Sample Code

Capturing depth using the LiDAR camera

Access the LiDAR camera on supporting devices to capture precise depth data.

Download

iOS 15.6+ | iPadOS 15.6+ | Xcode 16.0+

Overview

AVFoundation introduced depth data capture for photos and video in iOS 11. The data it provides is suitable for many apps, but may not meet the needs of those that require greater precision depth. Starting in iOS 15.4, you can access the LiDAR camera on supported hardware, which offers high-precision depth data suitable for use cases like room scanning and measurement.

This sample code project shows how to capture and render depth data from the LiDAR camera. It starts in streaming mode, which demonstrates how to capture synchronized video and depth data. When you tap the Camera button in the upper-left corner of the screen, the app switches to photo mode, which illustrates how to capture photos with depth data. In both modes, the app provides several Metal-based visualizations of the depth and image data.

Configure the sample code project

Run this sample code on a device that provides a LiDAR camera, such as:

- iPhone 12 Pro or later
- iPad Pro 11-inch (3rd generation) or later
- iPad Pro 12.9-inch (5th generation) or later

Configure the LiDAR camera

The sample app's `CameraController` class provides the code that configures and manages the capture session, and handles the delivery of new video and depth data. It begins its configuration by retrieving the LiDAR camera. It calls the capture device's `default( :for:position:)` class method, passing it the new `builtInLiDARDepthCamera` device type available in iOS 15.4 and later.

```
// Look up the LiDAR camera.
guard let device = AVCaptureDevice.default(.builtInLiDARDepthCamera, for: .video, po
    throw ConfigurationError.lidarDeviceUnavailable
}
```

After retrieving the device, the app configures it with a specific video and depth format. It asks the device for its supported formats and finds the best nonbinned, full-range YUV color format that matches the sample app's preferred width and supports depth capture. Finally, it sets the active formats on the device as in the following code example:

```
// Find a match that outputs video data in the format the app's custom Metal views r
guard let format = (device.formats.last { format in
    format.formatDescription.dimensions.width == preferredWidthResolution &&
    format.formatDescription.mediaSubType.rawValue == kCVPixelFormatType_420YpCbCr8B
    !format.isVideoBinned &&
    !format.supportedDepthDataFormats.isEmpty
}) else {
    throw ConfigurationError.requiredFormatUnavailable
}

// Find a match that outputs depth data in the format the app's custom Metal views r
guard let depthFormat = (format.supportedDepthDataFormats.last { depthFormat in
    depthFormat.formatDescription.mediaSubType.rawValue == kCVPixelFormatType_DepthF
}) else {
    throw ConfigurationError.requiredFormatUnavailable
}

// Begin the device configuration.
try device.lockForConfiguration()

// Configure the device and depth formats.
device.activeFormat = format
device.activeDepthDataFormat = depthFormat

// Finish the device configuration.
device.unlockForConfiguration()
```

Configure the capture outputs

The app operates in streaming or photo mode. To enable streaming output, it creates an instance of AVCaptureVideoDataOutput and AVCaptureDepthDataOutput to capture video sample buffers and depth data, respectively. It configures them as follows:

```
// Create an object to output video sample buffers.
videoDataOutput = AVCaptureVideoDataOutput()
captureSession.addOutput(videoDataOutput)

// Create an object to output depth data.
depthDataOutput = AVCaptureDepthDataOutput()
depthDataOutput.isFilteringEnabled = isFilteringEnabled
captureSession.addOutput(depthDataOutput)
```

```
// Create an object to synchronize the delivery of depth and video data.
outputVideoSync = AVCaptureDataOutputSynchronizer(dataOutputs: [depthDataOutput, vid
outputVideoSync.setDelegate(self, queue: videoQueue)
```

Because the video and depth data stream from separate output objects, the sample uses an `AVCaptureDataOutputSynchronizer` to synchronize the delivery from both outputs to a single callback. The `CameraController` class adopts the synchronizer's `AVCaptureDataOutputSynchronizerDelegate` protocol and responds to the delivery of new video and depth data.

To handle photo capture, the app also creates an instance of `AVCapturePhotoOutput`. It optimizes the output for high-quality capture and adds the output to the capture session.

```
// Create an object to output photos.
photoOutput = AVCapturePhotoOutput()
photoOutput.maxPhotoQualityPrioritization = .quality
captureSession.addOutput(photoOutput)

// Enable delivery of depth data after adding the output to the capture session.
photoOutput.isDepthDataDeliveryEnabled = true
```

After it adds the output to the session, it enables the delivery of depth data, which configures the capture pipeline appropriately. It can only enable depth delivery after adding the output to the capture session because the output needs to determine whether the pipeline configuration can deliver it.

Capture synchronized video and depth

After configuring the capture session's inputs and outputs as required, the app is ready to start capturing data. The app starts in streaming mode, which uses the video data and depth data outputs and an `AVCaptureDataOutputSynchronizer` to synchronize the delivery of their data. The app adopts the synchronizer's delegate protocol and implements its `dataOutputSynchronizer(_:didOutput:)` method to handle the delivery, as the following example shows:

```
func dataOutputSynchronizer(_ synchronizer: AVCaptureDataOutputSynchronizer,
                            didOutput synchronizedDataCollection: AVCaptureSynchroni
    // Retrieve the synchronized depth and sample buffer container objects.
    guard let syncedDepthData = synchronizedDataCollection.synchronizedData(for: dep
          let syncedVideoData = synchronizedDataCollection.synchronizedData(for: vid
```

```
    guard let pixelBuffer = syncedVideoData.sampleBuffer.imageBuffer,
          let cameraCalibrationData = syncedDepthData.depthData.cameraCalibrationDat

    // Package the captured data.
    let data = CameraCapturedData(depth: syncedDepthData.depthData.depthDataMap.text
                                  colorY: pixelBuffer.texture(withFormat: .r8Unorm,
                                  colorCbCr: pixelBuffer.texture(withFormat: .rg8Unc
                                  cameraIntrinsics: cameraCalibrationData.intrinsicM
                                  cameraReferenceDimensions: cameraCalibrationData.i

    delegate?.onNewData(capturedData: data)
}
```

The app retrieves the container objects that store the synchronized data from the `AVCaptureSynchronizedDataCollection`. It then unwraps the underlying video pixel buffer and depth data, and packages them for the app's Metal views to display.

Capture photos and depth

When you tap the app's Camera button in the upper-left corner of the user interface, the app switches to photo mode. When this occurs, the app calls its `capturePhoto()` method, which creates a photo settings object, requests depth delivery on it, and initiates a photo capture.

```
func capturePhoto() {
    var photoSettings: AVCapturePhotoSettings
    if  photoOutput.availablePhotoPixelFormatTypes.contains(kCVPixelFormatType_420Yp
        photoSettings = AVCapturePhotoSettings(format: [
            kCVPixelBufferPixelFormatTypeKey as String: kCVPixelFormatType_420YpCbCr
        ])
    } else {
        photoSettings = AVCapturePhotoSettings()
    }

    // Capture depth data with this photo capture.
    photoSettings.isDepthDataDeliveryEnabled = true
    photoOutput.capturePhoto(with: photoSettings, delegate: self)
}
```

When the framework finishes the photo capture, it calls the photo output's delegate method and passes it the `AVCapturePhoto` object that contains the image and depth data. The sample retrieves the data from the photo, stops the stream until the user returns to streaming mode, and,

similarly to the video case, packages the captured data for delivery to the app's user interface layer.

```
func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVC

    // Retrieve the image and depth data.
    guard let pixelBuffer = photo.pixelBuffer,
          let depthData = photo.depthData,
          let cameraCalibrationData = depthData.cameraCalibrationData else { return

    // Stop the stream until the user returns to streaming mode.
    stopStream()

    // Convert the depth data to the expected format.
    let convertedDepth = depthData.converting(toDepthDataType: kCVPixelFormatType_De

    // Package the captured data.
    let data = CameraCapturedData(depth: convertedDepth.depthDataMap.texture(withFor
                                  colorY: pixelBuffer.texture(withFormat: .r8Unorm,
                                  colorCbCr: pixelBuffer.texture(withFormat: .rg8Unc
                                  cameraIntrinsics: cameraCalibrationData.intrinsicM
                                  cameraReferenceDimensions: cameraCalibrationData.i

    delegate?.onNewPhotoData(capturedData: data)
}
```

See Also

Depth data capture

- Capturing photos with depth

  Get a depth map with a photo to create effects like the system camera's Portrait mode (on compatible devices).

- Creating auxiliary depth data manually

  Generate a depth image and attach it to your own image.

- {} AVCamFilter: Applying filters to a capture stream

  Render a capture stream with rose-colored filtering and depth effects.

{} Streaming depth data from the TrueDepth camera

Visualize depth data in 2D and 3D from the TrueDepth camera.

{} Enhancing live video by leveraging TrueDepth camera data

Apply your own background to a live capture feed streamed from the front-facing TrueDepth camera.

`class AVCaptureDepthDataOutput`

A capture output that records scene depth information on compatible camera devices.

`class AVDepthData`

A container for per-pixel distance or disparity information captured by compatible camera devices.

`class AVCameraCalibrationData`

Information about the camera characteristics used to capture images and depth data.