Sample Code

Debugging AVFoundation audio mixes, compositions, and video compositions

Resolve common problems when creating compositions, video compositions, and audio mixes.

Download

iOS 13.5+ | iPadOS 13.5+ | Xcode 13.4+

Overview

You can construct elaborate and complex compositions, video compositions, and audio mixes with entirely valid values that produce unexpected results. This sample code project visualizes the composition on the screen and adopts the video composition validation-handling protocol to debug common pitfalls in compositions and audio mixes.

Configure the sample code project

- Build the sample with Xcode 13.4.1 or later, and Swift 5.5 or later.
- This sample runs on physical devices with iPadOS 13.5 or later.

Build and run the app on any supported device. Use the play, pause, and scrubber controls to play and scrub through the composition in the upper portion of the screen. The debug view in the lower portion of the screen visualizes the composition, video composition, and audio mix.

Visualize the composition

Visualization is looking at a picture of a composition rather than looking at its code. This sample implements the `CompositionDebugView` class to present a visual description on the screen of the underlying AVComposition, AVVideoComposition, and AVAudioMix objects that form the composition. Developers can drop the CompositionDebugView into their own apps. It's a noninteractive view, and developers can extend it to draw their own video instructions. It helps in identifying any overlaps and gaps in the composition tracks, video instructions, and audio mix.

Synchronize the visualized composition

The PlayerViewController class has an outlet to the CompositionDebugView object in the Main.storyboard.

```
@IBOutlet weak var compositionDebugView: CompositionDebugView!
```

The PlayerViewController creates a player item to display the composition in the upper portion of the screen using an AVPlayerViewController that presents a native user interface to control playback. It creates this player item from the composition, video composition, and audio mix that the SimpleEditor creates from the video files in the project.

```
// Create a player item from the simple editor composition.
self.playerItem = AVPlayerItem(asset: self.editor.composition())
/*
 Set the player item's video composition and audio mix playback
 settings from the corresponding values in the simple editor.
*/
self.playerItem.videoComposition = self.editor.videoComposition()
self.playerItem.audioMix = self.editor.audioMix()
```

To synchronize its player item with the CompositionDebugView, the PlayerViewController calls the CompositionDebugView synchronize function, passing the player item as a parameter.

```
// Set the player item on the debug view to synchronize playback.
self.compositionDebugView.synchronize(with: self.playerItem)
```

The CompositionDebugView synchronize function uses the passed-in player item parameter to synchronize with its own drawing. It builds its visual display from the composition, video composition, and audio mix associated with the player item.

```
func synchronize(with playerItem: AVPlayerItem) {
    self.playerItem = playerItem
    if let composition = playerItem.asset as? AVMutableComposition {
        compositionTrackSegmentInfo = trackSegmentInfo(from: composition.tracks)
        duration = CMTimeMaximum(duration, composition.duration)
    }

    if let audioMix = playerItem.audioMix {
        volumeRampAsPoints = volumeRampPoints(from: audioMix, duration: duration)
    }

    if let videoComposition = playerItem.videoComposition {
        videoCompositionStages = videoCompStageInfo(from: videoComposition.instructi
    }

    drawingLayer.setNeedsDisplay()
    self.setNeedsDisplay()
```

Create the composition

The `SimpleEditor` class initialization creates an AVMutableComposition to merge the videos.

```
private lazy var editorComposition: AVMutableComposition = {
    var comp = AVMutableComposition()
```

Then the `SimpleEditor` initializer calls the `createComposition` function to stitch the provided video clips together. It inserts the clips into alternating video and audio tracks in the composition.

```
/*
 Place clips into alternating video and audio tracks in the composition,
 and overlap them with transitionDuration. Set up the video composition to cycle
 between "pass through A", "transition from A to B", and "pass through B".
*/
var nextClipStart = CMTime.zero
for clipIndex in 0..<clips.count {
    // Alternating targets: 0, 1, 0, 1, ...
    let asset = clips[clipIndex].asset
    let clipTimeRange = clips[clipIndex].availableTimeRange
    do {
        let clipVideoTrack = asset.tracks(withMediaType: AVMediaType.video)[0]
        try compVideoTrks[clipIndex % 2].insertTimeRange(clipTimeRange, of: clipVide
        let clipAudioTracks = asset.tracks(withMediaType: AVMediaType.audio)
        if clipAudioTracks.isEmpty {
            SimpleEditorUtils.display("Each clip must have an audio track.")
            return false
        }
        try compAudioTrks[clipIndex % 2].insertTimeRange(clipTimeRange, of: clipAudi
```

The sample overlaps the end of the first clip with the start of the second clip, and creates time ranges for a transition effect during the overlap period. The first clip ends with the transition, and the second clip begins with the transition. To set up the transition effect for the overlap period only, the sample enables compositing during the overlap, and disables it for the rest of the video composition. To do that, the sample creates a passthrough time range for each clip that excludes the transition period. This passthrough time range instructs the video compositor to pass through the frames without compositing.

```
passThroughTimeRanges[clipIndex] = CMTimeRangeMake(start: nextClipStart, duration: c
if clipIndex > 0 {
    passThroughTimeRanges[clipIndex].start = CMTimeAdd(passThroughTimeRanges[clipInc
                    transitionDuration)
}
passThroughTimeRanges[clipIndex].duration = CMTimeSubtract(passThroughTimeRanges[cli
                    transitionDuration)

/*
 The end of this clip overlaps the start of the next by
 transitionDuration.
*/
nextClipStart = CMTimeSubtract(CMTimeAdd(nextClipStart, clipTimeRange.duration), tra

// Retain the time range for the transition to the next item.
if clipIndex + 1 < clips.count {
    transitionTimeRanges[clipIndex] = CMTimeRangeMake(start: nextClipStart, duratior
}
```

Create the video composition and audio mix

The `SimpleEditor` class initialization creates an `AVMutableVideoComposition` to apply effects between clips, and an `AVMutableAudioMix` for mixing the audio tracks.

```
private var editorAudioMix = AVMutableAudioMix()

private lazy var editorVideoComposition: AVMutableVideoComposition = {
    var videoComp = AVMutableVideoComposition()
    // Every videoComposition needs these properties to be set:
    videoComp.frameDuration = CMTimeMake(value: 1, timescale: frameDuration)
    videoComp.renderSize = editorComposition.naturalSize
    return videoComp
}()
```

Then the `SimpleEditor` initializer calls the `createVideoCompAndAudioMix` function to add an opacity ramp transition between the video clips, and a volume ramp between the audio tracks.

```
instructions.append(passThroughInstruction)

if currIndex + 1 < clips.count {
    // Add a transition from clip i to clip i+1.
    let transitionInstruction = AVMutableVideoCompositionInstruction()
    transitionInstruction.timeRange = transitionTimeRanges[currIndex]
    let fromLayer = AVMutableVideoCompositionLayerInstruction(assetTrack:
                         compVideoTracks[alternatingIndex])
    let toLayer = AVMutableVideoCompositionLayerInstruction(assetTrack:
                     compVideoTracks[1 - alternatingIndex])
    // Sets an opacity ramp to apply during the specified time range.
    toLayer.setOpacityRamp(fromStartOpacity: 0.0, toEndOpacity: 1.0,
                         timeRange: transitionTimeRanges[currIndex])

    transitionInstruction.layerInstructions = [toLayer, fromLayer]

    instructions.append(transitionInstruction)

    // Add an audio mix to the first clip to fade in the volume ramps.
    let trackMix1 = AVMutableAudioMixInputParameters(track: compAudioTracks[0])
    trackMix1.setVolumeRamp(fromStartVolume: 1.0, toEndVolume: 0.0,
                             timeRange: transitionTimeRanges[0])

    trackMixArray.append(trackMix1)

    // Add an audio mix to the second clip to fade out the volume ramps.
    let trackMix2 = AVMutableAudioMixInputParameters(track: compAudioTracks[1])
    trackMix2.setVolumeRamp(fromStartVolume: 0.0, toEndVolume: 1.0,
                             timeRange: transitionTimeRanges[0])
    trackMix2.setVolumeRamp(fromStartVolume: 1.0, toEndVolume: 1.0,
                            timeRange: passThroughTimeRanges[1])
    trackMixArray.append(trackMix2)
}

editorAudioMix.inputParameters = trackMixArray
editorVideoComposition.instructions = instructions
```

Debug video compositions

The sample implements the `AVVideoCompositionValidationHandling` protocol methods in the `SimpleEditor` class to debug the video composition. These methods identify errors and indicate whether validation of a video composition needs to continue after finding specific errors.

The sample's implementation of each of these functions displays an appropriate error dialog, and returns `false` to stop any further validation of the video composition. See the `AVVideoCompositionValidationHandling` documentation for more information.

```
func videoComposition(_ videoComposition: AVVideoComposition, shouldContinueValidati
        SimpleEditorUtils.display("Empty time range detected during validation.")
        return false // Stop validation after finding errors.
}

func videoComposition(_ videoComposition: AVVideoComposition, shouldContinueValidati
        SimpleEditorUtils.display("Invalid time range detected during validation.")
        return false // Stop validation after finding errors.
}

func videoComposition(_ videoComposition: AVVideoComposition, shouldContinueValidati
        SimpleEditorUtils.display("Invalid value for \(key) detected during validati
        return false // Stop validation after finding errors.
}

func videoComposition(_ videoComposition: AVVideoComposition, shouldContinueValidati
        SimpleEditorUtils.display("Invalid track ID detected during validation.")
        return false // Stop validation after finding errors.
}
```

See Also

Built-in video compositing

{} Editing and playing HDR video

Support high-dynamic-range (HDR) video content in your app by using the HDR editing and playback capabilities of AVFoundation.

`class AVVideoComposition`

An object that describes how to compose video frames at particular points in time.

`class AVVideoCompositionInstruction`

An operation that a compositor performs.

`class AVVideoCompositionLayerInstruction`

An object used to modify the transform, cropping, and opacity ramps applied to a given track in a composition.

~~`class AVMutableVideoComposition`~~

A mutable video composition subclass.

Deprecated

~~`class AVMutableVideoCompositionInstruction`~~

A mutable video composition instruction subclass.

Deprecated

~~`class AVMutableVideoCompositionLayerInstruction`~~

An object used to modify the transform, cropping, and opacity ramps applied to a given track in a mutable composition.

Deprecated