AR

STRV

SESSION GOALS

- ARKit + RealityKit
- Challenges between 2D & 3D world
- Device capabilities
- 3D algebra & transformation matrix
- Explore common use cases and limitations
- Practical example at class
- Vision Pro in tha house

AR

01

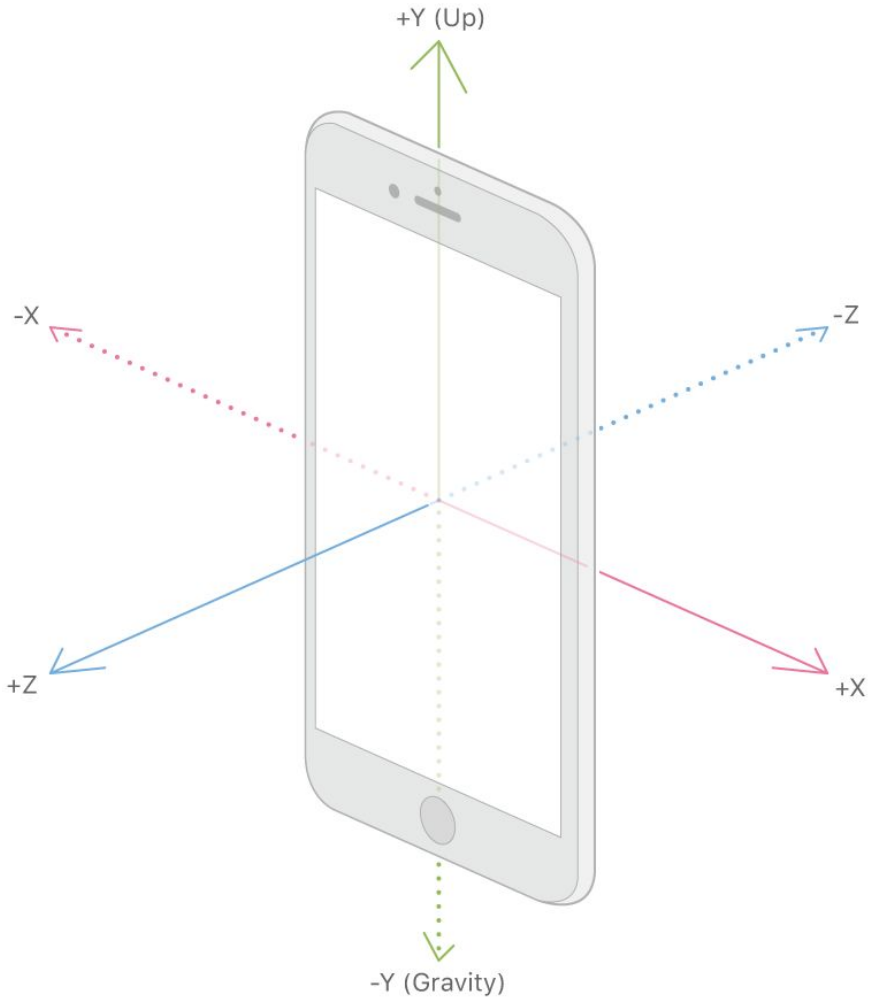
AUGMENTED REALITY (AR)

- To augment the perception of the real world
- Combines digital and physical worlds
  - Interactive in real-time
  - Anchors virtual elements to the physical world
- AR vs VR

- Performance bottleneck

HOW IT WORKS

- Hardware is important — processors, (3D) sensors (camera quality), accelerometer, …
- Sensors
  - Peripherals providing information
- Processors
  - Computing units analyzing provided information
- Tracking system
  - Detects surfaces, objects, faces, …
- ARKit, RealityKit, SceneKit, Vision, RoomPlan
- RealityComposer


+Y (Up)
-X
-Z
+Z
+X
-Y (Gravity)

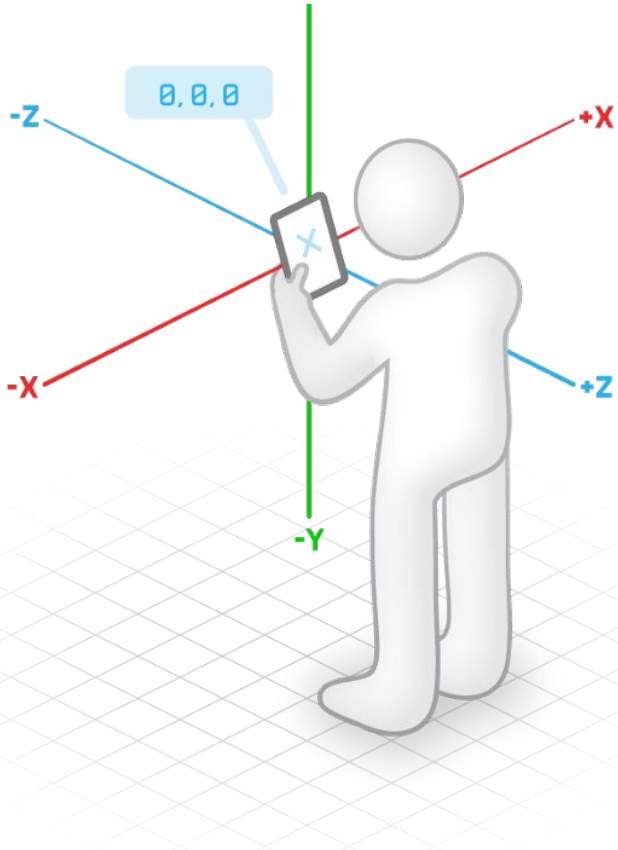
COORDINATE SPACES

World (Universe) Coordinate Space

- RHS (Right Hand Coordinate System)
- Immutable
- Base reference coordinate system
- The origin is at the location of the session's start

Object (Model) Coordinate Space

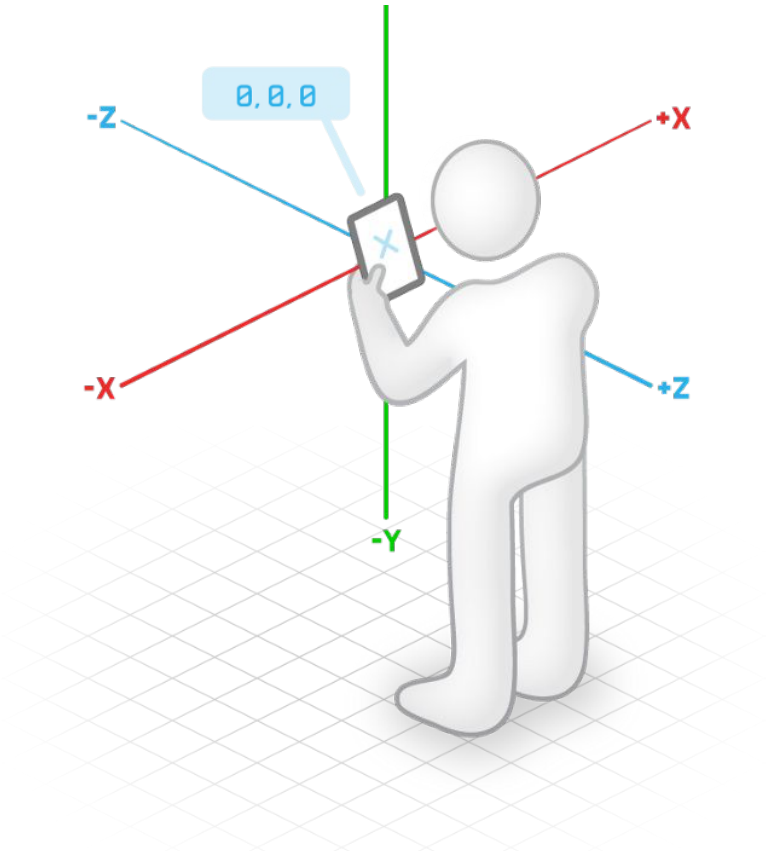
- Mutable
- The origin is determined upon creation
- Can be converted to world coordinate system (and from)

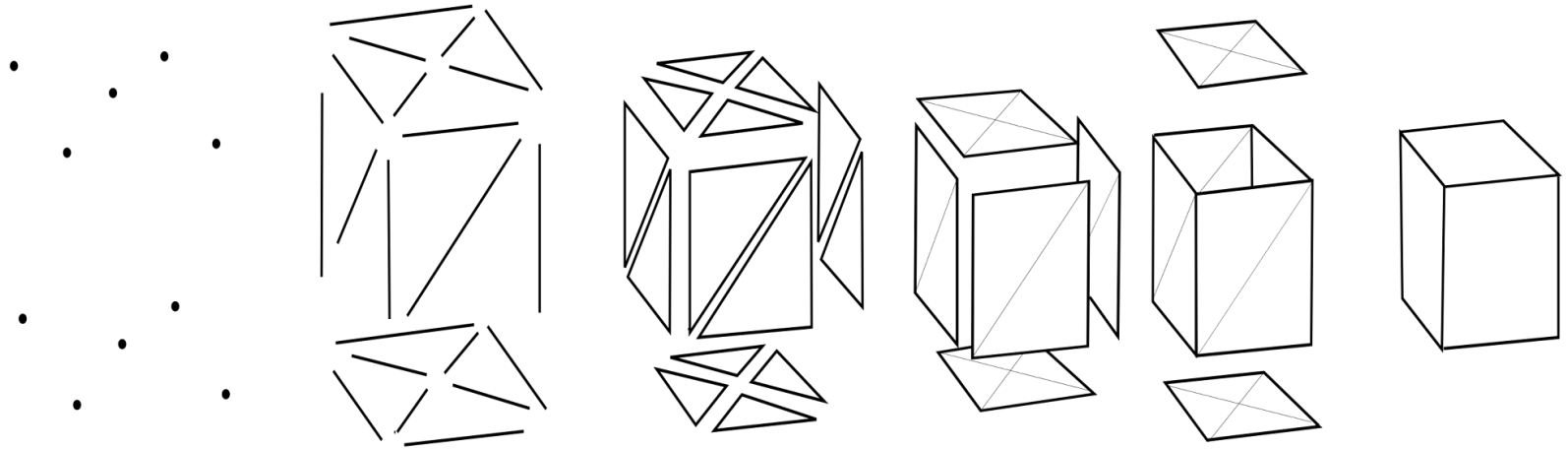


COORDINATE SPACES

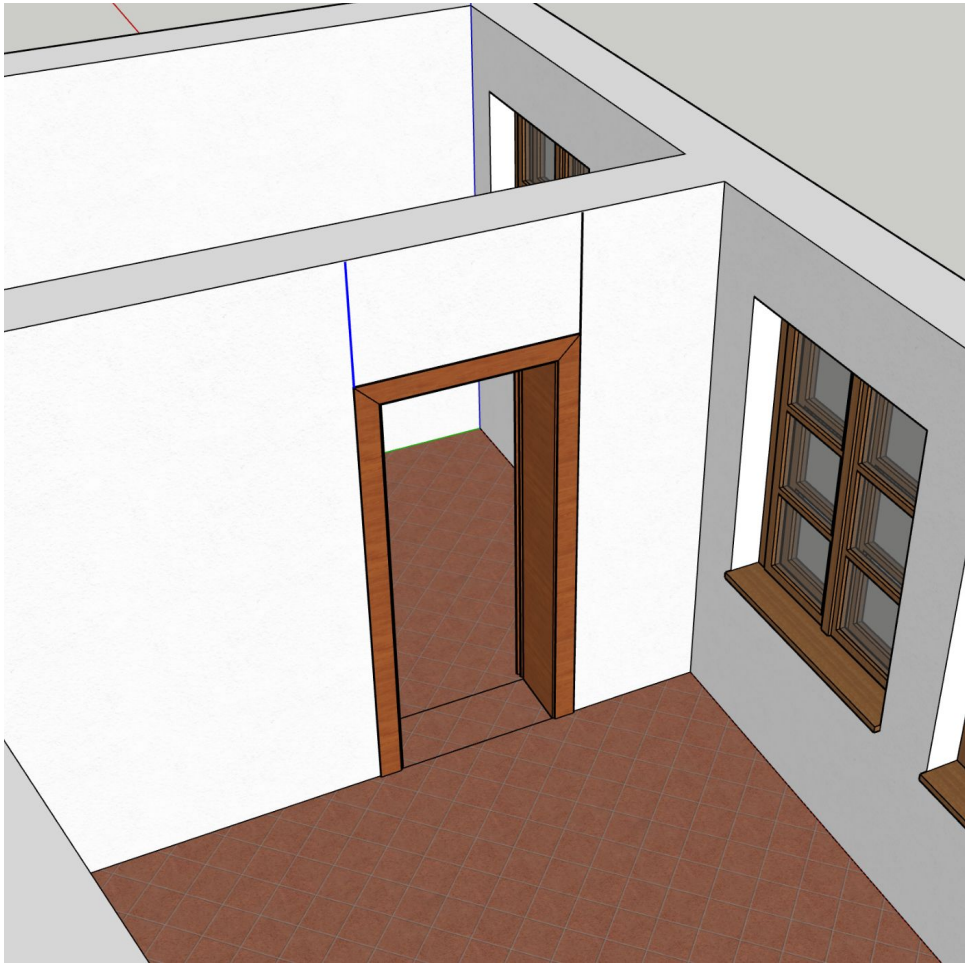
Screen Coordinate Space

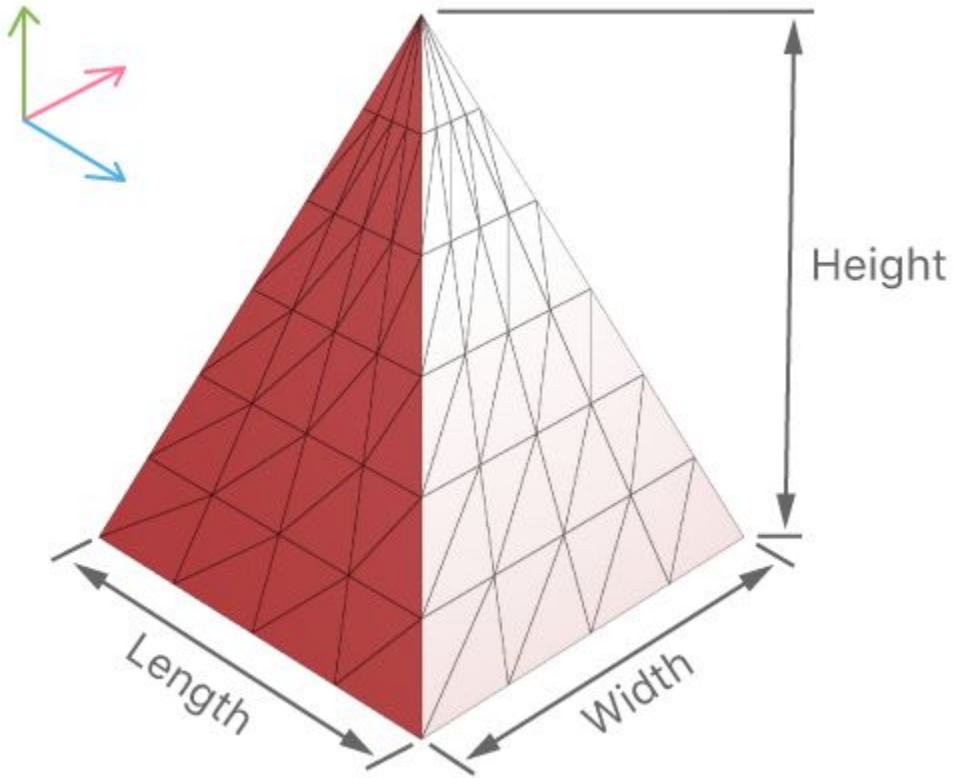
- 2D
- Positions on the device's screen





vertices edges faces polygons surfaces




Height
Length
Width

AUGMENTED REALITY

- Pokemon Go
- Ikea Place (Interior design)
- Education
  - Interactive learning environments
- Storytelling apps (tourist guides)
- Sports (NBA, F1)

ARKit

02

ARKit

- Creating augmented reality experience
- World tracking using the device's camera and motion sensors
    - Plane detection
    - Face & body tracking

VIEW (RENDERER)

- ARView (RealityKit), ARSCNView (SceneKit), ARSKView (SpriteKit)
- A container for digital content (scene)
- The simplest way to combine camera data with a digital content

ARSESSION & ARCONFIGURATION

- **ARSession**
  - Manages the virtual space: Movement detection, camera control, camera data analysis
  - The previous analogy: Mapping of the physical world

- **ARConfiguration**
  - Setting for the scene’s session
  - ARWorldTrackingConfiguration vs ARFaceTrackingConfiguration vs …

ARKit EXAMPLE

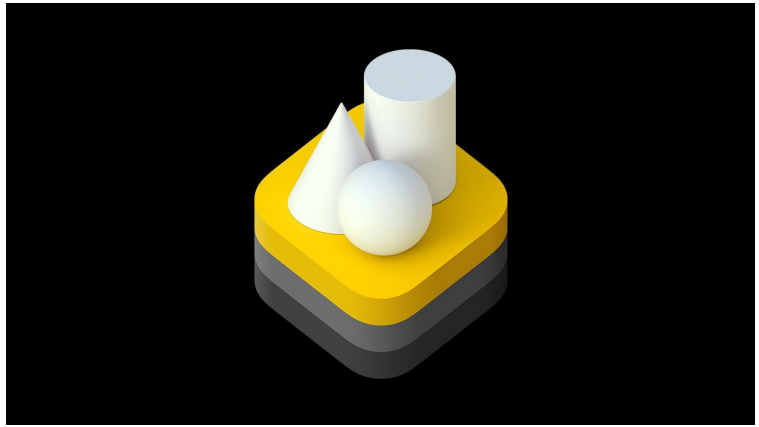
```
let arView = ARView(frame: .zero)

let config = ARWorldTrackingConfiguration()
config.planeDetection = [.horizontal]
arView.session.run(config)
```

RealityKit

03

RealityKit

- A higher-level framework for building AR experiences
- Simplifies rendering, physics, and animation for AR objects
- Built-in support for 3D models, materials, and animations
- Realistic rendering with physically-based shading
- Works seamlessly with Reality Composer



SUPPORTED FILE FORMATS

- DAE (Collada's Digital Asset Exchange)
- USDZ (Pixar's Zipped Universal Scene Description - preferred)
- rcproject / reality
- Optimizing 3D assets for performance
- Managing multiple anchors and scene complexity
- Object capture
- Many more, see here

RealityKit EXAMPLE

```
import RealityKit

// Add anchor to ARKit
arView.session.delegate = self

func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors {
        let anchorEntity = AnchorEntity(world: anchor.transform)

        // Add a 3D model to the anchor
        let box = ModelEntity(mesh: .generateBox(size: 0.1))
        anchorEntity.addChild(box)

        // Attach RealityKit's anchor to ARView
        arView.scene.addAnchor(anchorEntity)
    }
}
```

KEY CONCEPTS 04

COORDINATE SYSTEM

- World space vs local space
- ARAnchor, ARSession
- Transformation matrix
    - Position, rotation, scale

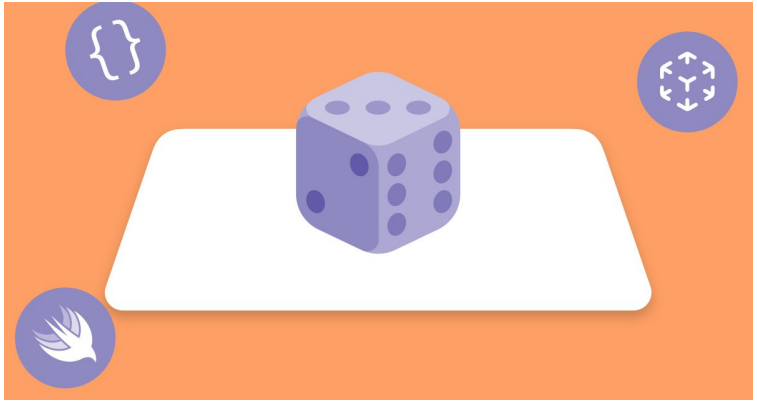
CHALLENGES

- Maintaining **accurate tracking** of the device's position and orientation in the real world
- AR experiences depend heavily on the **physical environment**
- Not all devices **support the same level of AR** capabilities
- **Mapping a 2D touch input** to a meaningful 3D location in the real world
- Ensuring **virtual objects are occluded** (hidden) by real-world objects when appropriate
- Creating **high-quality 3D models** and animations

BUILDING AR 05

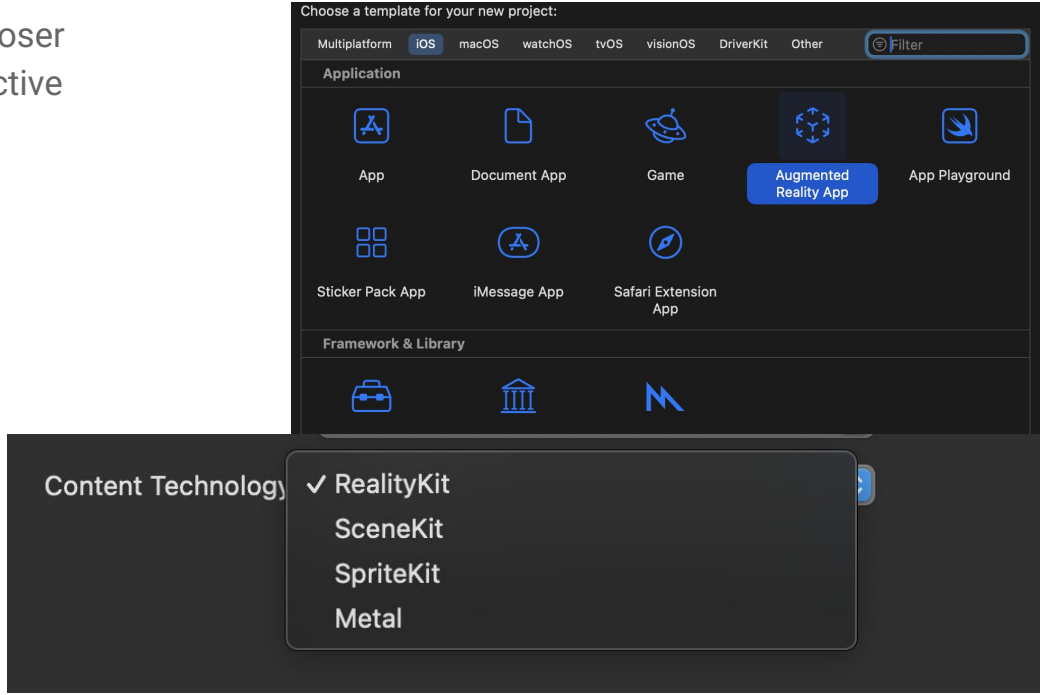
BUILDING AR EXPERIENCE

- New AR project
- Select content system
- Check requited HW (LIDAR)
- Add view with ARSession & rootAnchors
- Track real-time plane detection and object placement
- Interact with augmented objects



BUILDING AR EXPERIENCE

- Building AR scenes using Reality Composer
- Loading 3D models and creating interactive objects with RealityKit APIs
- Physics and animations in RealityKit
- Create custom entity programmatically
- Modifying position and rotation using transformation matrices

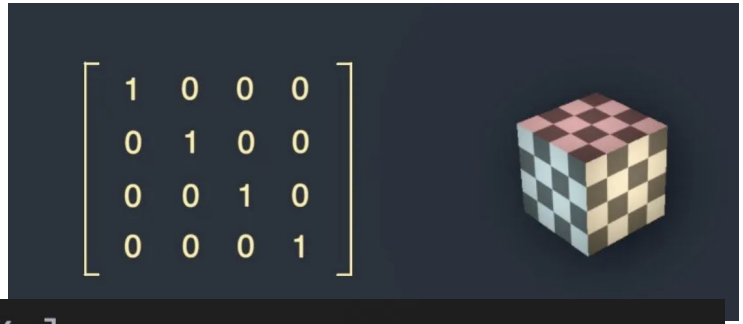


TRANSFORMATION MATRIX

06

TRANSFORMATION MATRIX

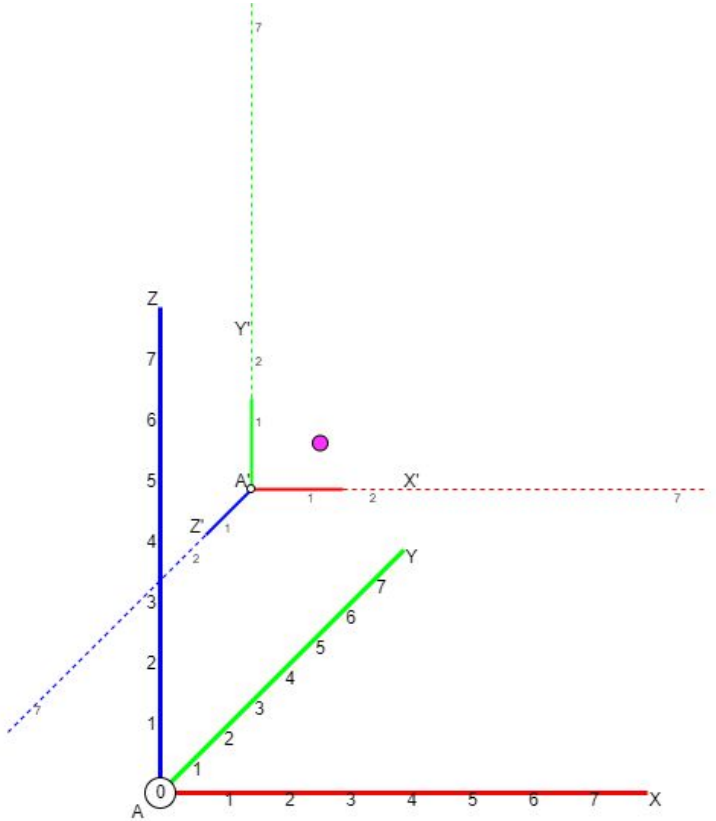
- Local & world coordinates
- Mathematical representation to represent position, rotation, and scale in 3D space
- simd_float4x4




```
[ ScaleX, ShearXY, ShearXZ, TranslationX ]
[ ShearYX, ScaleY, ShearYZ, TranslationY ]
[ ShearZX, ShearZY, ScaleZ, TranslationZ ]
[    0,       0,       0,       1       ] (homogeneous coordinate)
```

SPACES

- World point
  - Local origin + local point
- Example
  - (0.5, 0.5, 0.5) + (0.2, 0.2, 0.2) = (0.7, 0.7, 0.7)



TRANSFORMATION MATRIX

- Transformation Matrix = T * R * S
- T is translation. It is a 4x4 matrix.
- R is rotation. It is a 4x4 matrix.
- R is scale. It is a 4x4 matrix

Rotation around x_axis

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix}$$

Rotation around y_axis

$$R_y = \begin{bmatrix} \cos(t) & 0 & \sin(t) \\ 0 & 1 & 0 \\ -\sin(t) & 0 & \cos(t) \end{bmatrix}$$

Rotation around z_axis

$$R_z = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

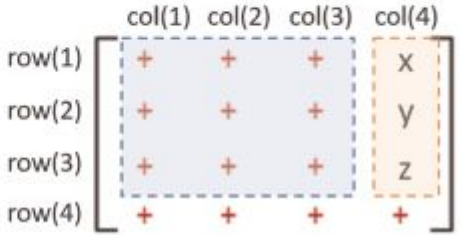
Scale 3D

$$S = \begin{bmatrix} scale & 0 & 0 \\ 0 & scale & 0 \\ 0 & 0 & scale \end{bmatrix}$$

Translation

$$V = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

4×4 transformation matrix:



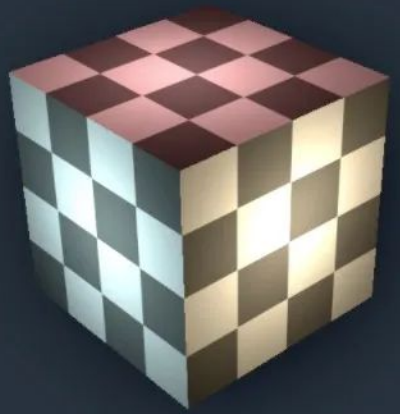
```
import ARKit
import simd

func extractComponents(from matrix: simd_float4x4) {
    let translation = matrix.columns.3 // Translation vector
    let rotation = simd_quaternion(matrix) // Convert rotation matrix to quaternion
    let scale = simd_length(matrix.columns.0) // Length of first column vector

    print("Translation: \(translation)")
    print("Rotation (quaternion): \(rotation)")
    print("Scale: \(scale)")
}
```

MATRIX MULTIPLICATION

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} i & j \\ k & l \end{bmatrix} = \begin{bmatrix} a.i + b.k & a.j + b.l \\ c.i + d.k & c.j + d.l \end{bmatrix}$$

SCALE

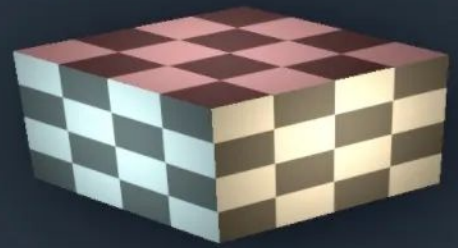
$$\begin{bmatrix} 1.7 & 0 & 0 & 0 \\ 0 & 1.7 & 0 & 0 \\ 0 & 0 & 1.7 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



SCALE

SCALE

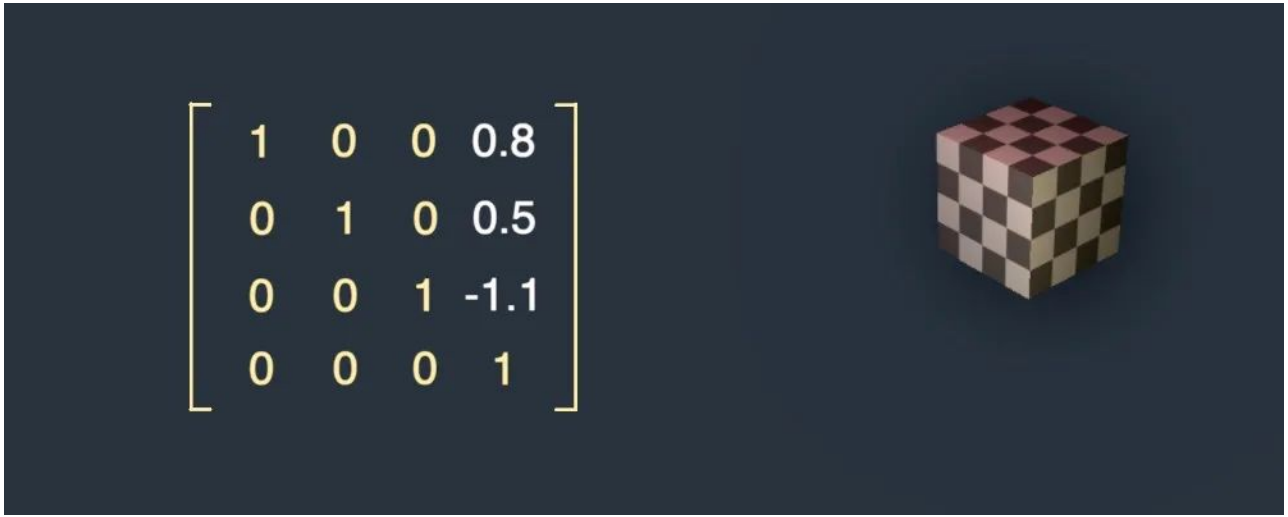
$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



MIRROR

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

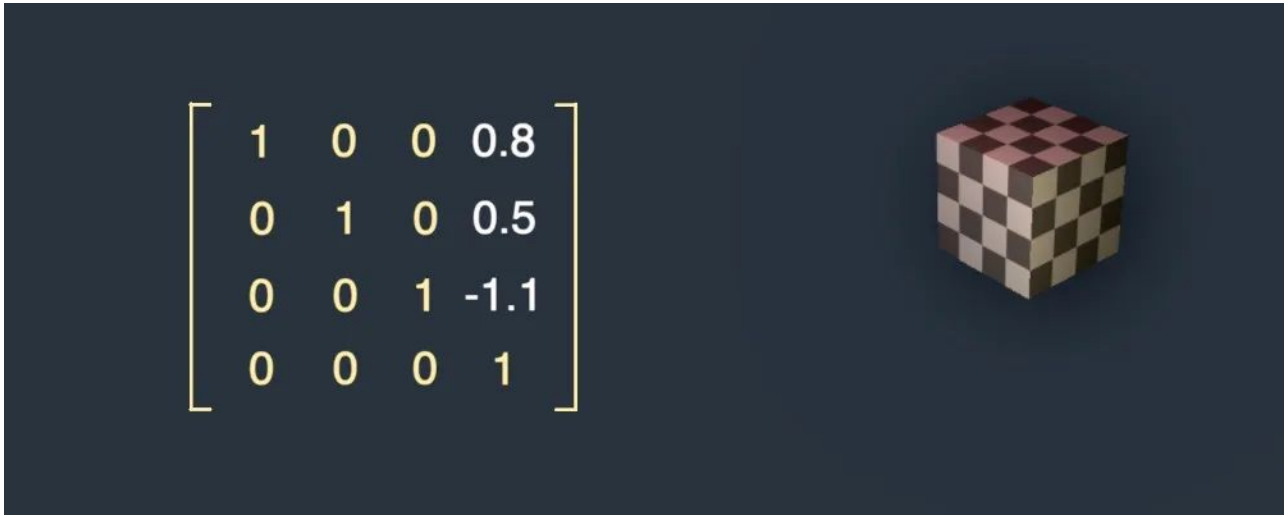
TRANSLATE



TRANSLATE EXAMPLE

```
// Identity matrix
let identity = matrix_identity_float4x4

// Translation matrix (move by [1, 2, 3])
let translation = simd_float4x4(SIMD4<Float>(1, 0, 0, 0),
                                SIMD4<Float>(0, 1, 0, 0),
                                SIMD4<Float>(0, 0, 1, 0),
                                SIMD4<Float>(1, 2, 3, 1))

// Apply the transformation to a point
let point = SIMD4<Float>(0, 0, 0, 1)
let transformedPoint = translation * point
print(transformedPoint) // Output: [1.0, 2.0, 3.0, 1.0]
```

TRANSLATE



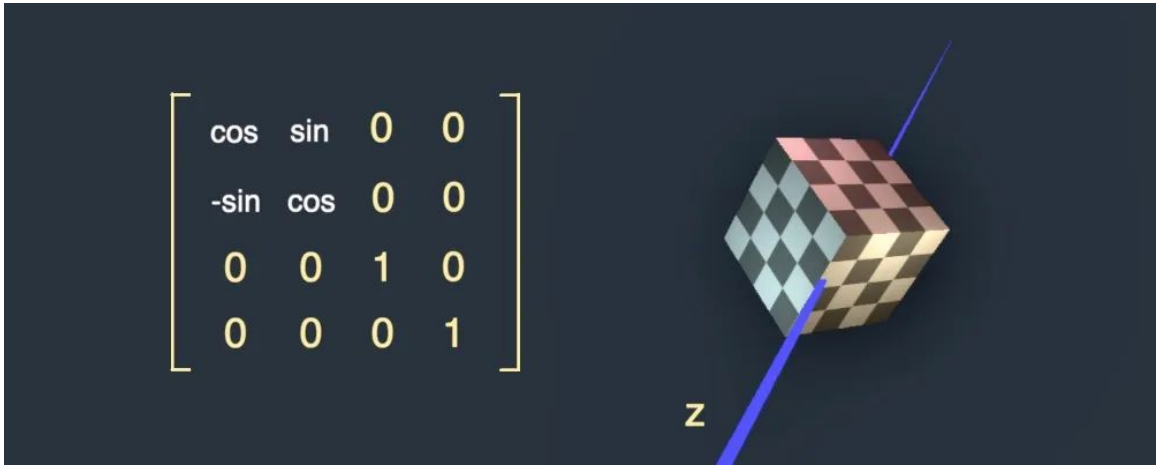
SHEAR

$$\begin{bmatrix} 1 & xy & xz & 0 \\ yx & 1 & yz & 0 \\ zx & zy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

SHEAR EXAMPLE

$$\begin{bmatrix} 1 & -0.7 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

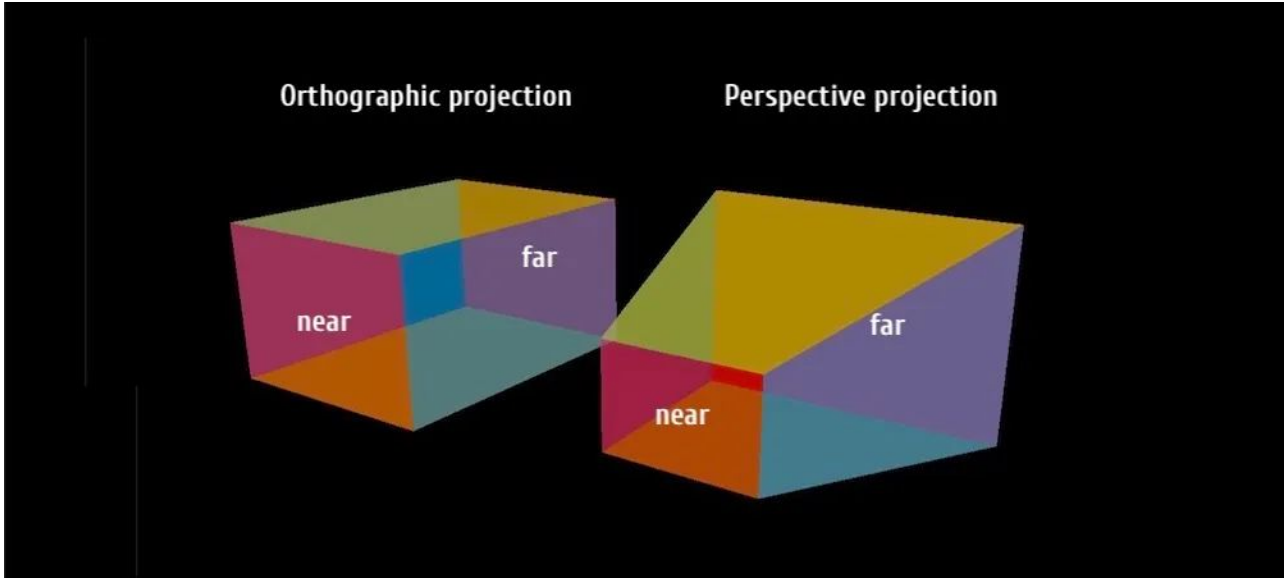
ROTATION



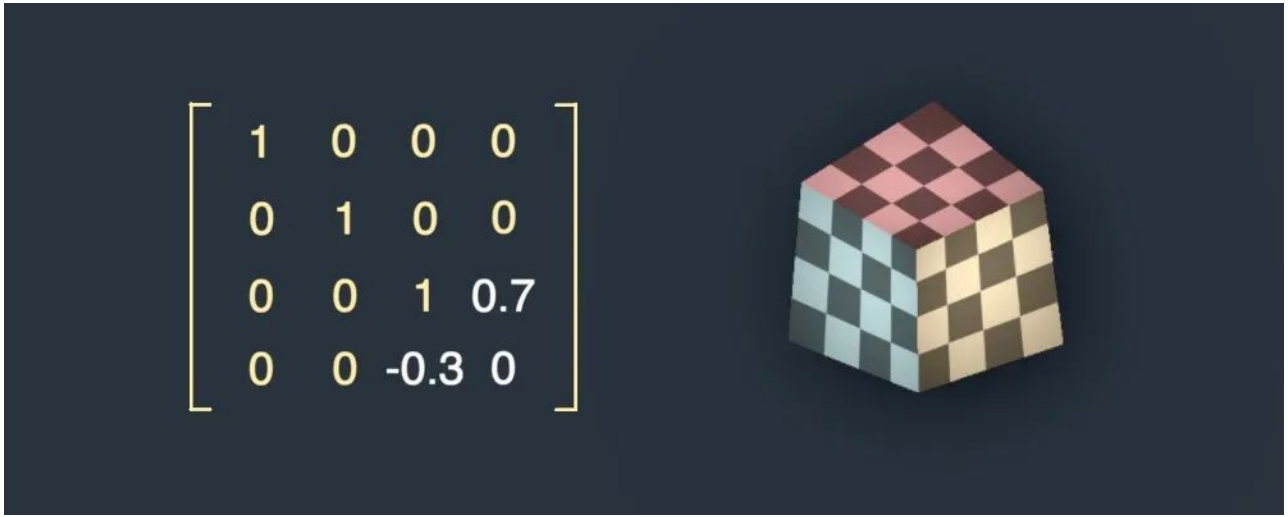
ROTATE EXAMPLE

```
// Update an entity's position
let entity = ModelEntity()
entity.transform.translation = SIMD3<Float>(0.5, 0.0, -1.0)

// Rotate the entity 90 degrees around Y-axis
let rotationMatrix = simd_float4x4(SIMD4<Float>(0, 1, 0, 0),
                                   SIMD4<Float>(-1, 0, 0, 0),
                                   SIMD4<Float>(0, 0, 1, 0),
                                   SIMD4<Float>(0, 0, 0, 1))
entity.transform.matrix *= rotationMatrix
```

PROJECTION


Orthographic projection
Perspective projection
far
near
far
near

PROJECTION



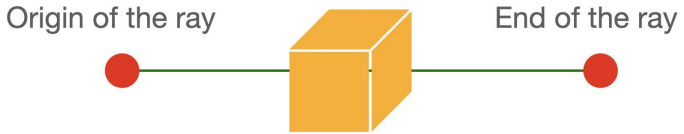
RAYCAST 07

RAYCAST

- Detect the intersection between a virtual ray and objects in the 3D environment

- Line projected from a source point

- Selecting, placing, or interacting with objects based on a point in 3D space
- Return multiple raycast results for the same query, allowing developers to choose the most relevant one
- Missing depth information



RAYCAST EXAMPLE

```
let arView = ARView(frame: .zero)
let touchLocation = CGPoint(x: 100, y: 100) // Example touch point on the screen

// Create a raycast query from the screen touch point
if let raycastQuery = arView.raycastQuery(from: touchLocation, allowing: .existingPlaneInf
    // Perform the raycast
    let results = arView.session.raycast(raycastQuery)

    if let firstResult = results.first {
        // Get the position of the intersection
        let position = firstResult.worldTransform.translation
        print("Intersection point: \(position)")
    } else {
        print("No intersection found")
    }
}
```

RAYCAST EXAMPLE

```
if let raycastResult = arView.session.raycast(raycastQuery).first {
    let position = raycastResult.worldTransform.translation
    let object = ModelEntity(mesh: .generateSphere(radius: 0.1))
    object.position = position
    arView.scene.anchors.append(AnchorEntity(entity: object)
}
```

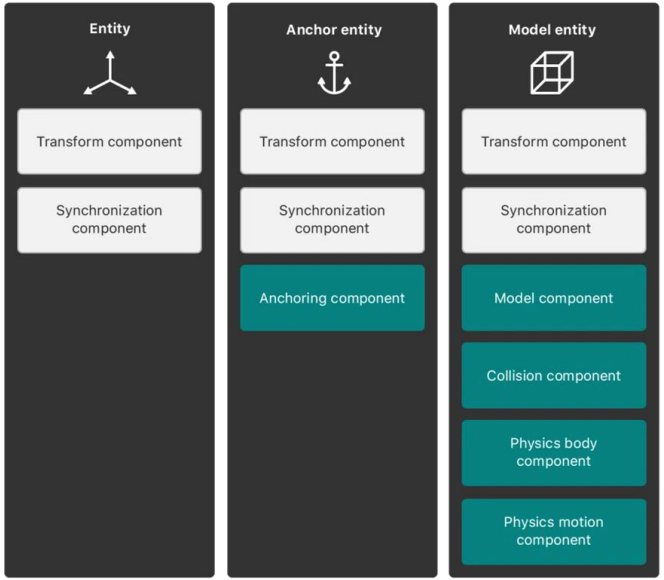
RAYCAST EXAMPLE

```
@objc func handlePanGesture(_ gesture: UIPanGestureRecognizer) {
    let location = gesture.location(in: arView)
    if let raycastQuery = arView.raycastQuery(from: location, allowing: .existingPlaneGeo
        let results = arView.session.raycast(raycastQuery)
        if let result = results.first {
            let newPosition = result.worldTransform.translation
            objectEntity.position = newPosition
        }
    }
}
```

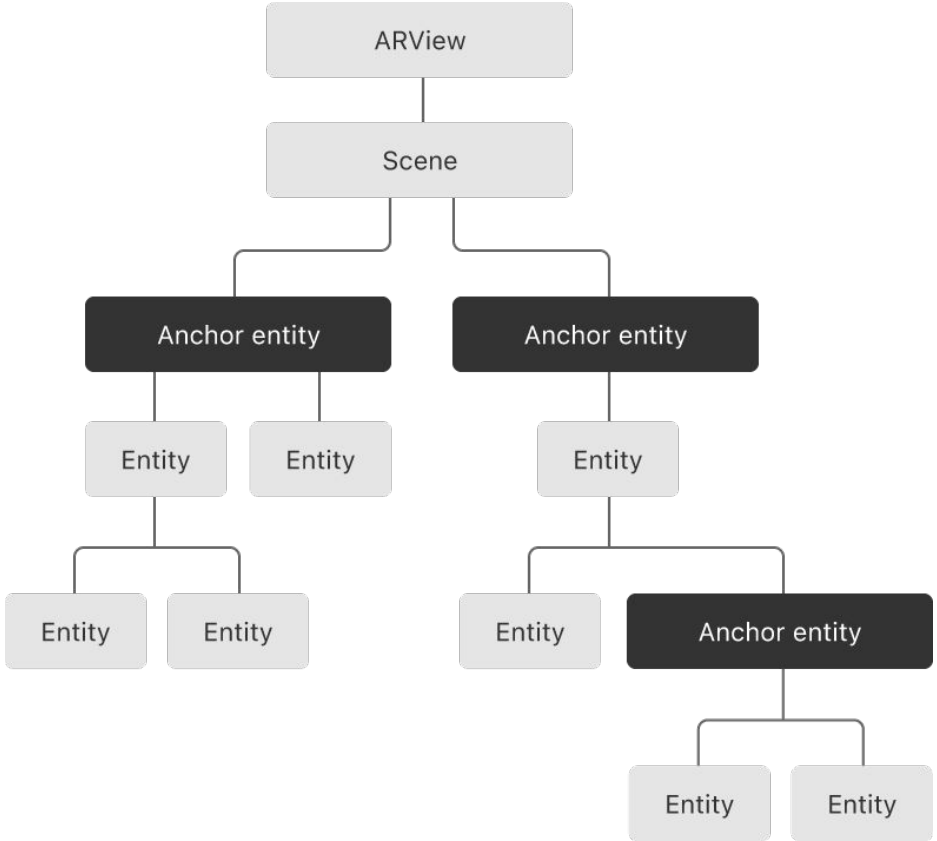
BUILDING BLOCKS 08

ECS SYSTEM

- Entity
  - Basic
  - ModelEntity
- Anchor
  - AnchorEntity
- Component
  - Behavior - physics & animation



ECS SYSTEM


ARView
Scene
Anchor entity
Anchor entity
Entity
Entity
Entity
Entity
Entity
Entity
Anchor entity
Entity
Entity

CODE EXAMPLE

```
// Create an anchor at the world origin
let anchorEntity = AnchorEntity(world: [0, 0, 0])

// Create a red box entity
let boxEntity = ModelEntity(
    mesh: .generateBox(size: [0.1, 0.1, 0.1]),
    materials: [SimpleMaterial(color: .red, isMetallic: false)]
)

// Add collision, physics, and input components
boxEntity.generateCollisionShapes(recursive: true)
boxEntity.components.set(PhysicsBodyComponent())
boxEntity.components.set(PhysicsMotionComponent())
boxEntity.components.set(InputTargetComponent())

// Add the box to the anchor and anchor to the scene
anchorEntity.addChild(boxEntity)
content.add(anchorEntity)
```

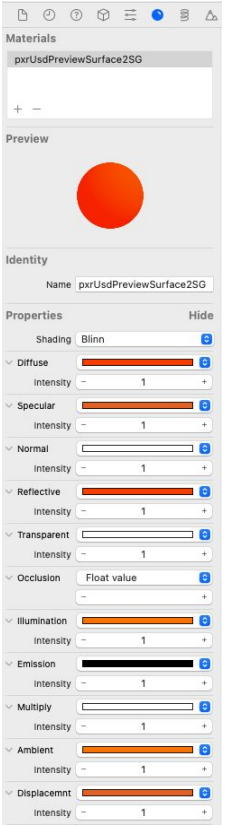
REALITY COMPOSER PRO

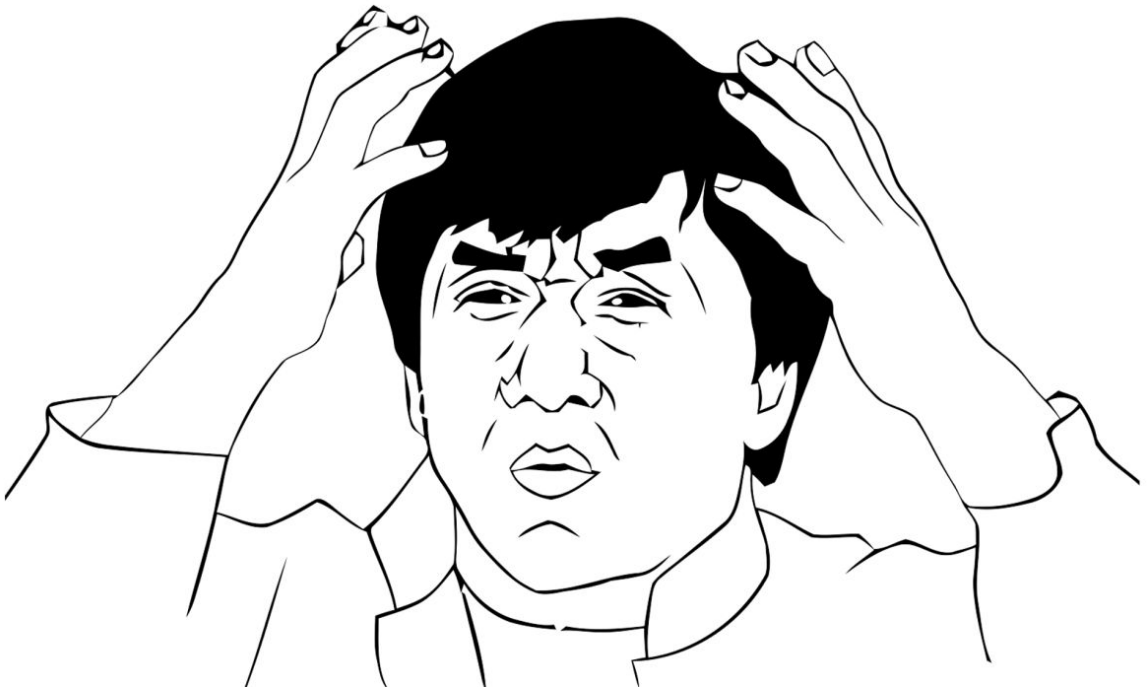
- 3D studio
- Direct integration with Xcode
- Create or export a .reality file using Reality Composer
- AnchorEntity.load(named:)
- Entity.load(named:)

```
class ViewController: UIViewController {
    var arView: ARView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Step 1: Set up the ARView
        arView = ARView(frame: self.view.bounds)
        self.view.addSubview(arView)

        // Step 2: Load the Reality Composer Model
        loadRealityComposerModel()
    }

    private func loadRealityComposerModel() {
        // The name of the scene group in your .reality file (usually m
        let modelName = "Scene" // Replace with your .reality file's sc

        // Attempt to load the model as an AnchorEntity
        do {
            let anchor = try AnchorEntity.load(named: modelName)
            // Add the loaded anchor to the ARView scene
            arView.scene.addAnchor(anchor)
            print("Model successfully loaded!")
        } catch {
            print("Failed to load the model: \(error.localizedDescripti
        }
    }
}
```

REALITY COMPOSER PRO

REALITY COMPOSER FOR IPAD





[Youtube: Texture Maps Explained]

QUESTIONS

STRV

THANK YOU!

STRV