Sample Code

Processing spatial video with a custom video compositor

Create a custom video compositor to edit spatial video for playback and export.

Download

iOS 26.0+ | iPadOS 26.0+ | macOS 26.0+ | visionOS 26.0+ | Xcode 26.0+

Overview

Spatial video content requires specialized processing to maintain its stereoscopic presentation or convert it to monoscopic output for different viewing contexts. While AVFoundation provides built-in video compositing capabilities for basic effects like transforms and fades, processing spatial video's dual-eye data streams demands more sophisticated control.

AVFoundation's `AVVideoCompositing` protocol provides the framework for building custom video compositors with full control over video processing. This sample demonstrates how to implement custom compositors that handle spatial video's tagged buffer data, enabling you to process stereoscopic content for both playback and export scenarios.

Configure the sample code project

To run the sample app, place one or more spatial video files on your device to process. The project includes a sample spatial video file for testing, and you can capture your own content on Apple Vision Pro or iPhone 16.

When you run the app, it prompts you to select a spatial video file through the system file picker on your platform.

Create a custom compositor

The sample provides two `AVVideoCompositing` implementations: `MonoOutputCompositor` and `StereoOutputCompositor`. Both process mono or stereo input, but output either monoscopic or stereoscopic video respectively.

To indicate that they support processing spatial video, both implementations override the `supportsSourceTaggedBuffers` property and provide a value of `true`.

```
/// A Boolean value that indicates whether the custom compositor supports source tag
let supportsSourceTaggedBuffers = true
```

The framework calls the compositor's `startRequest(_:)` method to process each video frame, passing an `AVAsynchronousVideoCompositionRequest` that provides access to the source frame data. The implementation handles both spatial and mono video sources:

```
func startRequest(_ request: AVAsynchronousVideoCompositionRequest) {

    // If no track identifier is found, cancel the request and return.
    guard let firstTrackIDNumber = request.sourceTrackIDs.first else {
        request.finishCancelledRequest()
        return
    }

    let firstTrackID = CMPersistentTrackID(truncating: firstTrackIDNumber)

    // Attempt to retrieve the tagged buffers in the source track.
    if let taggedBuffers = request.sourceTaggedDynamicBuffers(byTrackID: firstTrackI
        // Process the tagged buffers from stereoscopic source track.
        processTaggedBuffers(taggedBuffers: taggedBuffers, request: request)
    }
    // Attempt to retrieve the monoscopic video frame in the source track.
    else if let pixelBuffer = request.sourceFrame(byTrackID: firstTrackID) {
        // Process pixel buffer from monoscopic source track.
        processMonoscopicBuffer(sourcePixelBuffer: pixelBuffer, request: request)
    }
    // No source frames were found. Finish with an error.
    else {
        request.finish(with: CompositorError.invalidSource)
    }
}
```

The method first attempts to retrieve tagged buffers, which indicates the video data is spatial. If it finds no tagged buffers, it processes the source as monoscopic video. If neither attempt succeeds, it completes the request with an error.

> **Note**
>
> See the implementations for details about the processing each compositor performs.

Build a video composition object

The sample creates a video composition using the app's `VideoCompositionBuilder` structure. This type's `build()` method creates an `AVVideoComposition.Configuration` for the selected asset and indicates to use the custom compositor:

```
/// Builds a video composition object.
func build() async throws -> AVVideoComposition {

    // Create the video composition configuration object for the asset.
    var configuration = try await AVVideoComposition.Configuration(for: asset)
    // Specify the custom compositor implementation class to use.
    configuration.customVideoCompositorClass = compositorConfiguration.class

    ...

    return AVVideoComposition(configuration: configuration)
}
```

The configuration includes two key properties for spatial video:

`spatialVideoConfigurations`
: Contains `AVSpatialVideoConfiguration` objects describing the video data's camera baseline and disparity adjustment. Modify this only if your compositor's output differs from the source.

`outputBufferDescription`
: Configures tagged buffer output with `CMTag` arrays. Each top-level array element corresponds to one eye, specifying the tags for that eye's output buffers. For monoscopic output, leave this `nil` and set `spatialVideoConfigurations` to an empty array.

The builder configures the output based on whether the compositor produces stereo or mono output:

```
if compositorConfiguration.outputsStereo {
    // Wrap the instructions in the app's custom instruction type.
    configuration.instructions = configuration.instructions.compactMap {
        SpatialVideoCompositionInstruction(
            instruction: $0,
            spatialConfiguration: spatialConfiguration,
            projectionTag: projectionTag
        )
    }
    configuration.outputBufferDescription = [
        [.stereoView(.leftEye), projectionTag, .mediaType(.video)],
        [.stereoView(.rightEye), projectionTag, .mediaType(.video)]
    ]
} else {
    configuration.outputBufferDescription = nil
    configuration.spatialVideoConfigurations = []
}
```

Use a custom compositor for playback

Integrating the custom compositor for playback is straightforward. The sample's `SampleModel` class creates an `AVPlayer` with the video composition:

```
/// Creates a new player to play the movie at the specified URL.
/// - Parameter url: The URL of the movie file to play.
private func makePlayer(url: URL) async throws -> AVPlayer {
    let asset = AVURLAsset(url: url)
    let playerItem = AVPlayerItem(asset: asset)
    // Create a video composition for the currently user-selected custom compositor.
    if let videoComposition = try await makeVideoComposition(for: asset) {
        playerItem.videoComposition = videoComposition
        playerItem.seekingWaitsForVideoCompositionRendering = true
    }
    return AVPlayer(playerItem: playerItem)
}
```

The method creates a video composition using the selected compositor configuration. If the method creates a valid composition, it assigns it to the player item and enables rendering synchronization.

Export with an asset export session

The sample's `ExportSessionExporter` class demonstrates export using `AVAssetExportSession`. The export method selects the appropriate preset based on the video composition's output type:

```
func export(asset: AVAsset, videoComposition: AVVideoComposition? = nil) async throw

    // If this method receives a video composition that produces stereo output, use
    // Note: the `outputsStereo` property is an app-specific extension.
    let preset = if let videoComposition, videoComposition.outputsStereo {
        AVAssetExportPresetMVHEVC4320x4320
    }
    // Otherwise, use a standard HEVC preset.
    else {
        AVAssetExportPresetHEVCHighestQuality
    }

    // Attempt to create an export session with the selected preset.
    guard let exportSession = AVAssetExportSession(asset: asset, presetName: preset)
        throw ExportError.noExportSession
    }

    // If a valid video composition was passed to this method, set it on the export
    if let videoComposition {
        exportSession.videoComposition = videoComposition
    }

    ...
}
```

For stereo output, the method uses the MV-HEVC preset to maintain spatial video format. For mono output, it uses the standard HEVC preset. After creating the export session, it assigns the video composition and performs the export operation.

Export with an asset reader and writer

The sample's `ReaderWriterExporter` class provides fine-grained export control using `AVAssetReader` and `AVAssetWriter`. This approach requires more setup but offers greater flexibility over the export process.

The core export operation uses a read-write loop that processes each video frame:

```
// The read-write loop is executed in a dedicated dispatch queue.
writerInput.requestMediaDataWhenReady(on: DispatchQueue(label: "com.apple.spatialcom
    while writerInput.isReadyForMoreMediaData {

        guard let sampleBuffer = readerOutput.copyNextSampleBuffer() else {
            // A nil sample buffer indicates the end of the input.
            finishWritingAndResume()
            return
        }

        if let taggedBuffers = sampleBuffer.taggedBuffers, let taggedPixelBufferGrou
            // Send tagged buffers to writer input via tagged pixel buffer group rec
            // Make sure the tagged buffers are `CMTaggedDynamicBuffers` with `layer
            let wellFormedTaggedBuffers = taggedBuffers.ensureLayerIDTagsAndMakeDyna
            do {
                let pts = sampleBuffer.presentationTimeStamp
                if try !taggedPixelBufferGroupReceiver.appendImmediately(wellFormedT
                    finishWritingAndResume(error: .appendTaggedBuffersFailed)
                    return
                }
            } catch {
                finishWritingAndResume(error: .appendTaggedBuffersFailed)
                return
            }
        } else {
            // The reader output is a normal sample buffer. Send to writer input dir
            writerInput.append(sampleBuffer)
        }
        // Send async notification for progress update.
        let percent = (Double) (sampleBuffer.presentationTimeStamp.seconds / assetDu
        statusContinuation.yield(ExporterStatus.exporting(progress: percent))
    }
}
```

The loop reads sample buffers from the AVAssetReaderVideoCompositionOutput, which applies the custom video composition. For spatial video output, it uses tagged pixel buffer groups; for mono output, it appends sample buffers directly to the writer input.

See Also

Custom video compositing

`protocol` `AVVideoCompositing`

A protocol that defines the methods custom video compositors must implement.