Sample Code

Writing spatial photos

Create spatial photos for visionOS by packaging a pair of left- and right-eye images as a stereo HEIC file with related spatial metadata.

Download

macOS 14.0+ | Xcode 16.0+

Overview

Spatial photos are *High-Efficiency Image Codec* (HEIC) files containing a pair of left- and right-eye images, together with a stereo pair group and additional spatial metadata. *Spatial metadata* describes properties of the left- and right-eye cameras that captured the stereo scene. Adding spatial metadata to a stereo HEIC prompts Apple platforms to consider the image as *spatial* instead of just stereo, and opts the image into visual treatments on Apple Vision Pro that help minimize common causes of stereo viewing discomfort.

To learn more about why you might want to package a pair of stereo images as a spatial photo and the metadata values you provide, see Creating spatial photos and videos with spatial metadata.

This sample app demonstrates how to use Image I/O APIs to convert two separate images into a spatial photo, saving the output as a HEIC file. The app's `SpatialPhotoConverter` class performs this conversion process.

> Note
>
> This sample app is a macOS command-line application, but you can also perform the spatial photo conversion process on other Apple platforms such as visionOS and iOS.

Configure the sample code project

To run the app in Xcode, add the input file locations, output file location, and spatial metadata for your images as arguments to the project's scheme. Select Product > Scheme > Edit Scheme (Command-<), and add the arguments described below to Arguments Passed On Launch:

- `--leftImage` (or `-l`) followed by a path to a local left-eye image file
- `--rightImage` (or `-r`) followed by a path to a local right-eye image file
- `--outputImage` (or `-o`) followed by the output path for the spatial photo, with a file extension of `.heic` (for example, `~/Documents/spatial_photo.heic`)
- `--baseline` (or `-b`) to provide a baseline in millimeters (for example, `--baseline 64.0` for a 64mm baseline)
- `--fov` (or `-f`) to provide a horizontal field of view in degrees (for example, `--fov 80.0` for an 80-degree field of view)
- `--disparityAdjustment` (or `-d`) to provide a disparity adjustment value as a fraction of the image's width (for example, `--disparityAdjustment 0.02` for a 2% positive disparity shift)

By default, the project's scheme loads a pair of left- and right-eye images from the Xcode project folder: `Hummingbird_Left.png` and `Hummingbird_Right.png`. These images are renders of a 3D scene of a hummingbird model. The virtual cameras that created these renders were positioned 64mm apart, with a horizontal field of view of 80 degrees, which means the value for the `--baseline` argument is `64.0`, and the value of the `--fov` argument is `80.0`.

For these images, a disparity adjustment of +2% of the image width produces a comfortable depth effect when the spatial photo is presented in a window on Apple Vision Pro. This 2% disparity adjustment value positions the nearest object in the spatial photo — the hummingbird — just behind the front of the window, while still keeping an effective illusion of depth between the hummingbird and the background. The scheme's arguments express the +2% disparity adjustment with a `--disparityAdjustment` value of `0.02`.

Load the left- and right-eye images

The app starts by creating a `CGImageSource` for each of the left- and right-eye images. An initializer on `StereoPairImage` performs the following steps:

1. Opens the image as a `CGImageSource` by calling `CGImageSourceCreateWithURL`
2. Discovers the primary image index for the image source by calling `CGImageSourceGetPrimaryImageIndex`
3. Discovers the pixel width and height of the image by calling `CGImageSourceCopyPropertiesAtIndex` for the primary image index, and looking for `kCGImagePropertyPixelWidth` and `kCGImagePropertyPixelHeight` values in the returned properties dictionary

4. Stores the results of the above steps as properties on the `StereoPairImage`

```
init(url: URL) throws {

    guard let source = CGImageSourceCreateWithURL(url as CFURL, nil) else {
        throw ConversionError.couldNotOpenURLAsImageSource
    }
    self.source = source

    primaryImageIndex = CGImageSourceGetPrimaryImageIndex(source)

    guard let properties = CGImageSourceCopyPropertiesAtIndex(source, primaryImageIr
        throw ConversionError.couldNotCopyImageProperties
    }
    guard let width = properties[kCGImagePropertyPixelWidth] as? Int,
        let height = properties[kCGImagePropertyPixelHeight] as? Int else {
        throw ConversionError.unableToReadImageSize
    }
    self.width = width
    self.height = height

}
```

The app next validates that the left and right `StereoPairImage` have the same pixel width and height.

```
guard leftImage.width == rightImage.width, leftImage.height == rightImage.height els
    throw ConversionError.leftAndRightImageSizesDoNotMatch
}
```

Important

The left- and right-eye images in a spatial photo must have the same pixel size.

Define an extrinsic position for each image

The app converts the provided baseline distance from millimeters to meters.

```swift
// Convert the baseline from millimeters to meters.
let baselineInMeters = baselineInMillimeters / 1000.0
```

Then, the app uses this baseline distance to define a left extrinsic position (at the origin) and a right extrinsic position (offset from the origin by `baselineInMeters` in the positive x-axis) to describe how the left and right cameras are positioned relative to each other in 3D space.

```swift
let leftPosition: [Double] = [0, 0, 0]
let rightPosition: [Double] = [baselineInMeters, 0, 0]
```

Important

The distance in 3D space between the left- and right-eye camera extrinsic positions must always represent the baseline in meters between the two cameras that captured the images. The position of the right-eye camera must always be positioned to the right of the left-eye camera in the left-eye camera's coordinate space.

Compute a camera intrinsics matrix

The app uses the provided horizontal field of view, plus the known width and height of each input image, to compute a 3x3 camera intrinsics matrix that describes the characteristics of the left and right cameras.

```swift
let intrinsics = leftImage.intrinsics(horizontalFOV: horizontalFOV)
```

The app calculates the 3x3 intrinsics matrix using a convenience method, `intrinsics(horizontalFOV:)`, on `StereoPairImage`. This method defines an intrinsics matrix with the same focal length in both X and Y (to represent a spherical lens), with the principal point of the camera located at the center of the image, and no shear.

```swift
func intrinsics(horizontalFOV: Double) -> [Double] {
    let width = Double(width)
    let height = Double(height)
    let horizontalFOVInRadians = horizontalFOV / 180.0 * .pi
    let focalLengthX = (width * 0.5) / (tan(horizontalFOVInRadians * 0.5))
    // For a spherical pinhole camera, the focal length is the same in both X and Y.
    let focalLengthY = focalLengthX
    // The app assumes the principal point of the camera is located at the center of
    let principalPointX = 0.5 * width
    let principalPointY = 0.5 * height
    return [
        focalLengthX, 0, principalPointX,
        0, focalLengthY, principalPointY,
        0, 0, 1
    ]
}
```

The app already validated the left-eye and right-eye images to confirm that they have the same pixel size. As a result, the app calculates the intrinsics matrix for the left-eye image and uses it as the intrinsics matrix for both images below.

Encode the disparity adjustment

The app encodes the provided disparity adjustment value as a signed integer, converting it from a signed floating-point value in the range `[-1.0, +1.0]` (as a fraction of image width) to a signed integer value in the range `[-10000, +10000]`.

```
let encodedDisparityAdjustment = Int(disparityAdjustment * 1e4)
```

Define an image properties dictionary for each image

Before adding the left-eye and right-eye images to the image destination, the app defines a properties dictionary for each image, which expresses the spatial metadata for that image as expected ImageIO properties.

```
let leftProperties = propertiesDictionary(
    isLeft: true,
    encodedDisparityAdjustment: encodedDisparityAdjustment,
    position: leftPosition,
    intrinsics: intrinsics
)
let rightProperties = propertiesDictionary(
    isLeft: false,
    encodedDisparityAdjustment: encodedDisparityAdjustment,
    position: rightPosition,
    intrinsics: intrinsics
)
```

A convenience method, `propertiesDictionary(isLeft:encodedDisparityAdjustment:position:intrinsics:)`, defines these property dictionaries.

```
func propertiesDictionary(
    isLeft: Bool,
    encodedDisparityAdjustment: Int,
    position: [Double],
    intrinsics: [Double]
) -> [CFString: Any] {
    return [
        kCGImagePropertyGroups: [
            kCGImagePropertyGroupIndex: 0,
            kCGImagePropertyGroupType: kCGImagePropertyGroupTypeStereoPair,
            (isLeft ? kCGImagePropertyGroupImageIsLeftImage : kCGImagePropertyGroupI
            kCGImagePropertyGroupImageDisparityAdjustment: encodedDisparityAdjustmen
        ],
        kCGImagePropertyHEIFDictionary: [
            kIIOMetadata_CameraExtrinsicsKey: [
                kIIOCameraExtrinsics_Position: position,
                kIIOCameraExtrinsics_Rotation: Self.identityRotation
            ],
            kIIOMetadata_CameraModelKey: [
                kIIOCameraModel_Intrinsics: intrinsics,
                kIIOCameraModel_ModelType: kIIOCameraModelType_SimplifiedPinhole
            ]
        ],
        kCGImagePropertyHasAlpha: false
    ]
}
```

The properties dictionary has two sub-dictionaries for each image:

- A groups dictionary (`kCGImagePropertyGroups`) that defines the image as part of a stereo pair group, and specifies a disparity adjustment for that group
- A HEIF dictionary (`kCGImagePropertyHEIFDictionary`) that defines the extrinsics and camera model for the camera that created the image

For the groups dictionary, the convenience method defines a single stereo pair group with the following properties:

- A `kCGImagePropertyGroupIndex` of `0`, because this is the first and only group in the output file

- A `kCGImagePropertyGroupType` of `kCGImagePropertyGroupTypeStereoPair` to indicate that this group defines a stereo pair of images
- A flag indicating if this image is the left-eye image (`kCGImagePropertyGroupImageIsLeftImage`) or the right-eye image (`kCGImagePropertyGroupImageIsRightImage`) in the stereo pair group
- The `kCGImagePropertyGroupImageDisparityAdjustment` to use when presenting this stereo pair group

Important

The stereo pair group for a spatial photo must always define a disparity adjustment offset. Disparity adjustment and group index are provided as part of the image properties for each image, so the app includes the same index and disparity adjustment value in the groups dictionary for both the left- and right-eye images.

For the HEIF dictionary, the convenience method defines:

- A camera extrinsics (`kIIOMetadata_CameraExtrinsicsKey`) dictionary that contains the extrinsic position and rotation for the camera
- A camera model (`kIIOMetadata_CameraModelKey`) dictionary that contains the camera intrinsics matrix and camera model type for the camera that captured the image

The camera extrinsics dictionary specifies an identity rotation to indicate that the app defines camera extrinsics with a position offset only.

Important

The camera model dictionary for the left- and right-eye images in a spatial photo must always define a camera model type of either `kIIOCameraModelType_SimplifiedPinhole` or `kIIOCameraModelType_GenericPinhole`.

The properties dictionary also specifies a `kCGImagePropertyHasAlpha` value of `false` to indicate that the system should ignore any alpha channel data in the source image when adding that source image to the image destination.

Write the images to an output image destination

The app calls `CGImageDestinationCreateWithURL` to create an output `CGImageDestination` at the provided URL for the output spatial photo. The app creates the image destination with the uniform type identifier for a HEIC image and an expected image count of 2 to indicate that the app writes both a left- and right-eye image to a single HEIC file.

```
let destinationProperties: [CFString: Any] = [kCGImagePropertyPrimaryImage: 0]
guard let destination = CGImageDestinationCreateWithURL(
    outputImageURL as CFURL,
    UTType.heic.description as CFString,
    2,
    destinationProperties as CFDictionary
) else {
    throw ConversionError.unableToCreateImageDestination
}
```

The app creates an image destination with a `destinationProperties` dictionary that specifies a primary image index of `0` for the output HEIC file. This primary image index specifies which image in the output HEIC file is preferred for display when an app or system needs a single image to represent the file's content (for example, on a nonstereo platform such as iOS). However, not all apps and operating systems use the primary image index when displaying a HEIC file, and instead display the first image in the HEIC by default. For this reason, the app sets the primary image index to `0`, so that apps that use the primary image index select the same image as apps that don't.

Next, the app calls `CGImageDestinationAddImageFromSource` to copy the left- and right-eye image sources into the image destination, passing in an appropriate properties dictionary for each image. The app adds the left-eye image first, which means the left-eye image is the primary image for the output HEIC file, because it appears at image index `0` in the output file.

```
CGImageDestinationAddImageFromSource(
    destination,
    leftImage.source,
    leftImage.primaryImageIndex,
    leftProperties as CFDictionary
)
CGImageDestinationAddImageFromSource(
    destination,
    rightImage.source,
    rightImage.primaryImageIndex,
    rightProperties as CFDictionary
)
```

It's valid to write either the left- or right-eye image as the first image in a spatial photo. visionOS detects the appropriate images to use for left- and right-eye presentation based on the `kCGImagePropertyGroupImageIsLeftImage` or `kCGImagePropertyGroupImageIsRightImage` properties in the groups dictionary for each image, regardless of the order in which the images are added to the HEIC file. If the system should prefer the right-eye image as the primary image to display when it shows the spatial photo in a nonstereo environment, modify the app to add the right-eye image to the image destination first, so that it appears at index `0` in the output HEIC file.

Finally, the app calls `CGImageDestinationFinalize` to write the image destination to disk as a self-contained spatial photo.

```
guard CGImageDestinationFinalize(destination) else {
    throw ConversionError.unableToFinalizeImageDestination
}
```

See Also

Spatial Photos

Creating spatial photos and videos with spatial metadata

Add spatial metadata to stereo photos and videos to create spatial media for viewing on Apple Vision Pro.