A Reality(Kit) show

Tibor Kántor (POSSIBLE)

Agenda for today

Introduction

Overview of the framework

A glimpse of Reality Composer

Observations and found issues

Introducing RealityKit

Previously on the Apple AR show

You had to

1. Build 3D (2D) AR apps, you had to use, ARKit with SceneKit (SpriteKit)
2. Use specialised AR SceneKit (SpriteKit) Views
3. Implement and act on ARKit delegate methods
4. Add models by hand to found planes, images etc.
5. Handle interaction, physics, etc
6. Implement audio playback solution if needed
7. Create your own network layer

New season, introducing: RealityKit

A new rendering engine

- SceneKit, SpriteKit, now RealityKit

Built with Augmented Reality in mind, it’s “AR First”

- More realistic rendering
- Designed for Swift (unlike it’s predecessors)
- Available on MacOS and iOS
- Built on ARKit
    - Most of the ARKit features are accessible in a more simple way
    - Yet, all ARKit features are accessible if needed (some in the old way)

Standalone app for fast content creation: RealityComposer

More realistic rendering?

Physically based shading

- Built with Metal, optimised for Apple GPUs
- AR focused: nicely blending with the real world

Automatic:

- Shadowing
- Motion blur
- Depth of field
- Camera Noise
- Spatial audio

Framework overview

Basics

Physical based materials

- Control the colour, roughness and metalness

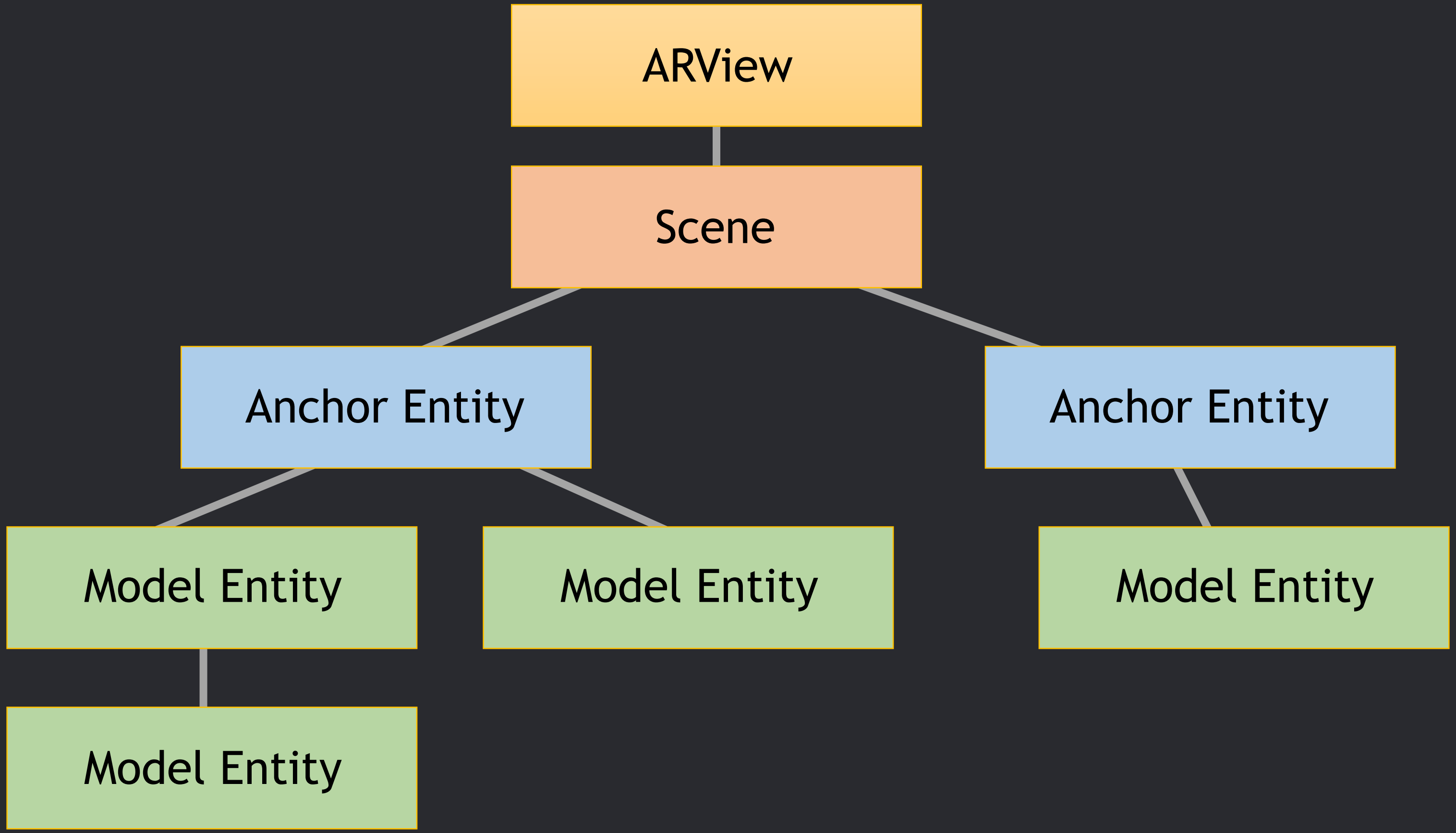
Supported animations

- Skeletal
- Transform (translate, scale, rotate)

Physics engine

- Collision detection
- Rigid body simulation, mass & friction
- AR enabled -> planes act as physical planes

Framework building blocks



Framework building blocks

ARView

- Base view for RealityKit
- Sets up the environment (configures ARKit)
- Handles gestures
- Backed by ARSession which is accessible

Scene

- Container for the AR objects (Entities)
- Associated with an ARView, created automatically

ARView
Scene

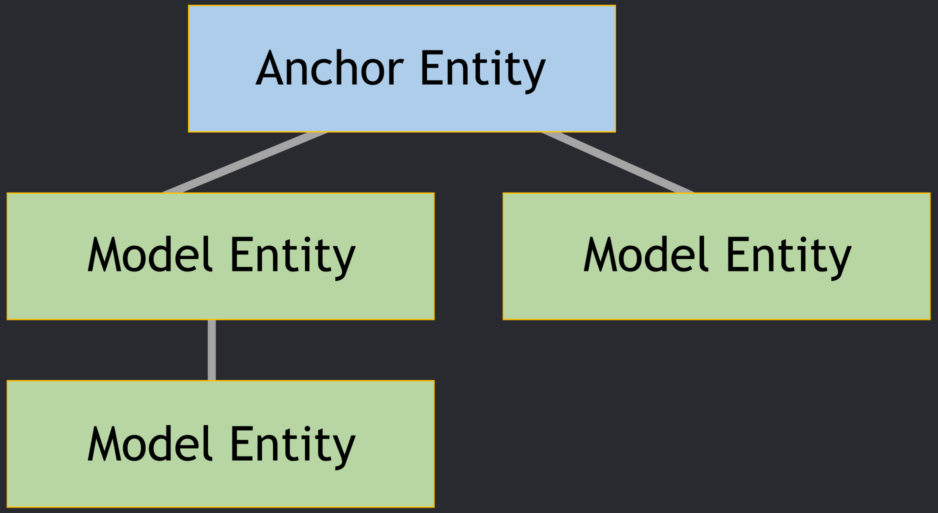
Framework building blocks

Anchor Entity

- Attached to a real world object, which is automatically tracked
- Plane, face, camera, image, object, body, ARAnchor

Model Entity

- Building blocks for AR experience
- Establishes scene structure
- Parent-child relation
- Provides transform hierarchy
- Like SCNNode



Entity-component system

The framework is built upon the entity-component paradigm

→ Like Unity or Unreal engine

There are entities and installable components

Forces composition over inheritance

Extensible: write your own components and install them

Entity-component system

Actually `entity` is the base class in RealityKit, `ModelEntity` and `AnchorEntity` are derived classes with different components

A Component is a class / struct conforming to the `Component` protocol:

- Defines geometry
- Defines behaviour

Adding components with subscript:

```
metalBall.components[CollisionComponent] = …
```

Entity-component system

Some important components:

- `ModelComponent` - visual representation: geometry & materials, built in meshes: box, sphere and plane
- `AnchoringComponent` - has an anchor in the real world
- `TransformComponent` - contains the trasform (matrix)
- `CollisionComponent` - can handle collisions
- `PhysicsBodyComponent` - defines the entity behaviours in physics simulation

Entity-component system

Not all entities need all components

Create your own for custom behaviour:

1. Implement `Component` protocol
2. Call the static `registerComponent()` before use
3. Set it as usual with subscript

Components are synchronised in a multi device environment automatically (in this case they also have to conform to `Codable`)

Other important & interesting stuff

Occlusion material

- Reveals video passthrough
- You can simulate real world objects with it

Handling scene events through Combine

- Update, collision, audio playback, etc.
- Subscribe to the scene’s appropriate events
- Thankfully the Scene is accessible from all entities

Audio

Audio playback on tracked objects

Spatial understanding: as you get closer to an object the played audio gets louder

Really simple to use:

1. Prepare an audio resource for playback on an entity or play it, which will return a `AudioPlaybackController`
2. Control the playback with that `AudiPlaybackController`
3. Handle audio events from the scene by subscribing to them

Network synchronisation

Multi device

Supports ARKit’s collaborative map building

Built upon multipeer connectivity

In theory just create a `MultipeerConnectivityService` object from `MCSession` and set it to the RealityKit’s session

In reality(kit) sometimes it really does work 😀

Reality Composer

Reality Composer

Standalone App for iOS and MacOS

Can be started from Xcode

Content library for built in objects

Layout previsualisation

Simple interactions and animations

Xcode integration

Reality Composer limitations

No hierarchies

Limited objects (eg. no plane for instance)

Limited material options: no images for primitives

Limited action sequences, no concurrencies

Xcode integration

From the Reality composer Xcode creates a reality file and generates code

Entities are accessible from code simply:

```
let poolAnchor = try! Experience.loadPool()
let whiteBall = poolAnchor.whiteBall
whiteBall?.playAnimation(named: "Pull")
```

USDZ files

RealityKit can be only fed with usdz and reality files

Other formats like .obj, .fbx, .dae aren’t supported!

Usdz file

- Universal Scene Description Zip
- Made by Pixar, open source format
- Originally made for collaboration on large movie scenes
- Bigger support: Maya, 3DS max, Unity exporter
- USDZ tools: https://developer.apple.com/augmented-reality/quick-look/

Use:

- Code: `ModelEntity.loadModel(named: ”MyUSDZMonster”)`
- Drag and drop into RealityComposer

Observations and experience

Observations

- The new paradigm shifts away from creating the experience in code and moves to a more 3D modelling approach
- For a more complex use case, you still have to dive into ARKit and implement it’s delegates and some features need some ARKit configuration (eg. People Occlusion)
- Math is much more user friendly with simd than with SCNVectors
- 3D debugging is not available
- There were some changes since the first beta and WWDC, but Apple didn’t update the docs in some cases

Issues found

- Grounding shadows don’t work for built in primitives (like boxes, spheres, planes), only for imported models
- Collision detection doesn’t work correctly if you specify collision shapes, it’s better to leave it to the framework to generate them
- Gesture recognizers don’t work with generated collision shapes
- You can not observe the collision for an object twice, although you can subscribe to it, and it will result in EXCBADACCES 😎

Issues found

- Built-in MutlipeerConnectivityService doesn’t work. Sometimes it does, but we couldn’t determine why and when does it work.
  - When it does work the synchronisation is to slow for an action game, probably it’s okay for turn based games, Apple demoed it with a memory card game
  - At least it generates a lot of SSL handshake errors to the log 😉
- Solution: don’t use it, instead use ARSession’s `didOutputData` delegate and send the `CollaborationData` by yourself and update the peer’s session with it
  - Even with this solution it looks like the ARAnchorEntities are sent really slowly, it’s better to send only their transformations

Conclusion

- RealityKit as a new framework has some flaws and issues, but
  - It’s much easier to use than SceneKit with ARKit
  - The rendering is indeed more realistic
  - For basic AR tasks you can achieve good results fast
  - RealityComposer is a great toolfor simple AR scenes
- If older iOS versions support is not needed, RealityKit is the way to go for AR

Thank you,

Questions?