

September 25th, 2018
6D.ai Beta SDK v0.15.1

Table of Contents

- [Introduction](#)
- [Hardware Requirements](#)
- [How to Install the 6D.ai SceneKit Sample App](#)
- [How to Use the 6D.ai SceneKit Sample App](#)
- [How to Install the 6D.ai Unity Sample App](#)
 - [How to set up the Photon Drawing Sample Scene](#)
- [How to Use the 6D.ai Unity Sample App](#)
 - [Basic Sample scene](#)
 - [Drawing Sample scene](#)
 - [Meshing Sample scene](#)
 - [Ball Pit Sample scene](#)
 - [Photon Drawing Sample scene](#)
- [Meshing API Guide](#)

Introduction

Happy Tuesday!

Today we introduce SDK version 0.15.1, now available for iPhone XR, XS and XS Max devices.

Our efforts are focused towards our next major milestone, and large amounts of changes under the hood to support compelling mesh persistence. Stay tuned for more updates on our Slack channel!

In other news, finalists for the first round of the 6Demo-God Challenge have been selected, but it's not too late to submit new pitches for demos, as another round is in the works!

More information on <https://www.6d.ai/6demo-god-challenge/>

Hardware Requirements

As of SDK version 0.15.1, the following devices are fully supported, including real-time meshing:

Chip	Supported iPhones Identify your iPhone model	Supported iPads Identify your iPad model
A12	XR, XS, XS Max	
A11	X, 8, 8 Plus	
A10X		Pro 12.9" (2nd Gen), Pro 10.5"
A10	7, 7 Plus	6th Gen (2018)
A9X		Pro 12.9" (1st Gen), Pro 9.7"

Not supported:

- iPhone 6S and lower
- iPad 5th Gen (2017) and lower
- iPad air family
- iPad mini family

Android support is still work in progress, we hope to share exciting updates in a near future!

How to Install the 6D.ai SceneKit Sample App

The following is a walk through for building and running the SceneKit SDK sample app on a compatible device.

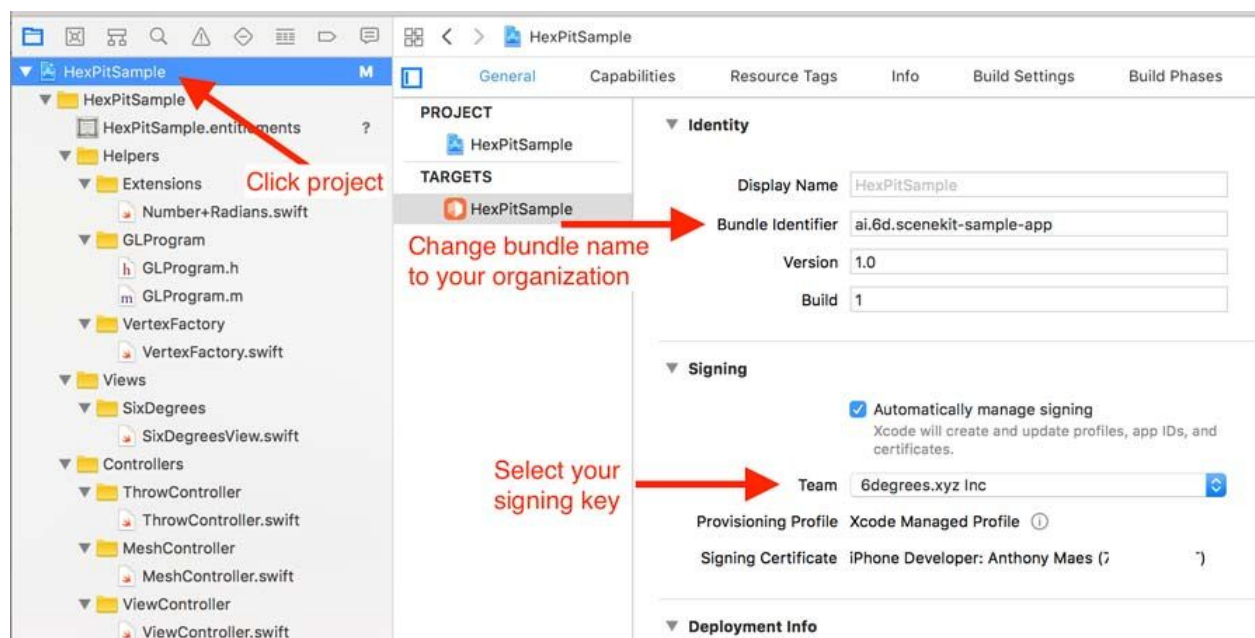
Requirements:

- iOS 12+ or iOS 11.4+
- Xcode 10+

1) Extract the sample app folder from the .zip file to your desired location.

2) Open HexPitSample.xcodeproj with Xcode

3) Select the project and change the Signing Team. You should also change the Bundle Identifier to your organization's name.

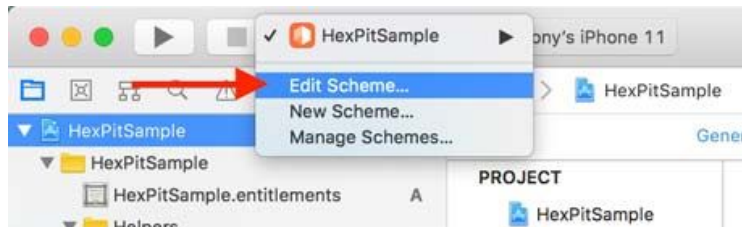


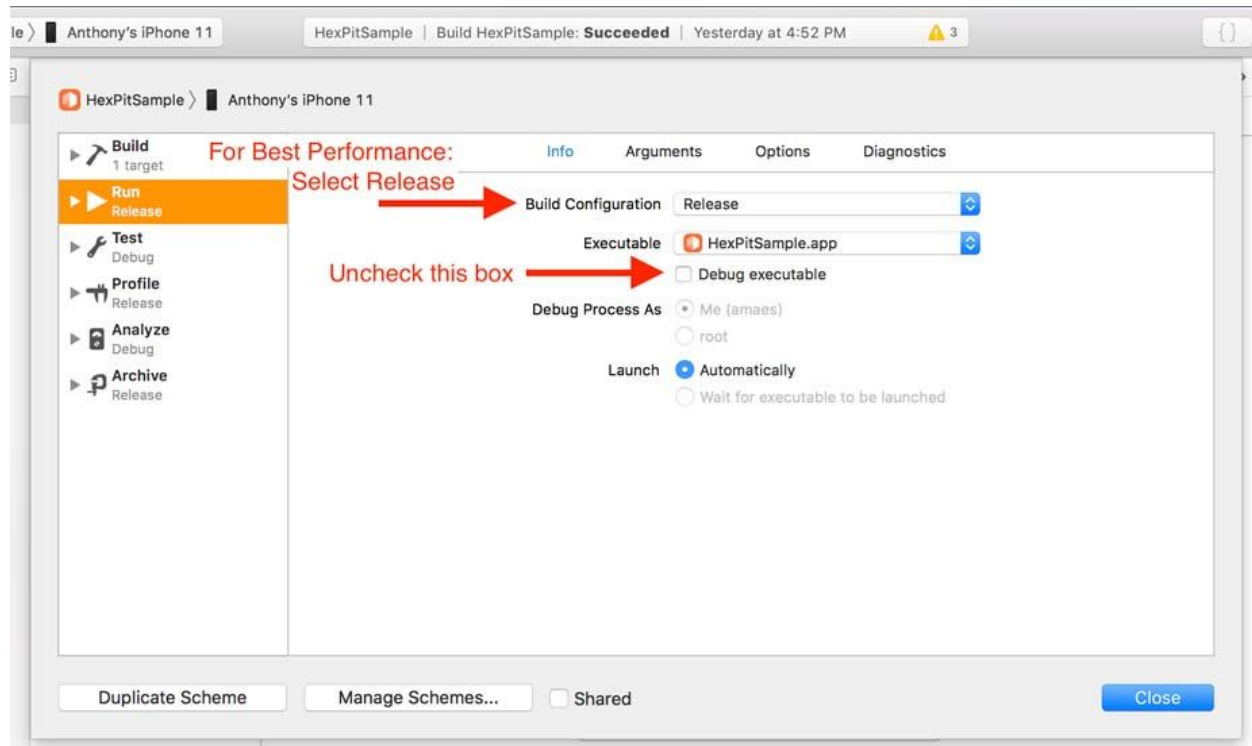
4) New for iOS 12 support, in the Capabilities tab, check "Access WiFi Information". Note this option only exists in Xcode 10+.



5) Add or replace HexPitSample/Supporting Files/SixDegreesSDK.plist with the one downloaded from [your developer profile](#).

6) For best performance, go into the Scheme settings and uncheck "Debug executable."





Everything should now be ready to build to the mobile device!

7) Connect your iPhone or iPad

8) Click the Run button.  The app should open on the mobile device automatically.

How to Use the 6D.ai SceneKit Sample App



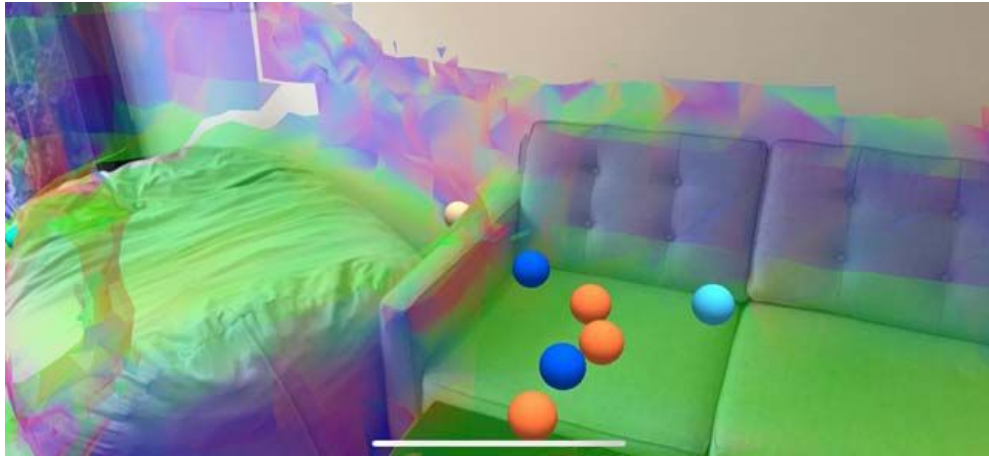
Open **6DSceneKitSample**. The app icon will be an orange 6D logo.

On the first launch, a pop-up will ask you to authorize usage of the phone's camera; another pop-up will also ask you to authorize usage of location data.

Every application build on top of the 6D SDK needs user consent to these two things, or the 6D SDK functions will not work properly.

⚠ Please make sure that **Location Settings** are **not disabled** under **Settings > Privacy**.

At the moment, the app only consists of a single scene demonstrating Meshing and Physics within SceneKit. Move around to mesh the environment, and tap the screen to throw balls.



The experience is similar to the [Ball Pit scene](#) in the Unity sample app.

How to Install the 6D.ai Unity Sample App

The following is a walk through for building and running the Unity SDK sample app on a compatible device.

Requirements:

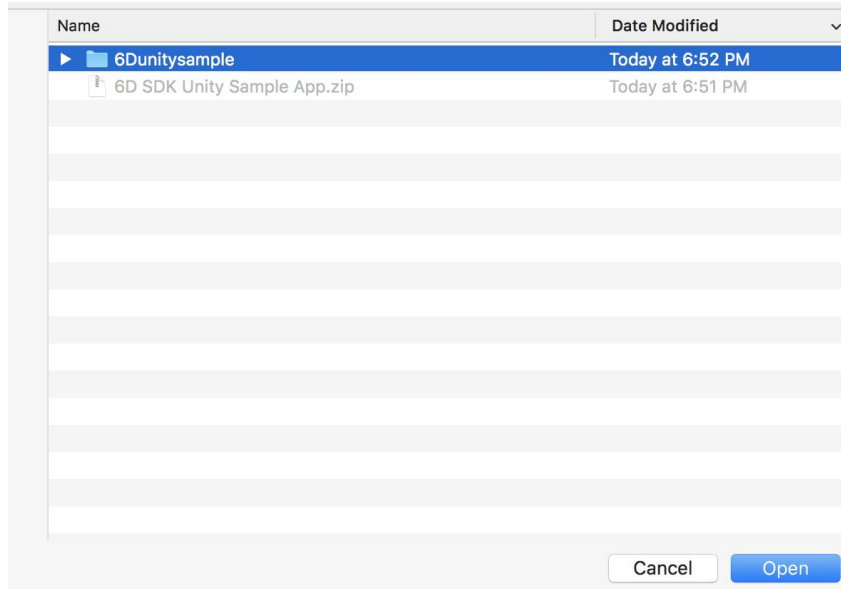
- iOS 12 or iOS 11.4+
- Xcode 10+
- Unity3D 2018.2+

1) Extract the sample app folder from the .zip file to your desired location.

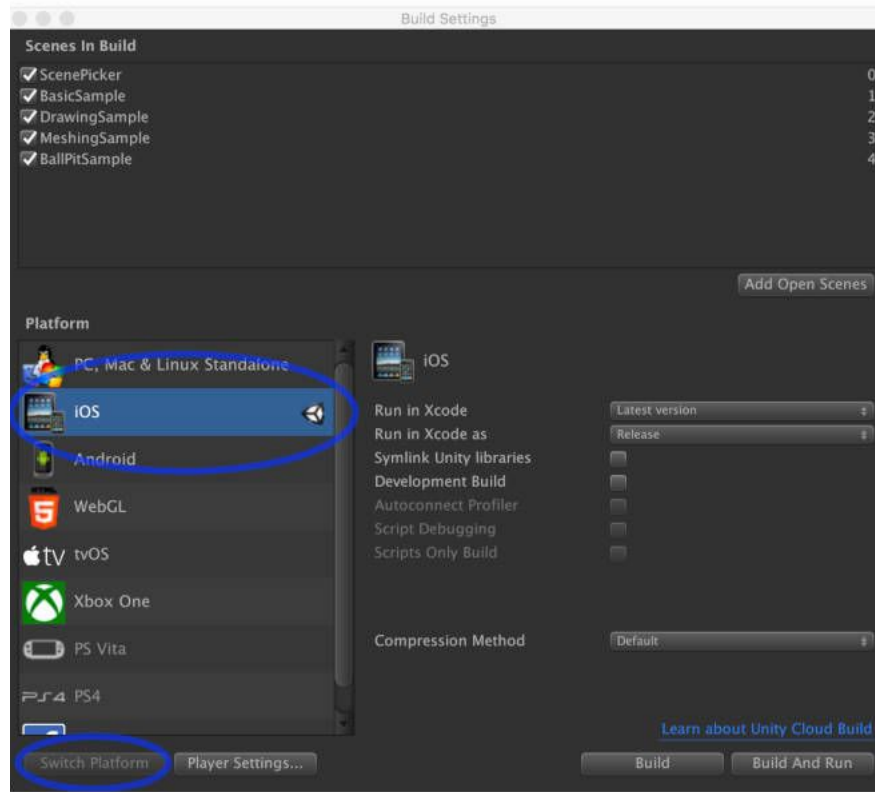
2) Replace Assets/Plugins/iOS/SixDegreesSDK.plist with the one downloaded from [your developer profile](#).

3) Connect your iPhone or iPad

4) Open Unity3D, then open the sample app folder as a new project.

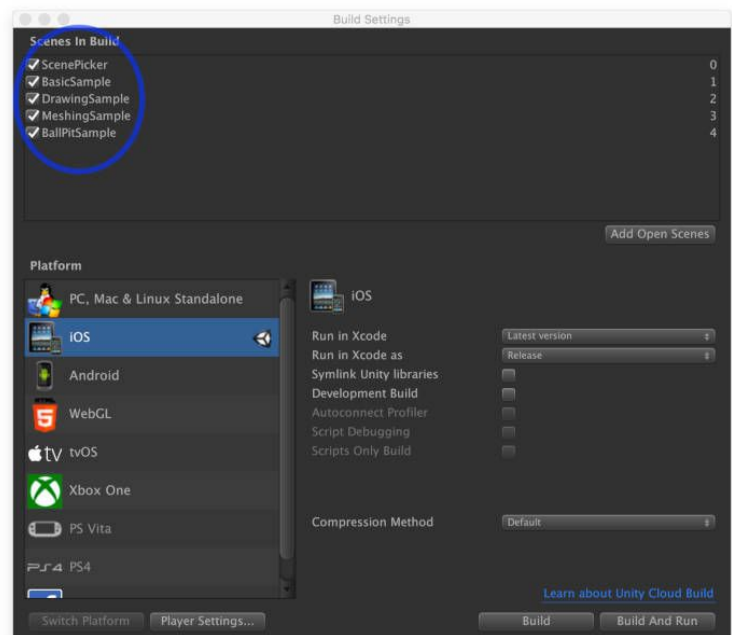


5) Navigate to *File>Build Settings* (Cmd+Shift+B)



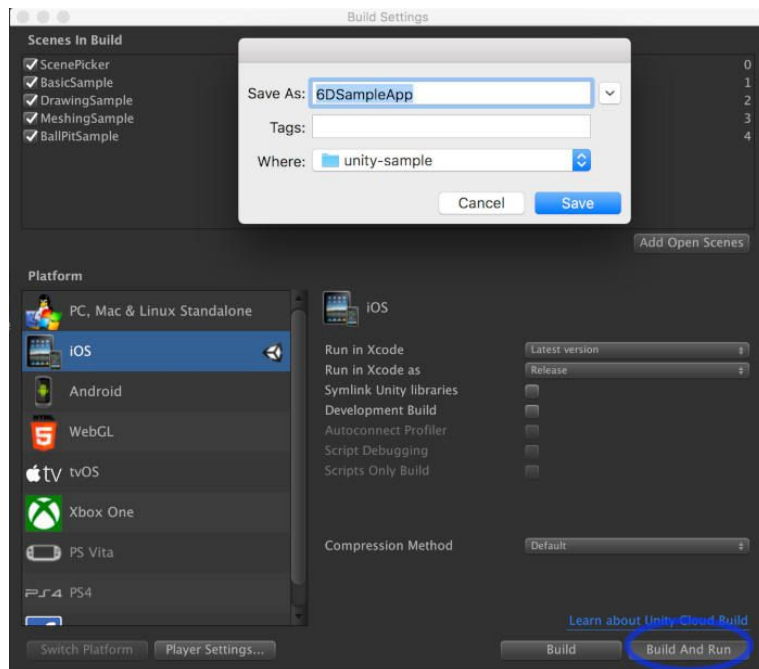
6) Verify that iOS is preselected as your build platform, if not then select iOS and click “Switch Platform”

7) All of the 6D.ai scenes should be automatically added and checked. If any are not checked, please click the corresponding checkbox to add to the build.



8) Select **Build and Run**.

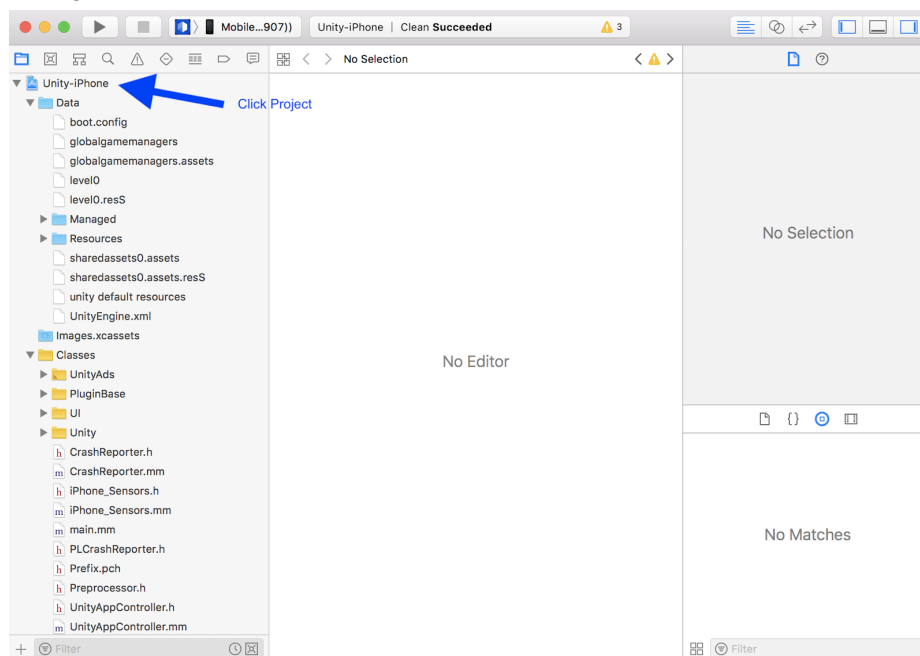
Save the build to your desired location and select Save.



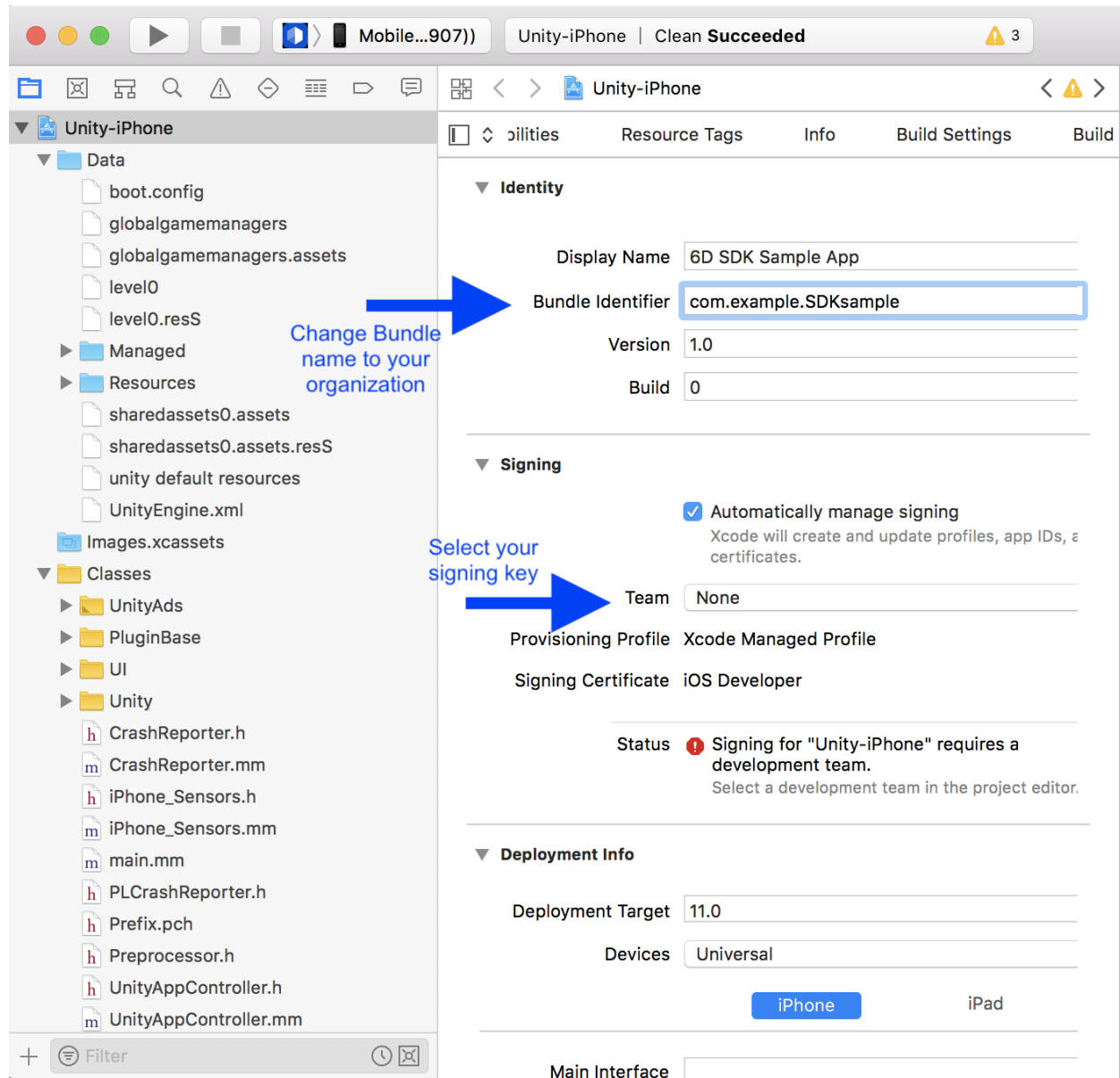
Xcode should open automatically.

You may encounter errors when Xcode opens. These can be cleared by changing your Signing settings.

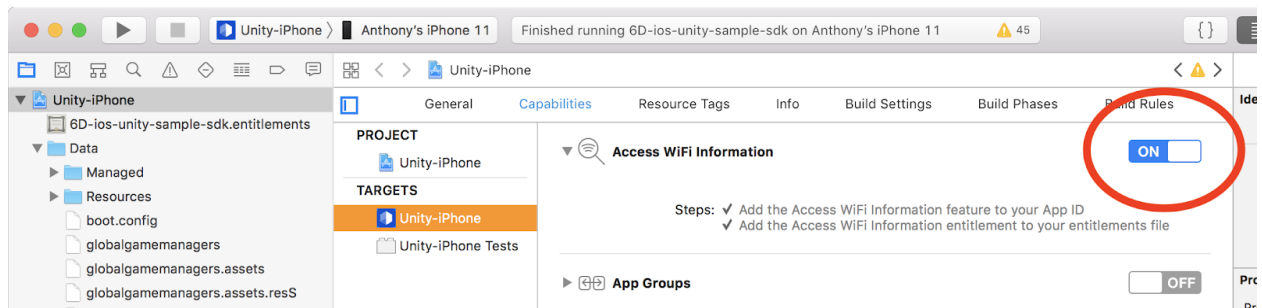
9) Select the project:



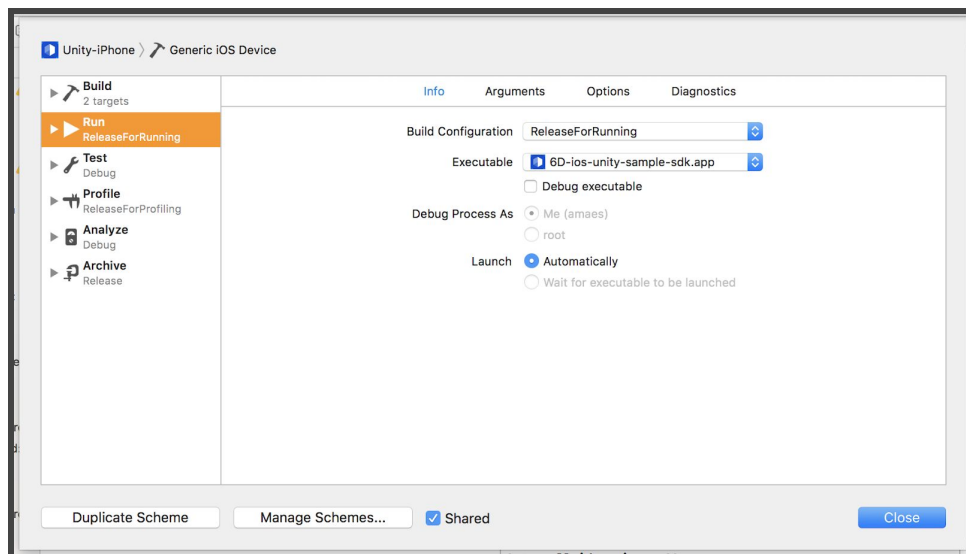
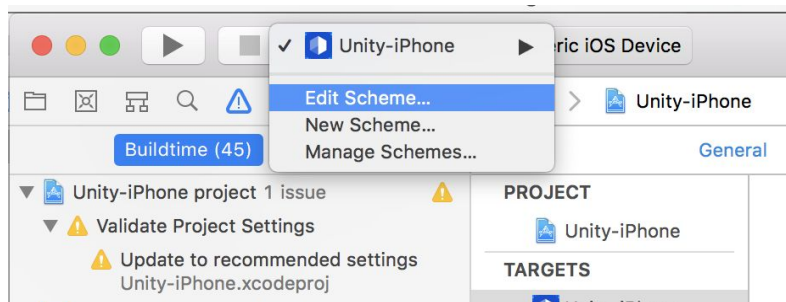
10) Then you can change the Signing Team. You should also change the Bundle Identifier to your organization's name.




11) New for iOS 12 support, in the Capabilities tab, check "Access WiFi Information".
Note this option only exists in Xcode 10+.



12) For best performance, go into the Scheme settings and uncheck "Debug executable."



Everything should now be ready to build to the mobile device!

13) Click the Run button.  **The app should open on the mobile device automatically.**

How to set up the Photon Drawing Sample Scene

Part of multiplayer is the real-time streaming of data between clients. In the Drawing Sample Scene, this is implemented using Unity's built-in NetworkManager, which requires clients to be connected to the same local network via WiFi.

A big limitation of this scene is the impossibility to do multiplayer on cellular network, or on WiFi networks with a firewall configured to block such connections. For this reason we added an alternative implementation of that sample scene, using **Photon**.

Photon is a popular service for cloud-based real-time data streaming, available as a Unity plugin known as PUN (Photon Unity Networking). The steps below describe the additional steps to configure Photon properly in order to use the Photon Drawing Sample Scene.

1) Sign up for a Photon account.

Go to this link: <https://www.photonengine.com/en-US/Photon> and log in if you have an account, or create one if you don't.

2) After logging in, create a new application:



The screenshot shows the Photon dashboard for a new application named "MyTestGame". A magnifying glass highlights the "App ID" field, which contains the value "5553d25-ded4-4430-922833180". The dashboard includes a "Go back to the dashboard" link, a "Properties" section with fields for Name, Url, and Description, and a "Concurrent Users" table.

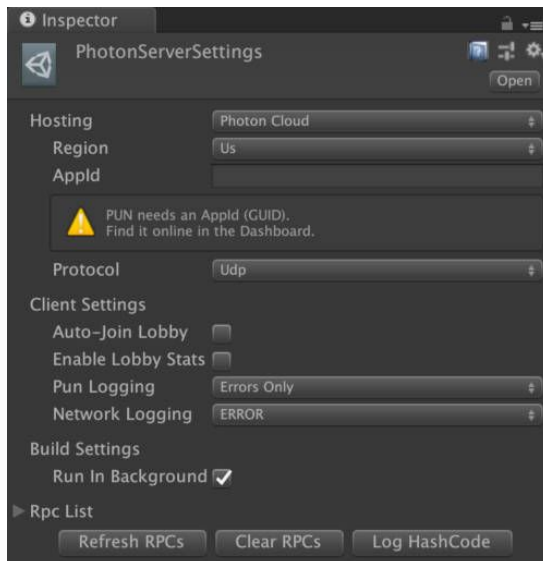
Concurrent Users	
Subscription	0 CCU
One-Time	0 CCU
Coupon	0 CCU
Total	20 CCU CCU Burst is not allowed.

At the bottom, there are buttons for "Edit Properties" and "Delete Application".

Provide the necessary info in the fields. Now get your App ID from the dashboard.

3) Configure the Photon settings in Unity:

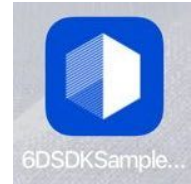
Under *Assets/Photon Unity Networking/Resources* you should find the *PhotonServerSettings*. Paste the App ID from the dashboard into the AppId field below.



4) Build and Run the project again.

You're done! The Photon Drawing Sample Scene is ready to use!

How to Use the 6D.ai Unity Sample App



Open **6DSDKSampleApp**. The app icon will be a blue 6D logo.

On the first launch, a pop-up will ask you to authorize usage of the phone's camera; another pop-up will also ask you to authorize usage of location data.

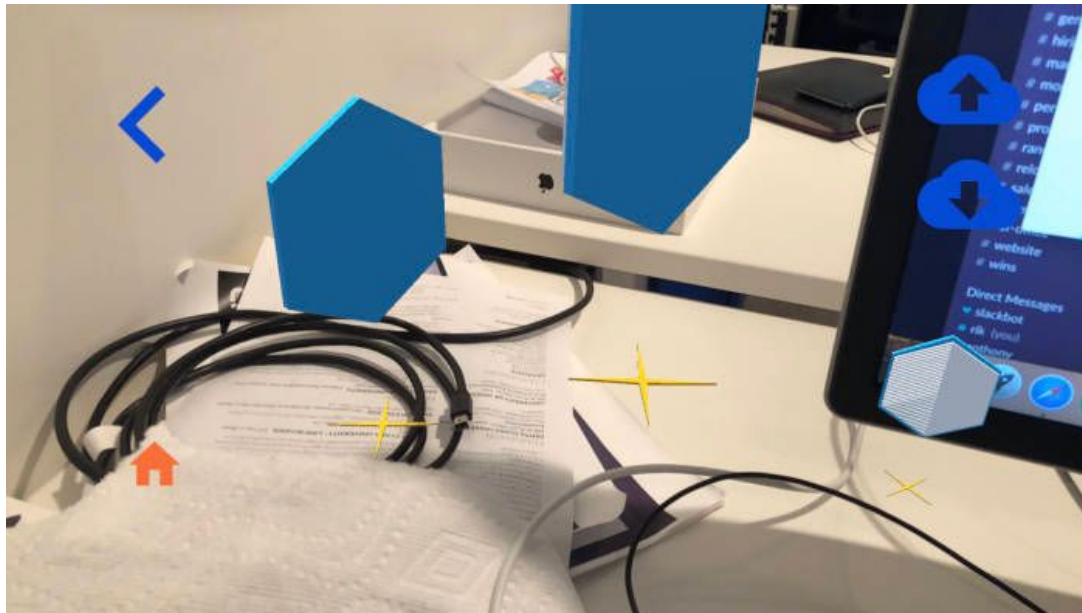
Every application build on top of the 6D SDK needs user consent to these two things, or the 6D SDK functions will not work properly.

⚠ Please make sure that **Location Settings** are **not disabled** under **Settings > Privacy**.



The app opens to a scene selector with a selection of 5 scenes you can choose from, emphasizing different aspects of the SDK:

1. The Basic Sample Scene is symbolized with a 6D hexagon icon.
2. The Drawing Sample Scene is symbolized with a palette icon.
3. The Meshing Sample Scene is symbolized with a 6D hexagon icon with a terrain icon.
4. The Ball Pit Sample Scene is symbolized with a three-sphere icon.
5. The Photon Drawing Sample Scene is symbolized with a palette icon and a Photon logo.



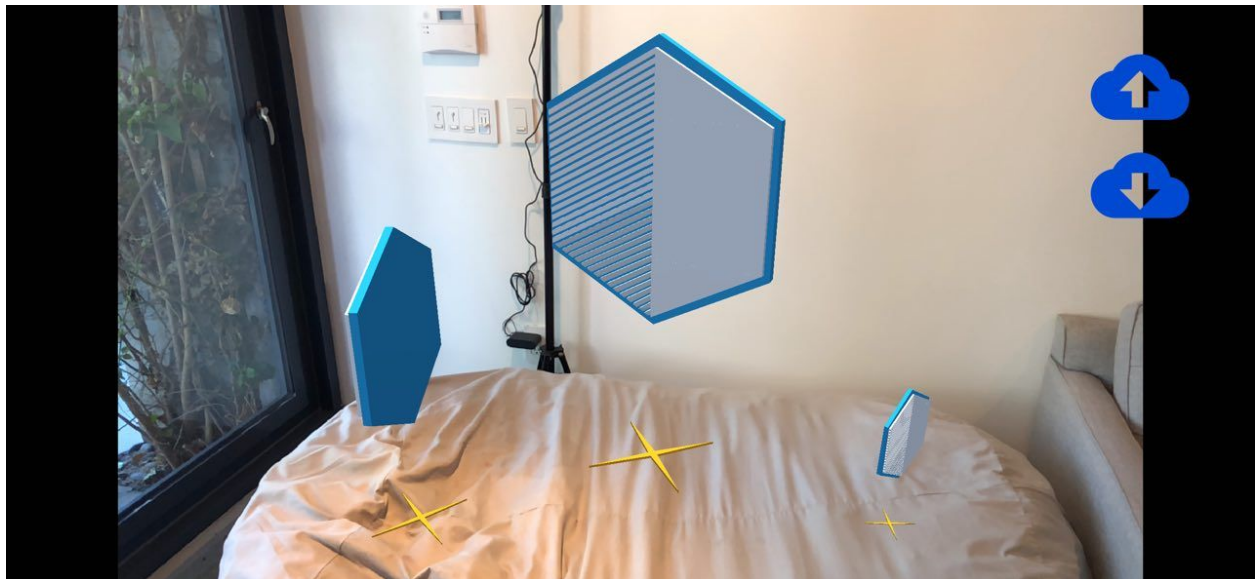
Here is the Basic Sample Scene open.

You can see the left arrow on the top left of the screen. This allows you to transition back to the scene picker menu.

How to Use the Basic Sample scene

The purpose of this sample is to demonstrate a minimal AR setup with persistence. The other samples are more advanced; a good handle of this one is recommended before exploring the others. Don't worry, it's simple!

The app presents itself as a camera viewport with two cloud buttons and three rotating 6D logos floating in AR.



Note that the logos are floating at a fixed arbitrary position, and there is no occlusion.

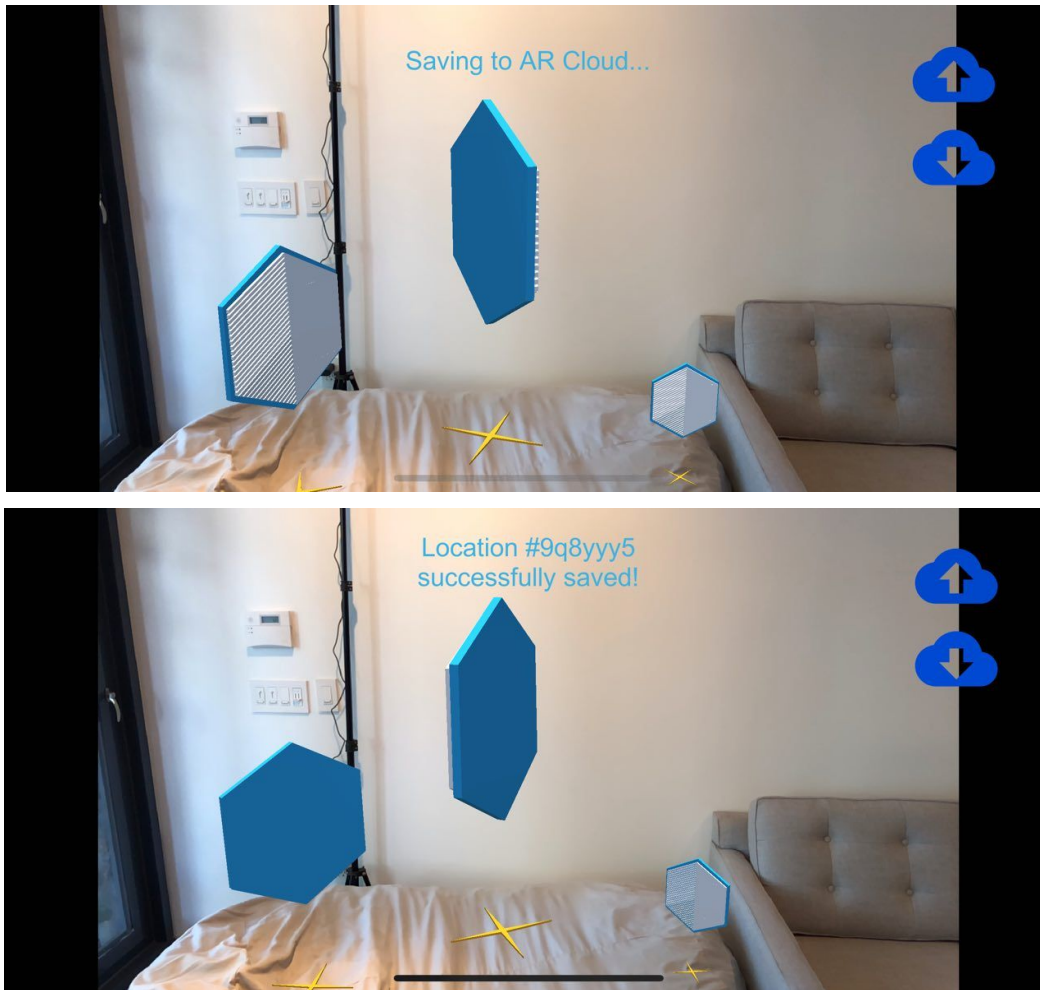
The top button (cloud with arrow pointing up) performs an API call to **Save To AR Cloud**.
The bottom button (cloud with arrow pointing down) performs a call to **Load From AR Cloud**.

As you use the app and explore the real world with your phone's camera, the 6D SDK scans and maps the area. You can control when to save and load those scans with those two buttons.

When you load, a phase called "relocalization" allows the 6D SDK to find where it is, relative to a previously saved map of the same location. This allows the coordinate system of the AR experience to be identical between sessions and devices, allowing the persistence of objects (since 3D coordinates of virtual objects map to the same physical location) and multiplayer (since the location map can be used by multiple devices simultaneously).

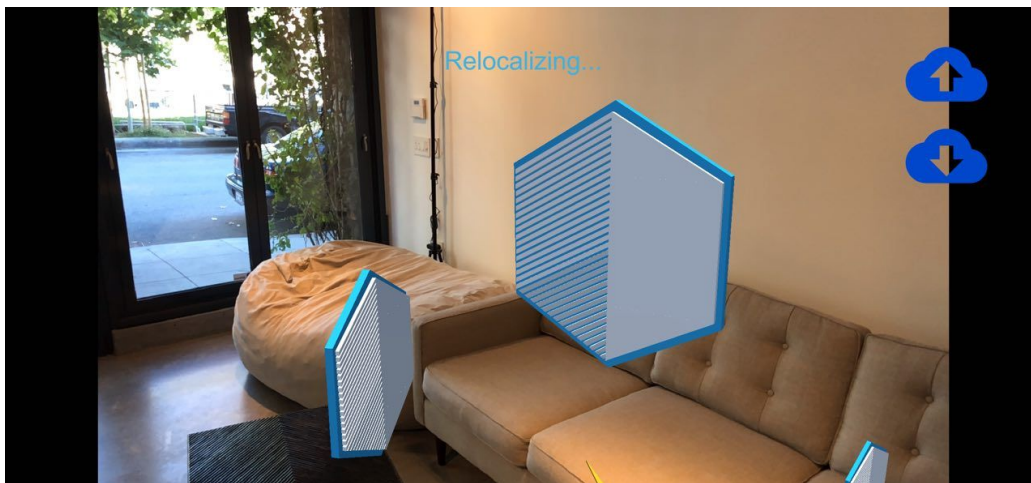
In this sample, there is no obvious indication that mapping is happening. It's all in the background, and invisible. Walk around your room, move your camera around. Everything that enters your field of view is getting mapped and can be pointed at later for relocalization.

After a few seconds of scanning, hit the **Save to AR Cloud** button (arrow up).



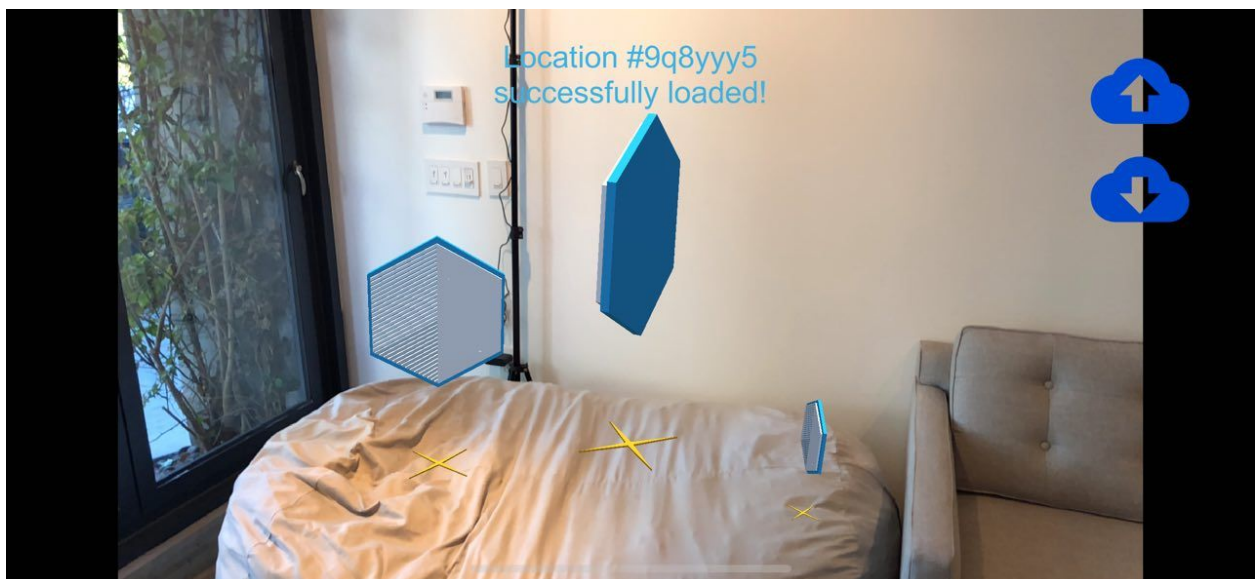
You can now close and kill the app.

Open the app again. You'll find that the logos are now floating at a different fixed arbitrary location. Immediately hit the **Load from AR Cloud button** (arrow down).



When the prompt says "Relocalizing..." make sure that you point your camera at one of the areas you had previously scanned.

After a successful relocalization, you'll notice that the logos teleported back to their original location. This is **persistence**.



In the background, mapping is now resuming, allowing you to save again and make subsequent relocalizations faster and more accurate.

How to Use the Drawing Sample scene

The scene presents itself as a camera viewport with two cloud buttons and a color picker.

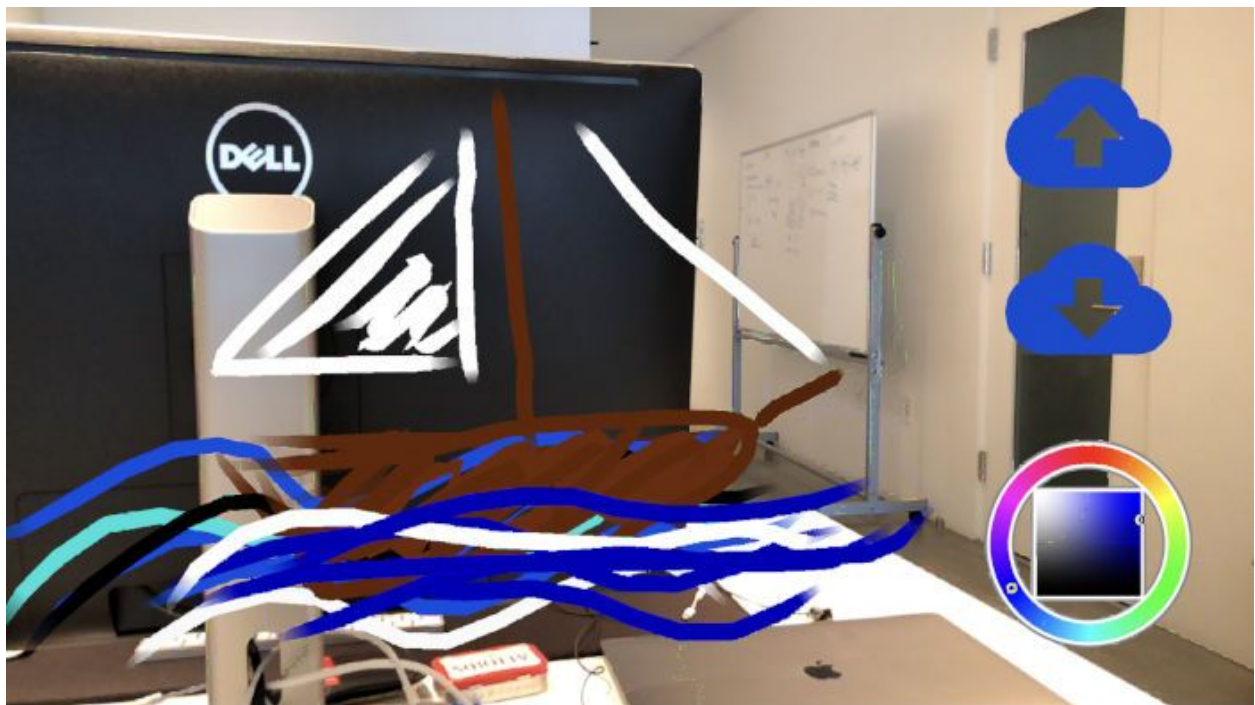


The top button (cloud with arrow pointing up) performs the API call to **Save To AR Cloud**.

The bottom button (cloud with arrow pointing down) performs the call to **Load From AR Cloud**.

The color picker changes the selected brush color.

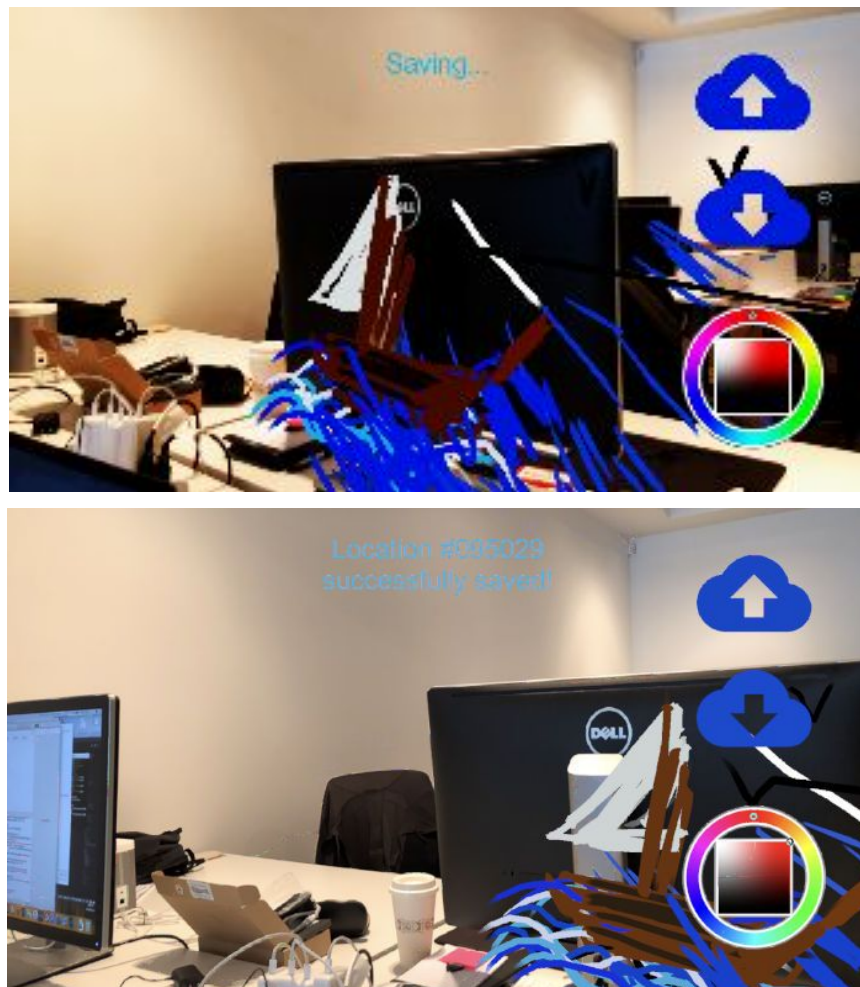
Pick a color and start dragging your finger across the screen. You're now drawing in the air!



You'll notice that the line exists in space at a fixed distance to your phone. To change depth, move your device closer or further.

While you're drawing, the 6D SDK scans the area and builds a "location map" of the environment, that can be saved using the **Save To AR Cloud** API.

After you've drawn and scanned around a bit, hit the top button (arrow pointing up) to save your map.

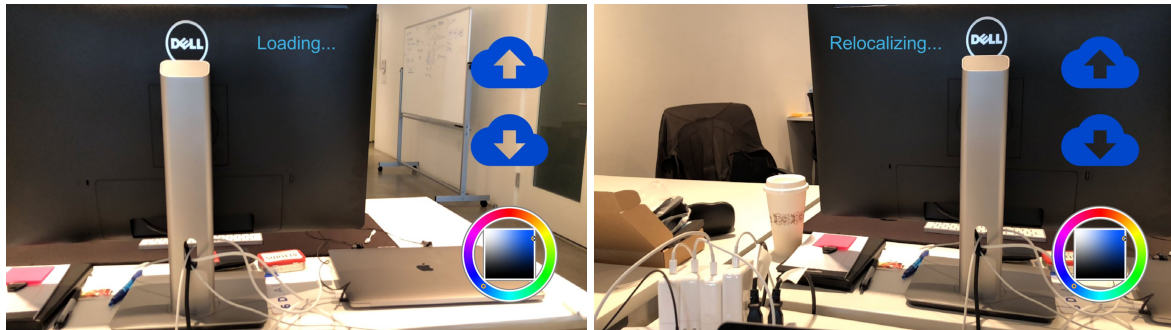


Messages at the top will appear and tell you when the map is successfully saved, indicating a location ID (or display errors if applicable).

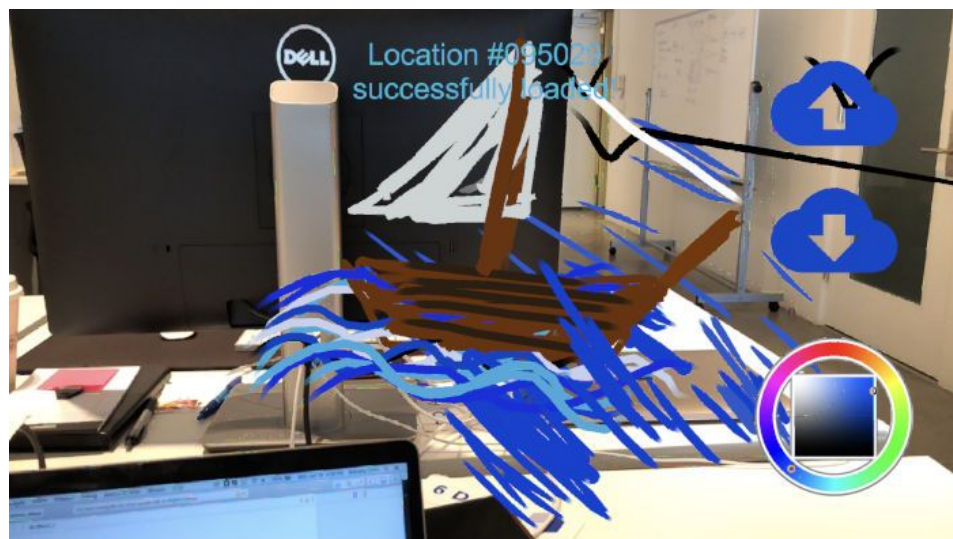
In this app, drawing data is also serialized and saved to our basic content server.

The first time you run the app, scan the area to capture as much of the environment around you as possible. Best results are obtained with SLAM friendly textures. The environment shown in the screenshots is particularly poor: low contrast, few details.

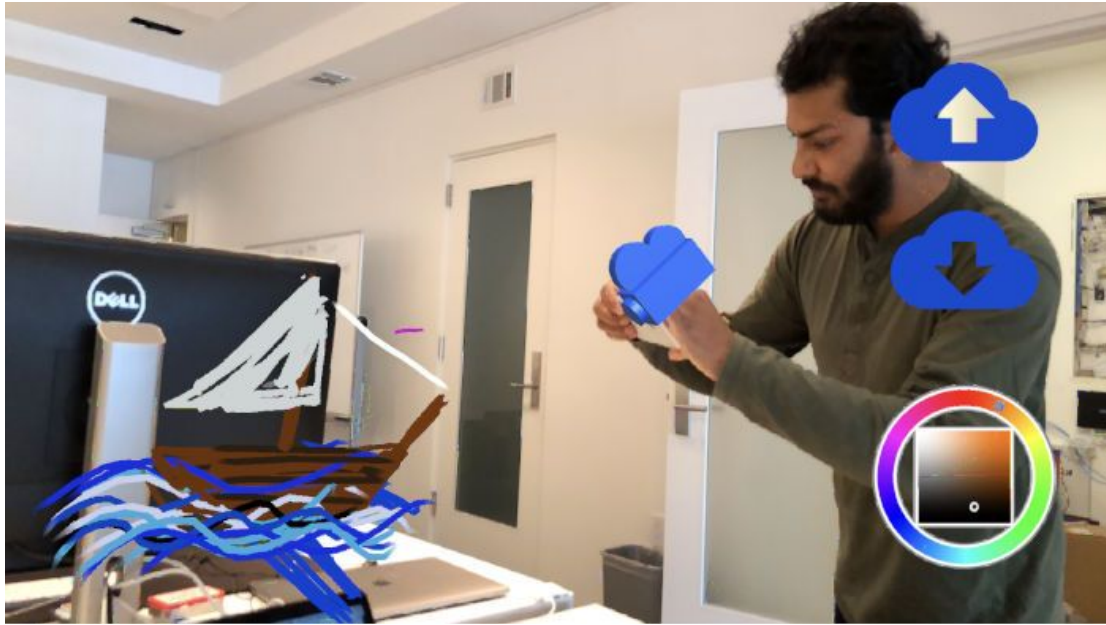
From that point, you can kill the app and open it again, or open it from a different device. Then you can hit the **Load from AR Cloud** button (arrow pointing down) to download the previously saved location.



Once the location data is downloaded, relocalization is attempted continuously, until the system successfully recognizes where it is in the saved location map data's coordinate system. After a successful relocalization, the app will download the location's saved drawing data and render it where it was painted. This is **persistence**.



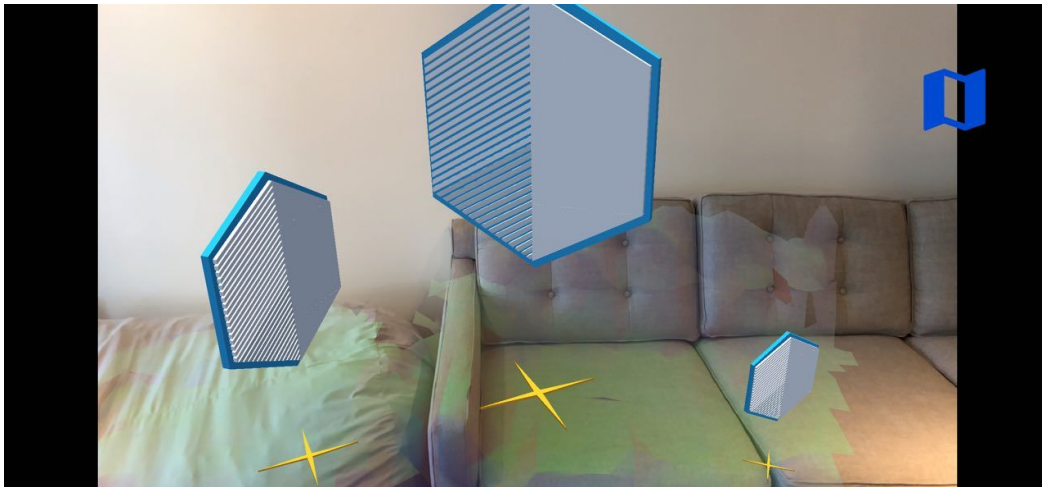
Additionally, when two devices relocalize against the same scene and share the same wifi network, they communicate their location to each other & drawing info. This is symbolized by a movie camera floating in real time at the location of the device. The color of that camera reflects the selected brush color of that player.



The distance between the physical device and the floating camera is indicative of the precision of that particular successful relocalization attempt. This is **multiplayer**.

How to Use the Meshing Sample scene

The scene presents itself as a camera viewport with a mesh toggle button, enabled by default, and three rotating 6D logos just like the Basic App.



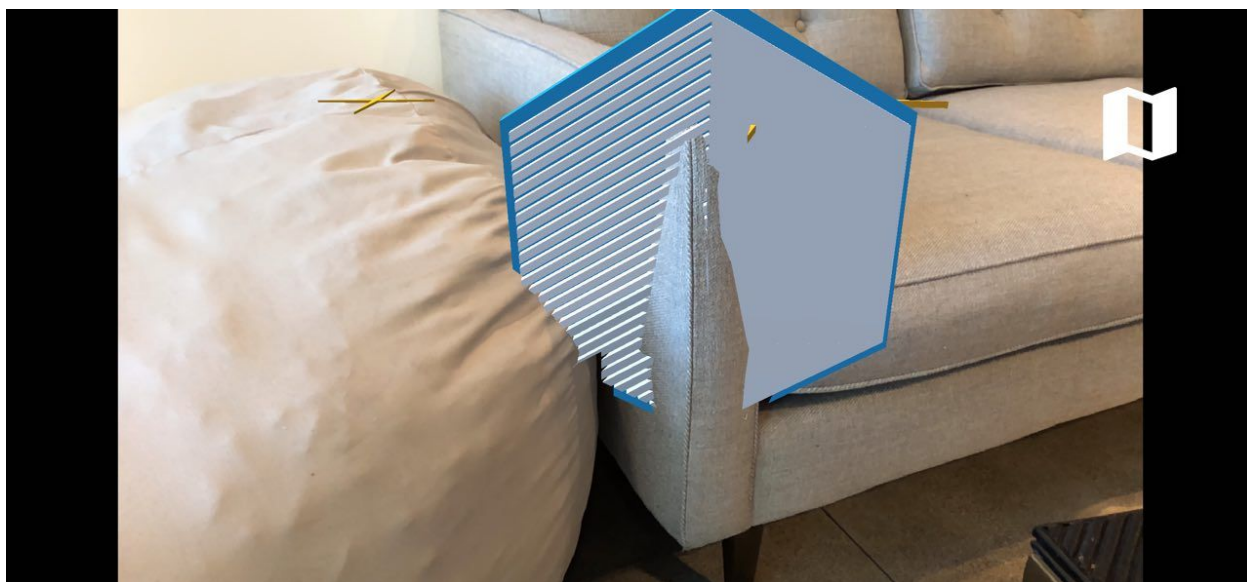
In addition, a colored tint starts spreading over objects: that's the real time world mesh.

In this application, you can drag the 6D logos around and reposition them.



You'll notice that the logos' rotating yellow star always rests on top of the mesh as you drag them, no matter how irregular the surface is. This is **real world interaction**.

Use the toggle mesh button to make the mesh rendering disappear and see how the virtual objects still interact and get occluded by, or intersected with, real objects like furniture.



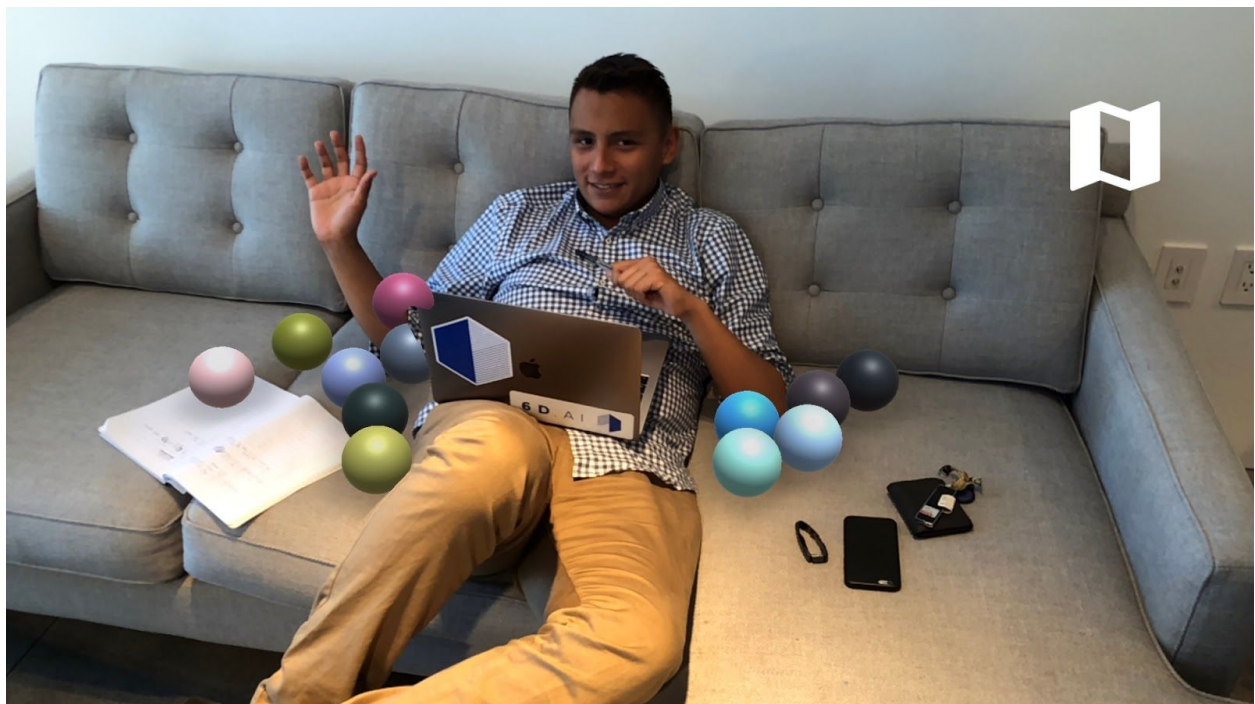
This is **occlusion**.

How to Use the Ball Pit scene

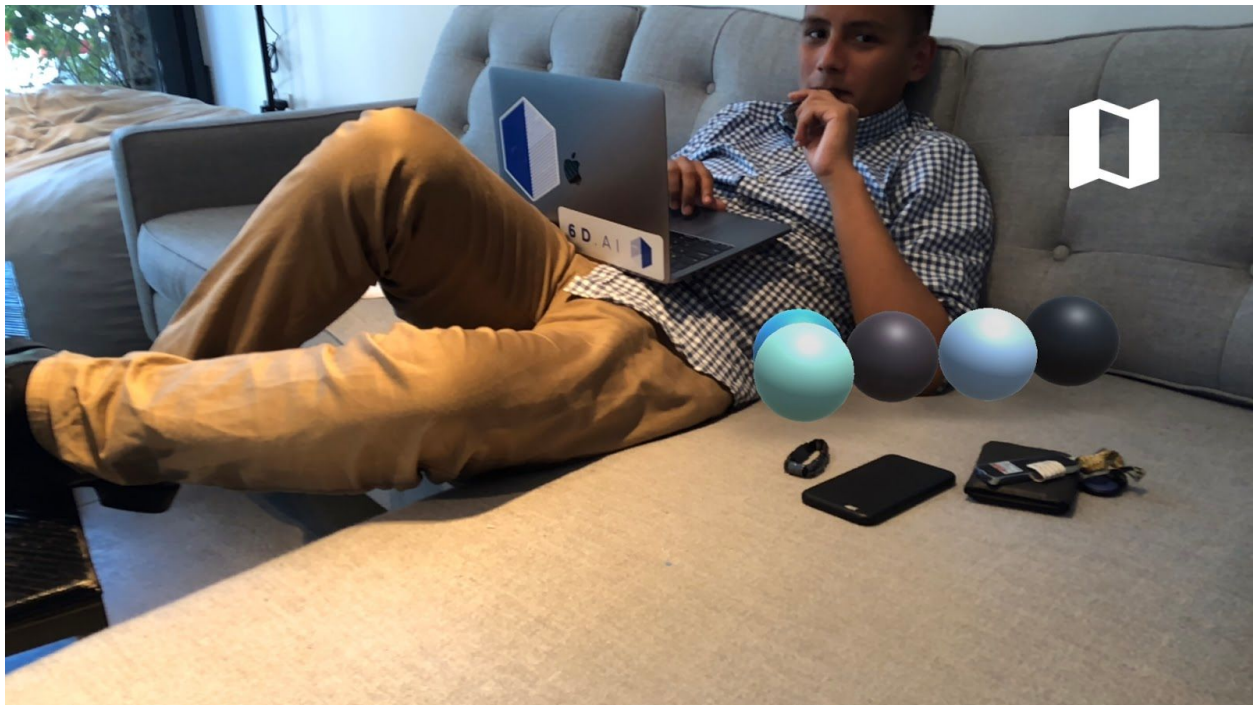
The scene presents itself as a camera viewport with a mesh toggle button, enabled by default. One can instantiate balls by tapping the screen anywhere. The ball will be instantiated at that position, and a force in the z direction will be added.



You can also toggle the mesh to make the mesh rendering disappear to get a more lifelike feel to the pseudo-game.



This app also supports occlusion (a real life object blocking the digital objects). The balls also cast shadows on the surface they are on.



We have the scene preset with a limit of thirty balls that can be instantiated before they start getting recycled (older ones get deleted); feel free to up this limit through Unity when you build the app. The purpose of this example is to show physics and interactions with the real world!

How to Use the Photon Drawing Sample scene

Before using this scene, ensure you have completed the [Photon setup](#) described above.

This scene presents itself identically to the [Drawing Sample scene](#).

You'll notice you can't draw anything from the get go. You will want to first save the map because we use the locationID as the room name for Photon.

Now you should be able to draw just like the previous drawing app. Also, anyone who wants to play multiplayer will just need to load the map, and they will automatically join the room you are in.

Now that you have the map saved for that location, the next time you start the app, you can either start from scratch by saving a new map, or load your existing map!

Using Photon, two players using cellular data can now play multiplayer in the same location since they will have the same locationID.

It should be noted that because the locationID depends on WiFi first, then geolocation, a player using WiFi and another using cellular data cannot play together, even if they are in the same location.

Meshing API Guide

Meshing is at the moment only generated on-the-fly, in the current AR session. In later SDK releases we'll include cloud-based meshing in order to allow sessions to use meshes generated from other devices.

`SixDegreesSDK_HasRealTimeMesh()` returns true if the current device can generate meshes on the fly.

Mesh information comes in three different buffers:

- Block buffer (int)
- Vertex buffer (float)
- Face buffer (int)

Those buffers must be allocated on the app side, using the sizes suggested by `SixDegreesSDK_GetMeshBlockInfo()`, and passed as pointers to `SixDegreesSDK_GetMeshBlocks()` to be populated.

A block is a cube of 22.4cm by 22.4cm by 22.4cm (8.82in); each triangle of the mesh is fully contained inside a block. The size of that cube is returned by `SixDegreesSDK_GetMeshBlockSize()`, expressed in meters (consistent with the coordinate system).

Each block is described in the block buffer by 6 consecutive integers: [x, y, z, vertex_count, face_count, version]. (x, y, z) represents the coordinates in space where the cube is, with (0, 0, 0) being the cube containing geometry between 0 and 0.224 on all 3 axes.

The vertex count and face count indicate how many vertices and triangles are contained in each block. The version value increments every time triangles in the block are updated.

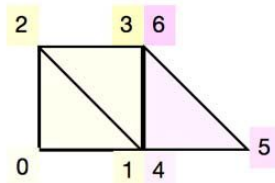
Each vertex is described by 6 consecutive floats: [x, y, z, normal_x, normal_y, normal_z]. (x, y, z) represents the vertex's coordinates in space, (normal_x, normal_y, normal_z) is its normal vector used for lighting and physics.

Each face is made of 3 consecutive indices [a, b, c], referring to 3 vertices, starting with 0 for the first one.

block buffer						
	x	y	z	vertex_count	face_count	version
block 0	0	0	0	4	2	1
block 1	1	0	0	3	1	1

vertex buffer						
	x	y	z	normal_x	normal_y	normal_z
vertex 0	0.0	0.0	0.0	0.0	1.0	0.0
vertex 1	0.224	0.0	0.0	0.0	1.0	0.0
vertex 2	0.0	0.0	0.224	0.0	1.0	0.0
vertex 3	0.224	0.0	0.224	0.0	1.0	0.0
vertex 4	0.224	0.0	0.0	0.0	1.0	0.0
vertex 5	0.448	0.0	0.0	0.0	1.0	0.0
vertex 6	0.224	0.0	0.224	0.0	1.0	0.0

face buffer			
	a	b	c
face 0	0	1	2
face 1	1	3	2
face 2	4	5	6



The diagram above details a simple example of block, vertex, and face buffers returned by `SixDegreesSDK_GetMeshBlocks()`. In this example, the mesh is a polygon spanning 2 blocks (= 12 int values), described by 7 vertices (= 42 floats) and 3 faces (= 9 int indices).

The first block (yellow), at coordinates (0, 0, 0), has a square made of 2 faces (= 6 indices) and 4 vertices (= 24 floats). The second block (pink), at coordinates (1, 0, 0), has a triangle made of 1 face (= 3 indices) and 3 vertices (= 18 floats).

You can notice that some vertices are duplicated (vertex 1 and 4, for instance), because they are part of a different block.

Blocks are 22.4cm a side because they are made of 8x8x8 voxels, a number that facilitates parallel processing at a voxel size that gives the best tradeoff of precision and accuracy with our current depth estimation.

The sample apps split meshes into chunks made of blocks so each Mesh object (in Unity) or geometry node (in SceneKit) is roughly 1 meter a side. This allows calls to `SixDegreesSDK_GetMeshBlocks()` to only update a subset of the total extent of the mesh, which isn't an issue for rendering but causes significant slowdowns for physics colliders.

Blocks also allow very efficient iterations through the mesh, for instance to find the Y value of the ground plane, without reading every single vertex.