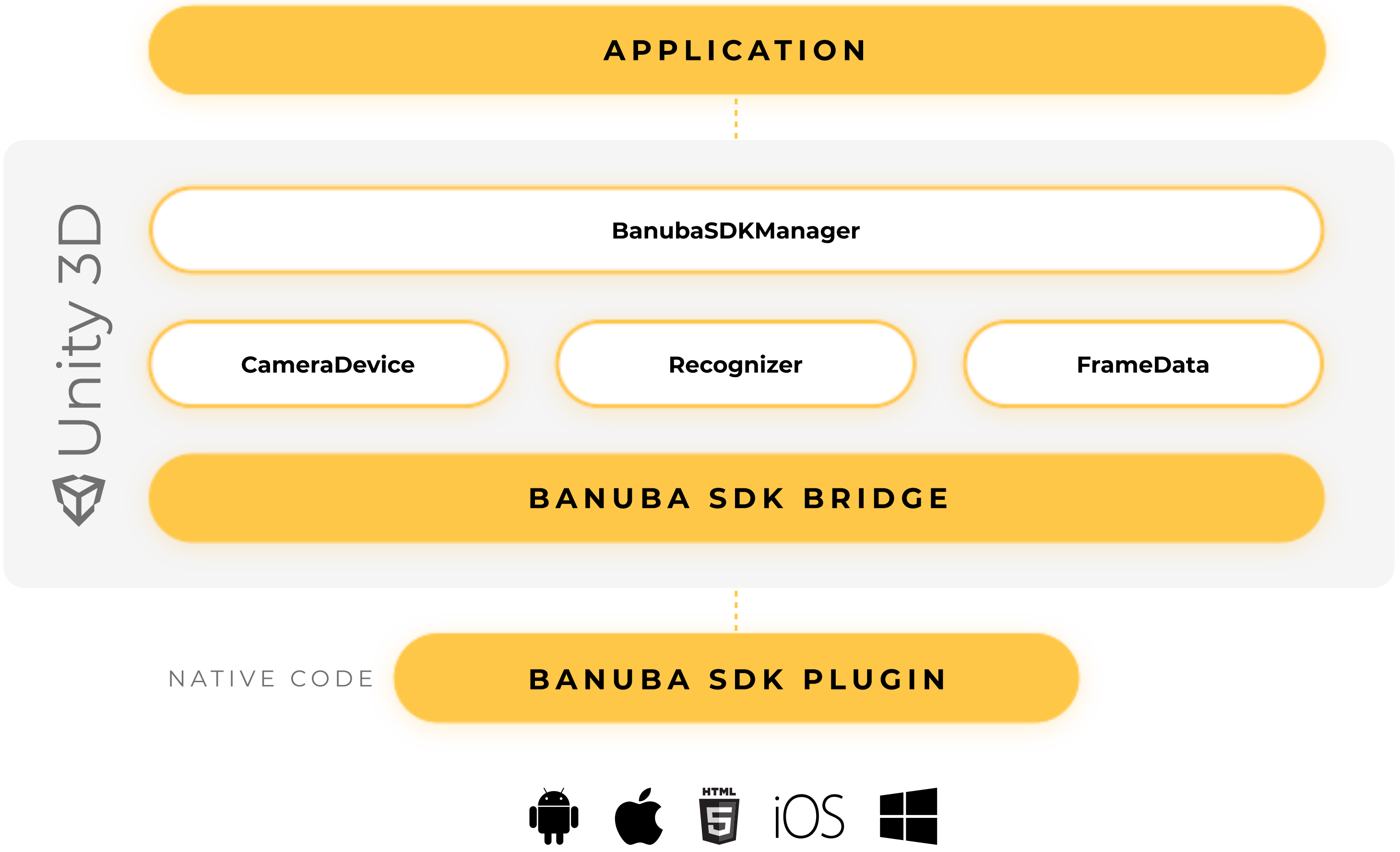
Overview

Introduction to Face AR plugin for Unity



Banuba Face AR plugin for Unity is a native library compiled for the following platforms:

- Windows
- MacOS
- iOS
- Android
- Web

This library exports pure C interface represented in script `BanubaSDKBridge.cs` as C# bindings. BanubaSDKBridge provides the following common methods to initialize the face recognition engine.

Banuba Face AR plugin static environment initialization:

- bnb_recognizer_env_init
- bnb_recognizer_env_release

These methods should be called once at the start and the end of the game respectively.

Once the environment is initialized, you can create the native recognizer object with the following methods:

- bnb_recognizer_init
- bnb_recognizer_release

You can use the already implemented wrapper from `Recognizer.cs` for releasing the memory.

The recognizer object init method needs a path to its resources. They are placed in `Assets/StreamingAssets` folder, and unity does not compress resources placed there which is important. Full path to `Assets/StreamingAssets` is platform dependent. Unity provides it as `Application.streamingAssetsPath` property. We recommend using only one instance of the recognizer object to decrease memory consumption.

Once the recognizer object is created successfully, you can change options and enabled features using the following methods:

- bnb_recognizer_set_features
- bnb_recognizer_set_fov

- bnb_recognizer_set_max_faces
- bnb_recognizer_set_offline_mode

All these methods are properly documented in `BanubaSDKBridge.cs`

To process an image, you need to create the native frame representation object using following methods:

- bnb_frame_data_init
- bnb_frame_data_release

You can use already implemented wrapper from `FrameData.cs` for releasing the memory.

Once the frame data instance is created, you need to set an image to it using `bnb_frame_data_set_bpc8_img` for BPC8 (RGB and similar) images and `bnb_frame_data_set_yuv_img` for YUV images. After that, you can process the frame data with a prepared recognizer instance by calling `bnb_recognizer_process_frame_data` .

When processing is completed, the frame data object will be filled up with data according to the enabled recognizer features. Only `recognizer_feature_frx_id` is enabled by default, so you can check if a face was detected on the current frame by extracting the face data structure from the frame data object by calling `bnb_frame_data_get_face` . For back-compatibility reasons, the frame data always contains one face with index 0. You should additionally check the face rectangle flag like `face.rectangle.hasFaceRectangle > 0` , and it will show if the face was detected.

When the face is detected, the face data object contains `verticies` of face mesh and face `landmarks` . Substructure `camera_position` contains projection and model-view maticies, and also affine coefficients extracted from them.

Face Mesh

- Landmarks of face mesh packed into `float` array of (x, y) coords one-by-one
- Verticies of face mesh packed into `float` array of (x, y, z) coords one-by-one
- UV coords of face mesh packed into `float` array of (x, y) coords one-by-one (`bnb_frame_data_get_tex_coords` method)
- Indicies of face mesh verticies packed into `int` array (`bnb_frame_data_get_triangles`

method)

Face mesh usage example is placed in `FaceMeshController.cs` script.

Action Units

Action Units (or blendshapes) can be enabled by setting the recognizer feature `recognizer_feature_action_units_id` . Once done, you can extract action units from `FrameData` object for the detected face using the method `bnb_frame_data_get_action_units` . Action units are packed into `float` array. The indexes of elements in this array correspond to enum `bnb_action_units_mapping_t` entries.

Action Units usage example is placed in `GrootController.cs` script.

Last updated on 12/23/2019

← REFERENCE GETTING STARTED →