ARKit Integration Guide

ARDemo

This project is a starting point showing how to integrate DMSDK and ARKit. It starts an ARSession and configures SceneKit integration with ARKit.

Getting Camera Frames from ARKit and Processing With DMSDK

To receive frames as ARFrames, the view controller sets itself as the ARSession's delegate, and implements the session (_ session: ARSession, didUpdate frame: ARFrame) method. These frames can be used with DMSDK's ImageReader class, which is designed specifically for doing detections one frame at a time.

When an ARFrame is provided to the delegate method, a CMSampleBuffer image is obtained using the captureImage property of the frame. The frame then has a detection performed on it by the DMSDK ImageReader, and any results are returned.

Retrieving Detection Locations From DMSDK

Once a result is returned, the application will need to know where in the frame the detection happened to enable AR content to be placed properly. The detection region can be obtained using the result's detectionRegion function. This region is provided by DMSDK as a path, but it can be converted to a point by first converting it to a rect using the boundingBox function of CGPath, and then using midX and midY to obtain a point that lies in the middle of the path.

Translating Between Coordinate Spaces

DMSDK and AVFoundation work in a normalized 2D space, where all coordinates are between 0.0-1.0, representing a location as a percentage of the camera frame. Because ARKit works in view coordinates, these normalized frame coordinates need to be translated to coordinates in the AR view's coordinate space.

ARKit can provide a translation matrix to change frame coordinate space to the view coordinate views, and DMSDK's normalized coordinate can be translated into ARKit space using the following code:

```
let midX = path.boundingBox.midX
let midY = path.boundingBox.midY
var point = CGPoint(x: midX, y: midY)
point.y *= CGFloat(self.sceneView.bounds.height)
point.x *= CGFloat(self.sceneView.bounds.width)
```

After this code is executed, point will be a value in the AR view's coordinate space that is suitable for hit testing and placing content. The demo application uses this technique to show how to use ARKit's hit testing and placement.

Tuning For Specific Use Cases

ARKit is a large, evolving framework that provides many ways to build an AR app. This demo is a basic integration, and your use case may require a different way of using ARKit. This demo is not an exhaustive demo of ARKit, ARKit contains many features and developers are encouraged to study the ARKit documentation.

World Tracking

This demo makes use of ARKit's world tracking feature to try to keep an object's position stable relative to the rest of the scene. ARKit will map the world around the phone in an attempt to better understand how the scene is constructed.

World tracking usually provides a more stable AR experience, but the constraints of ARKit can cause several challenges:

- Until world tracking has enough information to create an initial scene understanding, any objects placed in the scene may have an unreliable position as scene understanding shifts. The AR sample deals with this issue by pausing Digimarc processing and presenting a user prompt until world tracking is complete.
- World tracking can become confused by repeated content or a lack of detail in the world around it. We have observed this when the phone is pointed at similar objects like boxes on a store shelf. This can also cause content placed in the scene to shift its position.

In cases like this, turning world tracking off may provide a better experience. Without world tracking, it is more difficult to place objects that need to be tightly integrated into the scene but it reduces the chance of objects moving in the scene. Refer to the iOS Developer ARKit documentation to learn more about world tracking and different hit testing techniques, and experiment with which one works best.

Anchoring Content To Moving Objects

Tracking Digimarc-enhanced objects while they are moving can be challenging under the current version of ARKit. The ImageReader will provide new locations in every frame it detects Digimarc content, which will track an object as it moves. However, because a Digimarc Barcode typically covers the entire surface of an enhanced object the detected location may jump between different locations on the object. This variability can make it difficult for AR content to continuously track a Digimarc-enhanced object as it moves in the real world. We recommend the following two approaches to improve the experience:

- If ARKit's image recognition or 3D object recognition is suitable for your application, try using these in addition to DMSDK's Digimarc Barcode detector. DMSDK can provide product or serialization metadata, while ARKit itself handles tracking the actual object's movements in the real world.
- Another approach is to track the distance between different detections, and if a maximum distance is exceeded, treat the new result as movement of the real object instead of just variance in detector results.

This is an area we will optimize with future releases of ARKit. In the meantime, experiment with the above approaches and consider designing your AR experience in a way that is not reliant on tracking physical object movement.

Capture Resolution

DMSDK prefers a capture resolution of 2592x1944 (or 2592x1936.) While the DMSDK will support lower resolutions, the preferred resolutions provide a good amount of detail for the optimal Digimarc Barcode detection experience.

Currently, ARKit capture resolution is approximately 1080p. If a future version of ARKit allows for higher capture resolutions, use of those resolutions will improve the DMSDK experience with ARKit.