

[ARKit](#) / [ARKit in visionOS](#) / Detecting Images in an AR Experience

Sample Code

Detecting Images in an AR Experience

React to known 2D images in the user's environment, and use their positions to place AR content.

[Download](#)

iOS 11.3+ | iPadOS 11.3+ | Xcode 16.0+

Overview

Many AR experiences can be enhanced by using known features of the user's environment to trigger the appearance of virtual content. For example, a museum app might show a virtual curator when the user points their device at a painting, or a board game might place virtual pieces when the player points their device at a game board. In iOS 11.3 and later, you can add such features to your AR experience by enabling image detection in ARKit: Your app provides known 2D images, and ARKit tells you when and where those images are detected during an AR session.

This example app looks for any of the several reference images included in the app's asset catalog. When ARKit detects one of those images, the app shows a message identifying the detected image and a brief animation showing its position in the scene.

Important

The images included with this sample are designed to fit the screen sizes of various Apple devices. To try the app using these images, choose an image that fits any spare device you have, and display the image full screen on that device. Then run the sample code project on a different device, and point its camera at the device displaying the image. Alternatively, you can add your own images; see the steps in [Provide Your Own Reference Images](#), below.

Enable Image Detection

Image detection is an add-on feature for world-tracking AR sessions. (For more details on world tracking, see [Tracking and visualizing planes](#).)

To enable image detection:

1. Load one or more [ARReferenceImage](#) resources from your app's asset catalog.
2. Create a world-tracking configuration and pass those reference images to its [detectionImages](#) property.
3. Use the [run\(:options:\)](#) method to run a session with your configuration.

The code below shows how the sample app performs these steps when starting or restarting the AR experience.

```
guard let referenceImages = ARReferenceImage.referenceImages(inGroupNamed: "AR Resources",
                                                                bundle: nil) else {
    fatalError("Missing expected asset catalog resources.")
}

let configuration = ARWorldTrackingConfiguration()
configuration.detectionImages = referenceImages
session.run(configuration, options: [.resetTracking, .removeExistingAnchors])
```

Visualize Image Detection Results

When ARKit detects one of your reference images, the session automatically adds a corresponding [ARImageAnchor](#) to its list of anchors. To respond to an image being detected, implement an appropriate [ARSessionDelegate](#), [ARSKViewDelegate](#), or [ARSCNViewDelegate](#) method that reports the new anchor being added to the session. (This example app uses the [renderer\(_:didAdd:for:\)](#) method for the code shown below.)

To use the detected image as a trigger for AR content, you'll need to know its position and orientation, its size, and which reference image it is. The anchor's inherited [transform](#) property provides position and orientation, and its [referenceImage](#) property tells you which [ARReferenceImage](#) object was detected. If your AR content depends on the extent of the image in the scene, you can then use the reference image's [physicalSize](#) to set up your content, as shown in the code below.

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    guard let imageAnchor = anchor as? ARImageAnchor else { return }
    let referenceImage = imageAnchor.referenceImage
    updateQueue.async {
```

```

// Create a plane to visualize the initial position of the detected image.
let plane = SCNPlane(width: referenceImage.physicalSize.width,
                      height: referenceImage.physicalSize.height)
let planeNode = SCNNode(geometry: plane)
planeNode.opacity = 0.25

/*
   `SCNPlane` is vertically oriented in its local coordinate space, but
   `ARImageAnchor` assumes the image is horizontal in its local space, so
   rotate the plane to match.
*/
planeNode.eulerAngles.x = -.pi / 2

/*
   Image anchors are not tracked after initial detection, so create an
   animation that limits the duration for which the plane visualization app
   */
planeNode.runAction(self.imageHighlightAction)

// Add the plane visualization to the scene.
node.addChildNode(planeNode)
}

DispatchQueue.main.async {
let imageName = referenceImage.name ?? ""
self.statusViewController.cancelAllScheduledMessages()
self.statusViewController.showMessage("Detected image \"\((imageName)\"")
}
}

```

Provide Your Own Reference Images

To use your own images for detection (in this sample or in your own project), you'll need to add them to your asset catalog in Xcode.

1. Open your project's asset catalog, then use the Add button (+) to add a new AR resource group.
2. Drag image files from the Finder into the newly created resource group.
3. For each image, use the inspector to describe the physical size of the image as you'd expect to find it in the user's real-world environment, and optionally include a descriptive name for your own use.

Note

Put all the images you want to look for in the same session into a resource group. Use separate resource groups to hold sets of images for use in separate sessions. For example, an art museum app might use separate sessions (and thus separate resource groups) for detecting paintings in different wings of the museum.

Be aware of image detection capabilities. Choose, design, and configure reference images for optimal reliability and performance:

- Enter the physical size of the image in Xcode as accurately as possible. ARKit relies on this information to determine the distance of the image from the camera. Entering an incorrect physical size will result in an `ARImageAnchor` that's the wrong distance from the camera.
- When you add reference images to your asset catalog in Xcode, pay attention to the quality estimation warnings Xcode provides. Images with high contrast work best for image detection.
- Use only images on flat surfaces for detection. If an image to be detected is on a nonplanar surface, like a label on a wine bottle, ARKit might not detect it at all, or might create an image anchor at the wrong location.
- Consider how your image appears under different lighting conditions. If an image is printed on glossy paper or displayed on a device screen, reflections on those surfaces can interfere with detection.

Apply Best Practices

This example app simply visualizes where ARKit detects each reference image in the user's environment, but your app can do much more. Follow the tips below to design AR experiences that use image detection well.

Use detected images to set a frame of reference for the AR scene. Instead of requiring the user to choose a place for virtual content, or arbitrarily placing content in the user's environment, use detected images to anchor the virtual scene. You can even use multiple detected images. For example, an app for a retail store could make a virtual character appear to emerge from a store's front door by detecting posters placed on either side of the door and then calculating a position for the character directly between the posters.

Note

Use the `setWorldOrigin(relativeTransform:)` method to redefine the world coordinate system so that you can place all anchors and other content relative to the reference point you choose.

Design your AR experience to use detected images as a starting point for virtual content. ARKit doesn't track changes to the position or orientation of each detected image. If you try to place virtual content that stays attached to a detected image, that content may not appear to stay in place correctly. Instead, use detected images as a frame of reference for starting a dynamic scene. For example, your app might detect theater posters for a sci-fi film and then have virtual spaceships appear to emerge from the posters and fly around the environment.

Consider when to allow detection of each image to trigger (or repeat) AR interactions. ARKit adds an image anchor to a session exactly once for each reference image in the session configuration's detectionImages array. If your AR experience adds virtual content to the scene when an image is detected, that action will by default happen only once. To allow the user to experience that content again without restarting your app, call the session's remove(anchor:) method to remove the corresponding ARImageAnchor. After the anchor is removed, ARKit will add a new anchor the next time it detects the image.

For example, in the case described above, where spaceships appear to fly out of a movie poster, you might not want an extra copy of that animation to appear while the first one is still playing. Wait until the animation ends to remove the anchor, so that the user can trigger it again by pointing their device at the image.