XZIMG Ltd

Magic Face 2021

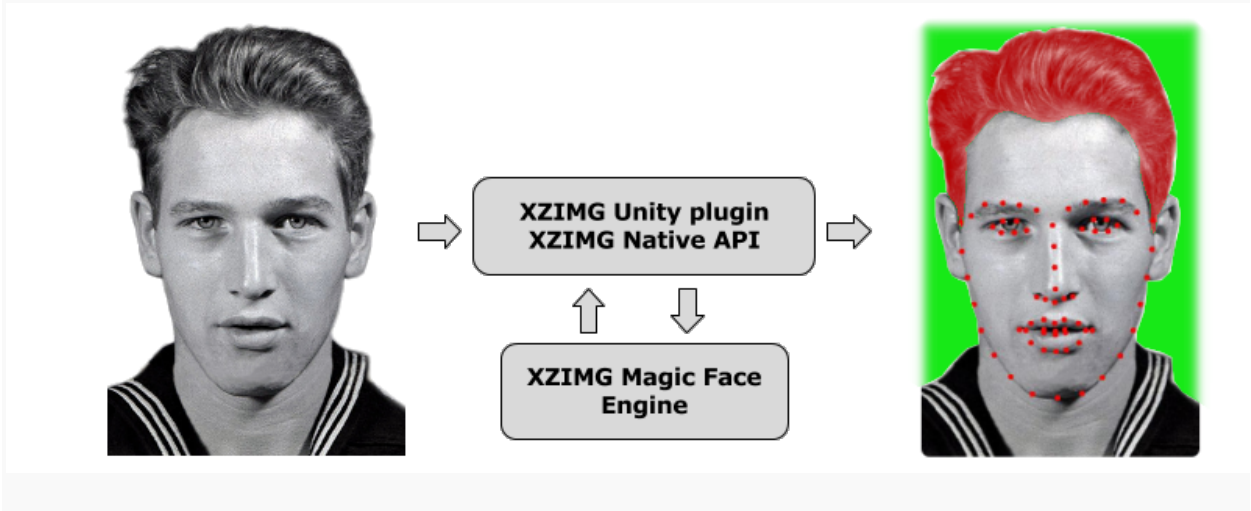
Documentation

Product:	**Magic Face**
Version:	**2021**
Creation Date:	**2020/05/01**
Revision Date	**2021/01/06**

Table of Contents

Overview

Magic Face, developed by XZIMG Limited, is a Software Development Kit (SDK) Solution which provides cross-platform and real-time facial features detection and tracking in the 3D Space using images captured from a video stream. Facial features can be detected in the image plane (2D), or in the 3D space using a projective camera model and a deformable face model. Efficient deep learning algorithms also offers state-of-the-art segmentation functionalities such as hair and background detection. Magic Face efficiency and robustness constitute an ideal solution for designing face-replacement and make-up Augmented Reality based applications.



Magic face 2.0, 2020, 2021 (and above) APIs offers:

- A real-time face feature tracker,
- A real-time mouth refiner (for iOS and OSX)
- A real-time segmentation engine which can segment hair and separate the background,
- A real-time emotion detector to detect standard emotions such as happiness, surprise, etc.
- A real-time eye tracking providing pupils coordinates in 2D and 3D.

Above deep learning / AI functionalities are available in real-time for both Android and iOS. Provided packages are structured as a Unity plugin (native libraries, scripts and materials) or sa a native SDK for dedicated platforms (Android Studio Project, XCode Projects, Javascript builder and samples). In each case a sample is provided to ease our client and partner integration phase.

Magic Face solution can export experiences on several platforms. Currently supported platforms are Windows, OSX, Android, iOS, and WebGL/HTML5 (using a separated builder/sample based on WebGL and Three.js). Some part of our code is based on proprietary AI components and operators while others are based on robust and efficient Computer Vision

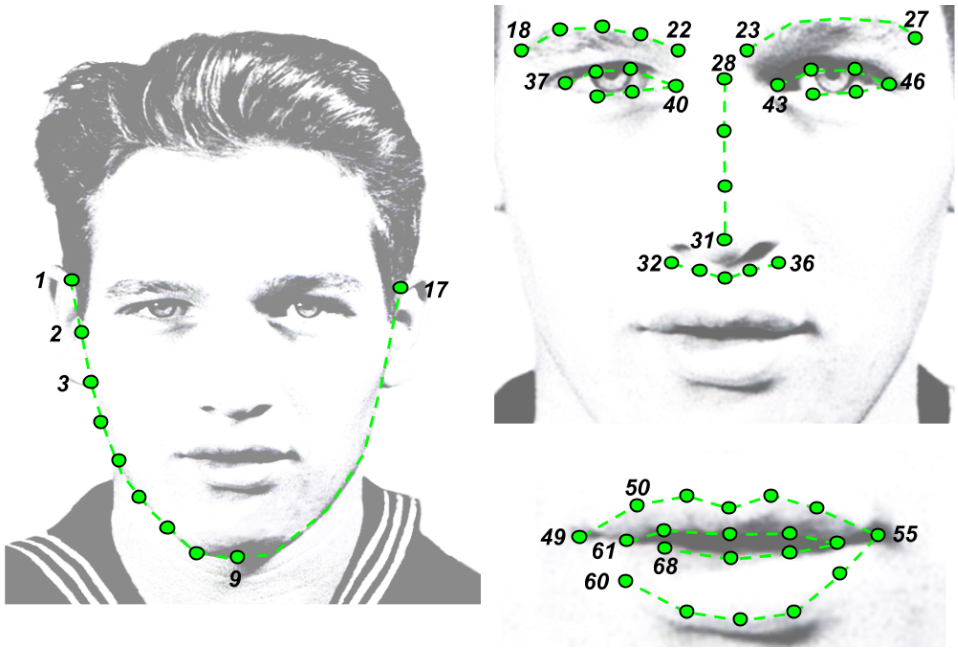
algorithms.

Trial version

If you are using the trial version of XZIMG Magic Face solution, a **watermark** will be displayed in the front of the rendering view. **Moreover**, the experience will stop automatically after a predefined period of time.

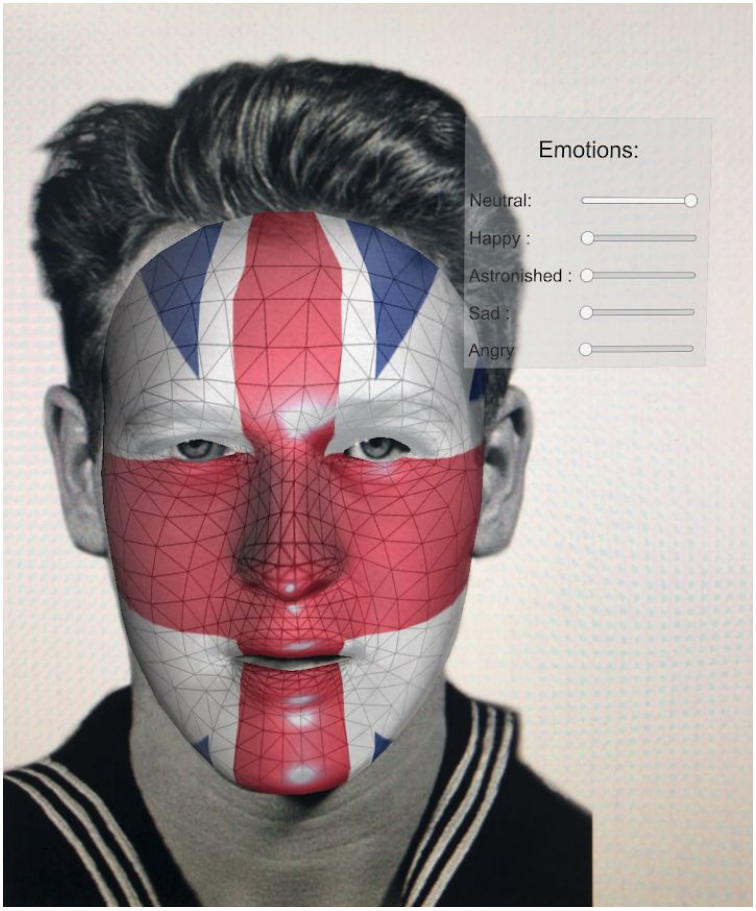
Face Features

One main objective of XZIMG Magic Face solution is to provide facial features locations in real-time. The face model is constructed based on **68 key landmarks** located inside the face and on its contours as illustrated bellow.



The 68 face features model (image coordinates).
This model includes face feature on the contour of the face.

The 68 face features controls a dense model which in turn is used as a support for mapping a texture. Face models are available in the `./Runtime/Resources/XZIMG/Models/` directory as *face-model.obj* and should contain **exactly 746 vertices** when imported in Unity. In case one needs to modify the scenario and add new effects on top of the face, the model has to be edited in a 3D editing tool (like blender, see explanation below). Face model textures and texture coordinates can be modified while leaving the vertices orders and faces (triangles) unmodified.



The 746 vertices model which includes the forehead.
Emotions values are displayed on the side.

Body Parts Segmentation

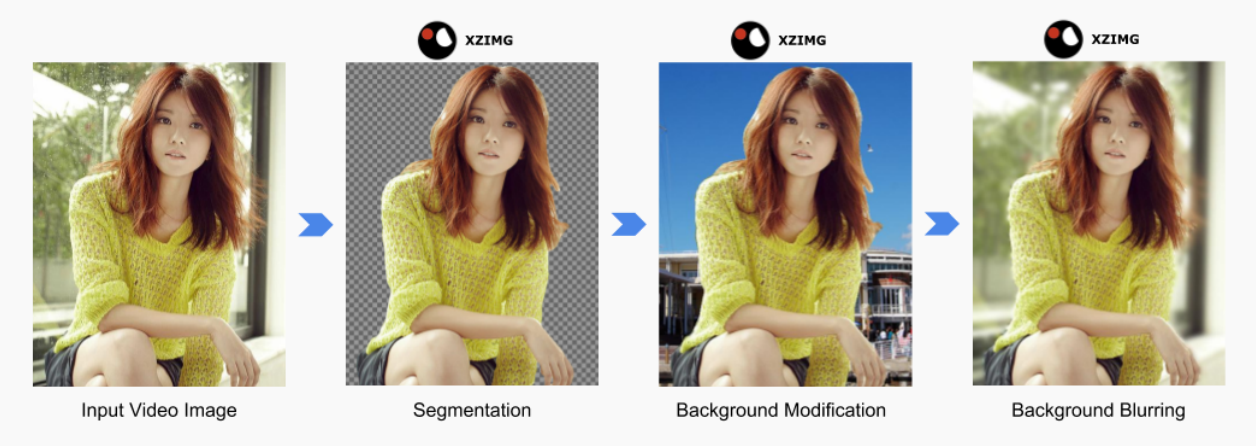
Magic Face 2.0 contains a segmentation engine which is based on recent deep learning developments. We are relying on powerful optimized RCNN (Recurrent Convolutional Neural Networks) to achieve real-time segmentation of the scene. We provide segmentation for hair and body (other body parts will be released progressively).

The segmentation is computed when calling the `xzimgMagicSegmentationProcess` API function. Resulting segmented mask is obtained by calling another API function named: `xzimgMagicSegmentationGetSegmentationImage`.

In Unity, several shaders are included in the `Asset/Resources/Shader/` folder. `VideoShaderHairDying.shader` is dedicate to dye the user hairs with preselected colors while `VideoShaderBodySegmentation.shader` will remove the background and replace it by a

transparent layer image. Note that the segmentation process is a computationally demanding task which relies extensively on CPU and/or GPU. As a consequence, the API won't deliver real-time results for older devices.

When using Native Magic Face SDK, you will have to build your own shaders on top of available API functions (you can use the shader provided in the Unity projects and modify them for the corresponding platforms).



Emotions Detection

XZIMG Magic Face 2.0 provides an emotion detection engine. This process can be activated by selecting the mode in the inspector window in Unity. This process is based on a CNN (Convolutional Neural Network) and has been optimized to work on Mobile phones with limited computations. Apart from neutral emotion, it provides several expressions such as happiness, angriness, sadness, surprise.

Unity Plugin

Unity Plugin Package

The Unity plugin consists of several libraries (organized per-platform), models (classifiers and 3D models to detect faces, segment body and hair, etc.), scripts to communicate between Unity and the XZIMG Magic Face API and shaders to render video images and deformable objects.

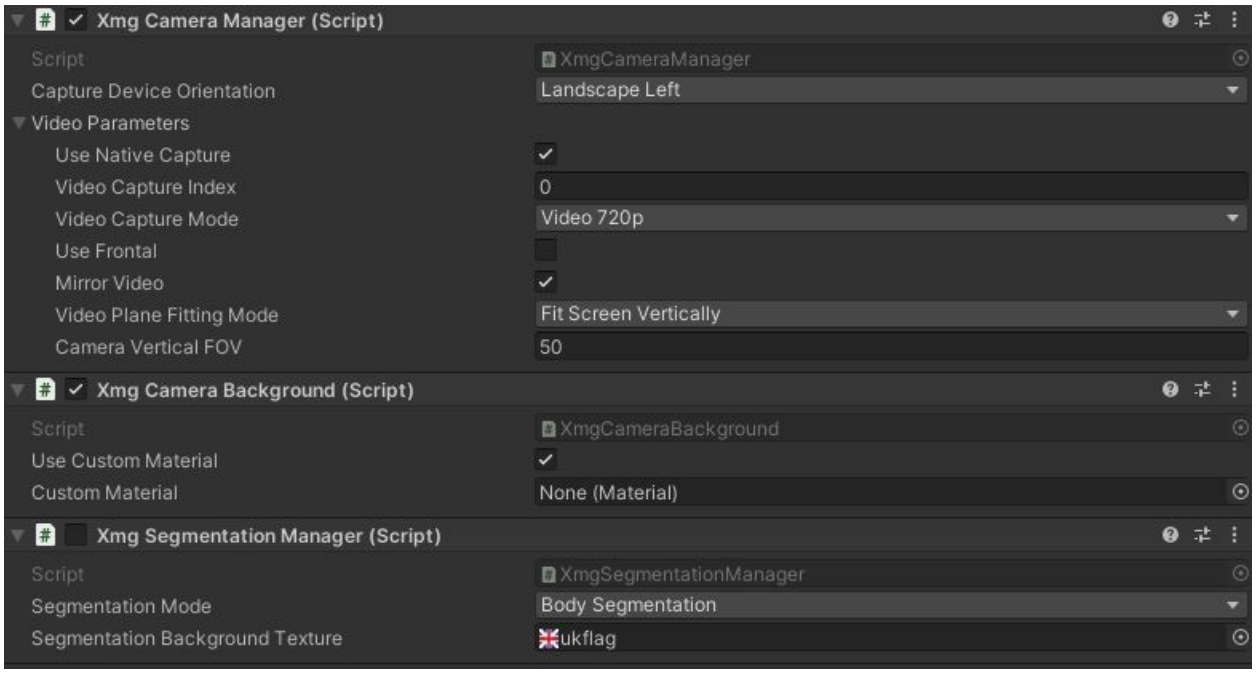
The Unity plugin contains a sample (in `./Assets/Sample`) which is ready to launch for early testing. One can modify and increment this sample to understand underlying functionalities. The main scene - *Sample-MagicFace3D.unity* - is provided in the `./Assets/Sample/Scene/` directory.

In the main Unity folder `./Assets/XZIMG-MagicFace` are stored:

- the native libraries (`Plugins/`),
- the scripts (`./Runtime/Scripts`),
- the Shaders (`./Runtime/Resources/XZIMG/Shaders/`),
- and the models (`./Runtime/Resources/XZIMG/Models`)

Tuning Parameters in Unity

Several parameters are exposed to let the user change the tracker properties through Unity's inspector sections. We first consider the parameters related to the camera and what happens to the video feed (shaders, segmentations).



Camera Manager Section

Capture Device Orientation: Indicates the capture device physical orientation in case you are using a real camera that is positioned in portrait mode (parameter only available on Desktop platforms).

Use Native Capture: Indicates if the application uses Unity webcamTexture class or the internal native video capture module provided by XZIMG (available on Mobiles).

Video Capture Index: Select a camera from its index (for desktop PC).

Video Capture Mode: Camera resolution mode (VGA-640x480, HD-1280x720, HD-1920x1080).

Use Frontal: Use the frontal camera (change only for Mobiles).

Mirror Video: Mirror the video display (object positions and orientations are automatically transformed accordingly).

Video Plane Fitting Mode: In case screen aspect ratio and video aspect ratio are different, you can choose to fit the video plane horizontally or vertically.

Camera Vertical FOV: Adjust the camera vertical field of view of the camera you use. Default value is 45 degrees.

Camera Background Section

Use Custom Material: To specify a custom material for the background rendering

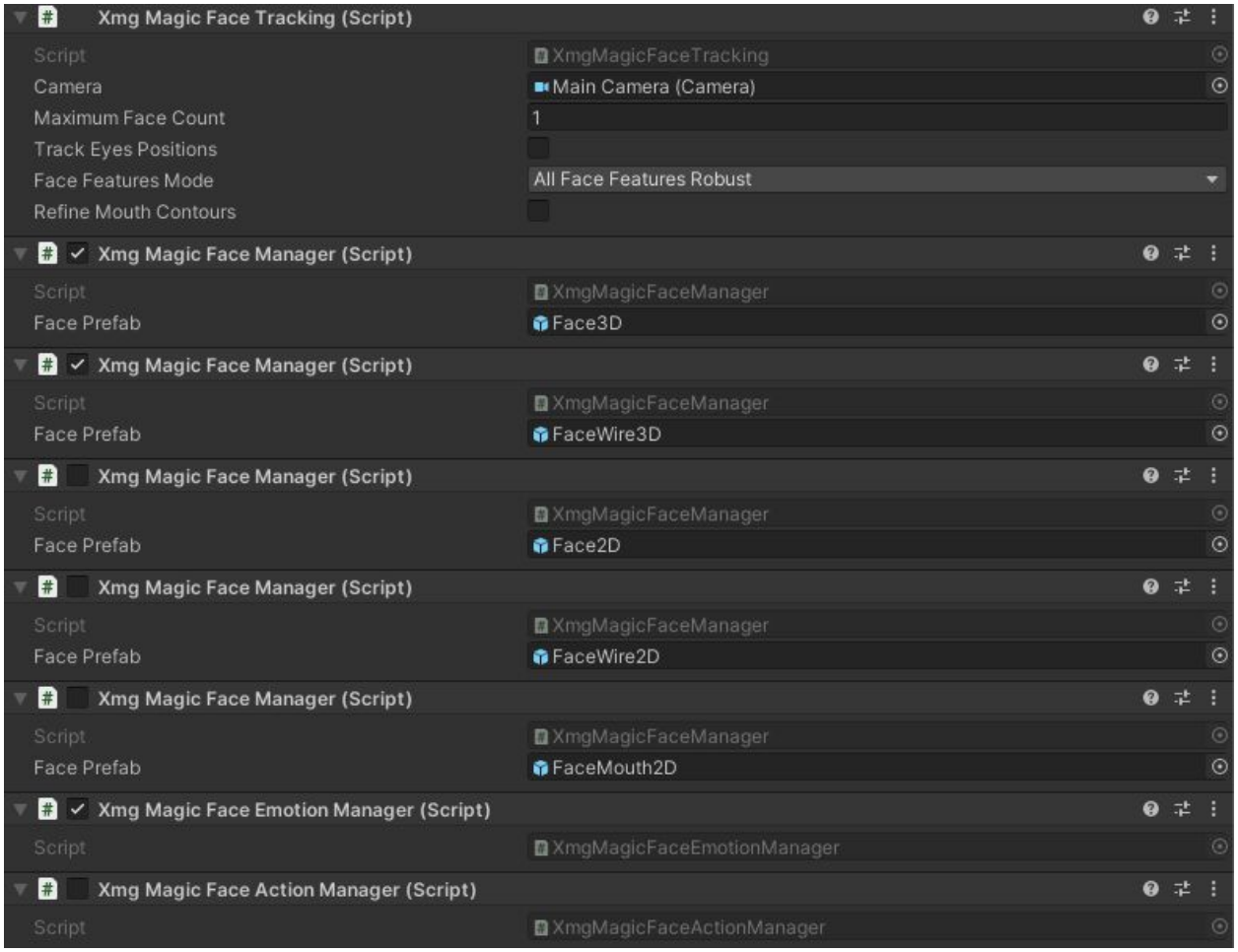
Camera Background Section

Segmentation Mode: If activated, the type of segmentation can be selected (hair, body, body robust, ...).

Segmentation Background Texture: The texture to be used in the background while segmenting.

Face Tracking, when activated, can be used to detect and add several effects on the face like eye positions, emotions, models rendered on the face… You can add several behaviour using

scripts. There exists several examples you can use and adapt to your use-cases.



Maximum Face Count: Maximum number of faces to be detected and tracked simultaneously.

Track Eye Positions: Engine will detect eyes position in 2D and 3D. Note: This feature won't work when `All Face Features Robust` mode is activated.

Face Features Mode: you can choose: `All Face Feature` (default) or `All Face Features Robust` (best results on ios, but requires an iphoneX or above)

Refine Mouth Contours: Highly accurate mouth tracking (available on ios, soon on Android)

Other scripts can be activated to load **prefabs** to:

- Render a textured 3D face (Face3D prefab),

- Render a wireframe on a 3D face (FaceWire3D prefab),
- Render a 2D face (Face2D, FaceWire2D),
- Render the mouth (FaceMouth2D),
- Render emotions using the script `xmgRenderFaceEmotionManager`.

Models

The unity plugins includes several models for deep-learning. They are located in the `Runtime/Resources/XZIMG/Models` folder. If you want to trim unused models, you will find a `README.txt` file in the folder to indicate which model is used depending on the functionality and the platform.

Platforms

MS Windows

When using XZIMG Magic Face on MS Windows, you might have issues when running the application (in the Editor or as a MS Windows application). If this is the case, please try to install the Microsoft Visual Redists. You can also verify that the plugin dynamic libraries exist (xzimgMagicFace.dll files in the `./Asset/Plugins/directory`)

Mobiles

With mobiles (Android / iOS), video capture class has been redefined internally for performance reasons. As a consequence, users should carefully set Use Frontal and Mirror Video parameters.

Android

Android version of the plugin is composed with a .jar and a .so files which aims to provide the videocapture and the tracking functionalities.

- The native Android plugin is stored in the `./plugins/Android/libs/arm64-v8a` directory.
- The videocapture.jar is stored in the `./plugins/Android/` directory.

iOS

iOS version consists in a static .a library that is linked with Unity using XCode. This process is done automatically when building a scene.

WebGL/HTML5

HTML5 experiences are not available with XZIMG Unity plugin. Instead, we have introduced a native player which relies on three js and WebGL. The package can be purchase separately, and is available for trial in the download section: www.xzimg.com/Downloads

How to Change Video Source

To replay videos, to support specific capture devices, …etc. you might want to use specific video frame as input to the face tracking engine. On desktop, default video capture is provided by Unity class WebCamTexture while on iOS and Android, we use a native implementation of the video capture which is hidden inside our plugin.

To Change the video source, you can follow the following steps:

- Check that the video capture is provided by the Unity layer and not natively, see the plugin parameters below:
- With the non native video capture mode activated, we can use the following function to launch a face detection/tracking on a given image:

```
private xmgMagicFaceBridge.xmgImage m_image;
xmgMagicFaceBridge.xzimgMagicFaceTrackNonRigidFaces(ref m_image,...)
```

- You can prepare the input image using the provided prepareImage() function, where the pixel handle describe the memory placement of the image:

```
xmgMagicFaceBridge.PrepareImage(ref m_image, captureWidth, captureHeight, 4,
m_myWebCamEngine.m_PixelsHandle.AddrOfPinnedObject());
```

Note: you have to input an image that has the same size as the size declared at plugin initialization: *xmgInitParams.m_processingWidth*, and *xmgInitParams.m_processingHeight*)

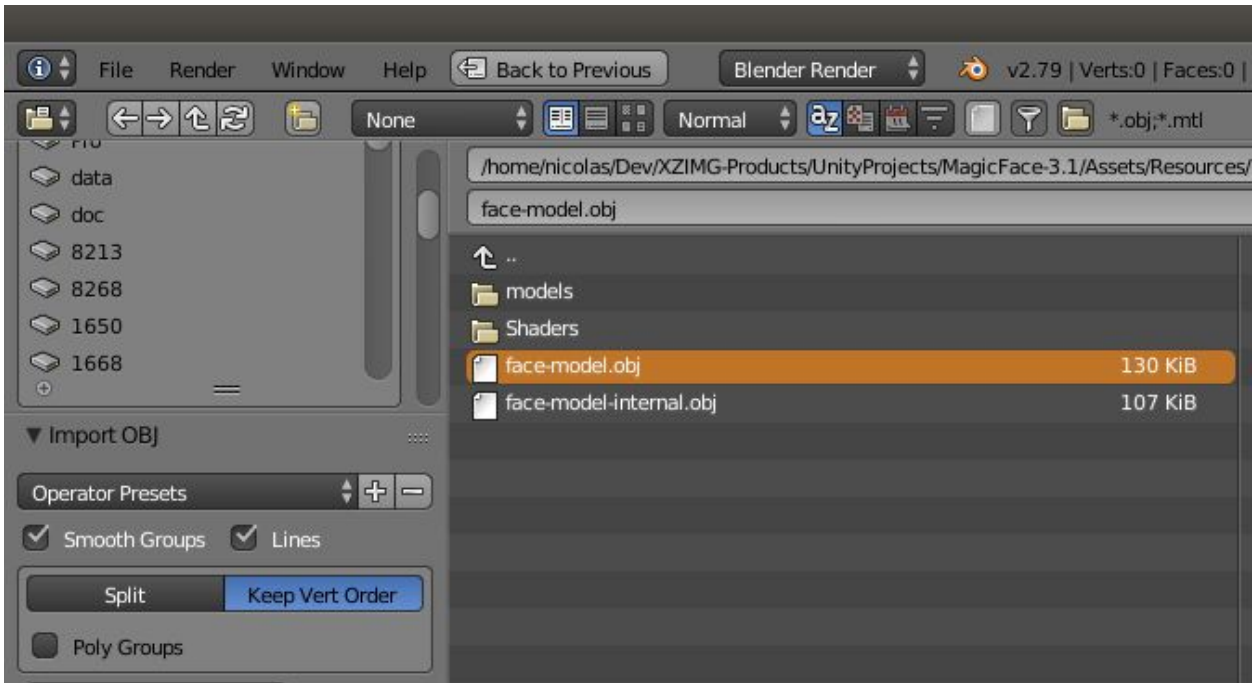
Plugin API functions

API function are detailed and documented in the `Script/Bribge/xmgMagicFaceBridge.cs` file.

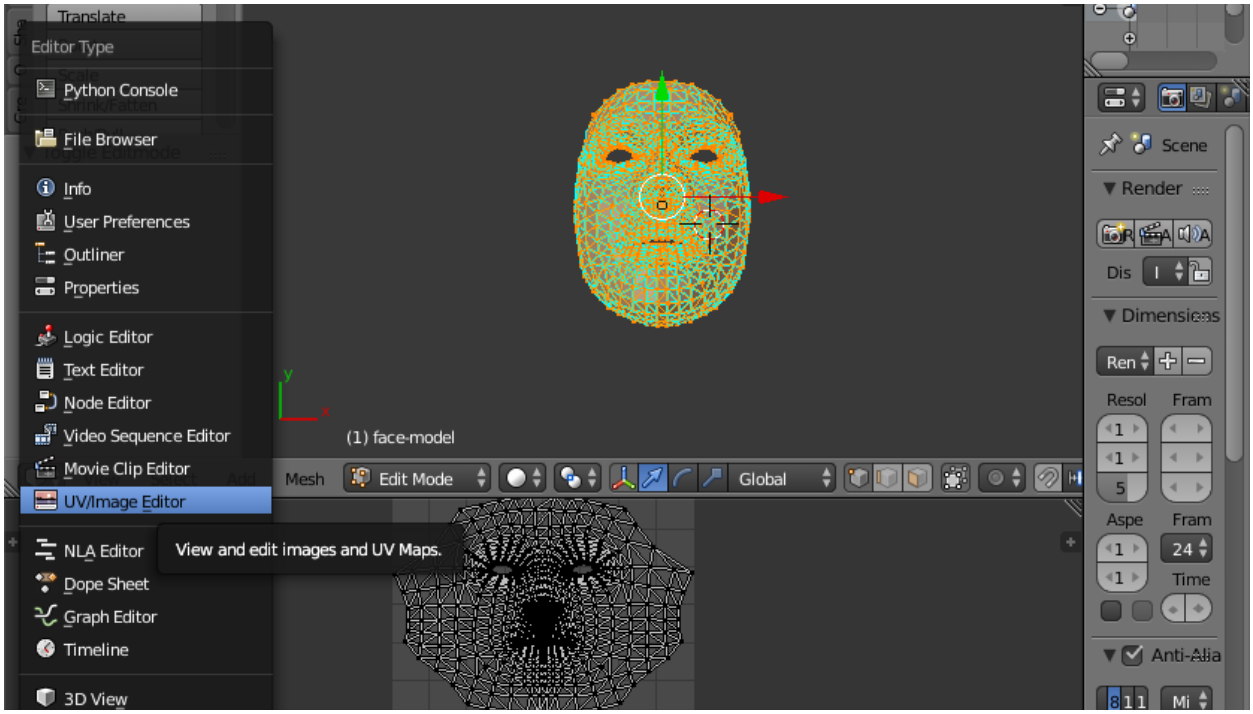
Modifying Face Texture

To modify provided face textures (eg. frenchflag.png, ukflag.png, ...etc) it is possible to redefine texture face object coordinates on top of a new selected texture image. Texture coordinates (uvs) are included in the *face-model.obj* geometry.

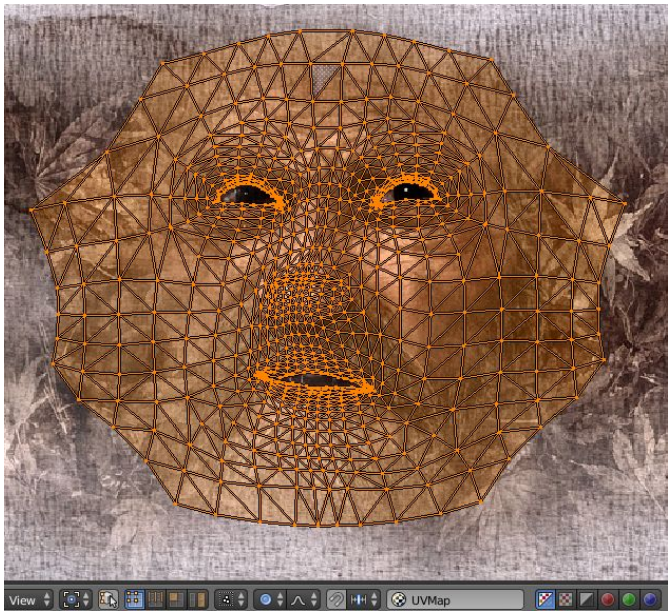
A. Import `face-model.obj` with your favorite 3D modeling tool (we are using blender). Within blender import menu, **keep vertices order** and choose **Y forward**, **Z Up** as illustrated bellow.



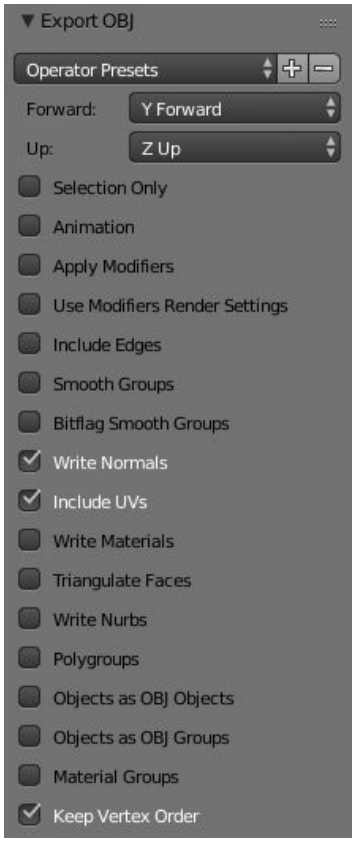
B. In blender, select the mesh and open the UV/Image editor. Unrolled UVs are then visible in the UV editor.

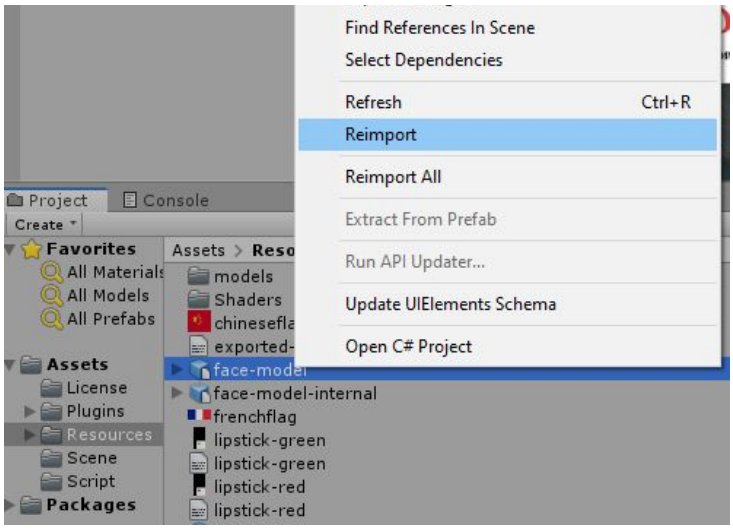


C. In blender, align UV coordinates to the selected texture image

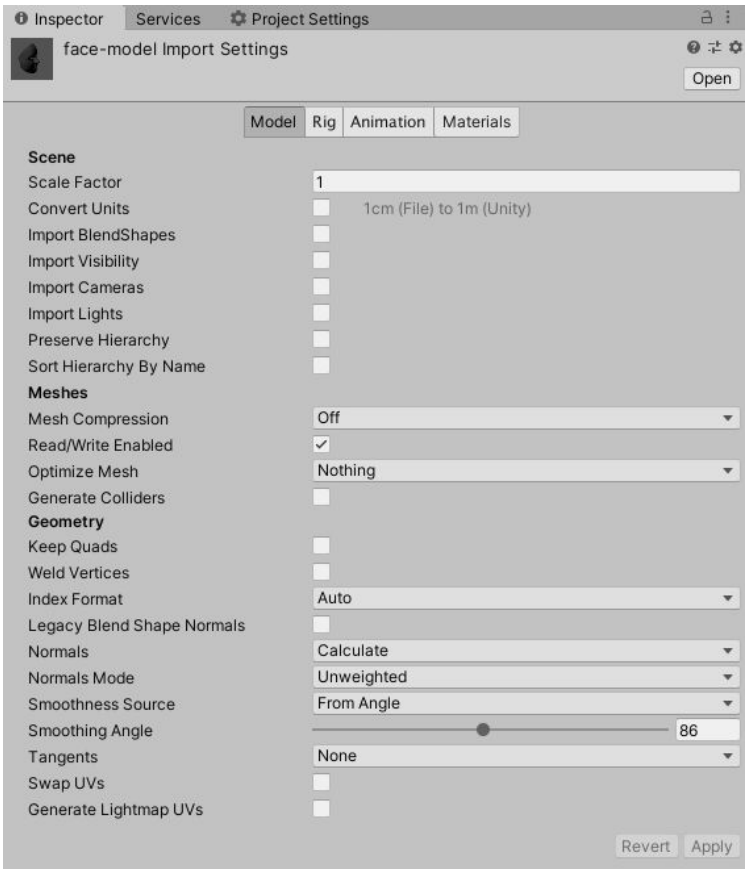


D. In blender, export the modified object as a .obj using the following export parameters



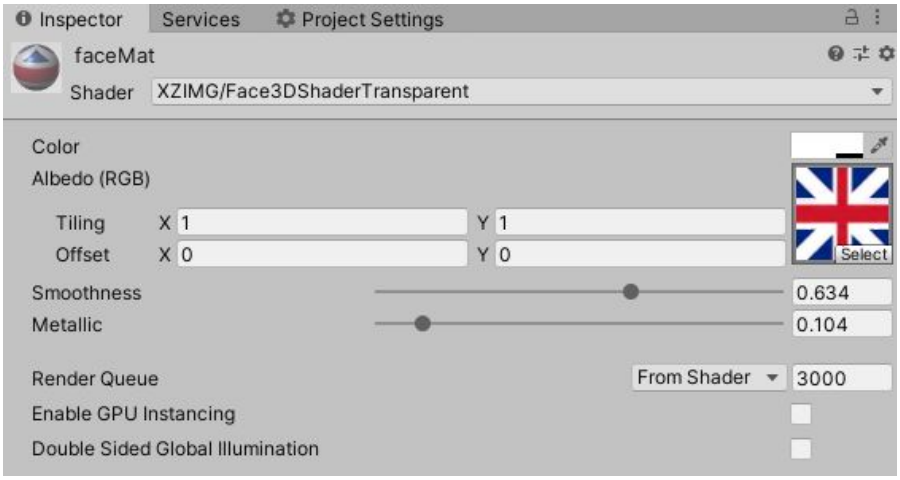


E. In Unity**,** copy and re-import the modified face-model.obj in the `./Runtime/Resources/XZIMG/Models/` directory.



Import settings for the .obj face models -
allowing to keep the same number of vertices when imported in Unity

F. In Unity, modify the corresponding material for the current face (default is faceMat) by setting the new texture image (uk flag bellow).

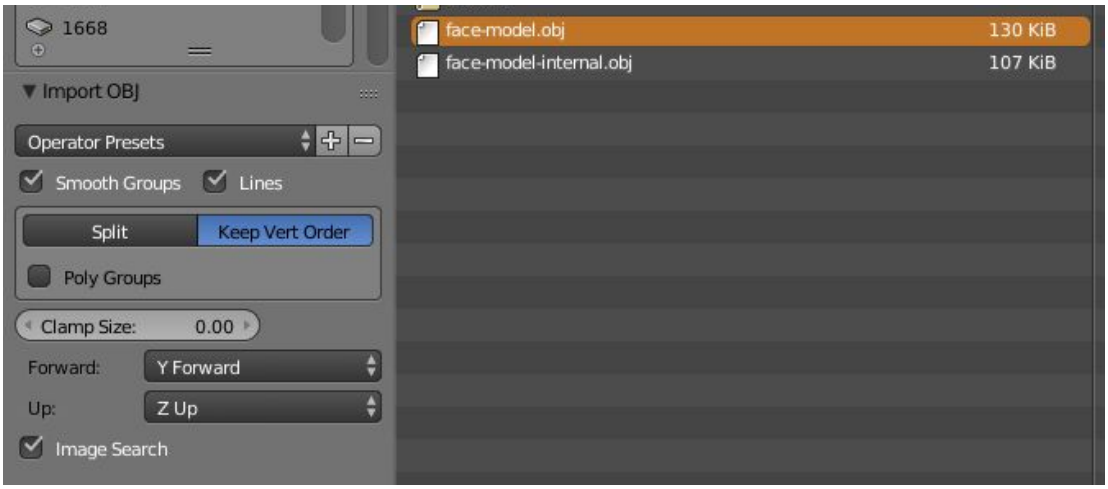


Rigging a Deformable Mesh

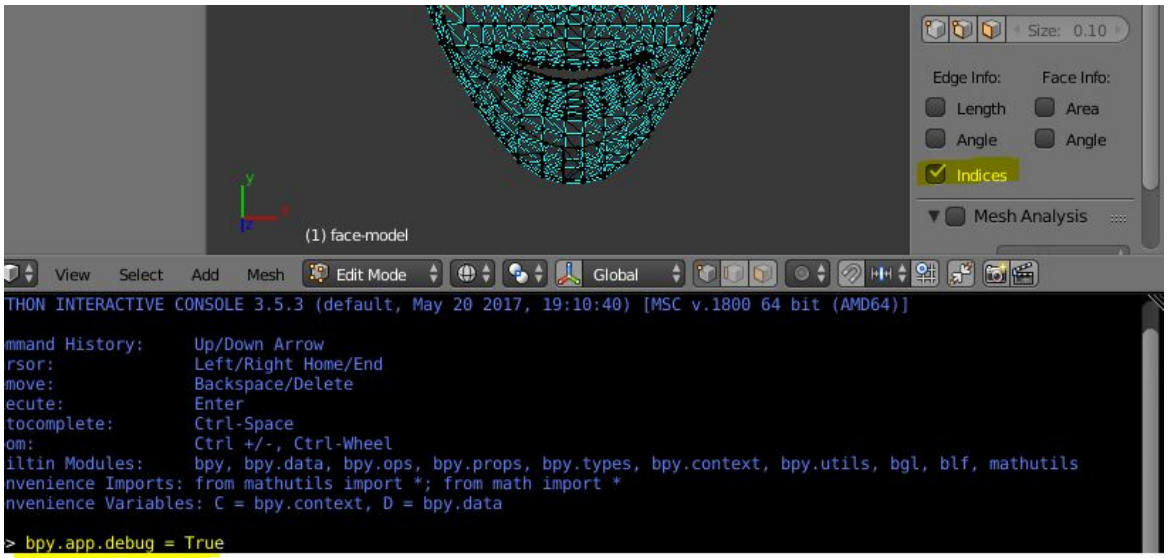
If you have your own deformable model and you need to rig it to existing face model. Follow steps below.

A. Identify the model vertices you want to use for the rigging: open the face-object in your favorite 3D modeling tool (we are using blender) and display the mesh vertex indices.

1. (in blender) Import the face-model.obj with options: **keep vertices order** and choose **Y forward**, **Z Up** as illustrated.



2. (in blender) activate then display vertices index using the terminal command: `bpy.app.debug = True` and activate the “indice” checkbox



You can have access to the face-model indices.

B. (in Unity) Once the indices are available, you can select those of interest to animate your own rigged deformable model.

1. In Unity scripts you can access to the current coordinates of the model vertices using the API structure `xmgMagicFaceObject::m_dataLandmarks3D` (ordered as floats `[x0, y0, z0, ... x729, y729, z729]`. An example of is given in the *xmgMagicFace3D.cs* file:

```
// -- Read and convert 3D vertices
int nbFaceFeatures = m_nonRigidData[o].m_faceData.m_nbLandmarks3D;
Vector3[] vertices3D = new Vector3[730];
for (int i = 0; i < nbFaceFeatures; i++)
{
    vertices3D[i].x = m_nonRigidData[o].m_dataLandmarks3D[3 * i];

    // mirror
    if (m_videoParameters.MirrorVideo)
        vertices3D[i].x = -vertices3D[i].x;
        // left handed
        vertices3D[i].y = -m_nonRigidData[o].m_dataLandmarks3D[3 * i + 1];
        vertices3D[i].z = m_nonRigidData[o].m_dataLandmarks3D[3 * i + 2];
}
```

2. Or you can use the pivot object geometry which has been updated with 3D vertices coordinates. They are accessible through the script as **m_renderedFaceObjects**.

```
if (m_renderedFaceObjects.Count > o && m_renderedFaceObjects[o].m_renderPivot)
{
    Mesh msh = m_renderedFaceObjects[o].m_renderPivot.GetComponent<MeshFilter>().mesh;
    msh.vertices = vertices3D;
...
```

Troubleshootings

MAC OS X Bundle Loading Error

If you can't load the .bundle library on Mac Os and encounter Error messages: "bundle corrupted, move to bin …". Please type the following command in the terminal (before launching the Unity project): **xattr -d com.apple.quarantine xzimgMagicFace.bundle**

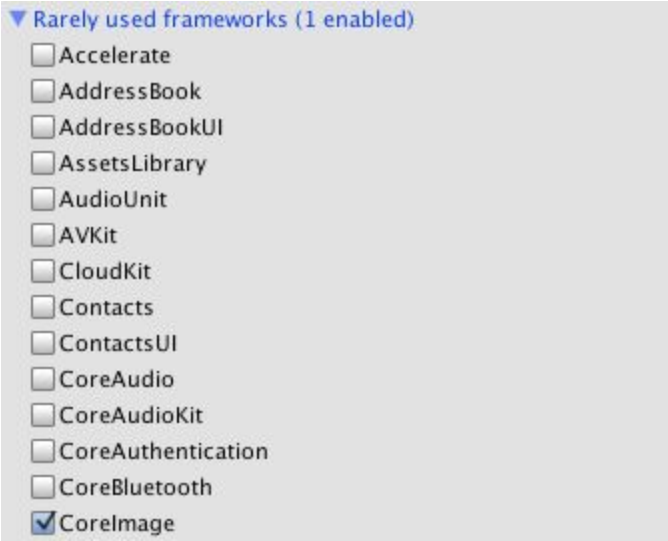
Note: We will soon notarize the internal library to simplify this process.

XCode Errors

If following error occurs in XCode

```
undef: _OBJC_CLASS_$_CIContext
Undefined symbols for architecture armv7:
  "_OBJC_CLASS_$_CIContext", referenced from:
      objc-class-ref in lto.o
ld: symbol(s) not found for architecture armv7
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Assert that CoreImage package is added when building the application with XCode. You can add this package once for all in Unity by selecting the iOS library and setting dependencies as illustrated.



Video Stream is upside down

On certain devices (identified are Nexus 6 series), the video is captured upside-down by the hardware layer of the capture. As a consequence, displayed video is inverted. To improve on this issue, we have added the parameter <m_flippedHorizontaly to the xmgImage class, you can set this parameter to true whenever it's required.

Mobile App. is too slow

Video capture module can be slow on some low end device because it relies on hardware implementation. We suggest that you reduce video capture resolution to VGA (640x480) on older devices to keep a decent frame rate.

Video Stream is blurred

If rendered video seems blurry, it's possible that the requested video resolution is not available for the phone you are using. In that case you can try to switch to different modes. For example, you can try to set the videoCaptureMode to 2 to get HD-720p resolution. For example, on a Google Pixel 2 it is recommended to use HD video format as VGA mode doesn't work properly.

macOS video capture

With macOS, it is not always easy to get the video capture working. We have done our best to deal with most camera within Unity, but in some cases, you might experience some issues. If that's the case, try to switch resolution modes with the videoCaptureMode parameter and check the **xmgVideoCaptureParameters.cs** script to verify that the video capture resolution is correct.

Exported experience is not working on windows

Please ensure that the Microsoft redistributable packages for Visual Studio 2017 are properly installed. If not, you can download them from Microsoft's website.

Native SDK and Samples

The native SDK and samples are available **for advanced users**. It allows to develop specific application within your favorite development frameworks, avoiding Unity Editor. The natives SDK and samples are available for iOS, Android and Javascript, and can take advantage of modern rendering capabilities.

These native samples are using native SDK libraries offering aforementioned face detection functionalities. The Android native SDK consists in a .so library file containing jni and C like exported API functions. The iOS native SDK contains a .a library that export API functions.

SDK API definition

For iOS, Android or HTML5, the SDK have a similar API. Functions are described below (or refer to the SDK main header file **xmgMagicFaceAPI.h**).

```
/// Initialize the detection engine
/// @params initParams contains initialization parameters
int ICV_DLL_SIGNATURE xzimgMagicFaceInitialize(
    xzimgSdk::xmgFaceTrackingInitParams &initParams);

/// Release the detection engine
void ICV_DLL_SIGNATURE xzimgMagicFaceRelease();

/// Pauses current tracking
/// @param pause: 1 to pause 0 to start
void ICV_DLL_SIGNATURE xzimgMagicFacePause(int pause);

/// Reset tracking engine to initialisation state
void ICV_DLL_SIGNATURE xzimgMagicFaceReset();

/// Detect Faces and find non rigid features
/// @param image input image
/// @param robustDetection robust detection mode (1 to activate)
/// @param detectionOrientation is from [0..3] describing steps from 0 to 3PI/2
anticlockwise (depends on screen orientation)
int ICV_DLL_SIGNATURE xzimgMagicFaceDetectNonRigidFaces2D(
    const xzimgSdk::expImage &image,
    int robustDetection,
    int detectionOrientation);
```

```
/// Detect/Track Face and find non rigid features in the image place
/// @param image input image
/// @param detectionOrientation is from [0..3] describing steps from 0 to 3PI/2
anticlockwise (depends on screen orientation)
int ICV_DLL_SIGNATURE xzimgMagicFaceTrackNonRigidFaces(
    const xzimgSdk::expImage &image,
    int detectionOrientation);

/// Get the current states of previously detected face objects
/// @param idxObject index of the face object
/// @param oFacesData returned data for the selected face
int ICV_DLL_SIGNATURE xzimgMagicFaceGetFaceData(
    int idxObject,
    xzimgSdk::xmgNonRigidPoseAPI &oFacesData);

/// Change face detection sensibility
/// @params level 0 .. 3 (0 is fast, 2 is slow but allows to find smaller faces)
int ICV_DLL_SIGNATURE xzimgSetFaceDetectionSensibility(int level);

/// Change the filter strength
/// @params level 0.. 10 (0 is very few filter, 5 is standard values, 10 is filter to its
maximum which will introduce delays i nthe rendering)
int ICV_DLL_SIGNATURE xzimgSetFilteringLevel(int level);

///Returns the triangles indices of the dense model
/// @param
int ICV_DLL_SIGNATURE xzimgGetTriangles(int *oTriangles, int *oNbTriangles);

///    Returns the uvs coordinates of the dense model
/// @param oUVs
/// @param oNbUVs
int ICV_DLL_SIGNATURE xzimgGetUVs(float *oUVs, int *oNbUVs);

/// Initialize the emotion engine (in addition of the face detector)
int ICV_DLL_SIGNATURE xzimgMagicFaceInitializeEmotion(
    xzimgSdk::xmgCNNInitParamsAPI &initParams);

/// Process current image and detect expression of detected faces
/// @return 0=neutral, 1=anger, 2=contempt, 3=disgust, 4=fear, 5=happy, 6=sadness,
7=surprise
int ICV_DLL_SIGNATURE xzimgMagicFaceProcessEmotion(
    int idxFace,
    float filter);

/// Number of available action units available
int ICV_DLL_SIGNATURE xzimgMagicFaceGetNumActionUnits();

/// Get action units values
int ICV_DLL_SIGNATURE xzimgMagicFaceGetActionUnits(int idx_face, float *actionUnits);
```

Data structures are useful to initialize the face detection engine and to get results from the detection iterations. The are described below (or in the file xmgIncludeAPI.h).

```
namespace xzimgSdk
{

/// Exposed image class
class expImage
{
public:
        expImage()
        {
         // -- Default
                m_iWidth = 640;
                m_iWStep = 0;
                m_iHeight = 480;
                m_pImageData = nullptr;
                m_colorType = 4;
                m_type = 0;
                m_flippedHorizontaly = false;
        }

        /// Image Width
        int  m_iWidth;
        /// Image Height
        int  m_iHeight;
        /// Image Data
        void *m_pImageData;
        /// Image Width Step - Set to 0 for automatic calculation
        int  m_iWStep;
        /// XMG_BW = 0, XMG_RGB, XMG_BGR, XMG_YUV, XMG_RGBA, XMG_BGRA, XMG_ARGB
        int m_colorType;
        /// 0: unsigned char, 1: float, 2: double
        int m_type;
        /// Has the image to be flipped horizontally
        int m_flippedHorizontaly;
};


/// Specific video Capture when using openCV vidcap engine
class xmgInternalVideoCaptureOptions
{
    Public:
        /// 0 is 320x240; 1, is 640x480; 2 is 720p (-1 if no internal capture)
        int m_resolutionMode;
        /// 0 is frontal; 1 is back
        int m_frontal;
        /// 0 auto-focus now; 1 auto-focus continually; 2 locked; 3; focus to point
        int m_focusMode;
        /// 0 auto-focus now; 1 auto-focus continually; 2 locked; 3; focus to point
        int m_exposureMode;
        /// 0 auto-focus now; 1 auto-focus continually; 2 locked; 3; focus to point
        int m_whileBalanceMode;

        xmgInternalVideoCaptureOptions()
        {
                int m_resolutionMode = 1;
                int m_frontal = 1;
                int m_focusMode = 0;
                int m_exposureMode = 0;
```

```
                int m_whileBalanceMode = 0;
        }
};

/// Face Tracking initialization parameters
class xmgFaceTrackingInitParams
{
public:
        xmgFaceTrackingInitParams() {
                m_3DFacialFeatures = 0;
                m_processingWidth = 640;
                m_processingHeight = 480;
                m_nbFacialFeatures = 68;
                m_nbMaxFaceObjects = 1;
                m_fovVerticalDegree = 50.0f;
                m_detectEyePupils = 0;
                m_detectExpressions = 0;
        }
        int m_3DFacialFeatures;
        int m_processingWidth;
        int m_processingHeight;
        int m_nbFacialFeatures;
        int m_nbMaxFaceObjects;
        float m_fovVerticalDegree;
        int m_detectEyePupils;
        int m_detectExpressions;

        /// Pointer to the classifiers data
        const unsigned char *m_faceModelContent;

};

/// CNN parameters initialization parameters
class xmgCNNInitParamsAPI
{
public:
        xmgCNNInitParamsAPI()
        {
                /// Default
                m_inWidth = 128;
                m_inHeight = 176;
                m_in_channels = 3;
                m_compressionMode = 1;
                m_netType = 0;
                m_quantized = 0;
                m_num_threads = -1;
                m_normalizationMode = 0;
                m_rcnnWeight = 0.6;

        }

        /// Network input size
        int m_inWidth;
        /// Network input size
        int m_inHeight;
        /// Number of input channels
        int m_in_channels;
```

```
        /// Compression mode (0 is no compression, 1 indicates the model is compressed)
        int m_compressionMode;
        /// Type of network (0 is very shallow, 4 is quite deep)
        int m_netType;
        /// Type of network (0 is very shallow, 4 is quite deep)
        int m_quantized;
        /// Thread number:
        /// (-1) no multithreading - (0) automatic (1 per core) - (N) user defined
        int m_num_threads;

        /// Normalization mode (0) No normalization - (1)
        int m_normalizationMode;
        /// Influence of RCNN part
        float m_rcnnWeight;

        /// Size of the weights stored in memory
        int m_cnnWeightsSize;
        /// Pointer to weights stored in memory
        const unsigned char *m_cnnWeights;
};


/// Exposed shape and pose for the
class xmgNonRigidPoseAPI
{
public:
        xmgNonRigidPoseAPI() {
                m_detected = 0;
                m_poseComputed = 0;
                m_nbLandmarks = 0;
                m_landmarks = nullptr;
                m_landmarks3D = nullptr;
                m_triangles = nullptr;
        }

        /// indicates if a face is detected
        int m_detected;
        /// indicates if a 3D pose is computed
        int m_poseComputed;

        /// translation (right handed)
        float m_t[3];
        /// euler orientation (left handed as defined in Unity)
        float m_euler[3];
        /// quaternion orientation
        float m_quat[4];
        /// rotation matrix (right handed)
        float m_matRot[3][3];

        /// landmarks number
        int m_nbLandmarks3D;
        /// landmarks number
        int m_nbLandmarks;
        /// 3D landmarks positions (x1, y1, z1, ...)
        float *m_landmarks3D;
        /// 2D image landmarks (x1, y1, z1, ...)
        float *m_landmarks;
```

```
        /// triangles number
        int m_nbTriangles;
        /// triangles indices
        int *m_triangles;
        /// key 3D landmarks positions of tracked facial features (x1, y1, z1, ...)
        float *m_keyLandmarks3D;

        /// Left right eyes coordinates in image plane
        float m_eyesPositions2D[4];

        /// Left right eyes coordinates in 3D space
        float m_eyesPositions3D[6];

        /// Left right eyes center coordinates in 3D space
        float m_eyesCenter3D[6];

        /// Detected face emotions
        float m_emotions[8];
};
```

High Level iOS API

The iOS API is objective-C based and developed with XCode. It is an example of how to use the SDK API described above. It is available in the sample entitled **MagicFace-OpenGLES**. The iOS API is implemented in **Sources/xmgAugmentedFaceIOSAPI.m**. Below are described main available functions. The sample give you an example of a working application based on this High level API functions.

```
/**
 *     Initialize the detection engine
 *     @param camWidth captured image size
 *     @param camHeight captured image size
 *     @param nbFaceFeatures face features model (51 or 68)
 */
int ICV_DLL_SIGNATURE Initialize(char *faceClassifiersContent,
                                 float fovxDegree,
                                 int camWidth, int camHeight,
                                 int procWidth, int procHeight,
                                 int nbFaceFeatures,
                                 int screenOrientation,
                                 int nbFaceObjects,
                                 bool detect3DFacialFeatures,
                                 bool detectPupils);

/**
*      Release the detection engine
*/
void ICV_DLL_SIGNATURE Release();


/**
```

```
 *     Initialize the GL Engine
 */
void ICV_DLL_SIGNATURE InitializeGL(bool useYUV);

/**
 *     Release the GL Engine
 */
void ICV_DLL_SIGNATURE ReleaseGL();

/**
 *     Add a 2D Texture to be fitted on the face
 *     @param ptrImage RGBA pixels of the image
 *     @param width
 *     @param height
 *     @param uvCoordinates texture coordinates between 0..1 corresponding to face features
 *     @param layerDepth layer index [0..10]
 *     @return >0 if success
 */
int ICV_DLL_SIGNATURE AddEffect(unsigned char *ptrImage,
                                int width, int height,
                                float *uvCoordinates,
                                int layerDepth);
int ICV_DLL_SIGNATURE AddEffect3D(unsigned char *ptrImage,
                                  int width, int height,
                                  float *uvCoordinates, int nbUVs,
                                  int layerDepth);

/** Replace the last added 3D effect or create one if empty*/
int ICV_DLL_SIGNATURE ReplaceEffect3D(unsigned char *ptrImage,
                                      int width, int height,
                                      float *uvCoordinates, int nbUVs,
                                      int layerDepth);

/** Clear all effects */
void ICV_DLL_SIGNATURE ClearEffects();

/**
 *     Add a 2D Texture in front of the video plane
 *     @param ptrImage RGBA pixels of the image
 *     @param width
 *     @param height
 *     @param deltaX translation of the image position (0..1)
 *     @param deltaY translation of the image position (0..1)
 *     @param layerDepth layer index [0..10]
 *     @return >0 if success
 */
int ICV_DLL_SIGNATURE AddStaticLayer(unsigned char *ptrImage,
                                     int width, int height,
                                     float deltaX, float deltaY,
                                     int layerDepth);

int ICV_DLL_SIGNATURE Add2DImage(unsigned char *ptrImage,
                                 int width, int height,
                                 float deltaX, float deltaY, int layerDepth);

/** Replace the last static layer or create one if empty*/
int ICV_DLL_SIGNATURE ReplaceStaticLayer(unsigned char *ptrImage,
```

```
                                        int width, int height,
                                        float deltaX, float deltaY, int layerDepth);

/**
 *     Render with openGL
 *     @param faceData face features information
 *     @param yuvImage yuv pixels to texture the video plane
 *     @param width
 *     @param height
 *     @param renderFaceFeatures render or not face features with blue circles
 *     @param renderStaticLayer render or not the static layer
 *     @param renderFaceTexture render or not the face texture
 */
void ICV_DLL_SIGNATURE Render(xzimgSdk::xmgNonRigidPoseAPI &faceData,
                              unsigned char *yuvImage,
                              int width, int height,
                              float scaleX, float scaleY,
                              bool renderFaceFeatures,
                              bool renderStaticLayer,
                              bool renderFaceTexture);


/**
 *     Detect/Track Face and find face features locations
 *     @param yuvImage yuv pixels to texture the video plane
 *     @param orientation face orientation for detection (0 landscape left, ...)
 *     @param xmgFaceData face result (filled when a face is detected)
 */
int ICV_DLL_SIGNATURE TrackNonRigidFaces(unsigned char *ptrYUVImage,
                                        int orientation,
                                        int width, int height);
```

Segmentation API on iOS

A dedicated SDK API and sample is available for using the segmentation engine. **Note** both face detection and segmentation can be merged into one single sample.

The segmentation sample is entitled **xzimgMagicSegmentation-sample** and provide an example to use body segmentation and hair segmentation on iOS.

Below is the SDK API for the segmentation (also available in **xmgMagicSegmentationAPI.h**). The sample calls the API function directly and one can modify it for his own projects. Note: once a segmentation mask is acquired eg. using the **GetSegmentationImage** function, it is advised to use proper shaders to prepare rendering effects. Several example shaders are available in the Unity project.

```
/// Initialize the segmentation engine
int ICV_DLL_SIGNATURE xzimgMagicSegmentationInitialize(xzimgSdk::xmgCNNInitParamsAPI
&initParams);

/// Release the detection engine
void ICV_DLL_SIGNATURE xzimgMagicSegmentationRelease();

/// Pauses current tracking
```

```
/// pause: 1 to pause 0 to start
void ICV_DLL_SIGNATURE xzimgMagicSegmentationPause(int pause);

/// Set Color
void ICV_DLL_SIGNATURE xzimgMagicSegmentationSetColor(int B, int G, int R, int A);

/// Segmentation on given image
/// in_image: Input image (any format?)
/// out_image: Output image:
/// - if single channel returns the (potentially rescalled) mask,
/// - if in color returns original image segmented with white pixels in background
/// - if 4 channels, returns original image segmented with alpha channel
int ICV_DLL_SIGNATURE xzimgMagicSegmentationProcess(const xzimgSdk::expImage &in_image, int
rotation);

int ICV_DLL_SIGNATURE xzimgMagicSegmentationGetSegmentationImage(xzimgSdk::expImage
&seg_image, int rotation);
```

High Level Android API

The Android API is Java based and developed with Android Studio. There is an existing sample in the solution that will help you to understand how to inherit from the face detection activity.

```
/**
* Initialize the augmented face engine
* @param width: size of the processed frames (should be the same as video capture
resolution)
* @param height: size of the processed frames (should be the same as video capture
resolution)
* @param fovx_degree: unused
* @param nbFaceFeatures number of face features to be detected: 51 or 68 (with contour)
*/
public void initialize(int width, int height, double fovx_degree, int nbFaceFeatures)
```

```
/**
* Release the augmented face engine
*/
public void release()
```

```
/**
* Detect face and track features in 2D
* @param yuvFrame Current image in native yuv pixel format
* @param orientation Orientation for the face detection (0 for landscape left, 1 for
portrait, ...)
* @param width size of the video frame
* @param height size of the video frame
* @return null if no face is detected
*    [0] indicates if a face is detected (>0)
```

```
*    [1..6] returns face pose (not computed in 2D case)
*    [7] indicates the number of features
*        [8..8+number of features*2]    returns face features coordinates
*        [9+ number of features*2]  indicates the number of triangles
*        [9+ number of features*2 +1 , ...] indicates the triangles indices
*/
public float [] trackFaceFeatures2D(ByteBuffer yuvFrame, int orientation, int width, int
height)
```

```
/**
* Add a colored image as a layer using GL Surface View
* @param context current context
* @param resourceId index of the image resource to be rendered
* @param x translate the image (normalized image coordinates)
* @param y translate the image (normalized image coordinates)
* @return zero or less if failed
*/
public int add2DImage(final Context context, final int resourceId, float x, float y)
```

```
/**
* Add a colored image as a layer using GL Surface View
* @param bitmap image to be displayed
* @param x translate the image (normalized image coordinates)
* @param y translate the image (normalized image coordinates)
* @return zero or less if failed
*/
public int add2DImage(Bitmap bitmap, float x, float y)
```

```
/**
* Add an effect to the face by rendering a GL layer
* @param context current context
* @param resImageIdx index of the image resource to be rendered
* @param resFaceFeaturesIdx index of the text resource to be used for identifying face
features locations
* @param type type of effect to be rendered (0 to add a texture on top of the face)
*/
public int addEffect(final Context context, final int resImageIdx, final int
resFaceFeaturesIdx, int type)
```

```
/**
* Add an effect to the face by rendering a GL layer
* @param bitmap image to be displayed
* @param faceFeaturesLocations array that contains image coordinates of face features
* @param type type of effect to be rendered (0 to add a texture on top of the face)
*/
public int addEffect(Bitmap bitmap, float [] faceFeaturesLocations, int type)
```

```
/**
* Get the rendered RGB image after applying the materials
* @return rgb byte buffer that contains the frame
*/
```

```
public ByteBuffer getAugmentedImageRGB()
```

```
/**
* Display/Hide face features
* @param renderFaceFeatures
*/
public void setRenderFaceFeatures(boolean renderFaceFeatures)

/**
* Display/Hide static images
* @param renderStaticImageLayers
*/
public void setRenderStaticImageLayers(boolean renderStaticImageLayers)
/**
* Display/Hide face textures
* @param renderFaceTextureLayers
*/
public void setRenderFaceTextureLayers(boolean renderFaceTextureLayers)
```

Contact

For any information or question regarding this product, please contact us at contact@xzimg.com