December 18th, 2018 🎄
6D.ai Beta SDK v0.19.1 Release Notes

Find additional documentation on the 6D.ai beta developer portal
https://dashboard.6d.ai

Introduction

Happy Holidays!

Accessing and configuring the underlying ARKit session to our iOS SDK has been a popular developer request from the beginning. We are excited to make it available to you today!

Because of the complexity of transforming ARKit coordinated to relocalized 6D coordinates, we are changing our 6D convention from Unity-style left-handed to ARKit-style right-handed. The sample code has been updated to reflect this change (adding conversion in Unity, taking it out in SceneKit). Your custom code written on top of our API, if any, might need updating.

This last release of 2018 also brings improvements in meshing accuracy and relocalization precision. On the sample app side, we now fully support up to 60 frames per second, greatly improving user immersion.

Changes

Changed API Methods

- `bool SixDegreesSDK_Initialize(void)`
  now returns false if the SDK will not initialize. Details will be found in the logs (it usually is a configuration error).
- `bool SixDegreesSDK_InitializeWithEAGL(void* eaglContext)`
  similarly now returns a bool.
- `int SixDegreesSDK_GetPose(float* poseDataOut, int bufferSize)`
  now returns the pose in right-handed world coordinates (flipping the Z axis from the previously used Unity coordinate convention).
- `int SixDegreesSDK_GetMeshBlocks(int* blockBuffer, float* vertexBuffer, int* faceBuffer, int blockBufferSize, int vertexBufferSize, int faceBufferSize)`
  similarly now returns the block coordinates, vertex coordinates and vertex normals in right-handed world coordinates.

New Expert API Methods

A new header in `SixDegreesSDK.framework` is now available: `SixDegreesSDK_advanced.h`
New API methods in this header are designed to provide configuration options and access to the underlying ARKit session, allowing options such as light estimation, plane detection, image detection, buffer access, etc. Because of potential issues in terms of performance, coordinate system complexities, etc. we only recommend those APIs for expert developers.

- `bool SixDegreesSDK_InitializeWithConfig(ARWorldTrackingConfiguration* configuration)`
  allows enabling additional ARKit features. The SDK imposes constraints on the configuration, and InitializeWithConfig() will only return true if:
  - Autofocus is disabled
  - World alignment is ARWorldAlignmentGravity
  - The video format aspect ratio is 16:9
- `ARSession* SixDegreesSDK_GetARKitSession(void)`
  returns the ARSession object managed by the 6D SDK. We recommend using that object to access values in the current ARFrame, but do not set a different delegate object or use start, pause and stop controls.
- `void SixDegreesSDK_GetARKitTransform(float* transformDataOut, int bufferSize)`
  returns the coordinate transform from the native ARKit coordinate system to the 6D world coordinate system. This will be an identity matrix before relocalization. Use that transform to adapt coordinantes, vectors and transforms to the mesh and pose provided by 6D.

Right-handed Coordinates Conventions

Sample apps have been updated to reflect the change in coordinates conventions in the SDK methods. If you use a custom script in your project on top of our API, e.g. in Unity, make sure that you update it to reflect that change. If you use our stock classes, no change should be necessary.

To flip the Z axis in a transform such as a camera pose, use the following matrix Q:

1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	-1.0	0.0
0.0	0.0	0.0	1.0

The same matrix Q is used to go from 6D to Unity and back. Please note that the math to change a pose P is: $P_{unity} = Q_{6D \rightarrow unity} * P_{6D} * Q_{unity \rightarrow 6D}$

You can avoid using matrix multiplications in your code by just flipping the sign of the right matrix indices; you'll notice this is what our sample code does:

m_{11}	m_{21}	$-m_{31}$	x
m_{12}	m_{22}	$-m_{32}$	y
$-m_{13}$	$-m_{23}$	m_{33}	-z
0.0	0.0	0.0	1.0

Technology Improvements

Keep in mind that relocalization accuracy and success rate depend considerably on the computer-vision-friendliness of the environment: detailed, non-repetitive textures; stable and sufficient lighting; non-reflective materials; and absence of motion.
Meshing quality is also sensitive to those factors but tends to perform well in a larger variety of environments.

- **Relocalization** is now more accurate, with a median error half the size it used to be. This also means that some false positives are eliminated, potentially increasing the average time to relocalization success.
- **Meshing** is less noisy again, comparable to 0.15.x levels.
- Background texture updates now run at 60 FPS.

Unity Package and Sample App

- **SDMesh.cs** changed to use a mesh chunk prefab instead of programmatically adding components to an empty game object. This means you can just swap the prefab in your scenes if you wish to use different components. Mesh chunks have their own dedicated layer (User Layer 9). The mesh now updates at a fixed interval (0.02 sec by default).
- Some prefab names changed to be more explicit. If the link breaks in your project, look for **AR Background** and **AR Scene**.
- The AR Background now has its own dedicated layer (User Layer 8), separate from the UI layer (Builtin Layer 5). This was a popular request from developers using 3D UI. The Render Queue order of the background's material was also reduced to 1990.
- The Sample App now targets a smooth 60 FPS instead of the Unity default 30 FPS.

Known Issues

- Framerate drops when attempting relocalization in highly detailed environments, which is made more obvious by the application otherwise running at 60 FPS. Optimizations will address this in future releases.
- ARKit tracking drift makes the previously scanned mesh look offset after a certain amount of motion. Continuous relocalization will address this in future releases.

Hardware Requirements

As of SDK version 0.19.1, the following iOS devices are supported:

Year	**Supported iPhones** Identify your iPhone model	**Supported iPads** Identify your iPad model
2018	iPhone XS iPhone XS Max iPhone XR	iPad Pro 12.9" (3rd Gen) iPad Pro 11" iPad (6th Gen)
2017	iPhone X iPhone 8 iPhone 8 Plus	iPad Pro 12.9" (2nd Gen) iPad Pro 10.5"
2016	iPhone 7 iPhone 7 Plus	iPad Pro 9.7"
2015		iPad Pro 12.9" (1st Gen)

Not supported:

- iPhone 6S (2015) and lower
- iPad 5th Gen (2017) and lower
- iPad Air family
- iPad Mini family

The SDK **will not initialize** on unsupported devices, even if they support ARKit.
The API method SixDegreesSDK_IsDeviceSupported() should be used at runtime to detect if the device is supported by the SDK.