Sample Code

Encoding video for low-latency conferencing

Configure a compression session to optimize encoding for video-conferencing apps.

Download

macOS 26.0+ | Xcode 26.0+

Overview

This sample code project demonstrates how to configure and use a `VTCompressionSession` object to encode video for low-latency conferencing.

Create a compression session

Create a `VTCompressionSession` object and specify the required dimensions and `codecType`. For low latency use, include the `kVTVideoEncoderSpecification_EnableLowLatencyRateControl` option and set it to `true` in the `videoEncoderSpecification`. Optionally, specify `sourceImageBufferAttributes` to provide a description of the source pixel buffers.

```
// Set `kVTVideoEncoderSpecification_EnableLowLatencyRateControl` in encoder
// specification when creating a compression session to request low-latency
// rate control.
let videoEncoderSpecification = [kVTVideoEncoderSpecification_EnableLowLatencyRateCo

// Specify the pixel format of the uncompressed video.
let sourceImageBufferAttributes = [kCVPixelBufferPixelFormatTypeKey: options.pixelFo
```

```
var compressionSessionOut: VTCompressionSession?
let err = VTCompressionSessionCreate(allocator: kCFAllocatorDefault,
                                     width: Int32(options.destWidth),
                                     height: Int32(options.destHeight),
                                     codecType: options.codec,
                                     encoderSpecification: videoEncoderSpecificatior
                                     imageBufferAttributes: sourceImageBufferAttribu
                                     compressedDataAllocator: nil,
                                     outputCallback: nil,
                                     refcon: nil,
                                     compressionSessionOut: &compressionSessionOut)
guard err == noErr, let compressionSession = compressionSessionOut else {
    throw RuntimeError("VTCompressionSession creation failed (\(err))!")
}
```

Configure the compression session

Get the suggested encoder settings dictionary for the encode preset.

```
/// Get the suggested encoder settings dictionary for encode preset.
/// - Parameters:
///   - session: A compression session.
///   - encodePreset: The `EncodePreset` enumeration.
private func getEncoderSettingsForPreset(session: VTCompressionSession, encodePreset
    var supportedPresetDictionaries: CFDictionary?
    var encoderSettings: [CFString: Any]?

    _ = withUnsafeMutablePointer(to: &supportedPresetDictionaries) { valueOut in
        VTSessionCopyProperty( session, key: kVTCompressionPropertyKey_SupportedPres
                               allocator: kCFAllocatorDefault, valueOut: valueOut )
    }

    if let presetDictionaries = supportedPresetDictionaries as? [CFString: [CFString
        let presetConstant = switch encodePreset {
        case .videoConferencing: kVTCompressionPreset_VideoConferencing
        }

        encoderSettings = presetDictionaries[presetConstant]
    }

    return encoderSettings
}
```

Set the encoder settings dictionary. Set `kVTCompressionPropertyKey_RealTime` to `kCFBooleanTrue` to indicate that this is a live encoding session. Specify the expected video source frame rate. Optionally specify the codec profile and level and update the average target bit rate. The low-latency rate control that the system enables during the `VTCompressionSession` object creation takes care of the other encoder configuration for low latency.

```swift
/// Configures a compression session for low-latency conferencing.
/// - Parameters:
///   - session: A compression session.
///   - options: The configuration options.
///   - expectedFrameRate: The expected frame rate of the video source.
private func configureVTCompressionSession(session: VTCompressionSession, options: (
    // Different encoder implementations may support different property sets, so
    // the app needs to determine the implications of a failed property setting
    // on a case-by-case basis for the encoder. If the property is essential for
    // the use case and its setting fails, the app terminates. Otherwise, the
    // encoder ignores the failed setting and uses a default value to proceed
    // with encoding.

    var err: OSStatus = noErr
    var variableBitRateMode = false

    if let presetTuple = options.presetTuple {
        // Try configuring the encoder using the preset.
        let encoderSettings: [CFString: Any]?
        encoderSettings = getEncoderSettingsForPreset(session: session, encodePreset

        if let encoderSettings {
            if encoderSettings[kVTCompressionPropertyKey_VariableBitRate] != nil {
                variableBitRateMode = true
            }

            // Set the encoder settings dictionary on the compression session.
            err = VTSessionSetProperties(session, propertyDictionary: encoderSetting
            try NSError.check(err, "VTSessionSetProperties failed")
        }
    }

    // Indicate real-time compression session, which conferencing requires.
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_RealTime, val
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_RealTime) fai
    }

    // Indicate the expected frame rate, if known. This is just a hint for rate
    // control purposes; the actual encoding frame rate matches the incoming
    // frame rate even if it doesn't match this setting. When
    // `kVTCompressionPropertyKey_RealTime` is `kCFBooleanTrue`, the video
    // encoder may optimize energy usage.
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_ExpectedFrame
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_ExpectedFrame
    }

    // Specify the profile and level for the encoded bitstream.
    if let profileTuple = options.profileTuple {
        var profileConstant: CFString?

        if options.codec == kCMVideoCodecType_H264 {
            if profileTuple.0 == .h264Main {
                profileConstant = kVTProfileLevel_H264_Main_AutoLevel
            } else if profileTuple.0 == .h264High {
                profileConstant = kVTProfileLevel_H264_High_AutoLevel
            }
        } else if options.codec == kCMVideoCodecType_HEVC {
            if profileTuple.0 == .hevcMain {
                profileConstant = kVTProfileLevel_HEVC_Main_AutoLevel
            } else if profileTuple.0 == .hevcMain10 {
                profileConstant = kVTProfileLevel_HEVC_Main10_AutoLevel
            }
        }

        if let profileConstant {
            err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Profi
            if err != noErr {
                print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Profi
            }
        }
    }

    if let destBitRate = options.destBitRate {
        if variableBitRateMode {
            // Specify the long-term desired variable bit rate in bits per second.
            err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Varia
            if err != noErr {
                print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Varia
            }

            // Set VBV maximum bit rate.
            err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_VBVMa
            if err != noErr {
                print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_VBVMa
            }
        } else {
            // Specify the long-term desired average bit rate in bits per second.
            // It's a soft limit, so the encoder may overshoot or undershoot and
            // the average bit rate of the output video may be over or under the
            // target.
            err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Avera
            if err != noErr {
                print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Avera
            }
        }
    }
```

Encode video frames

Call `VTCompressionSessionEncodeFrame(_:imageBuffer:presentationTimeStamp:duration:frameProperties:infoFlagsOut:outputHandler:)` and submit each uncompressed frame to the `VTCompressionSession` object for encoding. The object calls the `outputHandler` block for each encoded frame. Check whether a frame drop or error occurs after frame encoding.

Call `VTCompressionSessionCompleteFrames(_:untilPresentationTimeStamp:)` to indicate to the `VTCompressionSession` object that the app submitted all uncompressed video frames for encoding.

Perform compression and encoding

You can use the movie file `/Assets/video.m4v` to test this app. Copy the file to your desktop or other working directory, and then open Terminal to that directory and run the following command, where `xxx` is a unique string that Xcode generates. Use autocomplete before the `xxx` component to complete the path for that directory.

```
~/Library/Developer/Xcode/DerivedData/VTEncoderForConferencing-
xxx/Build/Products/Debug/VTEncoderForConferencing-Swift video.m4v
```

Pass the `--help` option for additional configuration options.

See Also

Compression

{} Encoding video for live streaming

Configure a compression session to encode video for live streaming.

{} Encoding video for offline transcoding

Configure a compression session to transcode video in offline workflows.

VTCompressionSession

An object that compresses video data.

VTDecompressionSession

An object that decompresses video data.

VTFrameSilo

An object that stores sample buffers from a multipass encoding session.

VTMultiPassStorage

An object that stores video encoding metadata from a multipass encoding session.