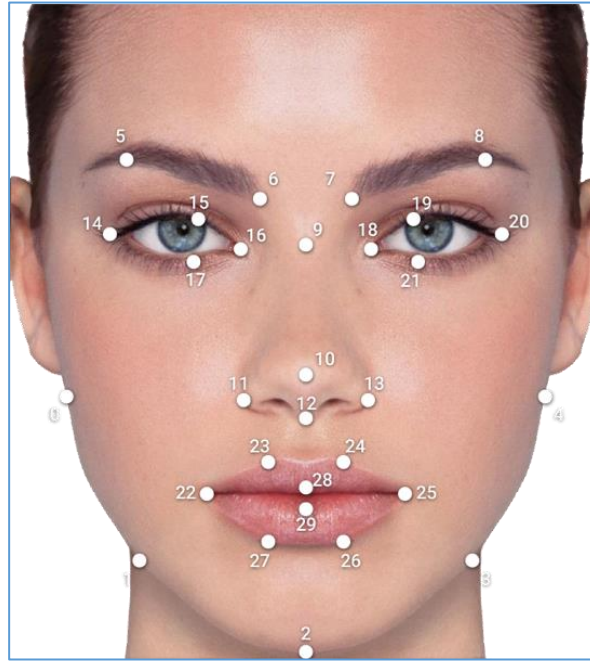

Figure 1. Positioning of Face landmarks by ULSee's Face Tracker

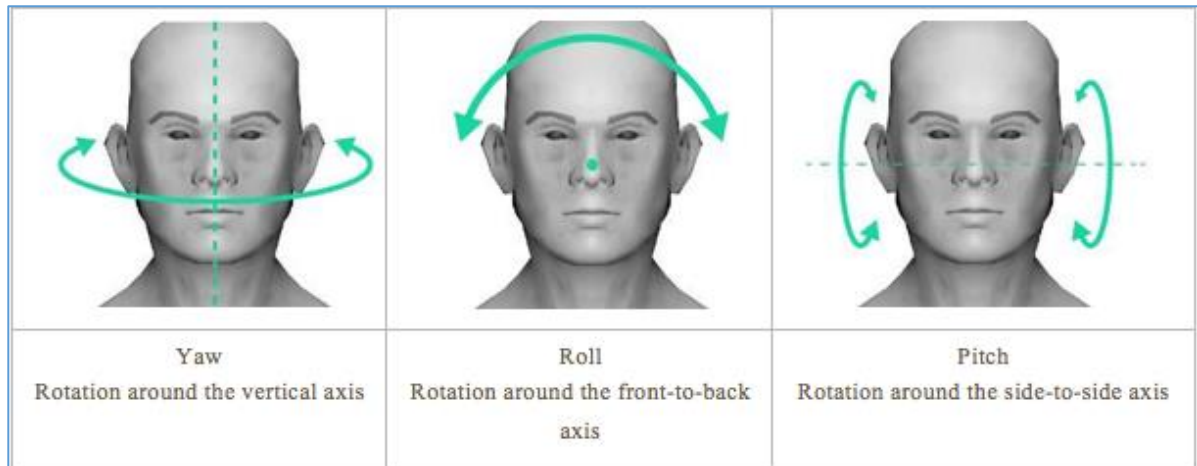| Yaw	Roll	Pitch
Rotation around the vertical axis	Rotation around the front-to-back axis	Rotation around the side-to-side axis

Figure 2. Definitions of Pitch, Roll and Yaw - Range of these properties are determined from the tracker performance indicators

Minimum System Requirements

a) Windows 8.1 (32-bit and 64-bit) or Windows 10 (32-bit and 64-bit) is recommended with i5 or i7 Processor

b) Macintosh OSX 10.11 or better is recommended with i5 or i7 Processor

c) iOS - iOS 9.0 and above (recommended)

d) Android - Quad-core Krait CPUs or better with Android 4.1 or higher (API Level 16 or higher) (recommended)

e) 8GB DDR3 RAM (for Windows and macOS) and 2GB (for iOS and Android)

f) Unity Editor version 5.6.0 and above

What is New in Version 1.3.20

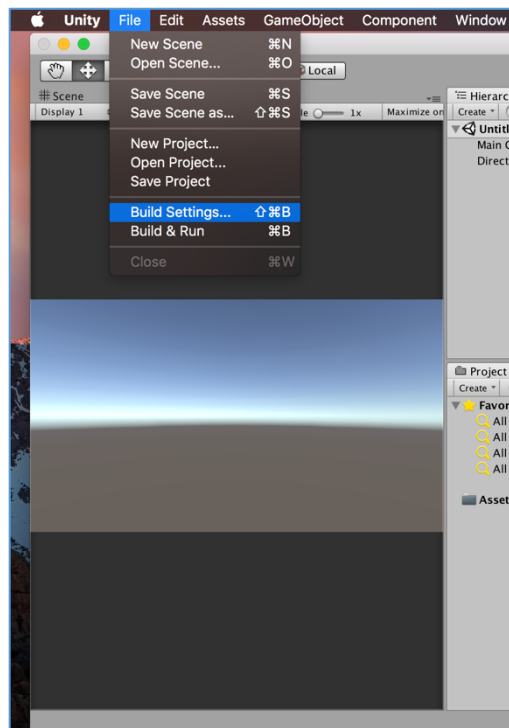Added Support for Bitcode at lower-level libraries for iOS builds

Adding the Face Tracker Plugin to your Unity Project

Step 1: Download the *ULSFaceTrackerLite1_3_20.unitypackage* file from Unity Store

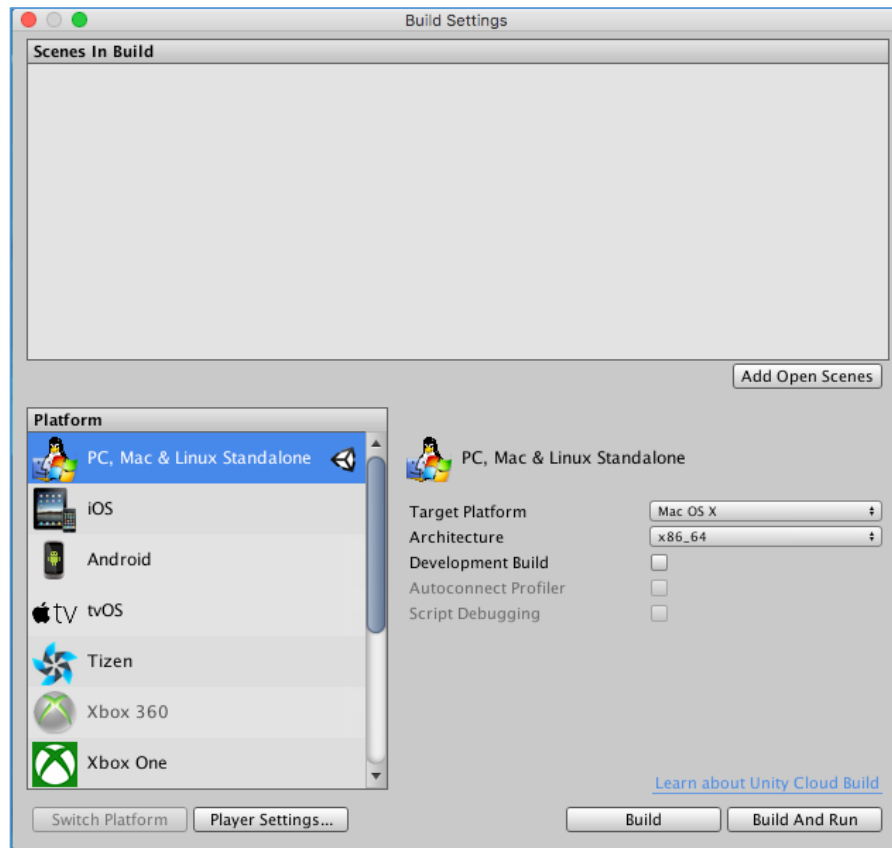**Step 2:** Create a new project in the Unity Editor (version 5.6.0f3 or above, for Windows or macOS)

Step 3: In the Unity Editor menu, click on *File → Build Settings*

Step 4: In the *Build Settings* window, under *Platform* options:

- **For Windows, macOS or Linux Desktops:** Click on *PC, Mac & Linux Standalone* (Unity Editor's default selection), before closing the window.

Note:

a) Default camera orientation is Landscape. For face tracking with real or virtual camera rotation, refer to Plugins.cs script and update *CameraDispatch.cs* script. The SDK Functions section provides the API description for reference.

b) Synchronization of camera input with face tracker output occurs in the same frame.

c) Hardware specifications would depend on how many objects the application must compute and draw

d) Desktop builds use CPU power

- **For iOS devices (iOS 9.0 and newer versions):** Click on *iOS* and then click on *Switch Platform* on the bottom left, before closing the window
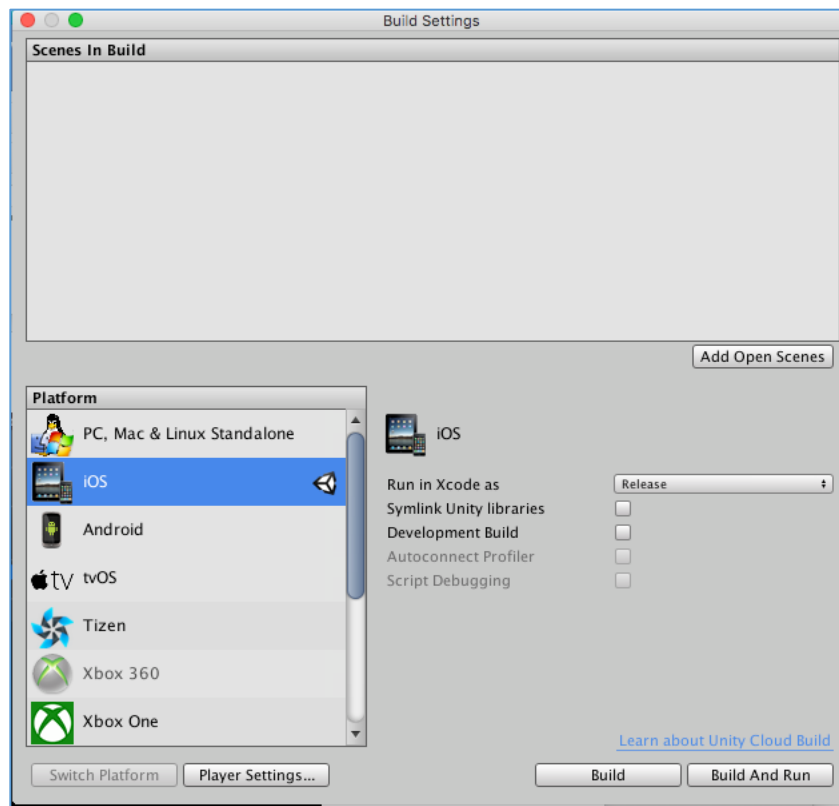
Note:

a) Under *Player Settings* → click on 🔲 → *Other Settings* —> Choose *Auto Graphics API* -or- uncheck *Auto Graphics API* and select only *OpenGLES2* -or- uncheck *Auto Graphics API* and select only *Metal*

b) In **Configuration** → *Camera Usage Description*, provide the desired description
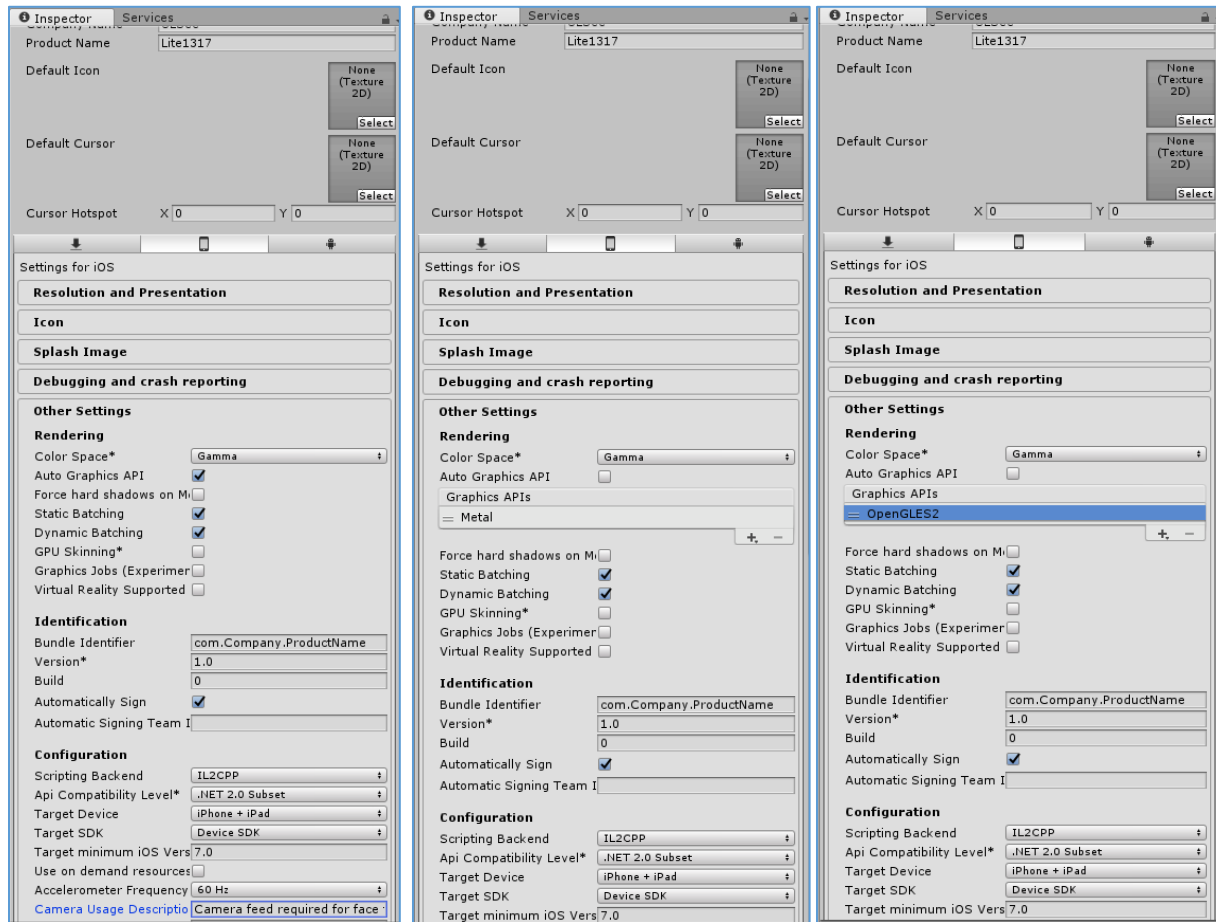
- **For Android devices (Android versions 4.1 API Level 16 and later):**

  a) Click on *Android* and then click on *Switch Platform* on the bottom left
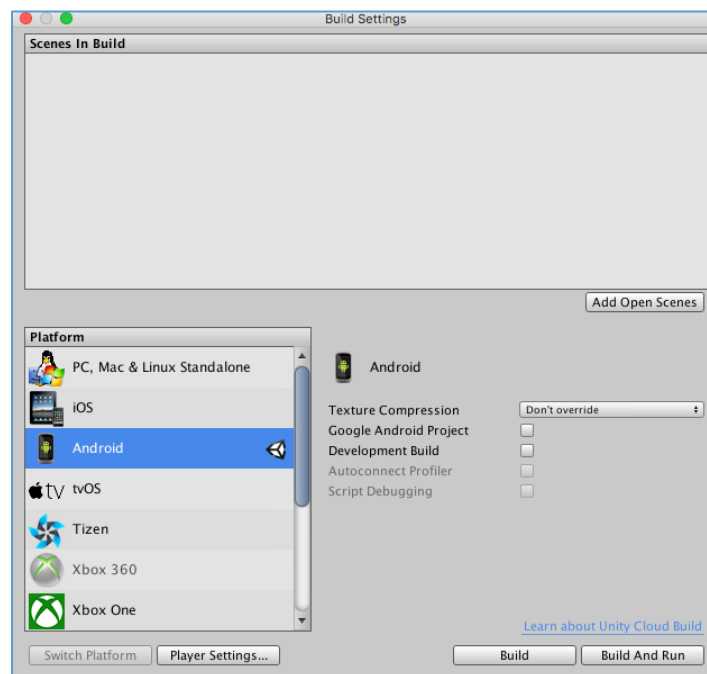
b) Click on *Player Settings* to reveal *Inspector*

Under *Other Settings*,

- *Identification* → *Bundle Identifier* → Change the default value to one of your preference
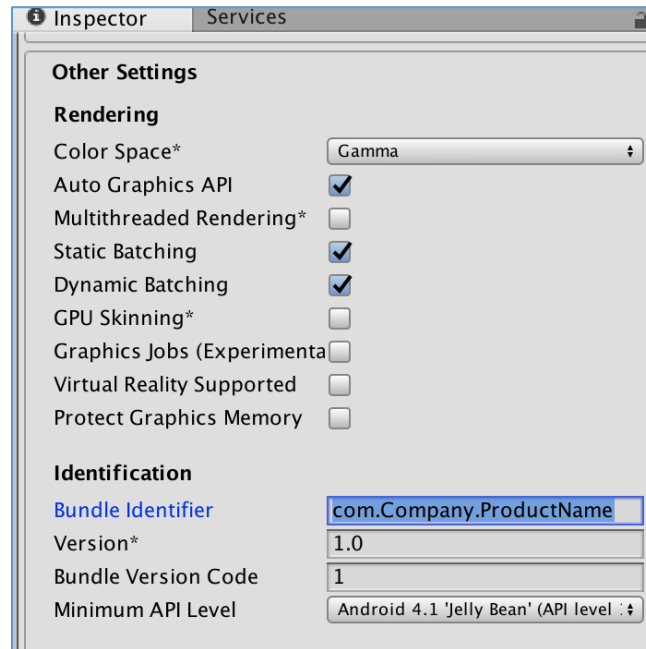
- *Identification*→ *Minimum API Level* → Android 4.1 'Jelly Bean' (API Level 16)
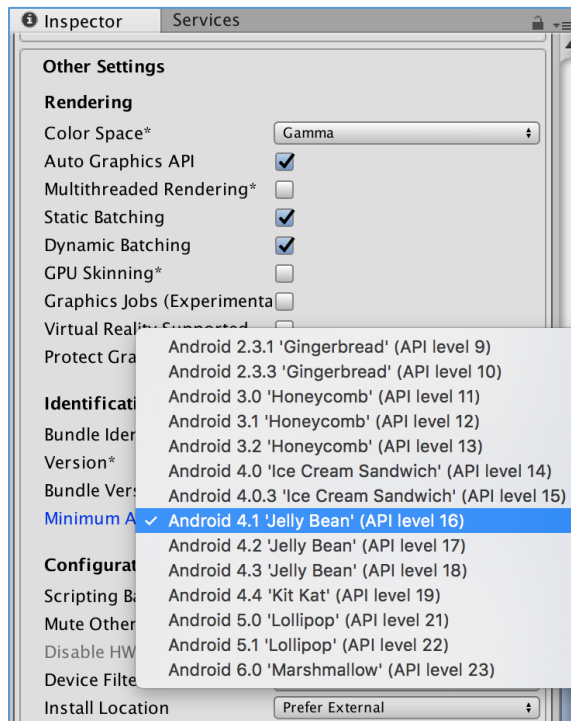
Note:

a) Under *Player Settings* → click on   → *Resolution and Presentation* →
Default Orientation should be adjusted based on developers' requirement
(Portrait. Portrait Upside Down, Landscape Right or Landscape Left)

b) To use the support for Landscape mode provided with the plugin, select
Landscape Right or *Landscape Left*

Settings for Android

Resolution and Presentation

Orientation

Default Orientation*   Landscape Right

Use 32–bit Display Buffer*   ☑
Disable Depth and Stencil*   ☐
Show Loading Indicator   Don't Show

* Shared setting between multiple platforms.

Settings for Android

Resolution and Presentation

Orientation

Default Orientation*   Landscape Left

Use 32–bit Display Buffer*   ☑
Disable Depth and Stencil*   ☐
Show Loading Indicator   Don't Show

* Shared setting between multiple platforms.

c) Android SDK Tools version 26.0.x or higher is recommended

Step 5: In the Unity Editor menu, click on *Assets* → *Import Package* → *Custom Package*
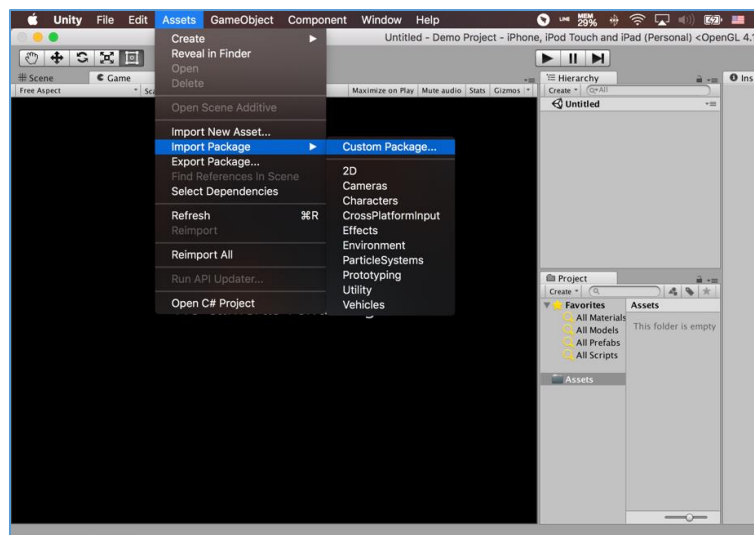
A small window titled *Import Unity Package* opens. Ensure that all contents under *ULSFaceTracker* are selected and click on *Import*.
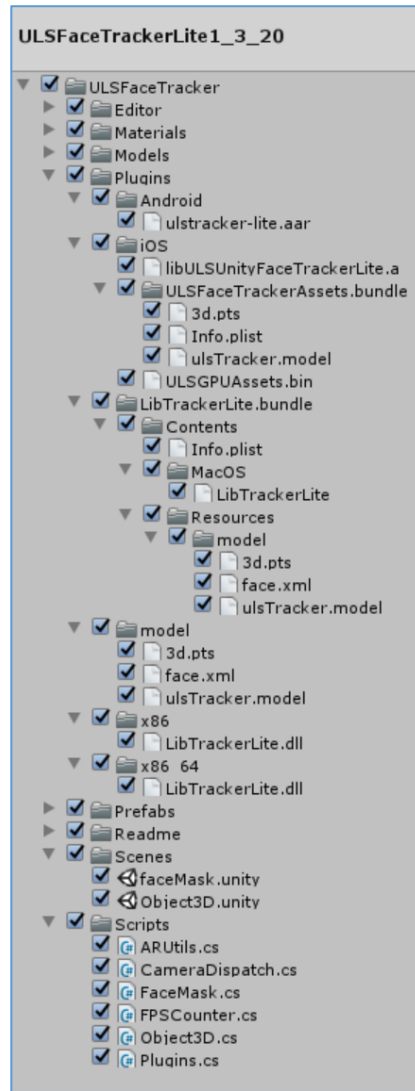
Step 6: The plugin package has now been imported

Step 7: You may now add your executable code to the *Assets* folder in Unity Editor

Step 8: Once your executable code has been added, select *Build Settings* from Unity Editor menu *File*

Note: In the *Build Settings* window**,** under *Scenes In Build***,** add the *faceMask* scene (to demonstrate adding of 2D objects to face image) and *Object3D* scene provided with the plugin package (to demonstrate adding of 3D objects to face image), by drag-and-drop from the *Scenes* folder of your Unity project.
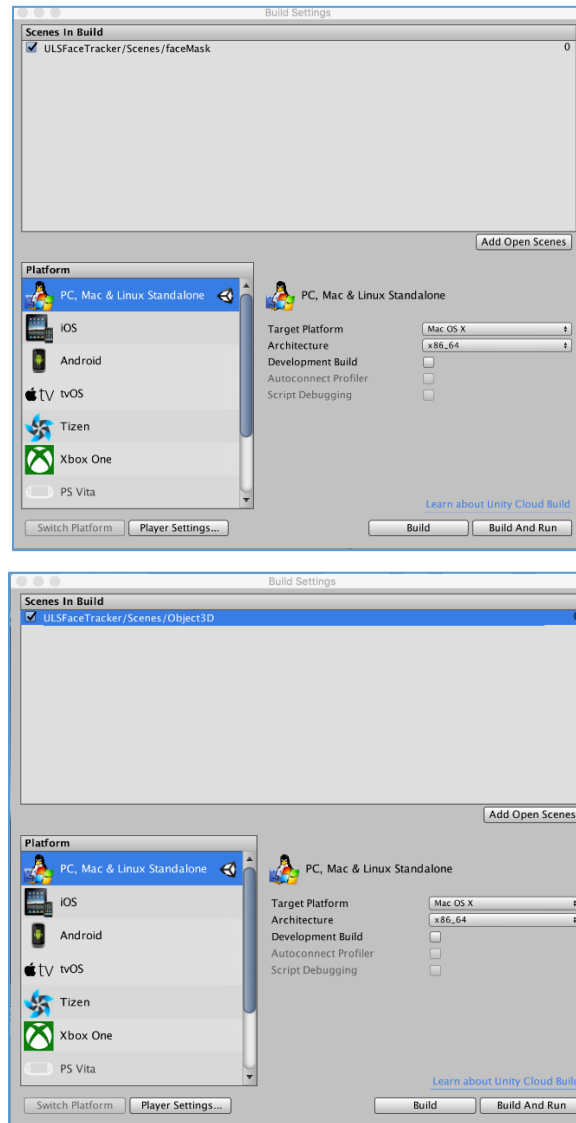
Step 10: Click on *Build And Run* to compile your code with the appropriate scene, based on your preference to use 2D objects or 3D objects with face image.

Note: If Xcode 8 is being used to build and run for iOS or Mac, please follow these instructions:

- If your application requires the WebCamTexture API, the reason must be mentioned in Info.plist located under *Supporting Files* folder of your Xcode project created by Unity. Add the *NSCameraUsageDescription* key with string value containing the description

- If your application doesn't require the WebCamTexture API, disable the camera code in your Xcode project created by Unity. This is a temporary measure. Proceed by opening Classes/Unity/CameraCapture.mm in your created Xcode project, go to the first line and replace "#if !UNITY_TVOS" and with "#if 0". This will effectively remove camera capture code and should help pass automatic submission check.

Quick summary on iOS 11.x and Xcode 9.x Compatibility

Latest Unity versions for macOS and Windows (Unity 2018.2.2) are recommended for development. Older Unity versions may work too, but issues may be encountered.

Known Issues Found in iOS and Android Builds

Initialization of *Switch Camera* function takes time. It is recommended that other functions not be called while *Switch Camera* function initialization is in process.

SDK Functions

In this section, we introduce and describe the Face Tracker functions that feature in our sample code. Functions are packaged in classes **ULSTrackerForUnity.Plugins** (reference script: *Plugins.cs*), **FaceMask** (reference script: *FaceMask.cs*) and **Object3D** (reference script: *Object3D.cs*)**:**

Initiate Camera Preview Texture for Face Tracker for Mapping of 2D or 3D objects to Device Camera Preview Image
[for FaceMask scene (reference script: FaceMask.cs) and Object3D scene (reference script: Object3D.cs)]

Function: Plugins.OnPreviewStart = <initial_camera_callback_function>

Function Description: This function is a callback API after the Face Tracker has been launched, to setup camera preview texture [and Augmented Reality (AR) matrix for mapping 3D objects] the first time, by passing the texture and camera rotation as parameters. We have provided this custom interface to access camera preview texture for the developer to use.

The default preview resolution is 640x480 pixels for mobile platforms and the resolution for desktops would be platform-dependent.

Custom Callback Function Interface:

void <initial_camera_callback_function> (Texture preview, int rotate)

Parameters of Custom Callback Function Interface:

Preview: Input Texture of camera preview

rotate: Input camera rotation in degrees

Mockup of an Intrinsic Camera Matrix for Perspective View

Calculations:

intrinsic_camera_matrix[0] = _focal_length

intrinsic_camera_matrix[4] = _focal_length

intrinsic_camera_matrix[2] = w/2

intrinsic_camera_matrix[5] = h/2

Description:

These calculations are to obtain the intrinsic camera matrix, based on your assumptive values for host device camera parameters – focal length, camera preview width and camera preview height.

Get Calibrated Field of View

Function: void UnityCalibration([In] float[] intrinsic_camera_matrix, float image_width, float image_height, [In, Out] float[] fovx, [In, Out] float[] fovy)

Function Description: This function is for the AR camera to obtain calibrated field of view.

Parameters:

intrinsic_camera_matrix: Input intrinsic Camera Matrix i.e.,

[focal_length_x, 0, image_center_x, 0, focal_length_y, image_center_y, 0, 0, 1]

image_width: Input image width

image_height: Input image height

fovx: Output field of view in degrees along the horizontal sensor axis

fovy: Output field of view in degrees along the vertical sensor axis

Get 3D Scale Vector

Function: void UnityGetScale3D([In, Out] float [] x, [In, Out] float [] y, [In, Out] float [] z)

Function Description: This function is to get the 3D scale vector compared to the original 3D model of Face Tracker.

Parameters:

x: X-axis of scale vector

y: Y-axis of scale vector

z: Z-axis of scale vector

Adjust Transform, Scale and Position to Map Face Tracking Points

Calculations for Windows and macOS Desktops

To adjust scale and position for mapping tracker points in 2 Dimensions, use the following calculations:

transform.localScale = new Vector3 (w, h, 1);

transform.localPosition = new Vector3 (w/2, h/2, 1);

transform.parent.localScale = new Vector3 (-1, -1, 1);

transform.parent.localPosition = new Vector3 (w/2, h/2, 0);

Camera.main.orthographicSize = h / 2

where,

w = camera preview width

h = camera preview height

To adjust scale and position for mapping tracker points in 3 Dimensions, include the following calculations:

ARCamera.fieldOfView = _fovy[0];

adjustMatrix = Matrix4x4.TRS (Vector3.zero, Quaternion.identity, new Vector3 (-1, -1, 1))

Calculations for iOS and Android

To adjust scale and position for mapping tracker points in 2 Dimensions, use the following calculations:

rotate += 90   **[choose this for Landscape Left view orientation of your app]**

rotate -= 90   **[choose this for Landscape Left view orientation of your app]**

int orthographicSize = w / 2

if (Screen.orientation == ScreenOrientation.LandscapeLeft || Screen.orientation ==

ScreenOrientation.LandscapeRight)

{

**   orthographicSize = h / 2**

}

transform.localScale = new Vector3 (w, h, 1)

transform.localPosition = new Vector3 (w/2, h/2, 1)   **[anchor: left-bottom]**

transform.parent.localPosition = Vector3.zero   **[reset position for rotation]**

transform.parent.transform.eulerAngles = new Vector3 (0, 0, rotate)   **[orientation]**

transform.parent.localPosition = transform.parent.transform.TransformPoint(-

transform.localPosition)   **[move to center]**

Camera.main.orthographicSize = orthographicSize

To adjust scale and position for mapping tracker points in 3 Dimensions, use the following calculations:

rotate += 90 [choose this for Landscape Left view orientation of your app]

rotate -= 90 [choose this for Landscape Left view orientation of your app]

if (Screen.orientation == ScreenOrientation.LandscapeLeft || Screen.orientation == ScreenOrientation.LandscapeRight)

{

Camera.main.orthographicSize = h/2

float v = (float)(w * Screen.height) / (h * Screen.width)

ARCamera.rect = new Rect ((1-v)*0.5f, 0, v, 1)　　　　　　　**[AR viewport]**

ARCamera.fieldOfView = _fovy[0]

}

else

{

    float aspect = ((float)Screen.height)/((float)Screen.width)*h/w

    float v = 1f / aspect

    Camera.main.orthographicSize = w/2 * aspect

    ARCamera.rect = new Rect (0, (1-v)*0.5f, 1, v)　　　　**[AR viewport]**

    ARCamera.fieldOfView = _fovx[0]

}

transform.localScale = new Vector3 (w, h, 1)

transform.localPosition = new Vector3 (w/2, h/2, 1)　　　　**[anchor: left-bottom]**

transform.parent.localPosition = Vector3.zero　　　　　**[reset position for rotation]**

transform.parent.transform.eulerAngles = new Vector3 (0, 0, rotate)　　**[orientation]**

transform.parent.localPosition = transform.parent.transform.TransformPoint(-transform.localPosition)　　　　　　　　　　　**[move to center]**

adjustMatrix = Matrix4x4.TRS (Vector3.zero, Quaternion.Euler(new Vector3(0,0,rotate)), new Vector3 (1, 1, 1))

Description: Camera orientation varies across Windows, macOS, iOS and Android platforms. The camera orientation calculations listed above would help in adjustment of camera orientation based on the operating system platform of the host device. Camera orientation is required for accurate mapping of face size and face tracking points, to add 2D or 3D objects to face image.

Initialize Face Tracker

Function: int UnityTrackerInit()

Function Description: This function is to initialize the camera and the Face Tracker with the activation key already integrated into the code by ULSee Inc.

An active Internet connection is required for tracker initialization, for a key validation algorithm built into the plugin. This algorithm requires an active Internet connection on the device. Unity should recognize the active connection too.

Return Values:

Activation Key failed (Activation Key Failure) or *Failed to initialize tracker (Tracker Initialization Failure*

Solution:

Switch off Wi-Fi on the device and switch it ON again, ensuring that the device is connected to the Internet, before trying again (re-launch Unity Editor if necessary) by double-clicking on any of the scenes in the Scenes folder and click on the Play button to check if rendering of 2D object and 3D objects on face image is operational.

Change Camera Setting

Function: void UnitySetupCamera(int width, int height, int fps, bool frontal)

Function Description: This function is to change camera setting.

Parameter:

width: camera width

width: camera height

fps: frame rate per second

frontal: front camera or rear camera

Parameter Values for *frontal*:

a) **true -** Front Camera
b) **false -** Rear Camera

Find Tracking Points and Continue Tracking Points

Function: int UnityTrackerUpdate([In, Out] byte[] image, int width, int height)

Function Description: Find facial landmarks (discriminant points) in face image and continue tracking these points.

Parameters:

image: Position of face image in x- and y-axes

width: image width

height: image height

Return Values: Integer numbers 1 to 30

Set-up Video Capture

Function: int UnityCreateVideoCapture(int index, int width, int height, int fps, int rotate)

Function Description: This function is to set-up the video capture based on camera ID, desired resolution, frame rate and rotation flag.

Parameters:

index: Camera ID like 0, 1, 2, 3, …

width: width of video capture file resolution

height: height of video capture file resolution

fps: frames per second

rotate: rotation flag to be set (for desktop Windows and macOS apps only)

[-1 is default landscape orientation, 0: rotate counter-clockwise, 1: flip and 2: rotate clockwise]

Update Video Capture

Function: void UnityUpdateVideoCapture()

Function Description: This function is to update the video capture.

Shut-down Video Capture

Function: void UnityCloseVideoCapture()

Function Description: This function is to shut-down video capture.

Obtain Two-Dimensional Positions of Face Landmark Points

Function: int UnityGetPoints([In, Out] float[] points2d)

Function Description: This function is to obtain (x, y) coordinate positions of 30 tracked landmarks as raw data i.e., $[x_1,y_1,x_2,y_2,...,x_{30},y_{30}]$.

Parameters:

points2d: Positions of face landmarks in x- and y-axes

Return Values and their meaning:

    a)   **0** – Failure

b) **Any non-zero value – **Address

Obtain Number of Face Landmark Points

Function: int UnityGetTrackerPointNum()

Function Description: This function is to obtain number of face tracking points.

Return Values and their meaning:

a) **0 – **No tracking points

b) **1 to 30 – **Number of tracking points

Obtain Confidence Level of Face Landmark Points

Function: int ULS UnityGetConfidence([In, Out] float[] conf)

Function Description: This function is to obtain confidence level for positions of face landmark points in 2 dimensions.

Parameters:

conf: Confidence level for positions of face landmark points in x- and y-axes

Parameter Values and their meaning:

a) **0 - **Failure

b) **Any non-zero value - **Confidence Level

Obtain Pitch Value

Function: float UnityGetPitchRadians ()

Function Description: This function is to obtain estimated angle of pitch.

Return Values: A floating point value between -180 and 180 (in radian equivalent)

Obtain Yaw Value

Function: float UnityGetYawRadians ()

Function Description: This function is to obtain estimated angle of yaw.

Return Values: A floating point value between -180 and 180 degrees (in radian equivalent)

Obtain Roll Value

Function: float UnityGetRollRadians ()

Function Description: This function is to obtain estimated angle of roll.

Return Values: A floating point value between -180 and 180 degrees (in radian equivalent)

Obtain Size of Face in Image Captured by Camera

Function: float UnityGetScaleInImage()

Function Description: This function is to obtain size of face in image captured by camera in real-time.

Return Values: A floating point value

Get Transformation Matrix to Align 3D Tracking Points with 2D Tracking Points

Function: void UnityGetTransform([In, Out] float [] transform, [In] float[] intrinsic_camera_matrix, [In] float[] distort_coeffs)

Function Description: This function is to obtain transform matrix to align 3D tracking points with 2D tracking points.

Parameters:

transform: Output Transform Matrix

intrinsic_camera_matrix: Input intrinsic Camera Matrix i.e.,

[focal_length_x, 0, image_center_x, 0, focal_length_y, image_center_y, 0, 0, 1]

distort_coeffs: Input vector of distortion coefficients ($k\_1, k\_2, p\_1, p\_2[, k\_3[, k\_4, k\_5, k\_6]]$) of 4, 5, or 8 elements

Note: If the vector is NULL/empty, zero distortion coefficients are assumed.

Enable/Disable Tracker

Function: void UnityTrackerEnable ([In] bool isTracking)

Function Description: This function is to enable or disable face tracker.

Parameter:

isTracking: A Boolean value to determine if tracker is to be enabled or disabled

Parameter Values:

    a)   **0 -** Disable tracking

    b)   **1 -** Enable tracking

Pause/Resume Camera Feed

Function: void UnityPauseCamera(bool paused)

Function Description: This function is to pause or to resume camera feed.

Parameter:

paused: A Boolean value to enable pausing of camera feed.

Parameter Values:

a) **true -** Pause Camera Feed

b) **false -** Resume Camera Feed

Enable/Disable Camera Flashlight

Function: void ULS_UnitySetFlashLight(bool enable)

Function Description: This function is to enable or to disable camera flashlight.

Parameter:

enable: A Boolean value to determine if camera flashlight is to be enabled or disabled

Parameter Values:

a) **true -** Enable Camera Flashlight

b) **false -** Disable Camera Flashlight

Obtain Status of Camera Flashlight

Function: bool UnityGetFlashLight()

Function Description: This function is to obtain the status of camera flashlight.

Return Values:

a) **true -** Camera Flashlight is enabled

b) **false -** Camera Flashlight is disabled

Terminate Face Tracker and Close Camera

Function: void UnityTrackerTerminate()

Function Description: This function is to terminate face tracker and close camera.
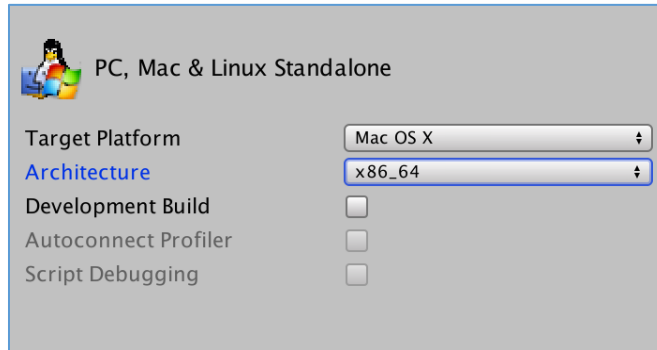
Troubleshooting

Builds-and-tests on macOS environment for Object3D and FaceMask scenes -

Builds for macOS platform:

Issue: Camera operation fails to launch even though Facetime can launch camera operation

Solution:

a) Type *sudo killall VDCAssistant* on *Terminal* console to terminate all camera processes

b) Build for 64-bit Architecture (not *Universal*)

Builds for Android platform:

Issue: Crash on Oppo R7

Solution: Toggle Wi-Fi OFF and ON

Builds for iOS platform:

Issue: Tracker initialization may fail sometimes due to Internet connectivity issues, LAN/ISP Firewall Settings

Solution: Toggle Wi-Fi OFF and ON, ensuring that the device is connected to the Internet, while testing for tracker initialization. It is also important to check if Internet connection is active on the test device and is not blocked by Lan/ISP firewall settings.

Builds-and-tests on Windows environment for Object3D and FaceMask scenes -

Builds for Android platform:

Issue: 3D mask is brighter than build for macOS platform

Solution: Screen interface settings are to be adjusted

FAQs

1) What exactly is the scale? When is scale equal to one?

A. ULSee's approach to scale determination is to use weak perspective projection in our camera coordinate system, to approximate perspective with scaled orthographic.

With function *(float)getScaleInImage*, one can obtain the scale measure of a face i.e., the size of the tracked face shape in image is equal to scale*basis_shape (basis shape is 128x128 i.e., the size of the face image getting trained). Therefore, when scale is equal to one, the size of tracked face shape is equal to the size of basis shape.

2) Regarding 3D Face Shapes, what is the camera coordinate system? Where is the origin in this system?

A. Camera coordinate system is the standard coordinate system of a camera, widely used in computer vision technology, with the origin at the center of projection (center of camera). Our application of coordinate systems is best described by the figure below:
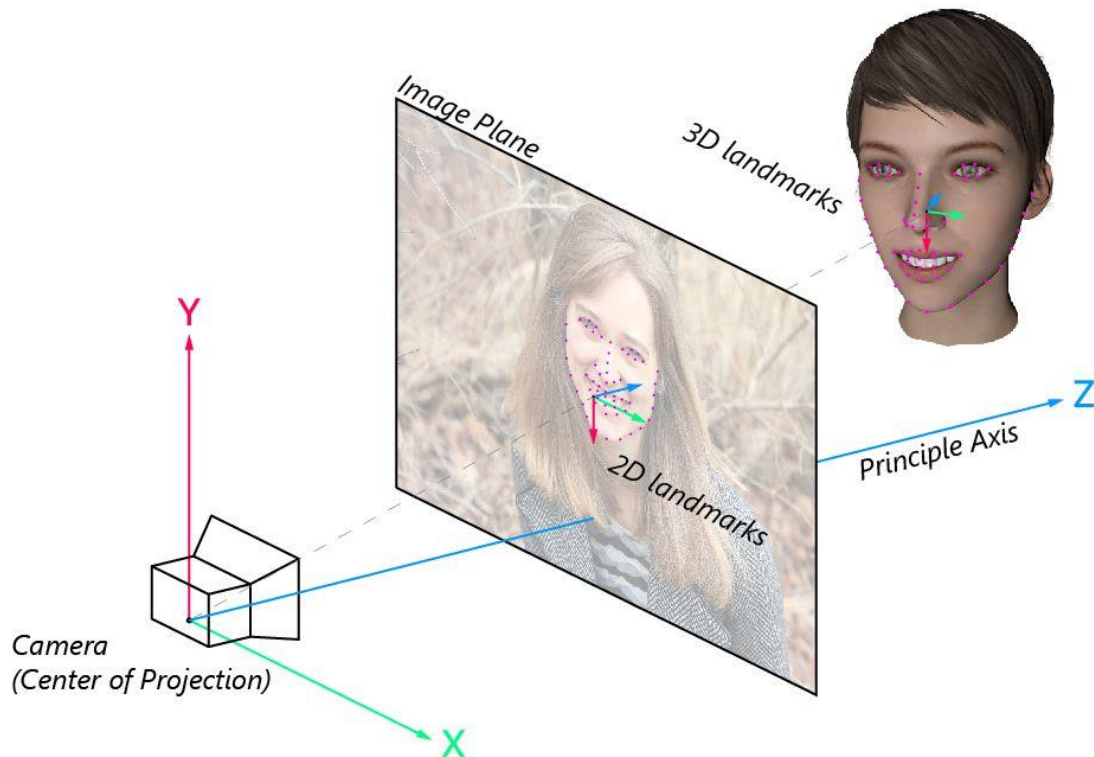

The origin is the center of projection (center of camera).

When facial landmark points are obtained in 3 dimensions (3D), the 3D normalized face shape can be created. This shape is related to head's 3D position. Rotation and translation between camera coordinate system and 3D head coordinate system makes it possible to obtain the face shape. With functions *UnityGetTransform()* and *UnityCalibration(),* it is possible to calibrate these two coordinate systems.

For a detailed description, click on this link.

3) Does ULSee deploy other frameworks to load 3D models for facial landmarks?

A. ULSee uses its proprietary algorithm to load 3D models for facial landmark determination.

4) Which version of the Android SDK is to be used to help with plugin integration into custom project?

A. It is advisable to use Android SDKs with minimum API Level 16 (Android 4.1 'Jelly Bean')

5) How can custom textures be added to be used with the plugin?

A. With reference to the screenshot provided below:
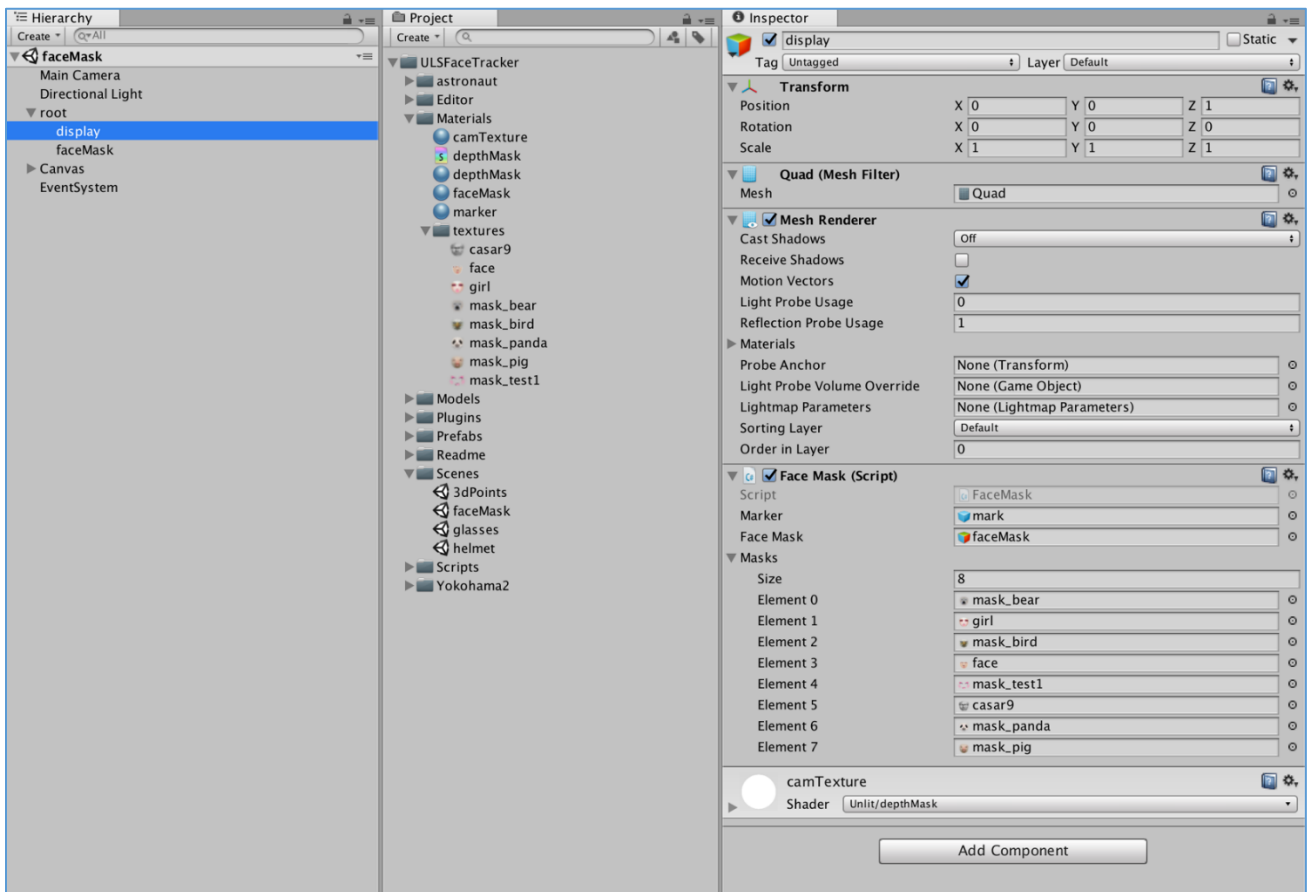
Add your textures to the *textures* folder and *Add Component*(s) with *Inspector*.

Notes for Further Reference

1) For mobile devices, the plugin uses the face detection provided with hardware on-device

2) Forehead and ear points are not available

3) Face Tracker is calibrated for camera position on top of device screen

4) Rotation angles of camera cannot be obtained on the Windows OS platform. To use camera in a vertical position, developers must adjust the rotation of camera preview texture in CameraDispatch.cs.

5) FaceMask and Object3D demos update mesh vertices with tracking points. The mesh can be replaced if tracking points are mapped to mesh vertices. Developers must update or calculate extra vertices.

6) From FaceMask.cs and Object3D scripts, developers can obtain UV coordinates from _mesh.uv. Developers may check the Unity manual available at location https://docs.unity3d.com/ScriptReference/Mesh.html

7) The 2D and 3D objects provided in the Unity plugin are examples for references to understand what types of objects are supported for applying on the faces for AR effects. The textures we have used are in PNG format.

8) There is no real 3D model in the source code for 3D mapping. The model at the source is a 2D surface mesh, which is applied on the face based on the face shape and 3D face tracking points' estimation from 2D face tracking points. The face mesh stretches out to fit the face regions covered by the face tracking points.