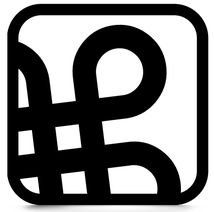


Macoun

Interaktive Audio Entwicklung in iOS

Rolf Wöhrmann

Ablauf

- Anforderungen an interaktive Audio Apps
- Überblick iOS Audio APIs
- Im Detail:
  - AVAudioSession, AudioUnits, Inter-App Kommunikation, CoreMIDI
- Weitere Aspekte:
  - Portabilität, Metal Data Processing, Swift & Audio

Anforderungen an interaktive Audio Apps

Anforderungen

- Geringe Latenz
- Zusammenspiel mit anderen Audio Events
- Konnektivität
- Synchronisation

Latenz

- Zeitspanne vom Touch zum Ton
- Wie wird gemessen?
- Was ist State-of-the-art? Gesamt Latenz ~30ms
  - Audio Block Größen: 512, 256, 128, 64, 32 (11ms-0.6ms)
  - Touch Event Delivery 10-15ms
  - HW Output Latency ~10ms

Zusammenspiel Audio Events

- Zusammenspiel von Audio Events auf mobilen Geräten:
  - Telefon klingelt
  - Alarm: Wecker, Timer, Kalender
  - Notifications
  - Background Audio, PIP Video
- Lösungsansatz von iOS: Audio Session Kategorien
- Mehrere Audio Apps spielen gleichzeitig: BackgroundAudio

Konnektivität

- Audio Apps untereinander
  - Echtzeit: Audio Streaming durch AudioBus, Inter-App Audio, Virtual CoreMIDI
  - Offline: Audio Pasteboards
  - Einbettung: AudioUnit Extensions
- Externe Geräte
  - Audio Interfaces, Mac Rechner, MIDI/OSC Controller

Synchronisation

- Synchronisation von Tempo & Timing
- MIDI Clock
  - Mehrere Apps untereinander
  - Mit externen Geräten via MIDI Interface
- Timing info Strukturen in Inter-App Audio & AU Extensions
- Mit Bildschirmgrafik

Überblick iOS Audio APIs & Coding im Detail

iOS Audio APIs

- AVFoundation & AVAudioEngine, AudioToolbox
- CoreMIDI
- CoreAudio, AudioUnit, HAL
- CoreAudioKit (UIWidgets)
- Other: Pasteboards, OpenAL, SpriteKit
- Non-API Technologies: BackgroundAudio, Inter-Device Audio
- 3rd Party: AudioBus, AudioShare, AudioCopy, Soundcloud, USB Audio

AVFoundation & AVAudioEngine

- Audio Playback & Recording mit geringer Interaktivität:
  - AVAsset, AVAudioPlayer, AVAudioRecorder
- Wichtigste Klasse: AVAudioSession
  - Realisiert Anforderung “Zusammenspiel mit anderen Audio Events”
- Neues API AVAudioEngine für Node basierte Audio Anwendungen

AudioToolbox

- Audio Format Spezifikation: AudioFormat.h
- Format Konvertierung: AudioConverter.h
- File Handling: AudioFile.h, AudioFileStream.h, ExtendedAudioFile.h
- Verarbeitungsnetzwerke: AUGraph.h
- Mid-level Playback: AudioQueue.h
- High-Level: AudioServices.h

Anforderung: Zusammenspiel mit anderen Audio Events

AVAudioSession Activation

- Wichtig:
  - Die Audio APIs werden weitgehend auf dem Simulator unterstützt.
  - Deren Echtzeitverhalten und Konfiguration entspricht nicht iOS Devices.
  - Bitte immer auf dem Device testen!
- Code Beispiel…

```
func activateAudioSession(preferredSampleRate: Double = 0.0,
    preferredBufferSize: Int = 0) {

    let session = AVAudioSession.sharedInstance()
    do {
        try session.setCategory(AVAudioSessionCategoryPlayback,
            withOptions:AVAudioSessionCategoryOptions.MixWithOthers)
        try session.setActive(true)

        if preferredSampleRate > 0.0 {
            try session.setPreferredSampleRate(preferredSampleRate)
        }
        if preferredBufferSize > 0 {
            try session.setPreferredIOBufferDuration(
                Double(preferredBufferSize) / session.sampleRate)
        }
    } catch let error as NSError {
        print(error.localizedDescription)
    }
}
```

AVAudioSession - Categories

- Die Session Category bestimmt das Zusammenspiel mit anderen Audio Apps und Audio Events

```
// From AVFoundation/AVAudioSession

public let AVAudioSessionCategoryAmbient: String
public let AVAudioSessionCategorySoloAmbient: String
public let AVAudioSessionCategoryPlayback: String
public let AVAudioSessionCategoryRecord: String
public let AVAudioSessionCategoryPlayAndRecord: String
public let AVAudioSessionCategoryAudioProcessing: String
public let AVAudioSessionCategoryMultiRoute: String
```

AVAudioSession - CategoryOptions

- Die Category Options verfeinern bestimmte Categories:

```
public struct AVAudioSessionCategoryOptions : OptionSetType {

    public static var MixWithOthers: AVAudioSessionCategoryOptions { get }
    public static var DuckOthers: AVAudioSessionCategoryOptions { get }
    public static var AllowBluetooth: AVAudioSessionCategoryOptions { get }
    public static var DefaultToSpeaker: AVAudioSessionCategoryOptions { get }
    @available(iOS 9.0, *)
    public static var InterruptSpokenAudioAndMixWithOthers:
                                            AVAudioSessionCategoryOptions { get }
}
```

AVAudioSession Notifications

- Wichtig: Preferred SampleRate & ioBufferDuration können nur angefragt werden. iOS entscheidet aus dem Gesamtkontext, welche Werte verwendet werden und ändert diese ggf. während der Laufzeit.
- Änderungen des Audio Setups müssen per App Laufzeit mittels Notifications abgefangen werden:

```
public let AVAudioSessionInterruptionNotification: String
public let AVAudioSessionRouteChangeNotification: String
```

[Xcode]

Anforderung: Geringe Latenz

AudioUnit Framework

- AudioUnit als zentrale Komponente für interaktives Audio
  - RemoteIO Callback erlaubt Audio Verarbeitung auf unterstem Level
- Als Basis Komponente für
  - App-internes Audio
  - Audio Streaming zwischen Apps: Inter-App Audio
  - Audio Extensions: AUv3

AudioUnit - Creation

- Neu in iOS9: AUAudioUnit Objective-C/Swift Klasse statt C-API

```
let description = AudioComponentDescription(
  componentType: kAudioUnitType_Output,
  componentSubType: kAudioUnitSubType_RemoteIO,
  componentManufacturer: kAudioUnitManufacturer_Apple,
  componentFlags: 0,
  componentFlagsMask: 0
)

try audioUnit = AUAudioUnit(componentDescription: description)
```

AudioUnit - Konfiguration

- Für jede AudioUnit kann ein internes Audio Format spezifiziert werden.

```
let format = AudioStreamBasicDescription(
    mSampleRate: samplingRate,
    mFormatID: kAudioFormatLinearPCM,
    mFormatFlags: kAudioFormatFlagIsFloat | kAudioFormatFlagIsPacked,
    mBytesPerPacket: bytesPerSample * channels,
    mFramesPerPacket: 1,
    mBytesPerFrame: bytesPerSample * channels,
    mChannelsPerFrame: channels,
    mBitsPerChannel: bytesPerSample * 8,
    mReserved: 0
)

try audioUnit.inputBusses[0].setFormat(AVAudioFormat(streamDescription: &format))
```

AudioUnit - Realtime Callback

- Der outputProvider Block springt direkt in den C-Echtzeit Code:

```
audioUnit.outputProvider = {(
    actionFlags: UnsafeMutablePointer<AudioUnitRenderActionFlags>
    timestamp: UnsafePointer<AudioTimeStamp>,
    frameCount: AUAudioFrameCount,
    inputBusNumber: NSInteger,
    inputData: UnsafeMutablePointer<AudioBufferList>
) -> AUAudioUnitStatus in

  return AudioC_remoteIOCallback(nil, actionFlags, timestamp, frameCount,
          UInt32(inputBusNumber), inputData)
}
```

AudioUnit - Los geht's!

- Jetzt muss nur noch das Audio Prozessen gestartet werden.

```
try audioUnit.startHardware()
```

- Und zum Beenden entsprechend:

```
try audioUnit.stopHardware()
```

[Xcode]

Coding auf dem Echtzeit Thread

- Keine Speicher(de)allokationen (malloc, new, free, delete…)
- Kein Objective-C oder Swift
- Keine “blocking calls”, Inter-Thread Kommunikation möglichst über non-blocking Queues/FIFOs, o.ä.
- Optimizer auf -O3, NEON 4 Byte Alignment beachten,
  Relax IEEE Compliance (-ffast-math) & denormals disabled
- Audio Signalverarbeitung (DSP) ist “Rocket Science”:
  Fragt einen Experten!

Anforderung:
Konnektivität - 1. MIDI

CoreMIDI

- Implementierung des klassischen MIDI Protokolls
- Wurde vor 30 Jahren erfunden zur Verbindung von elektronischen Musikinstrumenten
- Hohe Interaktivität: Austausch von Noten & Timing Informationen
- Virtual CoreMIDI: Apps untereinander
- Per externes MIDI Interface
- Via WIFI oder Bluetooth LE

CoreMIDI - Create Client

```
var clientRef: MIDIClientRef = 0
let appName = “MyApp”

guard MIDIClientCreateWithBlock(appName, &clientRef,
                                {(not: UnsafePointer<MIDINotification>) -> () in

    switch not.memory.messageID {
       case MIDINotificationMessageID.MsgSetupChanged:
          // ...
       default:
          break
    }
}) == noErr else {
  print("MIDIClientCreateWithBlock failed!”); return
}
```

CoreMIDI - Create Input Port

```
// create input port

var inPort: MIDIPortRef = 0

guard MIDIInputPortCreateWithBlock(clientRef, appName + ": In", &inPort,
 {(packets: UnsafePointer<MIDIPacketList>, refCon: UnsafeMutablePointer<Void>) -> ()
      in self.readProc(packets, refCon: refCon)
  }) == noErr else {print("MIDIInputPortCreateWithBlock failed!"; return}

// optional: create network session

let netSession = MIDINetworkSession.defaultSession()
netSession.enabled = true
netSession.connectionPolicy = MIDINetworkConnectionPolicy.Anyone
```

CoreMIDI - Connect to Sources

```
let n = MIDIGetNumberOfSources()
for i in 0..<n {
    let src = MIDIGetSource(i)
    let ret = MIDIPortConnectSource(inPort, src, nil)

    var cname: Unmanaged<CFString>?
    if MIDIObjectGetStringProperty(src, CoreMIDI.kMIDIPropertyDisplayName, &cname)
     != noErr {
        MIDIObjectGetStringProperty(src, CoreMIDI.kMIDIPropertyName, &cname)
    }

    let name = String(cname?.takeUnretainedValue()) ?? "Unknown Device"
    print("Connecting \(name) " + (ret == noErr ? "succeeded" : "failed"))
}
```

CoreMIDI - Receiving Data

- Das CoreMIDI C-API ist nicht auf Swift optimiert:

```
func readProc(packetList: UnsafePointer<MIDIPacketList>,
              refCon: UnsafeMutablePointer<Void>) {

    let packets = packetList.memory
    let packet: MIDIPacket = packets.packet

    var ap = UnsafeMutablePointer<MIDIPacket>.alloc(1)
    ap.initialize(packet)

    for _ in 0 ..< packets.numPackets {
        // Do something with it ...
        ap = MIDIPacketNext(ap)
    }
}
```

[Xcode]

Anforderung:
Konnektivität - 2. Audio

Externes Audio

- Headphone Connector
- USB Class-Compliant Audio Interfaces per Apples Camera Connection Kit
- MFI: Audio Interfaces am Lightning Port
- Inter-Device Audio (iOS9, OSX 10.11)
- Bluetooth Audio

Inter-App Audio

- Zunächst 3rd Party Lösung AudioBus über MachPorts realisiert.
  - Custom API, zentrale AudioBus App
- Als Inter-App Audio seit iOS 7 von Apple angeboten.
  - AU basiertes API, keine zusätzliche App notwendig
  - Host-Node Topologie
  - Übertragung von Noten-Informationen

iOS 9: AudioUnit Extensions

- Sind vergleichbar zu Plugins (Components) auf OSX.
- Werden via Standalone Apps im AppStore vertrieben.
- Sind eine Weiterentwicklung der Inter-App Audio AUs:
  - State, Parameter & FactoryPreset Handling
- Es können mehrere Instanzen einer AU Extension erzeugt werden.
- AU Extensions laufen in einem separaten Prozess, deren UI wird aber in die Host App integriert.

AU Extension Host - List AUs

- Beispiel eines AVAudioEngine basierten AU Hosts
- Auflisten aller verfügbaren AUs inkl. AU Extensions

```
var anyComponentDescription = AudioComponentDescription(
    componentType: kAudioUnitType_Effect,
    componentSubType: 0,
    componentManufacturer: 0,
    componentFlags: 0,
    componentFlagsMask: 0)

let availableEffects: [AVAudioUnitComponent] =
AVAudioUnitComponentManager.sharedAudioUnitComponentManager()
```

AU Extension Host - Create AU

- AU wird erzeugt, der AVAudioEngine hinzugefügt und die UI abgefragt

```
 var component: AVAudioUnitComponent = availableEffects[0]

AVAudioUnit.instantiateWithComponentDescription(component.audioComponentDescript
ion, options: []) {avAudioUnit, error in

  guard let avAudioUnit = audioUnit else {return}
  self.au = avAudioUnit
  self.engine.attachNode(avAudioUnit)
  self.engine.connect(avAudioUnit, to: self.mixerAU, format: nil)
  avAudioUnit.AUAudioUnit.requestViewControllerWithCompletionHandler {
  }
}
```

[Xcode]

Weitere Aspekte

Weitere Aspekte

- AUv3 - Ansatz für Portabilität iOS/OSX ?
  - Ja, aber nur Processing, kein UI
- 3D Touch
  - Multi-touch fähig, mehrere Druck Levels
- Audio auf dem GPU: Metal Data Processing
- Swift & Audio
  - Auf API Level ok, aber nicht auf dem Echtzeit Thread (gilt auch für Objective-C)

3D Touch

- UITouch Properties force & maximumPossibleForce

```
override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {

  guard let touch = touches.first else {return}

  if UIScreen.mainScreen().traitCollection.forceTouchCapability ==
          UIForceTouchCapability.Available {

    let normForce = touch.force / touch.maximumPossibleForce
    // do something with it...
  }
}
```

Metal & Audio - Klasse

- Metal Data Processing Klasse für Audio:

```
@interface AudioMetal() {

    id <MTLDevice> device;
    id <MTLLibrary> library;
    id <MTLCommandQueue> commandQueue;
    id <MTLFunction> func;

    id <MTLBuffer> paramsBuffer, outBuffer;

    // …
}
```

Metal & Audio - Init

```
device = MTLCreateSystemDefaultDevice();
commandQueue = [device newCommandQueue];

library = [device newDefaultLibrary];
func = [library newFunctionWithName:@"calcAudio"];

paramsBuffer = [device newBufferWithLength:sizeof(AudioMetalParams_t)
                    options:MTLResourceOptionCPUCacheModeDefault];
params = [paramsBuffer contents];
// setup params

outBuffer = [device newBufferWithLength:bytes
                    options:MTLResourceOptionCPUCacheModeDefault];
```

Metal & Audio - Compute 1

```
id <MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];
id <MTLComputeCommandEncoder> computeCE = [commandBuffer computeCommandEncoder];

NSError *error;
id <MTLComputePipelineState> compState =
          [device newComputePipelineStateWithFunction:func error:&error];

[computeCE setComputePipelineState:compState];

[computeCE setBuffer:paramsBuffer offset:0 atIndex:kParamBufferIdx];
[computeCE setBuffer:outBuffer offset:0 atIndex:kOutBufferIdx];
```

Metal & Audio - Compute 2

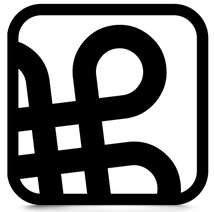
```
MTLSize threadsPerGroup = {VOICES, 1, 1};
MTLSize numThreadgroups = {1, 1, 1};

[computeCE dispatchThreadgroups:numThreadgroups
     threadsPerThreadgroup:threadsPerGroup];

[computeCE endEncoding];

[commandBuffer commit];

[commandBuffer waitUntilCompleted];

float *xp = [outBuffer contents];
// do something with xp
```

Zusammenfassung

- iOS ist ein hervorragendes mobiles Platform OS für interaktive Audio Anwendungen.
- Für geringe Latenz sollten eigene AudioUnits geschrieben werden (RemoteIO Callback).
- Konnektivität: Es stehen eine ganze Reihe von APIs zur Verfügung, die Inter-App und Inter-Device Konnektivität ermöglichen.
- Synchronisation: Klassische Protokolle als auch spezielle API calls stehen zur Verfügung

Fragen?

Vielen Dank



Macoun