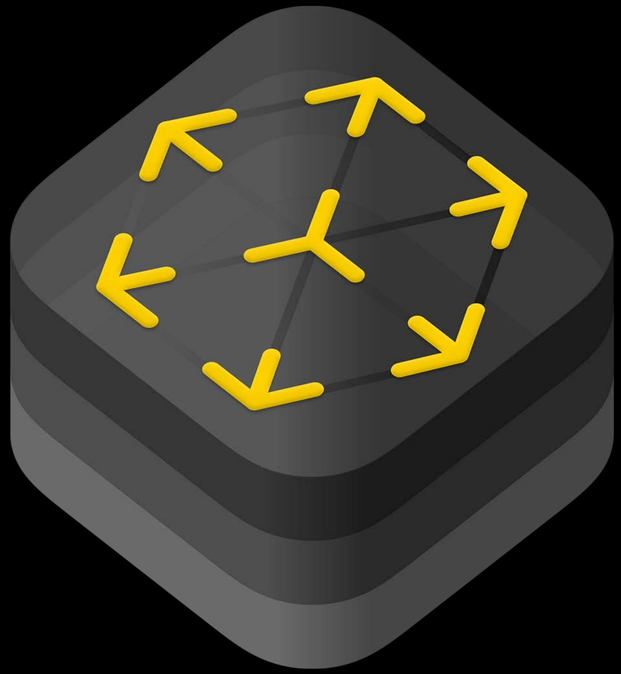
#WWDC19

Bringing People into AR

Adrian Lindberg
Tanmay Batra

RealityKit



ARKit 3

People Occlusion

Motion Capture

People Occlusion

Occlusion between people and rendered content

Overview

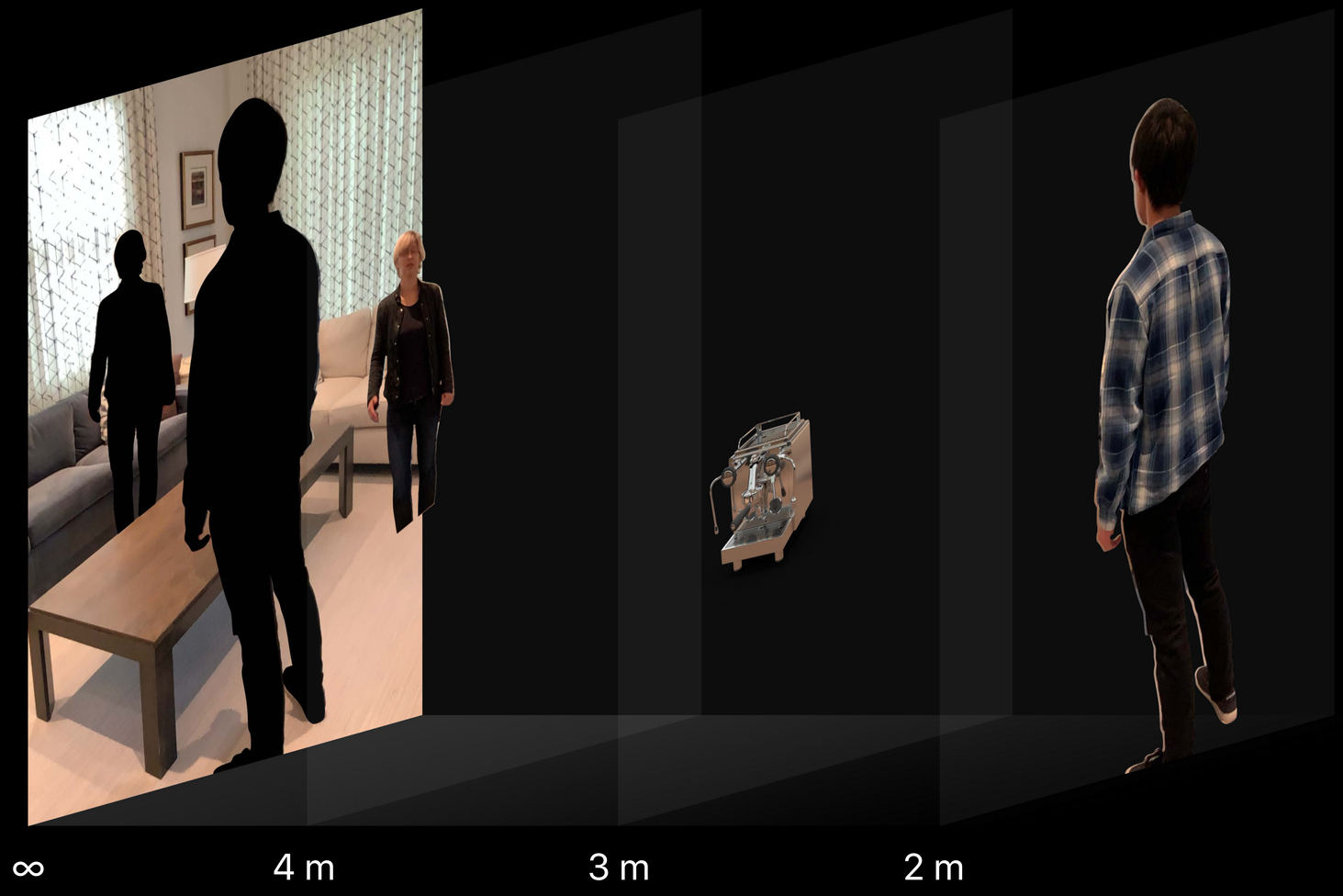
Overview

Overview

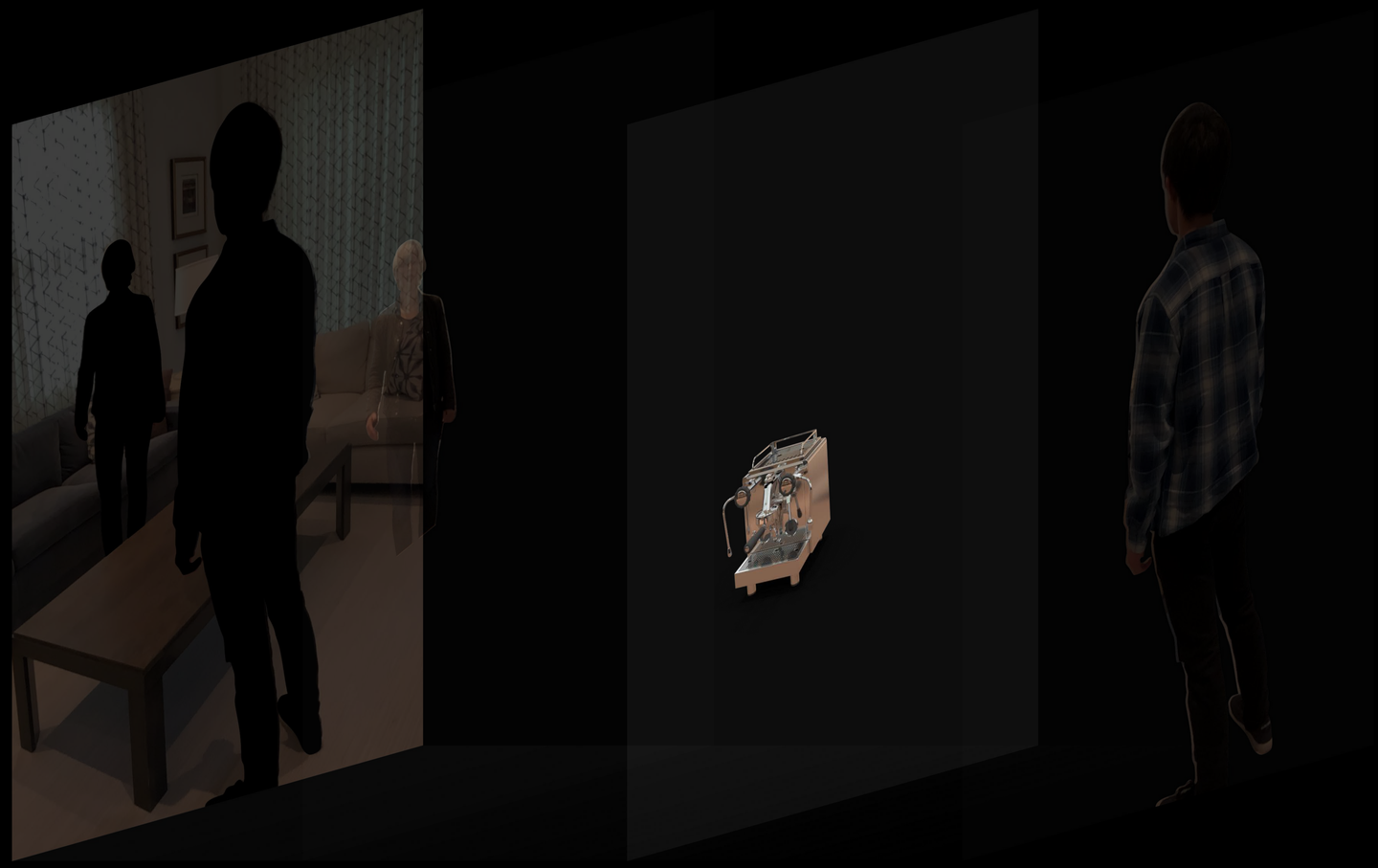
Overview

Overview

Overview

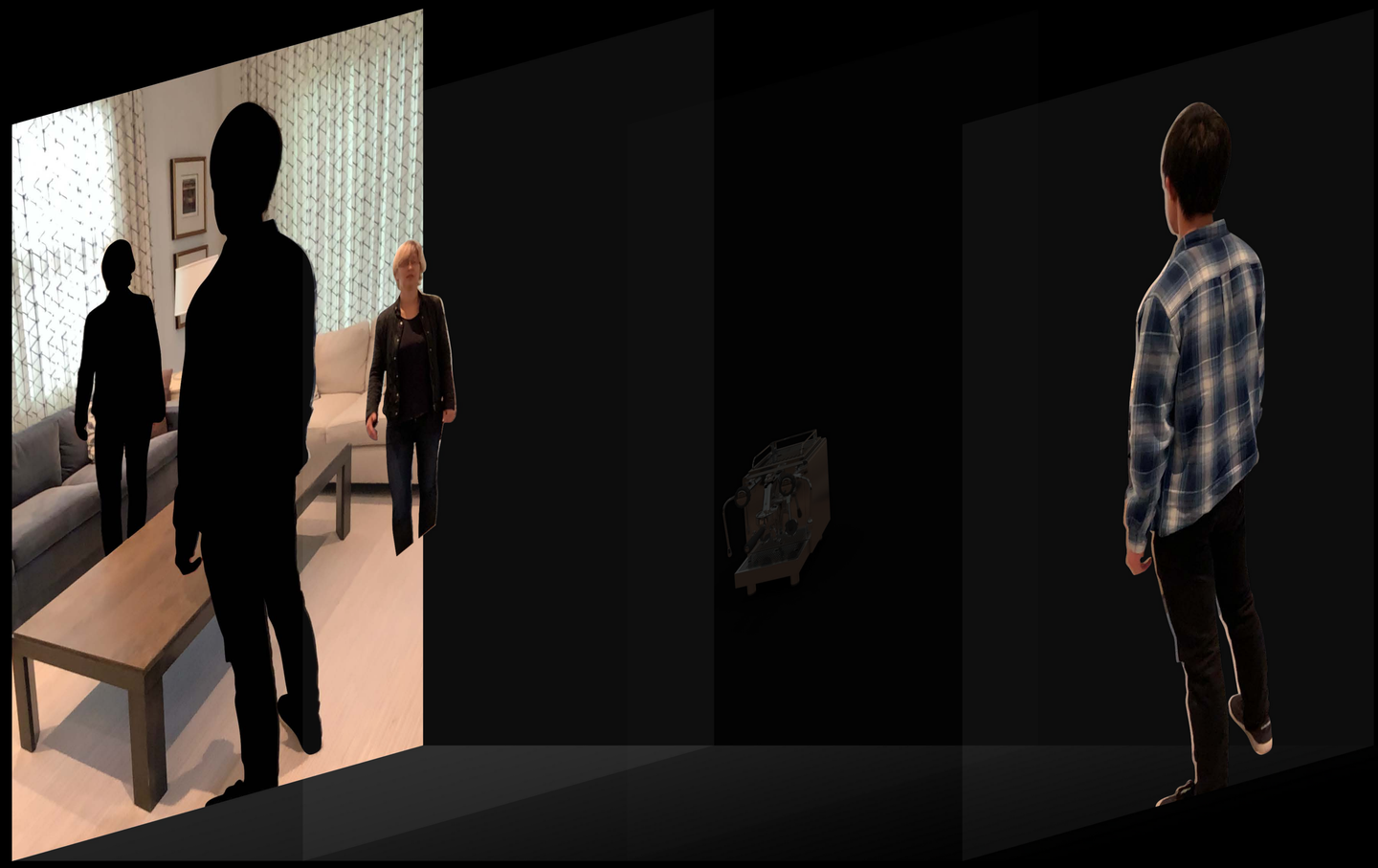


Overview



Rendered

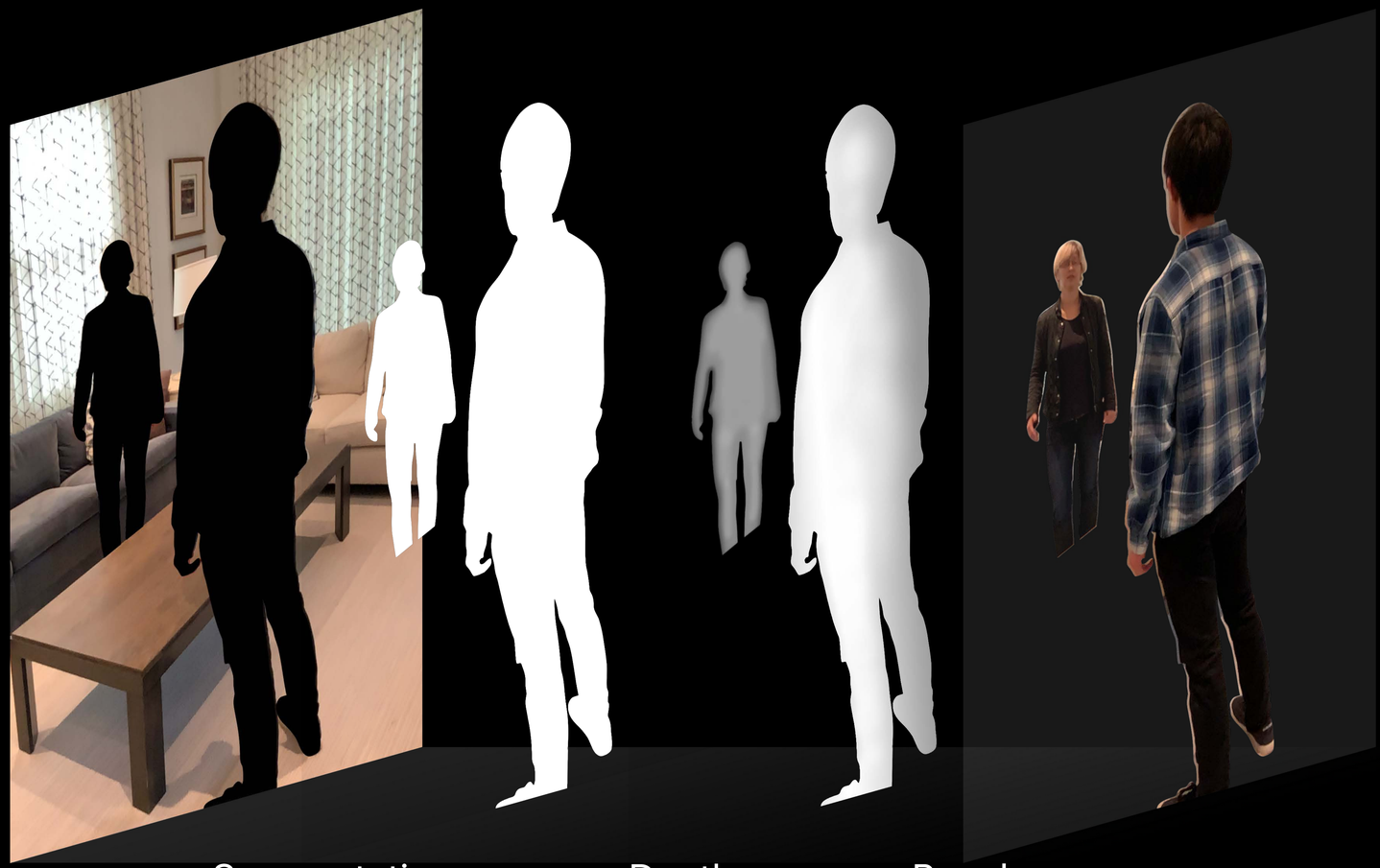
Overview



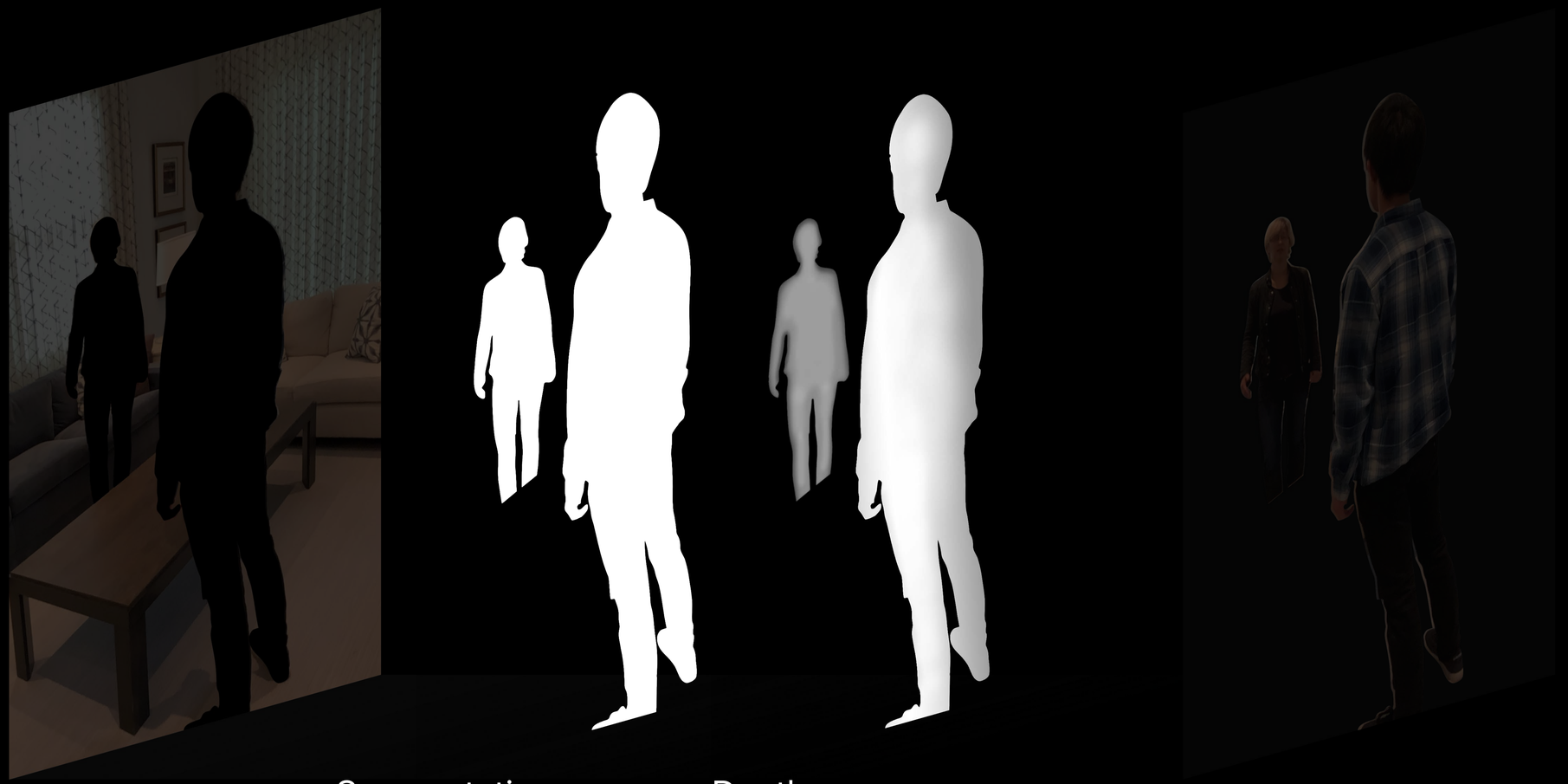
How It Works

People

How It Works



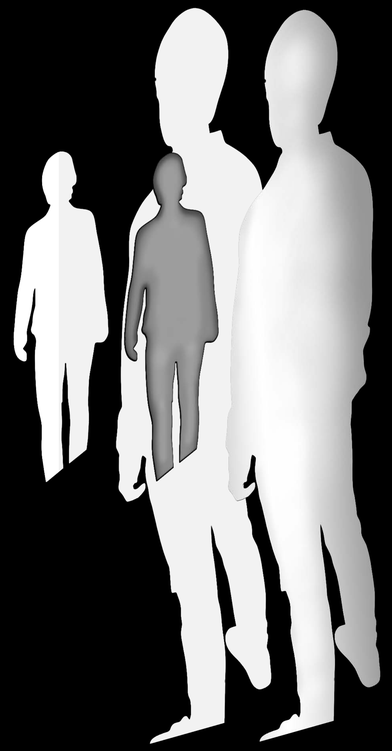
Machine Learning



Segmentation

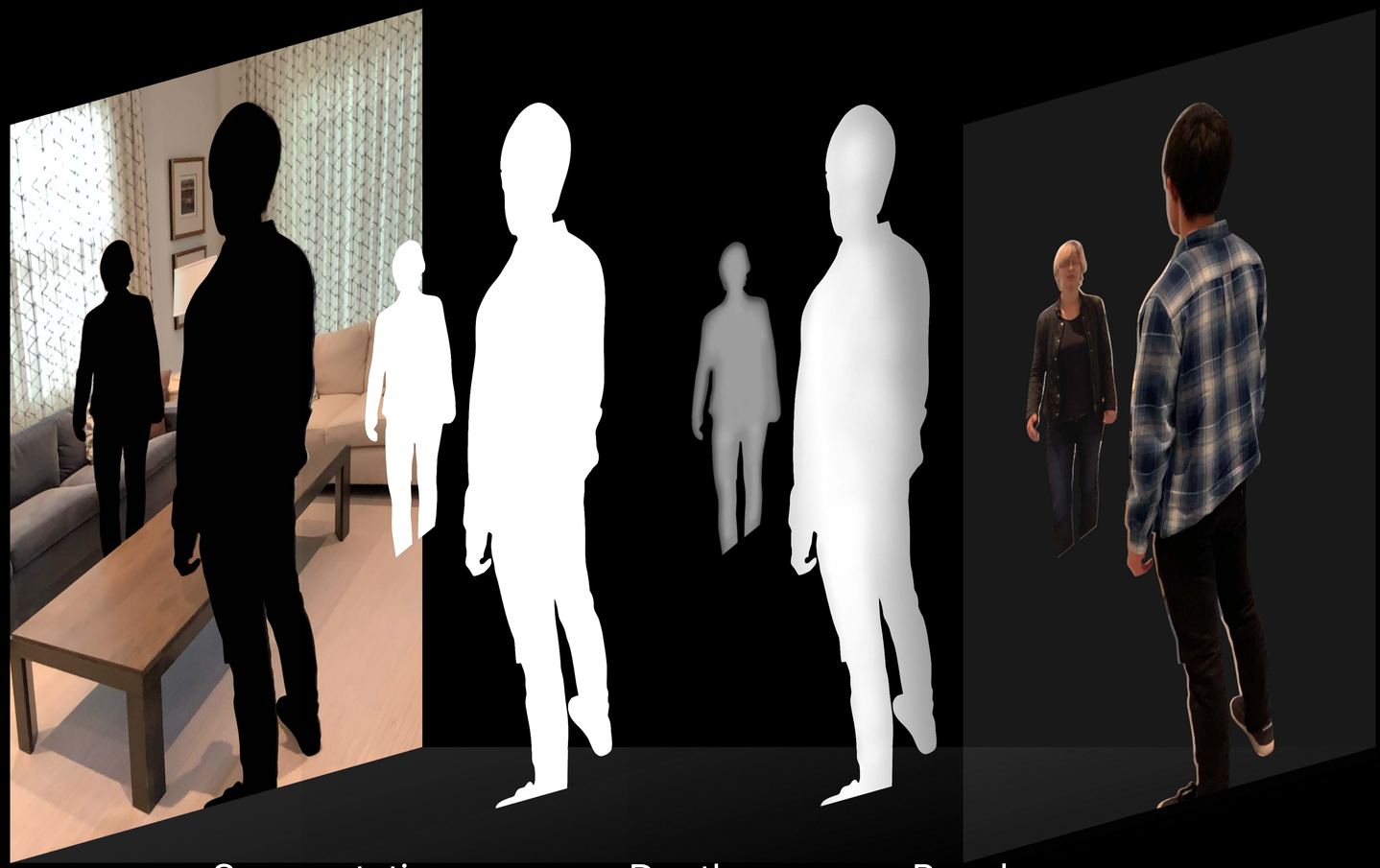
Depth

ARFrame



```
var segmentationBuffer : CVPixelBuffer? { get }
var estimatedDepthData : CVPixelBuffer? { get }
```

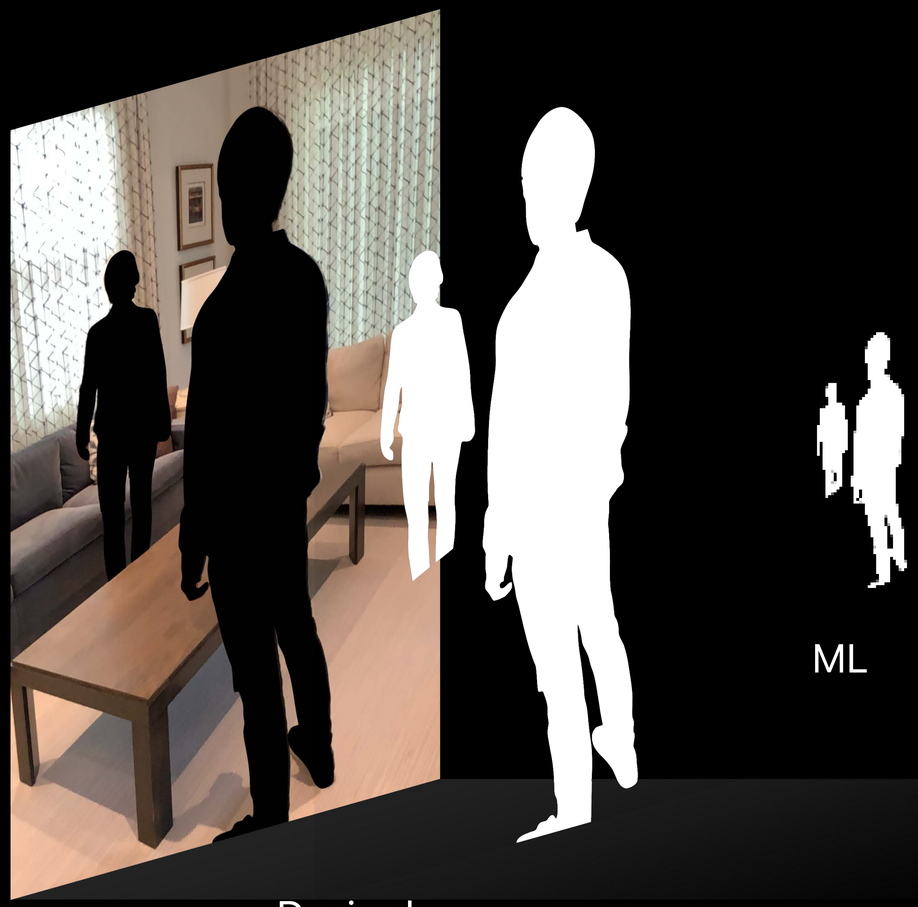
How It Works



How It Works

Desired

How It Works

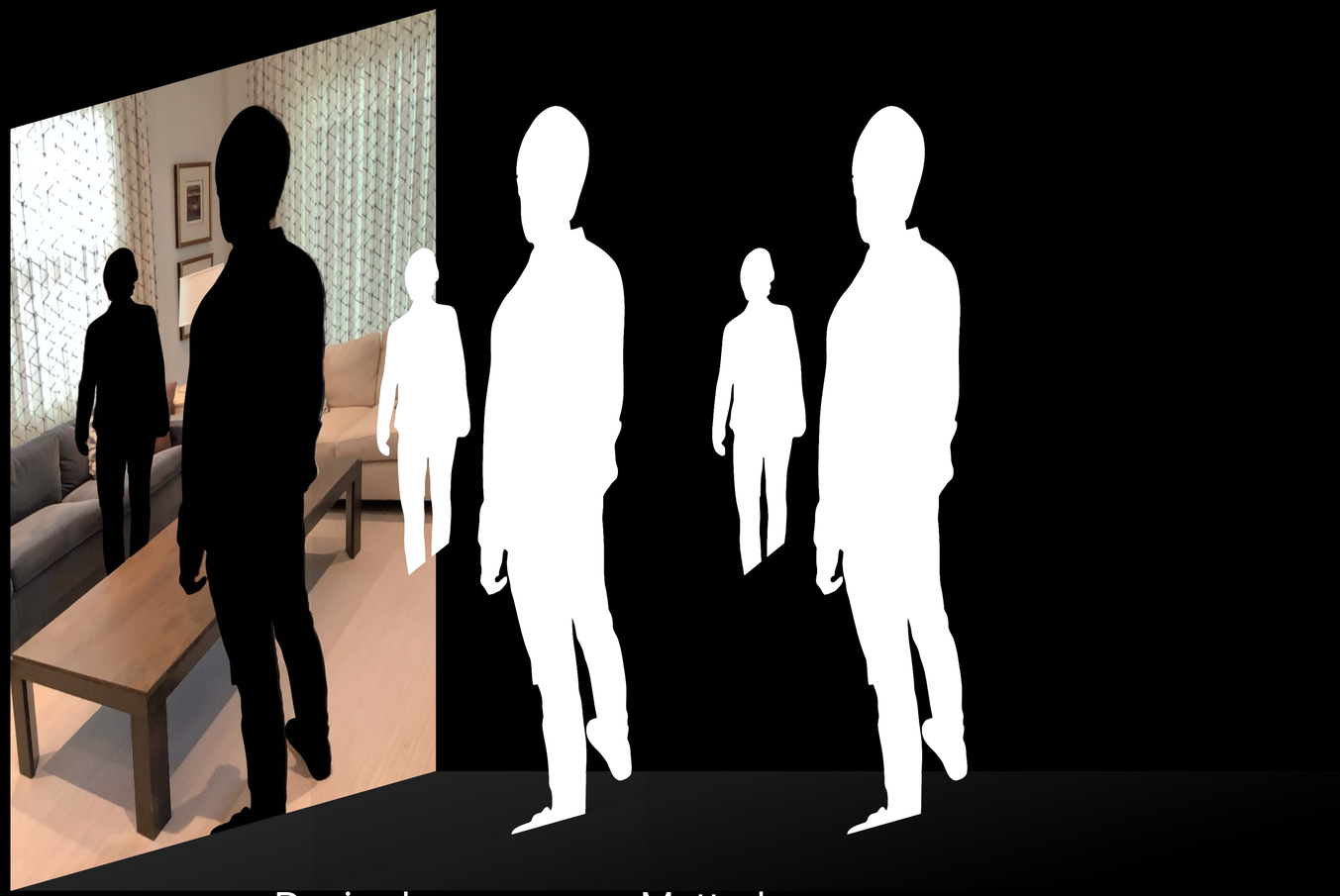


Desired

ML

How It Works

Matting

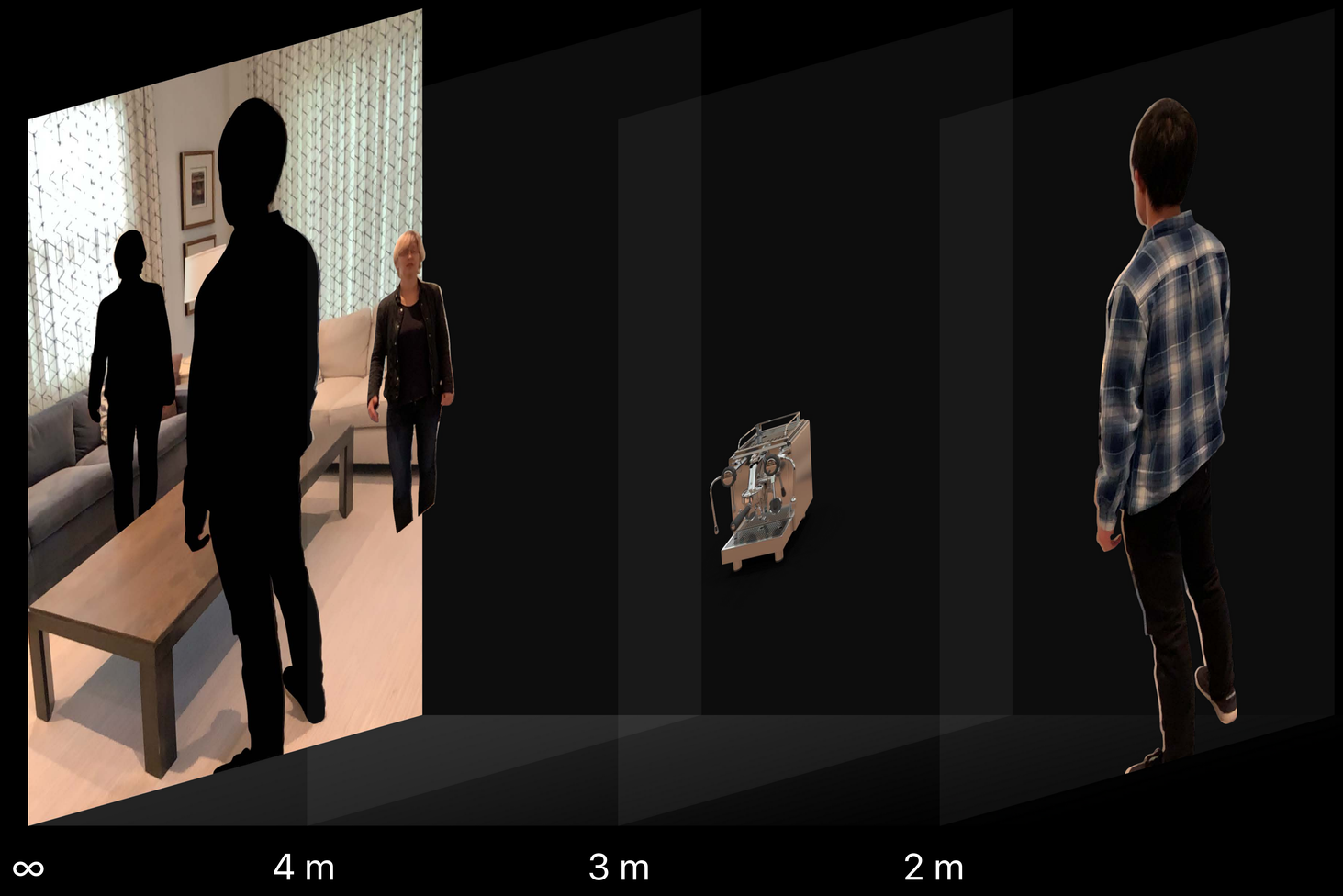


Desired Matted

Matting

Matted People

Matting



Composition

Composition

RealityKit

SceneKit

Metal

RealityKit

NEW

Provides new ARView

RealityKit

NEW

Provides new ARView

Easy to use API for photorealism in AR

RealityKit

NEW

Provides new ARView

Easy to use API for photorealism in AR

Built-in support for People Occlusion

```swift
override func viewDidLoad() {
  super.viewDidLoad()
  // Check If Supported
  guard let config = arView.session.config as? ARWorldTrackingConfiguration,
    ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Enable Frame Semantics
  config.frameSemantics = .personSegmentationWithDepth
}
```

```
override func viewDidLoad() {
  super.viewDidLoad()
  // Check If Supported
  guard let config = arView.session.config as? ARWorldTrackingConfiguration,
    ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Enable Frame Semantics
  config.frameSemantics = .personSegmentationWithDepth
}
```

```
override func viewDidLoad() {
  super.viewDidLoad()
  // Check If Supported
  guard let config = arView.session.config as? ARWorldTrackingConfiguration,
    ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Enable Frame Semantics
  config.frameSemantics = .personSegmentationWithDepth
}
```

Update Your Configuration

NEW

```
enum ARFrameSemantics {
    case PersonSegmentation // Only People, No Depth
    case PersonSegmentationWithDepth
}

class ARWorldTrackingConfiguration: ARConfiguration {
     var frameSemantics: ARFrameSemantics { get set }
}
```

ARView

Recommended way for new apps

ARView

Recommended way for new apps

Deep renderer integration

ARView

Recommended way for new apps

Deep renderer integration

Works with transparent objects

ARView

Recommended way for new apps

Deep renderer integration

Works with transparent objects

Built for optimal performance

SceneKit

SceneKit

SceneKit

Added support to ARSCNView

SceneKit

SceneKit

Added support to ARSCNView

Just enable frame semantics

SceneKit

SceneKit

Added support to ARSCNView

Just enable frame semantics

Does composition as a post-process

SceneKit

SceneKit

Added support to ARSCNView

Just enable frame semantics

Does composition as a post-process

May not work well with transparency

SceneKit

Custom Composition

Metal

Custom Composition

Integrate into your own renderer

Metal

Custom Composition

Integrate into your own renderer

Complete control of composition

Metal

Custom Composition

Integrate into your own renderer

Complete control of composition

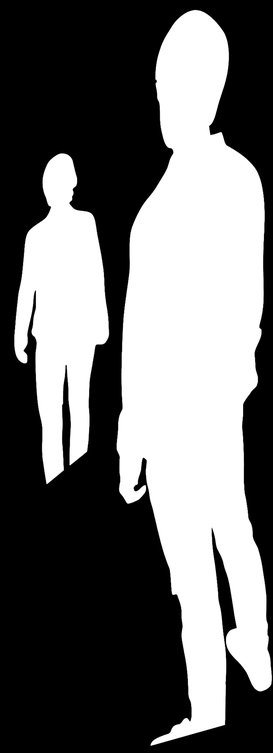
Provide simple access to required functionality

Metal

Custom Composition

ML

Custom Composition



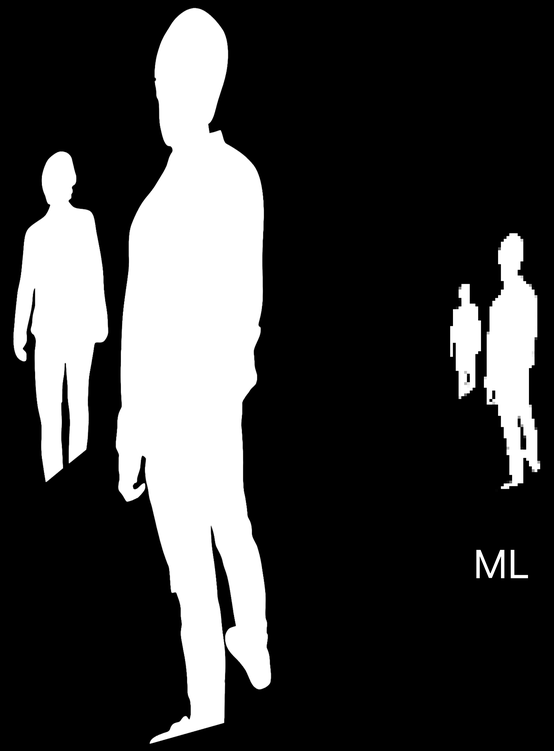
Matte

ML

Custom Composition

New class for generating matte

Uses Metal to provide a texture



```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition Part of the Rendering Code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Schedule Matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)



  // Custom composition code

  // Done
  commandBuffer.commit()
}
```

```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition Part of the Rendering Code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Schedule Matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)



  // Custom composition code

  // Done
  commandBuffer.commit()
}
```

```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition Part of the Rendering Code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Schedule Matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)


  // Custom composition code

  // Done
  commandBuffer.commit()
}
```

```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition Part of the Rendering Code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Schedule Matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)


  // Custom composition code

  // Done
  commandBuffer.commit()
}
```

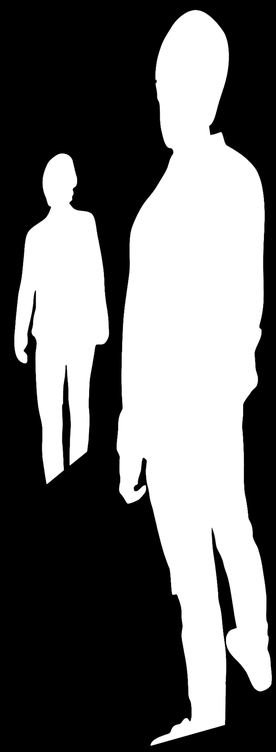
```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition Part of the Rendering Code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else {
    return
  }

  // Schedule Matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)



  // Custom composition code

  // Done
  commandBuffer.commit()
}
```

Custom Composition

NEW

```
class ARMatteGenerator: NSObject {
  func generateMatte(from: ARFrame,
                     commandBuffer: MTLCommandBuffer) -> MTLTexture
}
```

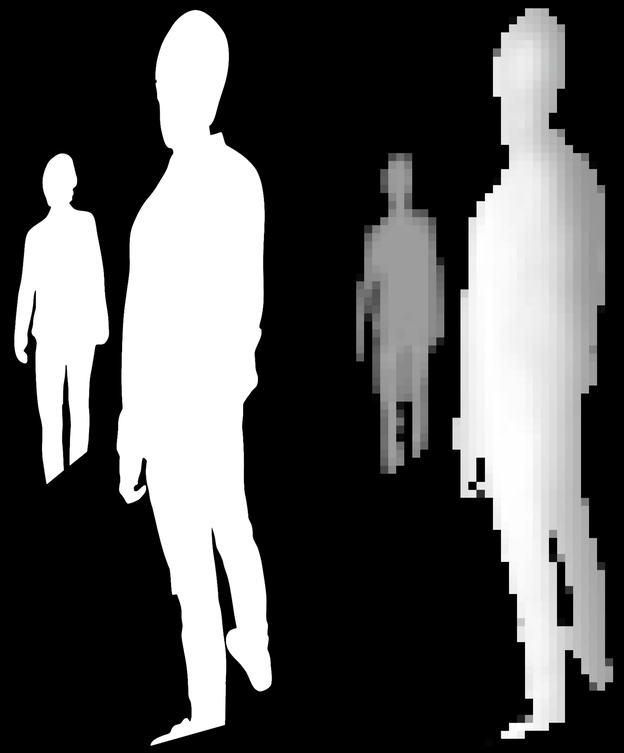
Custom Composition



Matte

ML

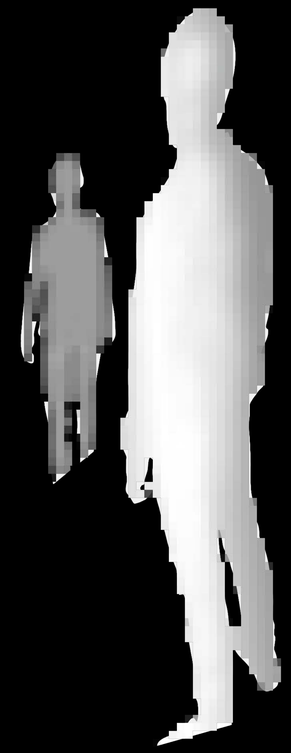
Custom Composition



Matte

ML

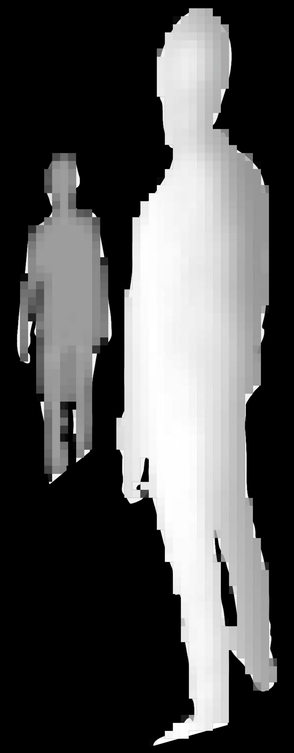
Custom Composition



Mismatch

Custom Composition

Matte and depth resolution mismatch

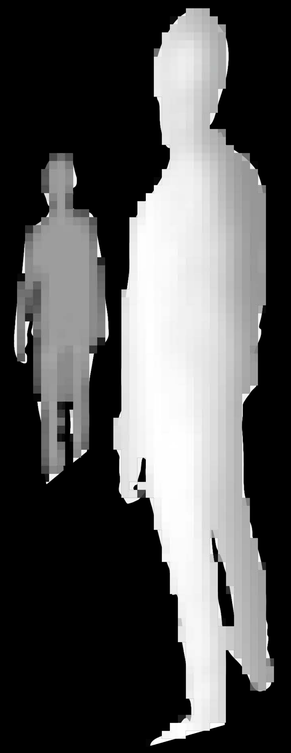


Mismatch

Custom Composition

Matte and depth resolution mismatch

Can’t be solved by alpha only



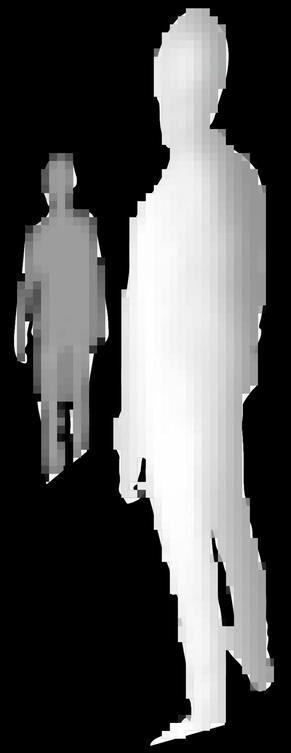
Mismatch

Custom Composition

Matte and depth resolution mismatch

Can’t be solved by alpha only

Need to modify estimated depth buffer



Mismatch

```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition part of the rendering code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else
{
    return
  }

  // Schedule matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)


  // Custom composition shader

  // Done
  commandBuffer.commit()
}
```

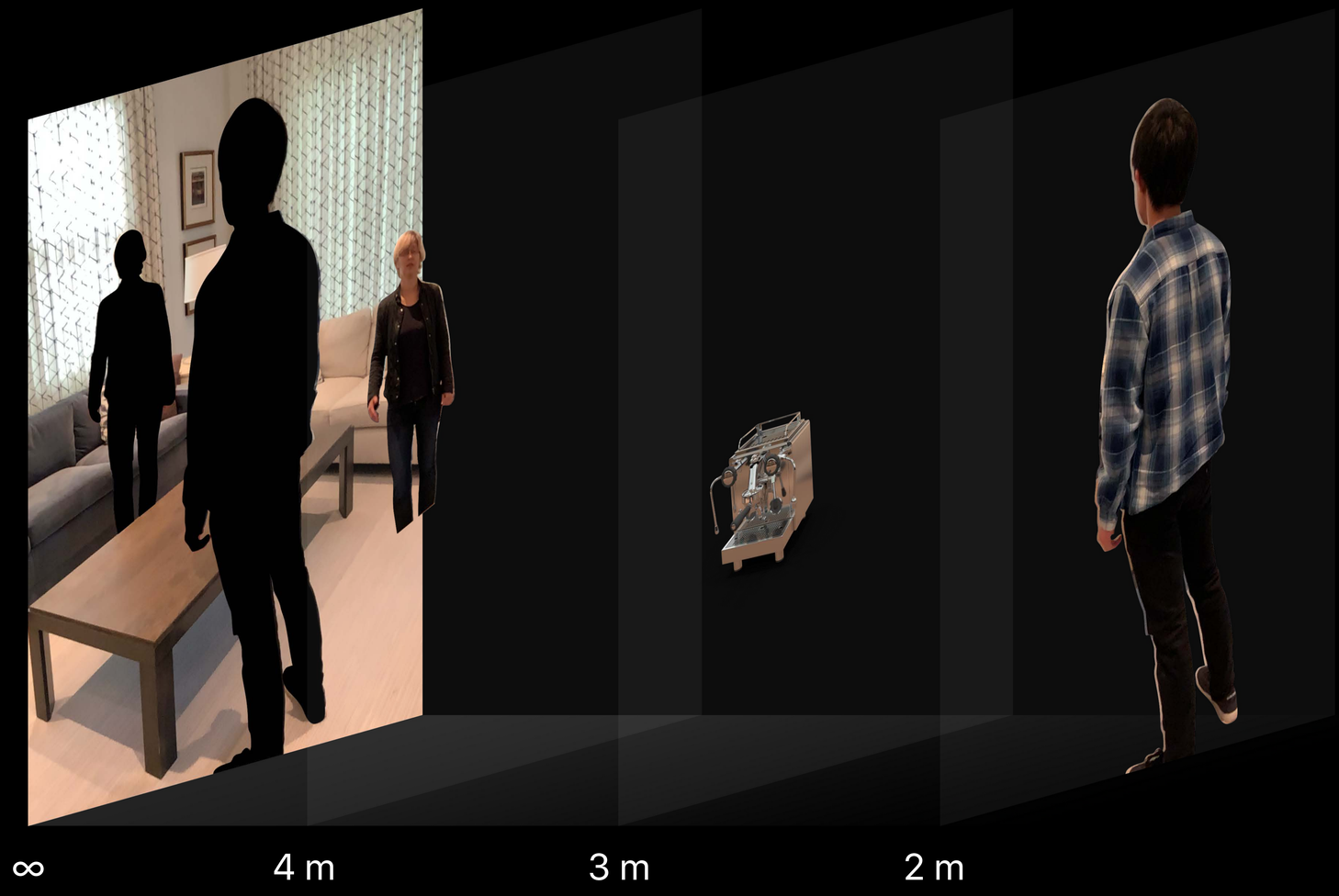
```
func compositeFrame(_ frame : ARFrame!, commandBuffer : MTLCommandBuffer!) {
  // Composition part of the rendering code
  guard ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) else
{
    return
  }

  // Schedule matting
  let matte = matte.generateMatte(from: frame, commandBuffer: commandBuffer)
  let dilatedDepth = matte.generateDilatedDepth(from: frame, commandBuffer: commandBuffer)

  // Custom composition shader

  // Done
  commandBuffer.commit()
}
```

Custom Composition

NEW

```
class ARMatteGenerator: NSObject {
  func generateDilatedDept(from: ARFrame,
                           commandBuffer: MTLCommandBuffer) -> MTLTexture
}
```

Custom Composition



Custom Composition

```
fragment half4 customComposition(…)
{
  half4 camera = cameraTexture.sample(s, in.uv);
  half4 rendered = renderedTexture.sample(s, in.uv);
  float renderedDepth = renderedDepthTexture.sample(s, in.uv);
  half4 scene = mix(rendered, camera, rendered.a);

  half matte = matteTexture.sample(s, in.uv);
  float dilatedDepth = dilatedDepthTexture.sample(s, in.uv);

  if (dilatedDepth < renderedDepth) { // People in front of rendered
    return mix(scene, camera, matte);
  } else {
    return scene;
  }
}
```

```
fragment half4 customComposition(…)
{
  half4 camera = cameraTexture.sample(s, in.uv);
  half4 rendered = renderedTexture.sample(s, in.uv);
  float renderedDepth = renderedDepthTexture.sample(s, in.uv);
  half4 scene = mix(rendered, camera, rendered.a);

  half matte = matteTexture.sample(s, in.uv);
  float dilatedDepth = dilatedDepthTexture.sample(s, in.uv);

  if (dilatedDepth < renderedDepth) { // People in front of rendered
    return mix(scene, camera, matte);
  } else {
    return scene;
  }
}
```

```
fragment half4 customComposition(…)
{
  half4 camera = cameraTexture.sample(s, in.uv);
  half4 rendered = renderedTexture.sample(s, in.uv);
  float renderedDepth = renderedDepthTexture.sample(s, in.uv);
  half4 scene = mix(rendered, camera, rendered.a);

  half matte = matteTexture.sample(s, in.uv);
  float dilatedDepth = dilatedDepthTexture.sample(s, in.uv);

  if (dilatedDepth < renderedDepth) { // People in front of rendered
    return mix(scene, camera, matte);
  } else {
    return scene;
  }
}
```

```
fragment half4 customComposition(…)
{
  half4 camera = cameraTexture.sample(s, in.uv);
  half4 rendered = renderedTexture.sample(s, in.uv);
  float renderedDepth = renderedDepthTexture.sample(s, in.uv);
  half4 scene = mix(rendered, camera, rendered.a);

  half matte = matteTexture.sample(s, in.uv);
  float dilatedDepth = dilatedDepthTexture.sample(s, in.uv);

  if (dilatedDepth < renderedDepth) { // People in front of rendered
    return mix(scene, camera, matte);
  } else {
    return scene;
  }
}
```

```
fragment half4 customComposition(…)
{
  half4 camera = cameraTexture.sample(s, in.uv);
  half4 rendered = renderedTexture.sample(s, in.uv);
  float renderedDepth = renderedDepthTexture.sample(s, in.uv);
  half4 scene = mix(rendered, camera, rendered.a);

  half matte = matteTexture.sample(s, in.uv);
  float dilatedDepth = dilatedDepthTexture.sample(s, in.uv);

  if (dilatedDepth < renderedDepth) { // People in front of rendered
    return mix(scene, camera, matte);
  } else {
    return scene;
  }
}
```

Availability

A12 and later

Availability

A12 and later

Indoor environments

Availability

A12 and later

Indoor environments

Occludes your own hands and feet

Availability

A12 and later

Indoor environments

Occludes your own hands and feet

As well as other people

Recap

Occlusion between people and rendered content

Recap

Occlusion between people and rendered content

Supported in RealityKit with ARView

Recap

Occlusion between people and rendered content

Supported in RealityKit with ARView

Backwards compatible with ARSCNView

Recap

Occlusion between people and rendered content

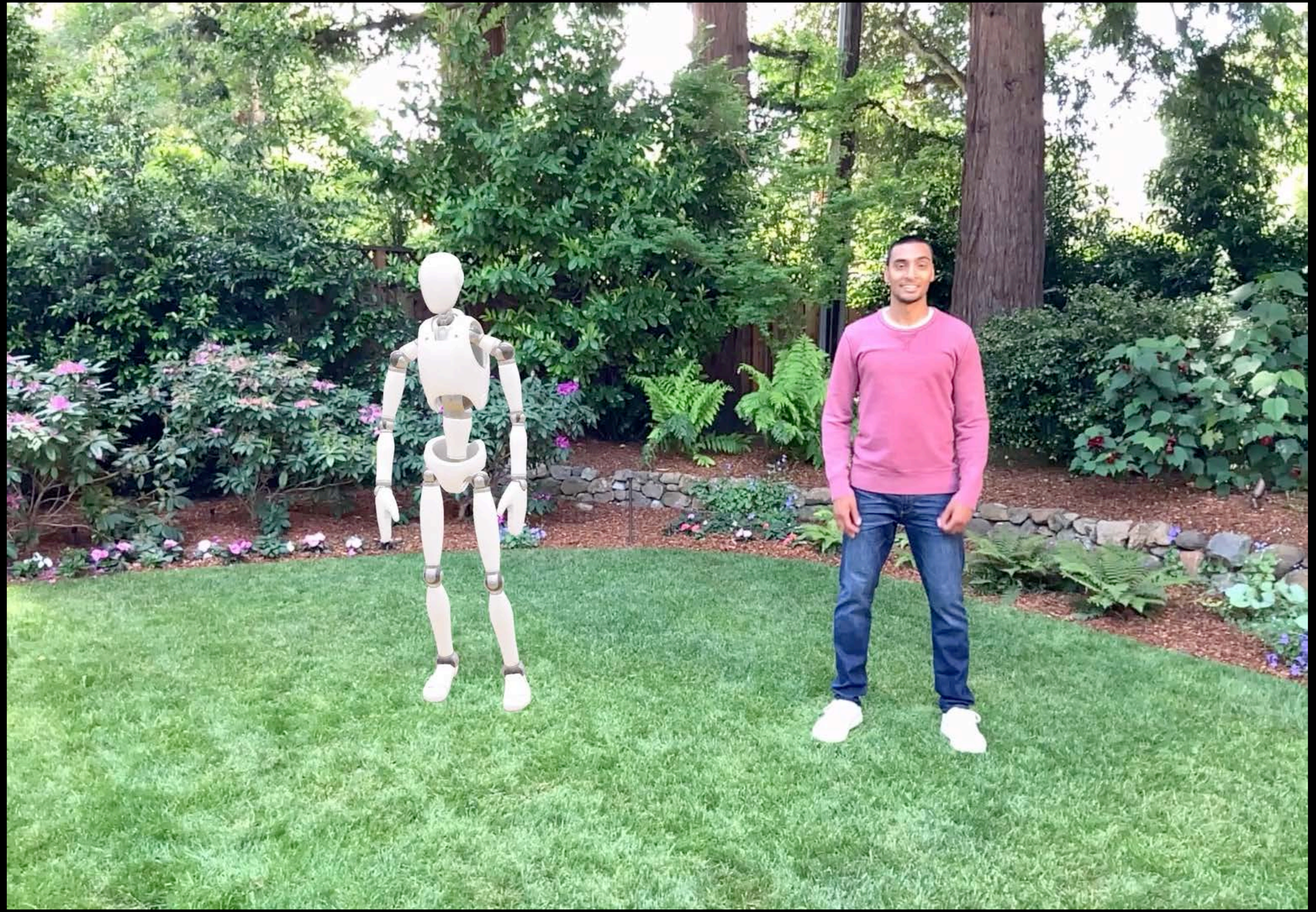
Supported in RealityKit with ARView

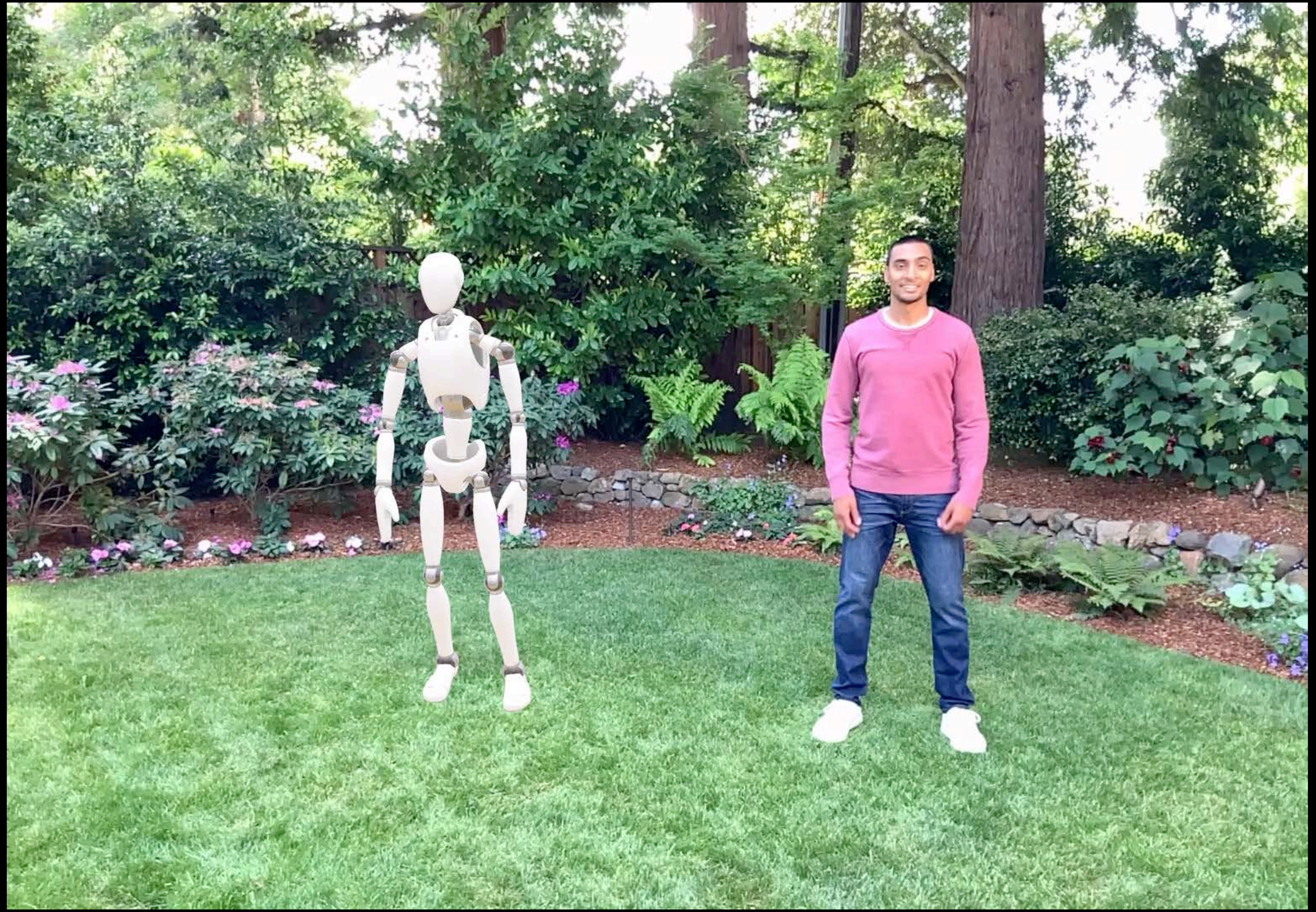
Backwards compatible with ARSCNView

Also enables custom composition through ARMatteGenerator

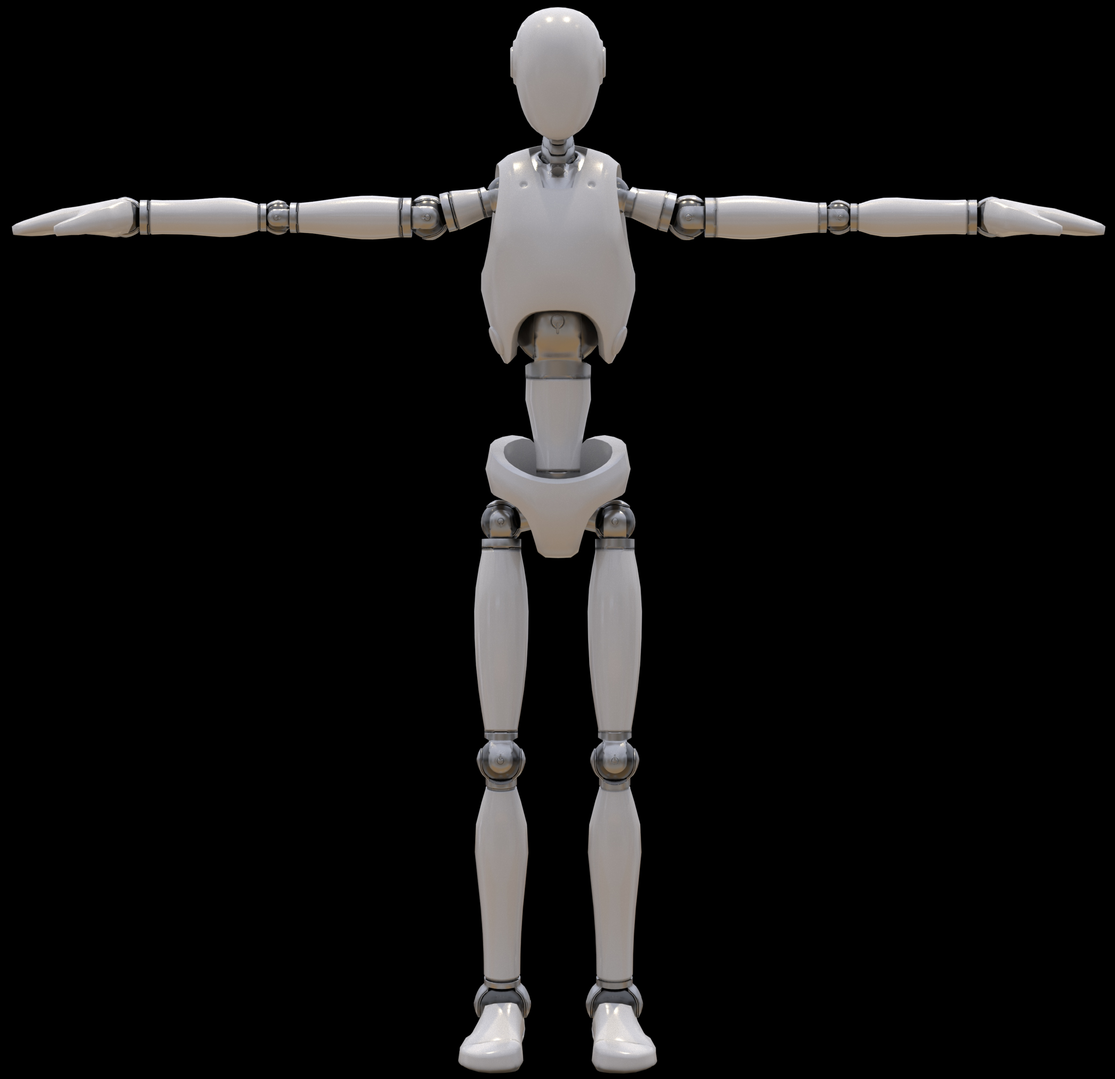
Motion Capture

Tanmay Batra

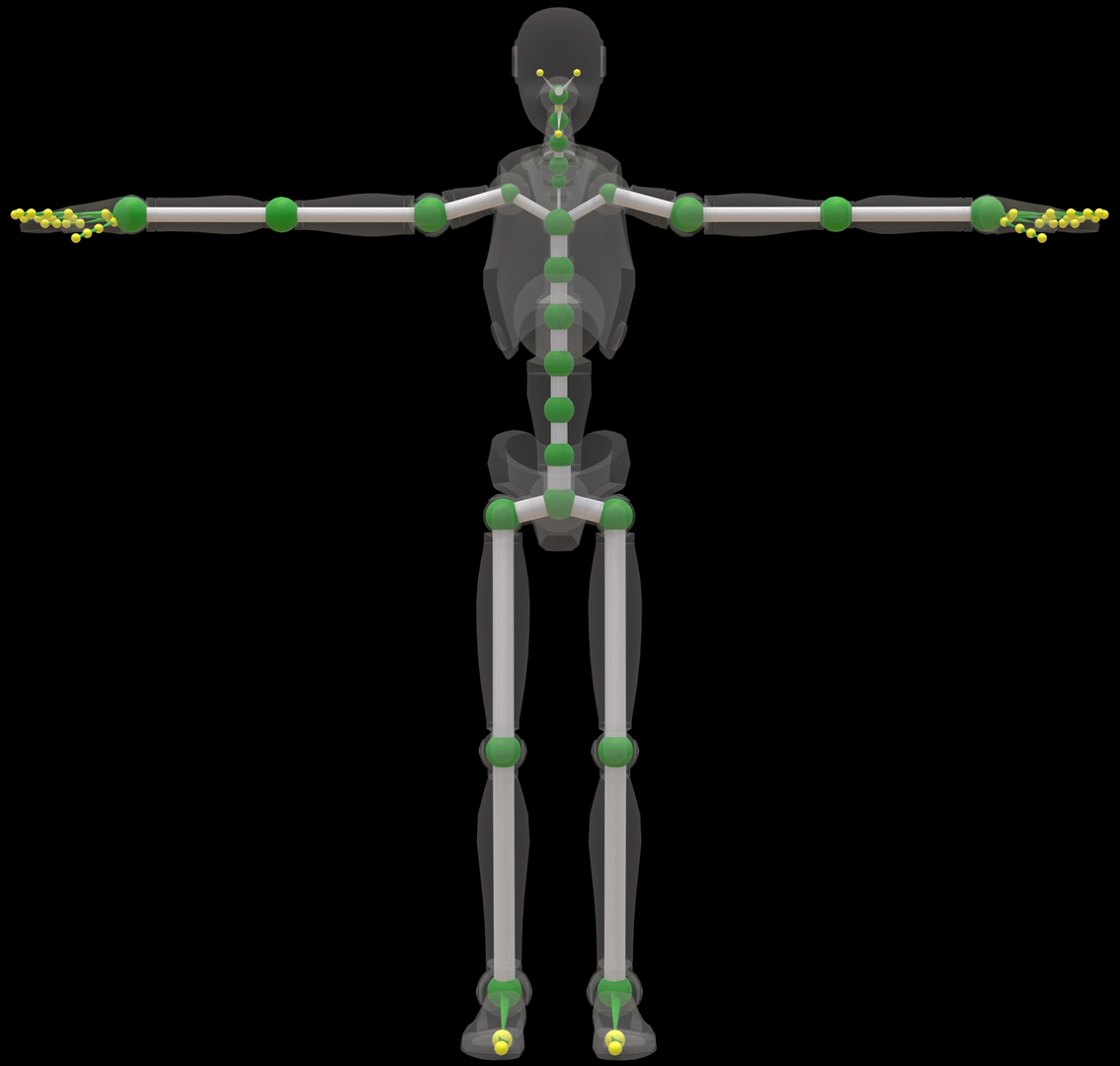




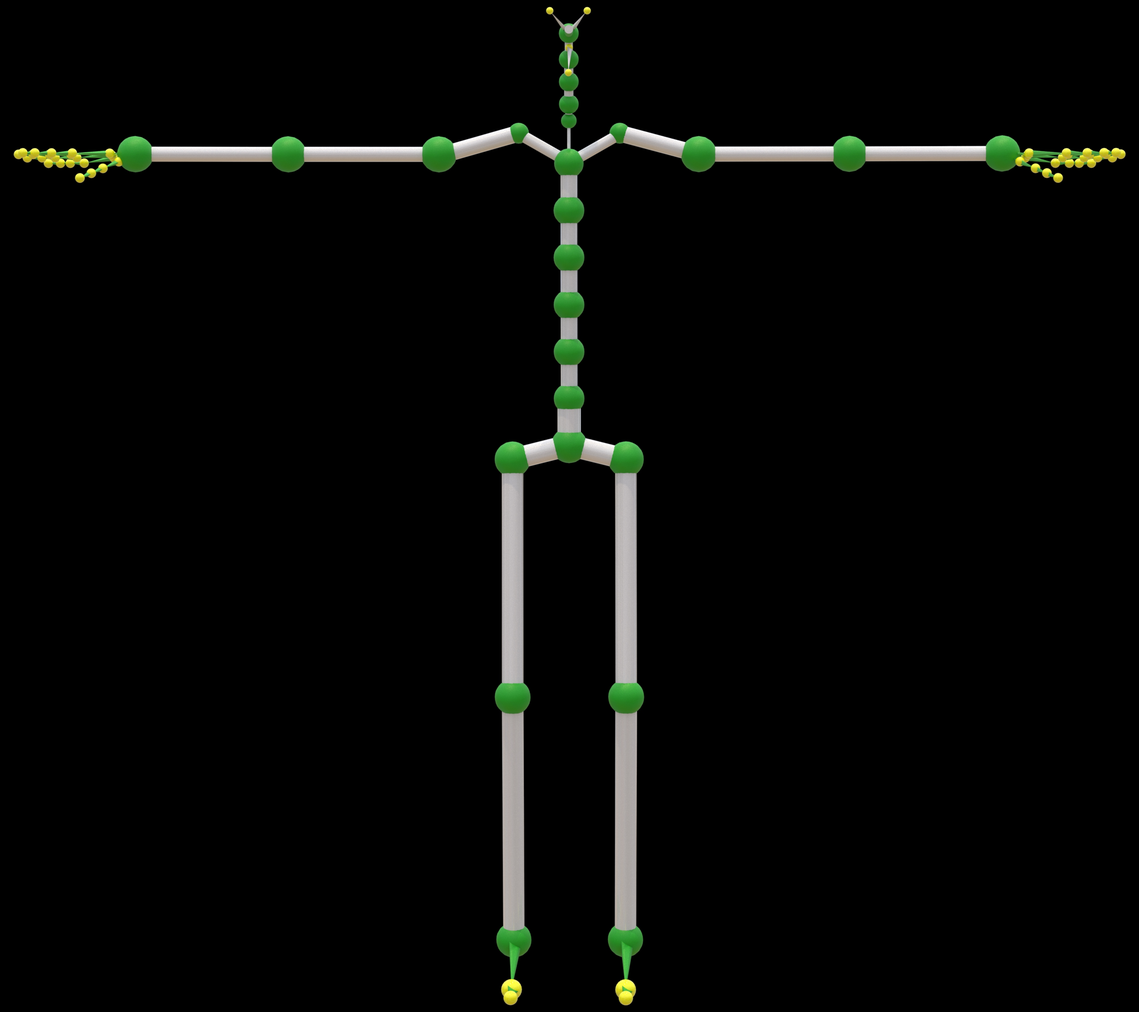
Overview



Overview



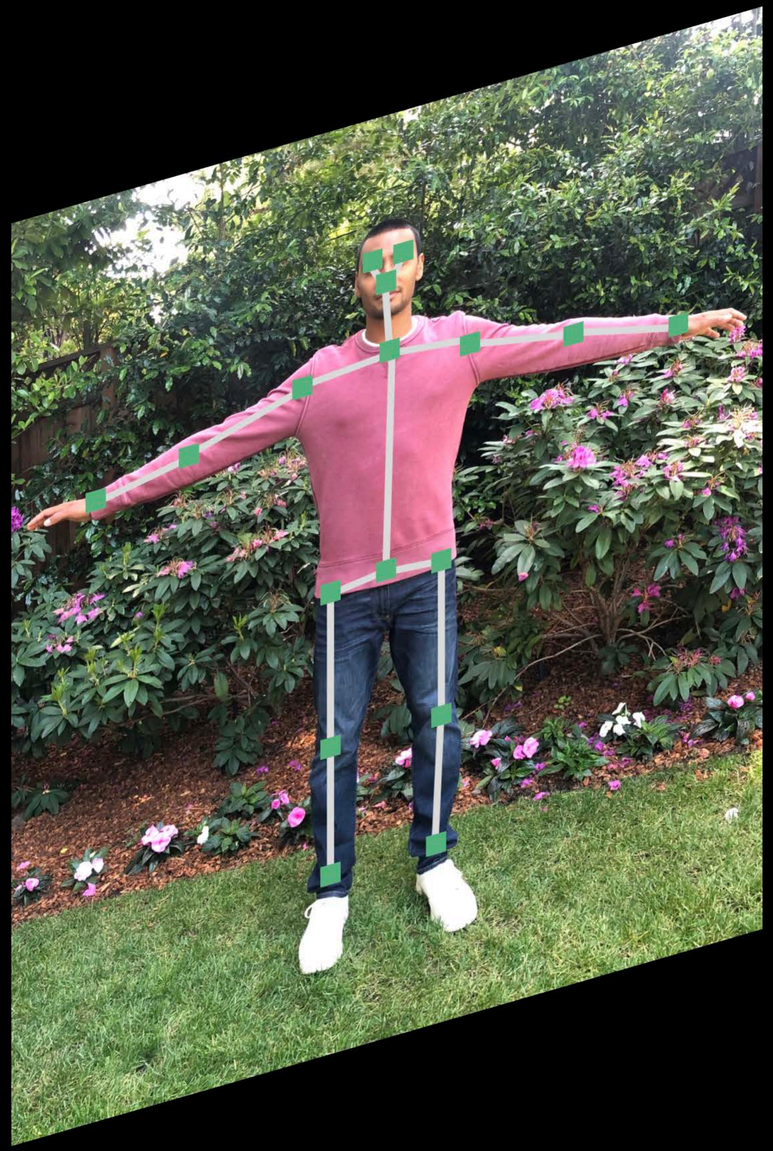
Overview



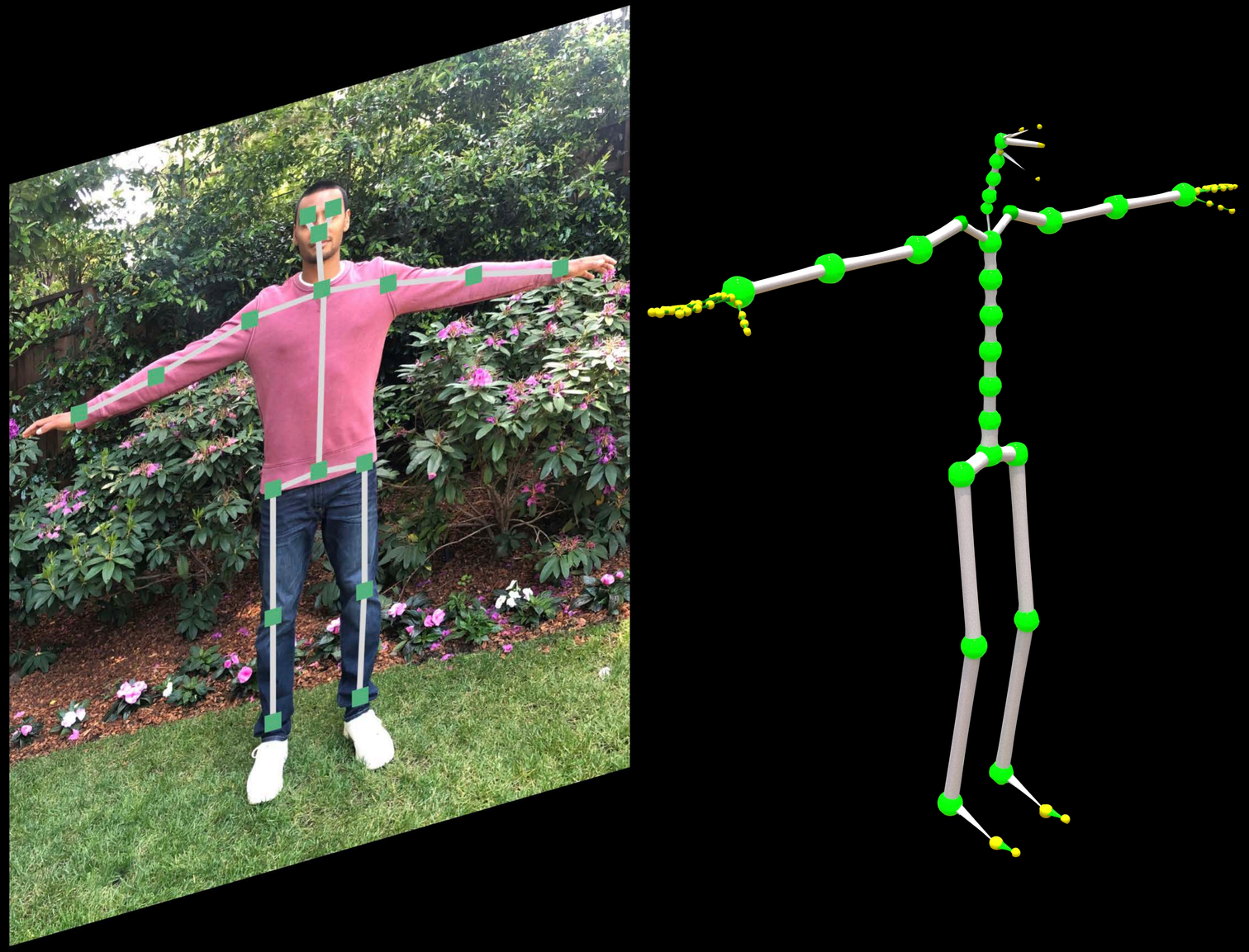
Skeleton

How Does This Work?

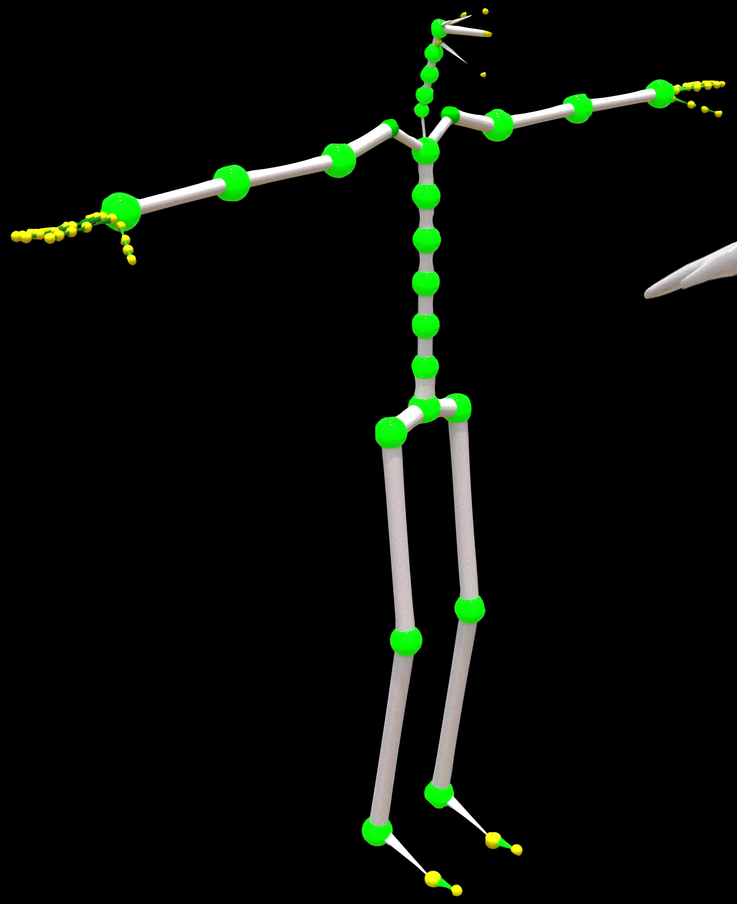
How Does This Work?

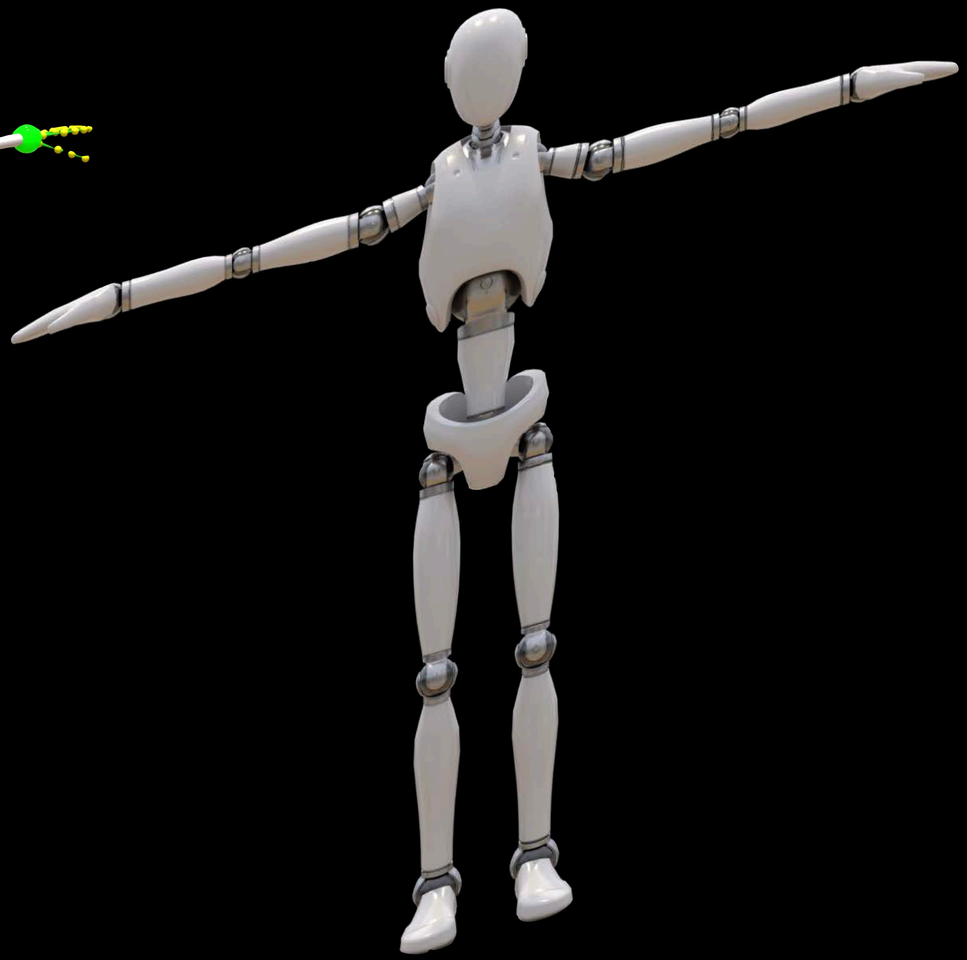


How Does This Work?



How Does This Work?





Motion Capture in AR

Tracks movements of people in real-time

Integrated with ARKit 3 and RealityKit

Powered by machine learning

Available on A12 and later

Motion Capture in AR

Use cases

Animate characters

Motion Capture in AR

Use cases

Animate characters

Action/Activity recognition

Motion Capture in AR

Use cases

Animate characters

Action/Activity recognition

Sports and fitness analysis

Motion Capture in AR

Use cases

Animate characters

Action/Activity recognition

Sports and fitness analysis

Interacting with virtual objects

Motion Capture in AR

Use cases

Animate characters

Action/Activity recognition

Sports and fitness analysis

Interacting with virtual objects

Semantic image analysis

How to Use It?

Motion capture in RealityKit

How to Use It?

Motion capture in RealityKit

Extracting data from skeleton in 3D

How to Use It?

Motion capture in RealityKit

Extracting data from skeleton in 3D

Extracting data from skeleton in 2D

Motion Capture in RealityKit

Motion Capture in RealityKit

NEW

Quickly animate characters

Motion Capture in RealityKit

NEW

Quickly animate characters

Simple and easy to use API

Motion Capture in RealityKit

NEW

Quickly animate characters

Simple and easy to use API

Add your custom rigged characters

Motion Capture in RealityKit

NEW

Quickly animate characters

Simple and easy to use API

Add your custom rigged characters

Easy to access tracked person

- Provided via AnchorEntity
- Automatically gathers motion transforms

Motion Capture in RealityKit

NEW

ARView

Motion Capture in RealityKit

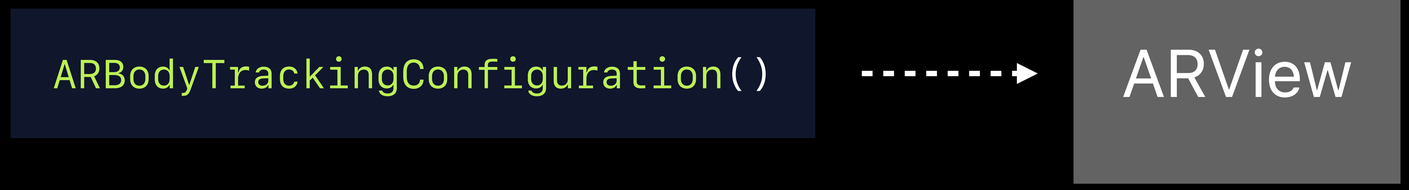
NEW

```
ARBodyTrackingConfiguration()
```

ARView

Motion Capture in RealityKit

NEW



Motion Capture in RealityKit

NEW

```
ARBodyTrackingConfiguration()
```

BodyTrackedEntity

Represents a person

Contains skeleton and position

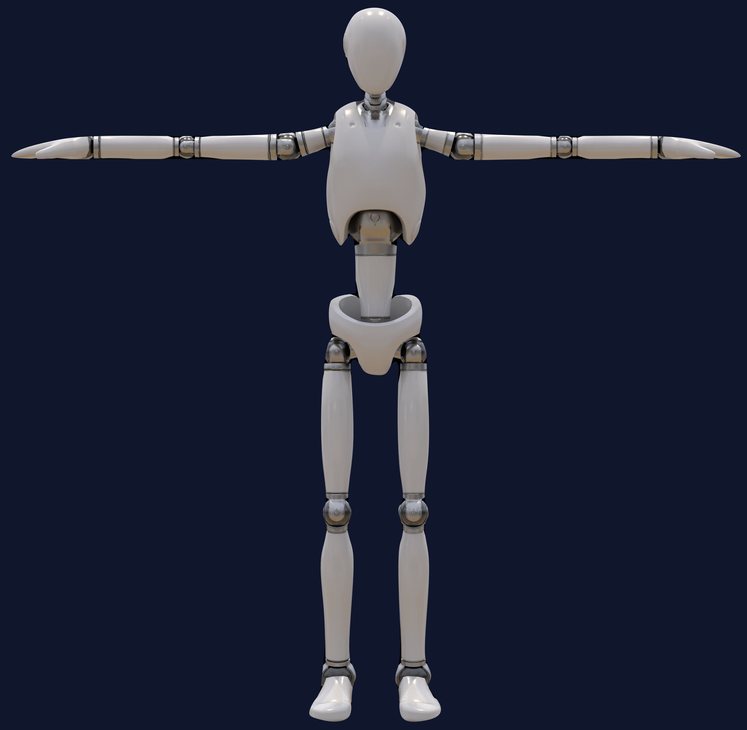
Updated every frame

Applies the skeleton to a .usdz model

```
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```

```
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```

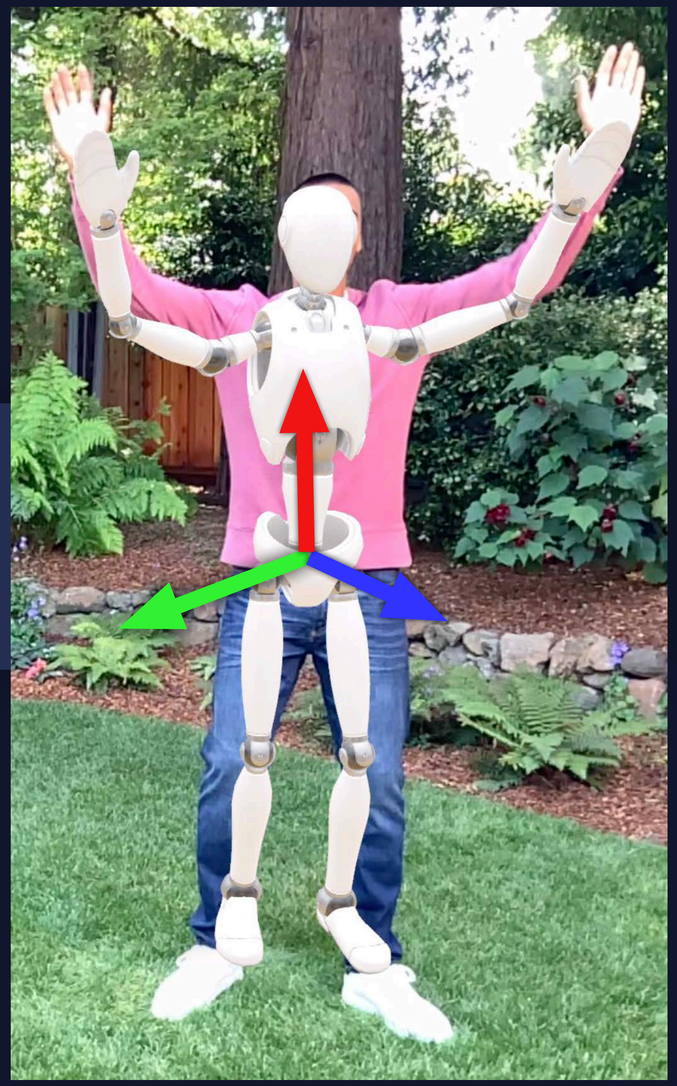
```swift
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```



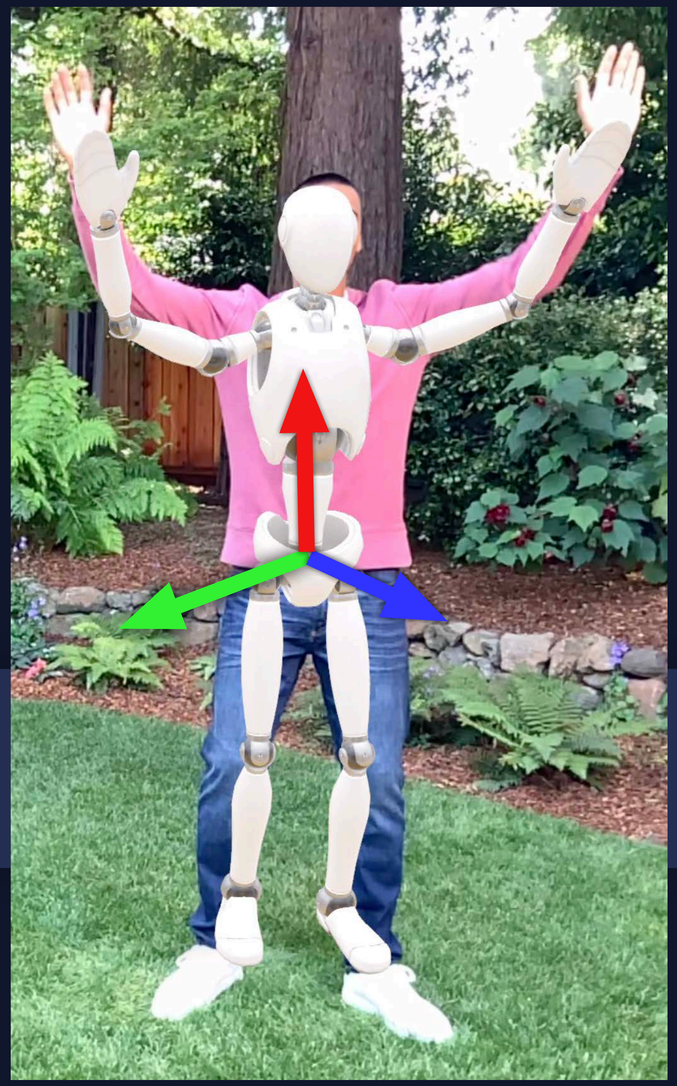
robot.usdz

```
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```

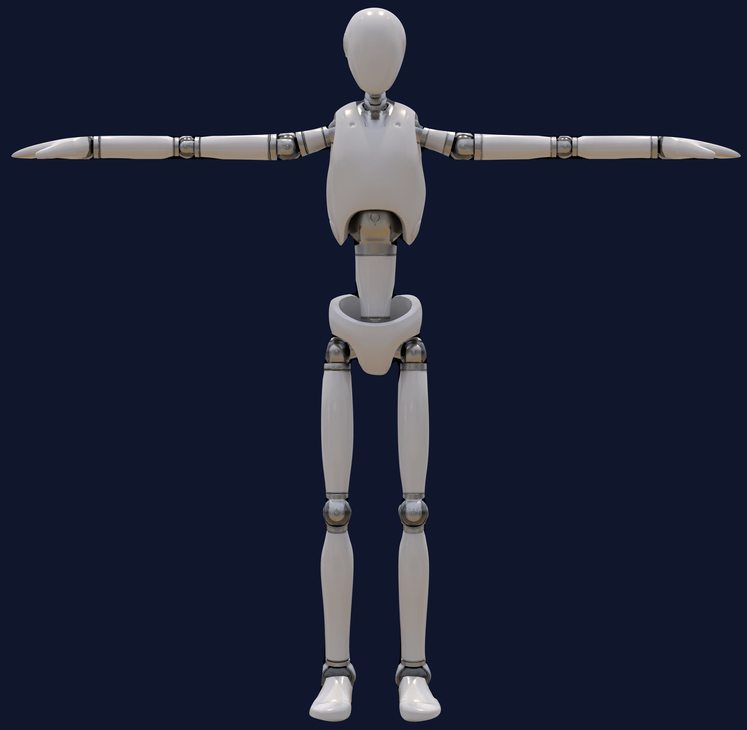
```
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```



```swift
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```



```swift
// Load Rigged Mesh and Tracked Person
Entity.loadBodyTrackedAsync(named: "robot")
    .sink(receiveCompletion : { // For catching failure/error },
          receiveValue: { (character) in
          guard let character = character as? BodyTrackedEntity
          else { return }

// Get the Location Where You Want to Put Your Character
let personAnchor = AnchorEntity(.body)
arView.scene.addAnchor(personAnchor)

// Add the Character to that Location
personAnchor.addChild(character)
```



robot.usdz

3D Skeleton Joints

3D Skeleton Joints

root
hips_joint
left_upLeg_joint
left_leg_joint
left_foot_joint
left_toes_joint
left_toesEnd_joint
right_upLeg_joint
right_leg_joint
right_foot_joint
right_toes_joint
right_toesEnd_joint
spine_1_joint
spine_2_joint
spine_3_joint
spine_4_joint
spine_5_joint
spine_6_joint
spine_7_joint

right_shoulder_1_joint
right_arm_joint
right_forearm_joint
right_hand_joint
right_handThumbStart_joint
right_handThumb_1_joint
right_handThumb_2_joint
right_handThumbEnd_joint
right_handIndexStart_joint
right_handIndex_1_joint
right_handIndex_2_joint
right_handIndex_3_joint
right_handIndexEnd_joint
right_handMidStart_joint
right_handMid_1_joint
right_handMid_2_joint
right_handMid_3_joint
right_handMidEnd_joint

right_handRingStart_joint
right_handRing_1_joint
right_handRing_2_joint
right_handRing_3_joint
right_handRingEnd_joint
right_handPinkyStart_joint
right_handPinky_1_joint
right_handPinky_2_joint
right_handPinky_3_joint
right_handPinkyEnd_joint
left_shoulder_1_joint
left_arm_joint
left_forearm_joint
left_hand_joint
left_handThumbStart_joint
left_handThumb_1_joint
left_handThumb_2_joint
left_handThumbEnd_joint

left_handIndexStart_joint
left_handIndex_1_joint
left_handIndex_2_joint
left_handIndex_3_joint
left_handIndexEnd_joint
left_handMidStart_joint
left_handMid_1_joint
left_handMid_2_joint
left_handMid_3_joint
left_handMidEnd_joint
left_handRingStart_joint
left_handRing_1_joint
left_handRing_2_joint
left_handRing_3_joint
left_handRingEnd_joint
left_handPinkyStart_joint
left_handPinky_1_joint
left_handPinky_2_joint
left_handPinky_3_joint

left_handPinkyEnd_joint
head_joint
jaw_joint
chin_joint
nose_joint
right_eye_joint
right_eyeUpperLid_joint
right_eyeLowerLid_joint
right_eyeball_joint
left_eye_joint
left_eyeUpperLid_joint
left_eyeLowerLid_joint
left_eyeball_joint
neck_1_joint
neck_2_joint
neck_3_joint
neck_4_joint

Extracting Data from Skeleton in 3D

Extracting Data from Skeleton

NEW

Detailed access to skeleton elements

Extracting Data from Skeleton

NEW

Detailed access to skeleton elements

Easy to use API

Extracting Data from Skeleton

NEW

Detailed access to skeleton elements

Easy to use API

Enable advanced use cases

Extracting Data from Skeleton

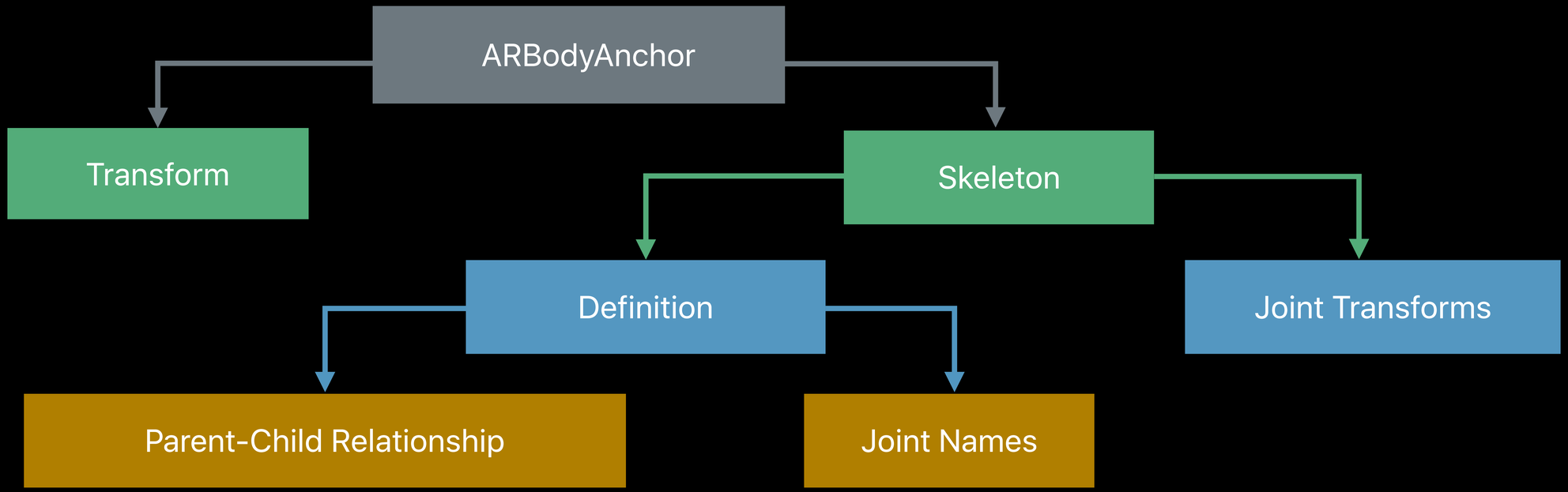
NEW

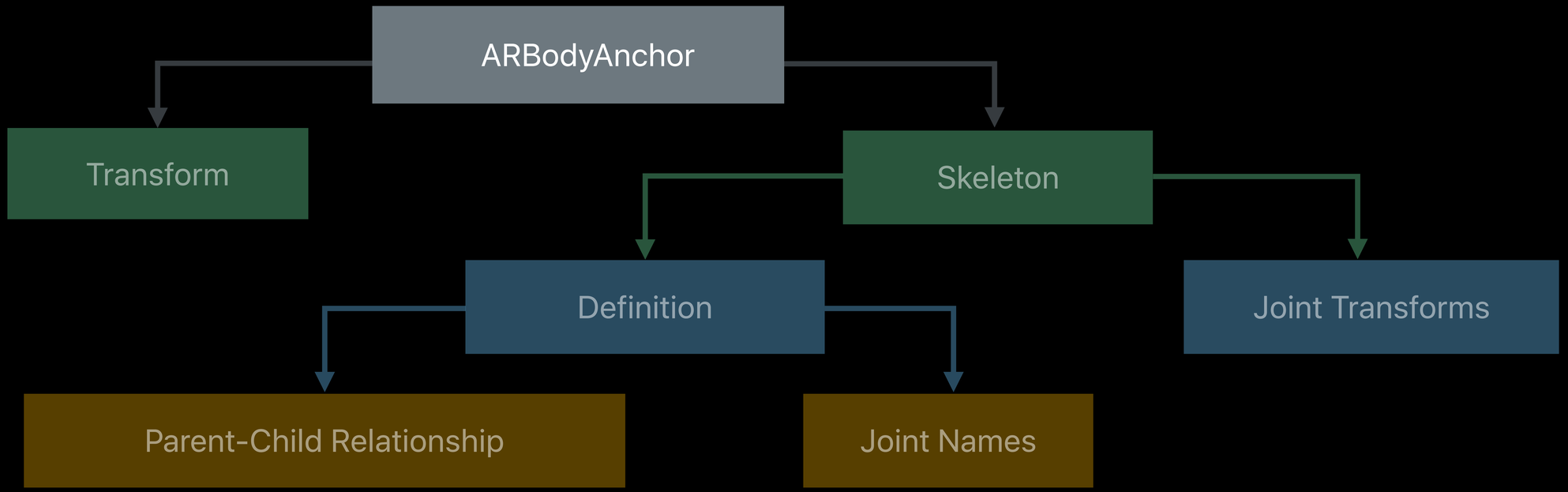
Detailed access to skeleton elements

Easy to use API

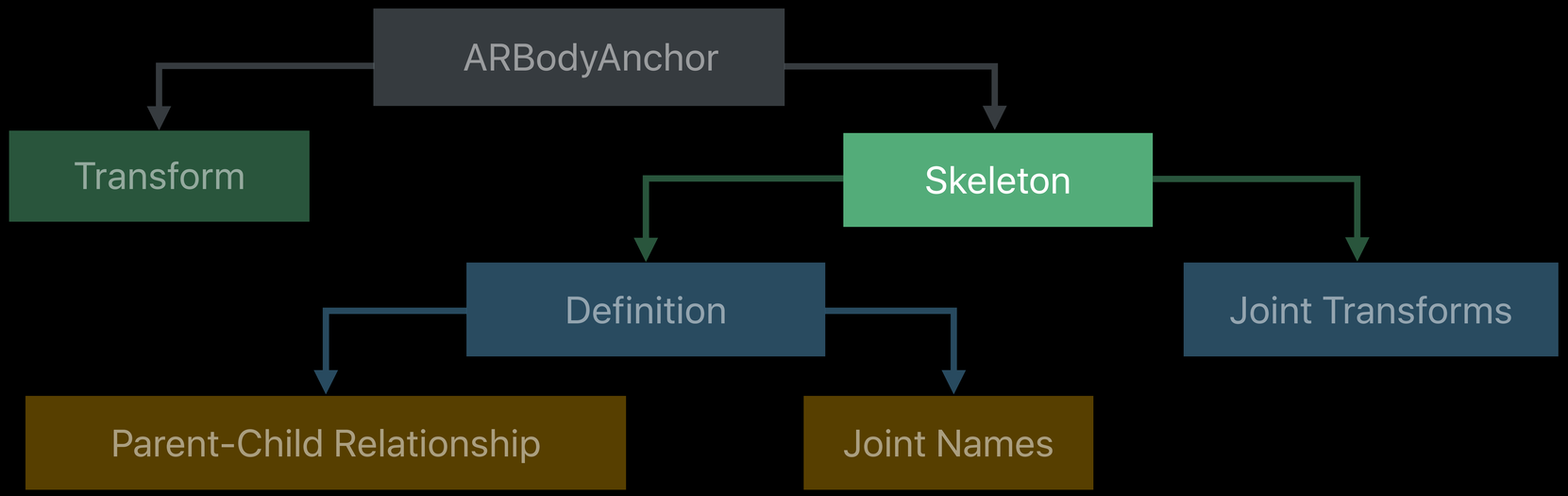
Enable advanced use cases

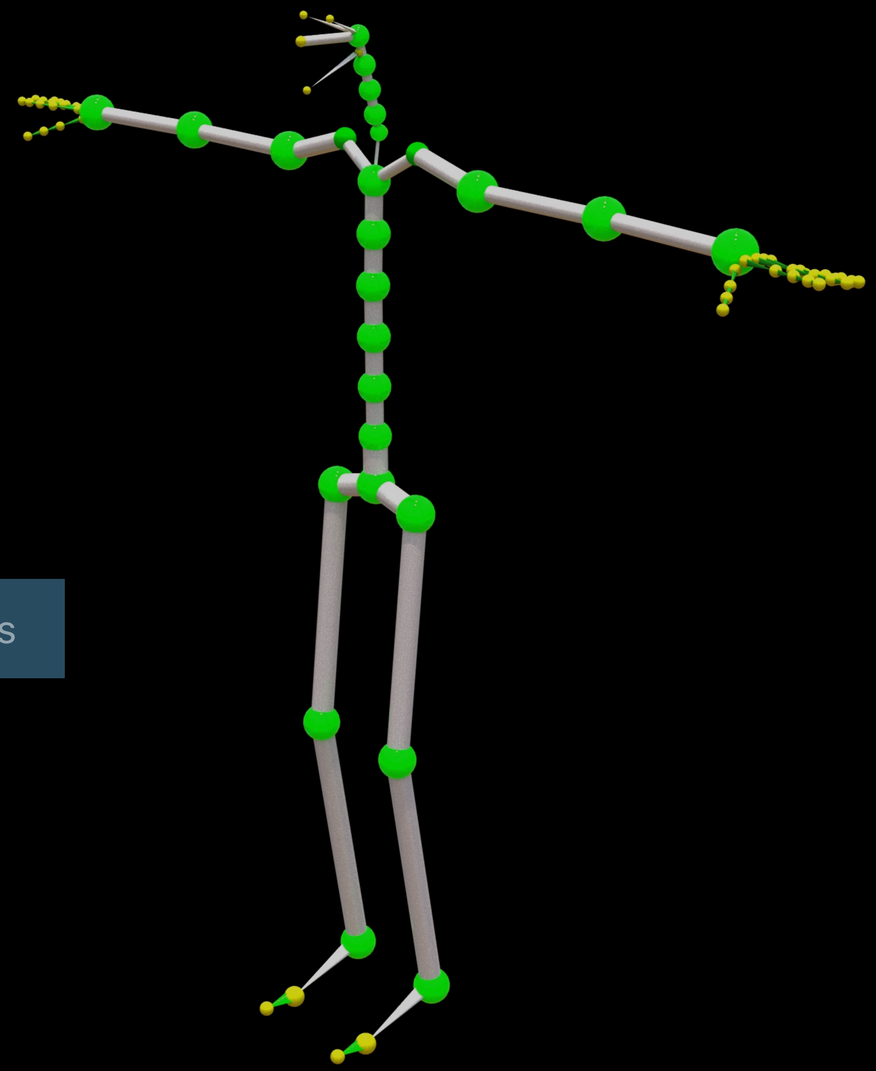
Provided via ARBodyAnchor


ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

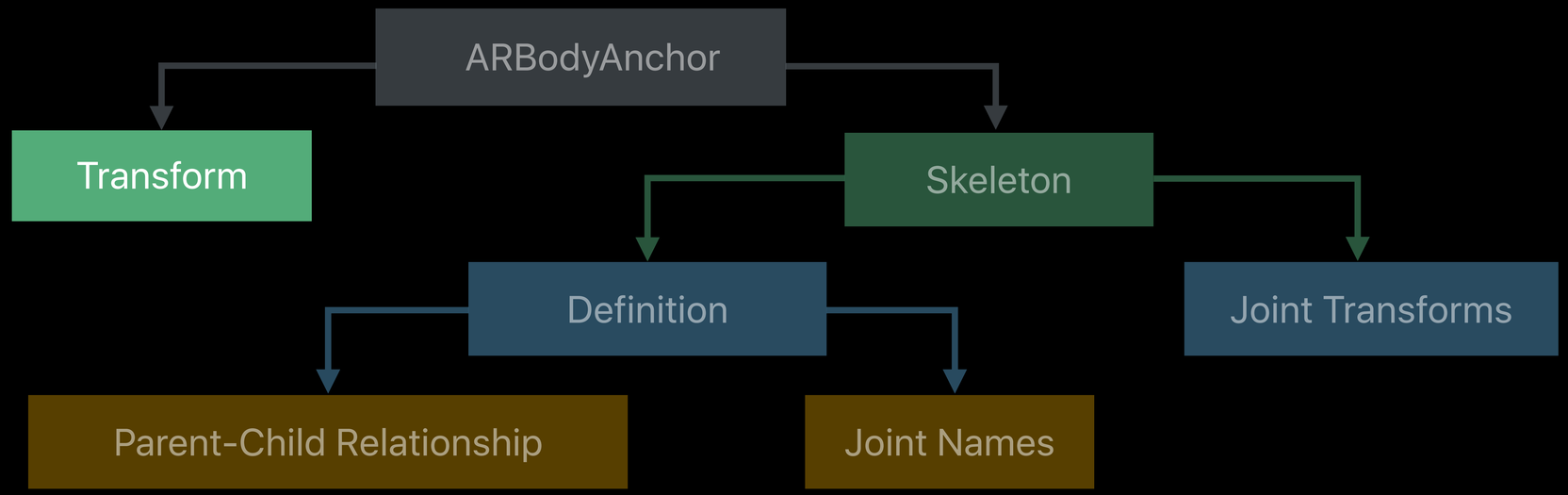


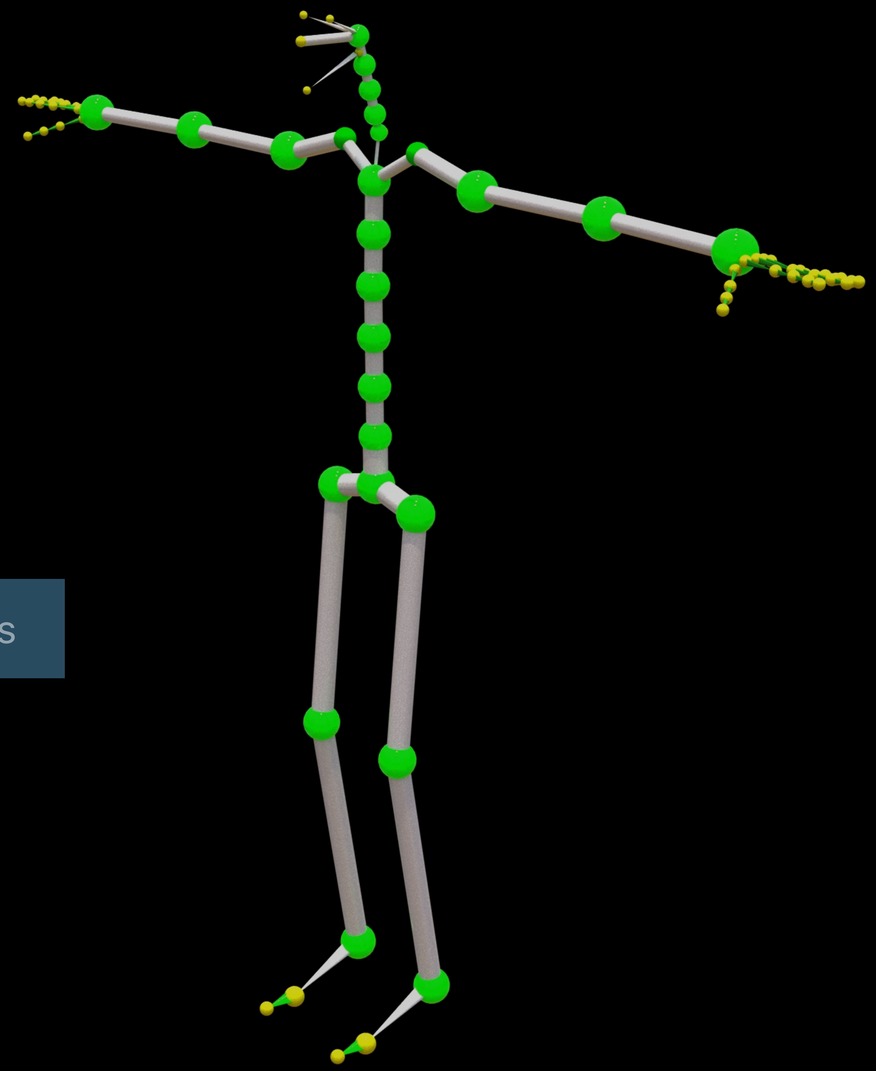

```
class ARBodyAnchor : ARAnchor
```



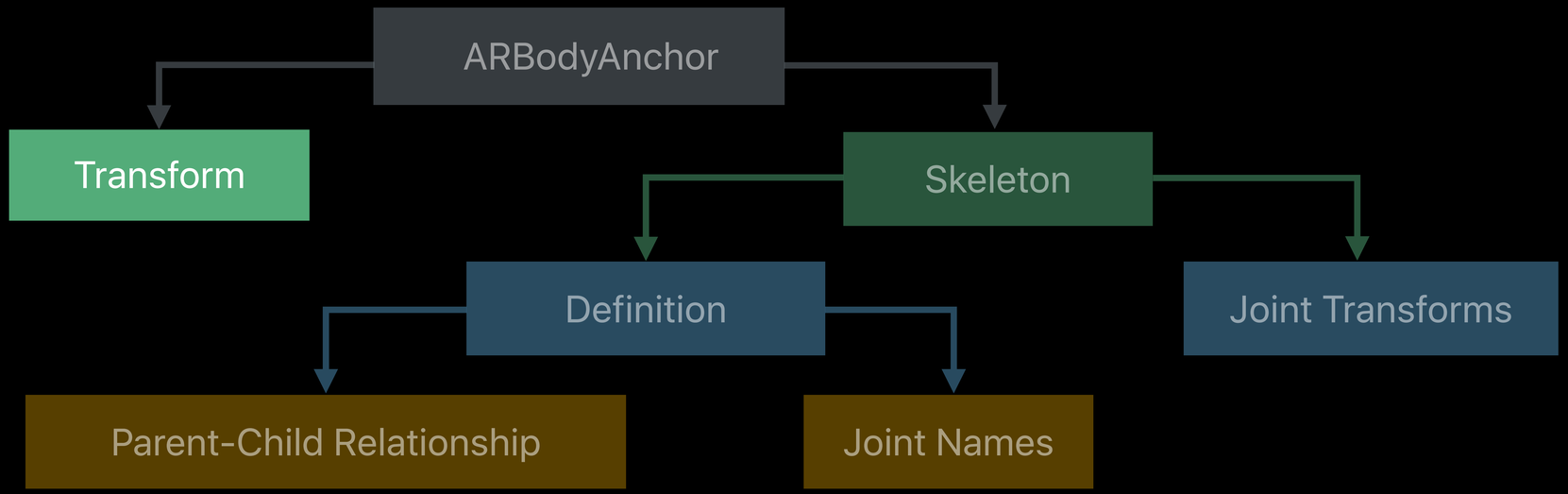


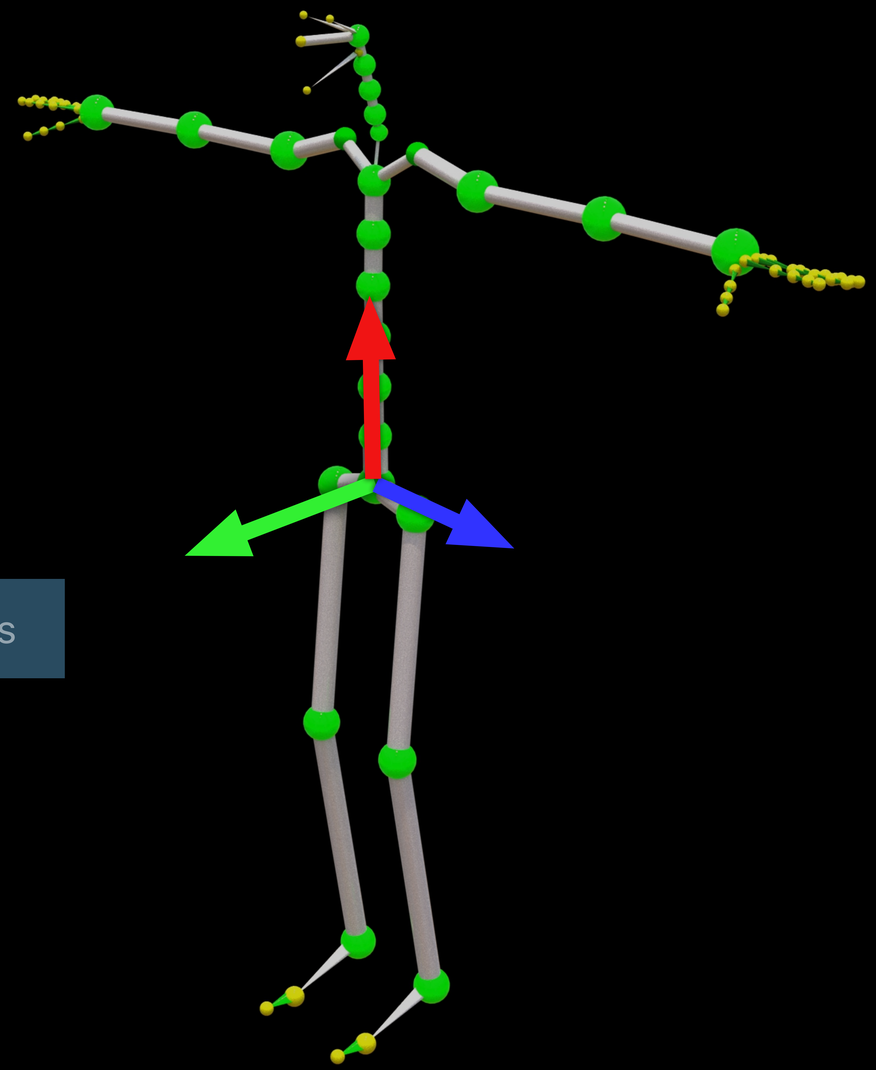
```
class ARSkeleton
```



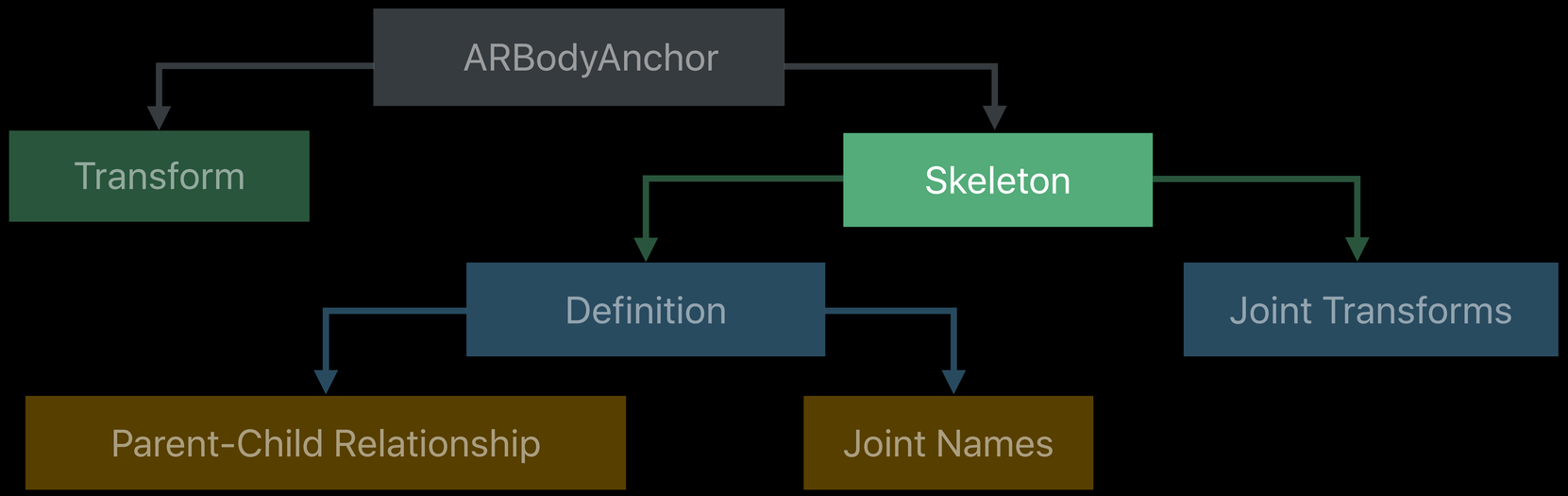


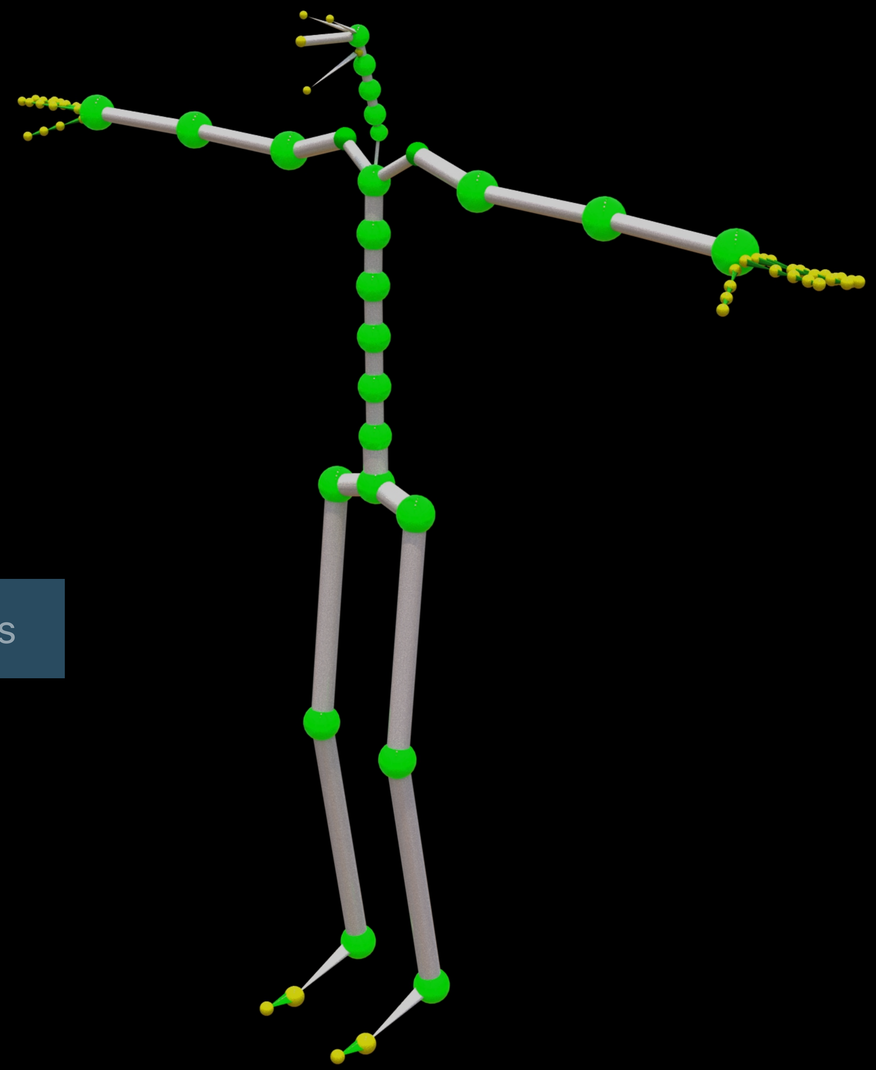
```
var transform: matrix_float4x4
```

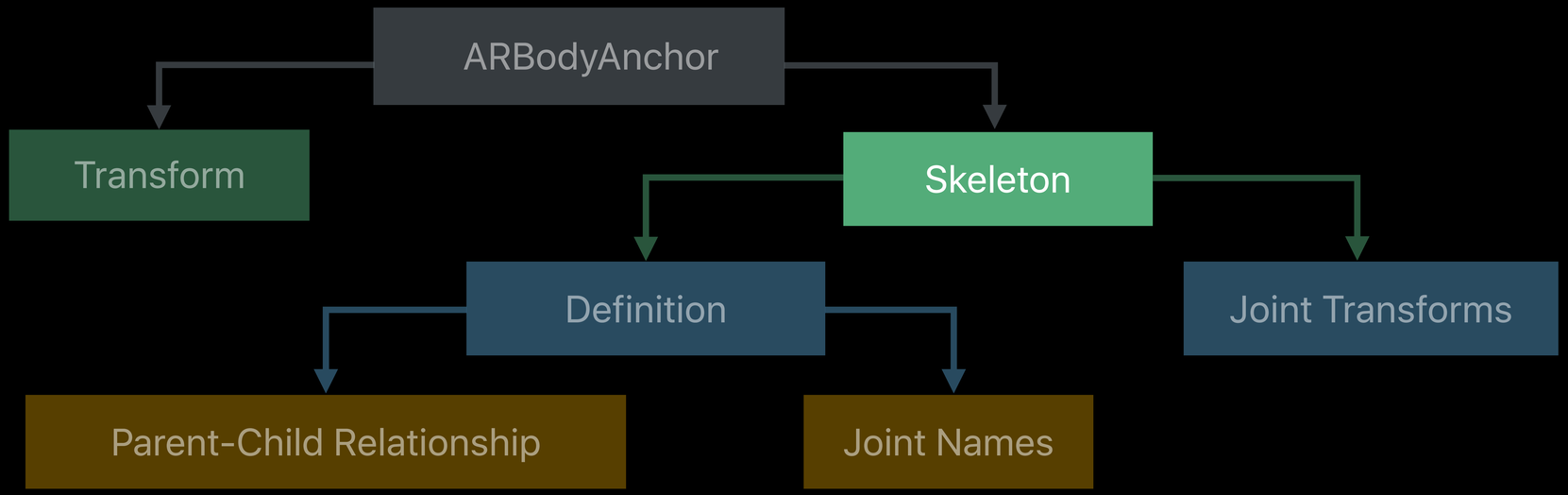


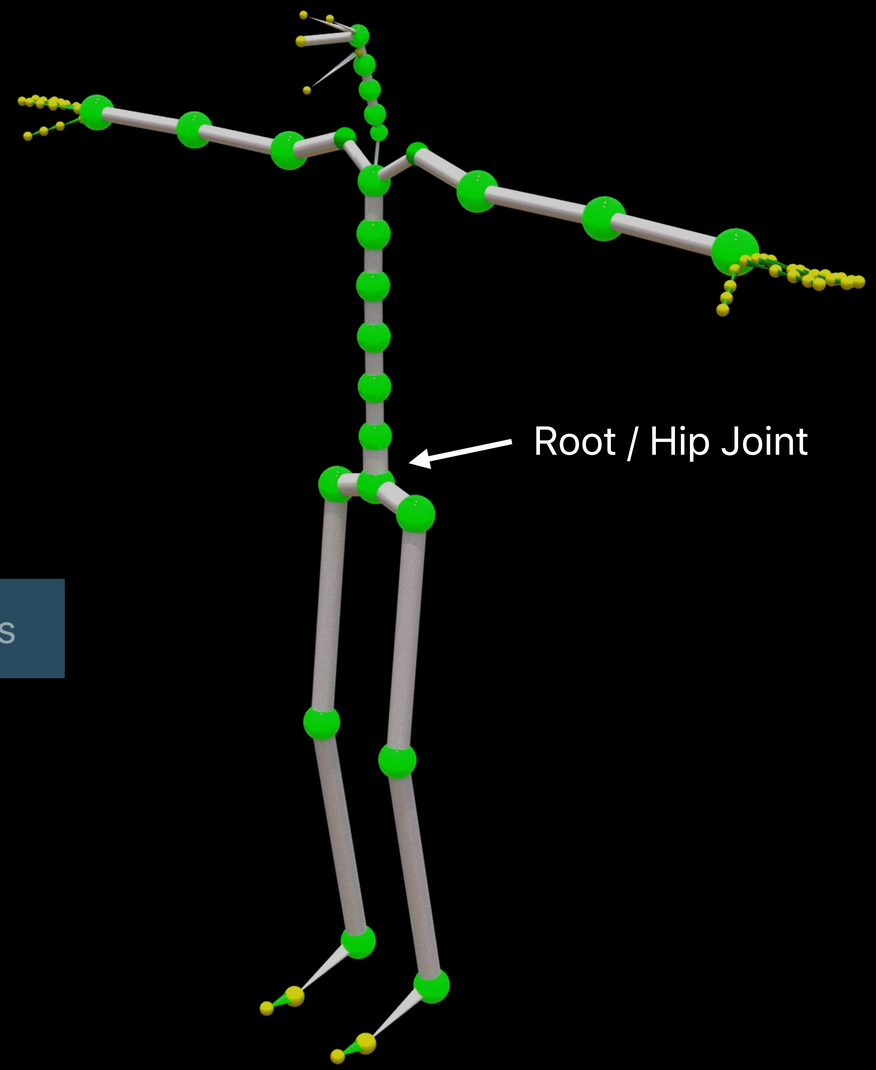


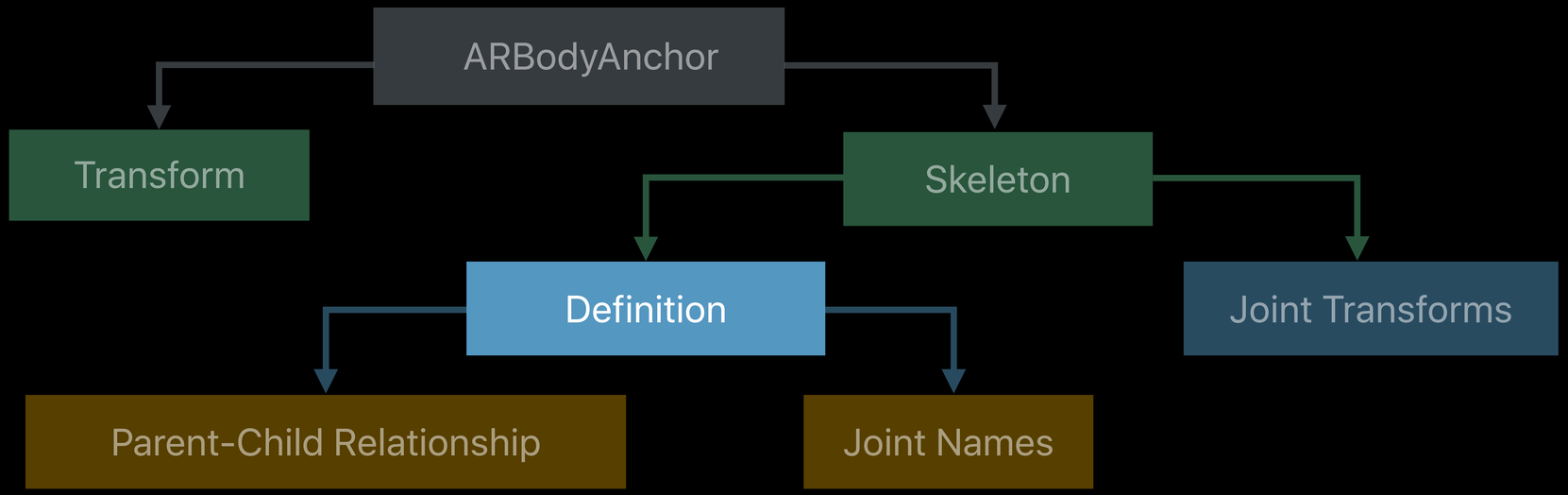
```
var transform: matrix_float4x4
```

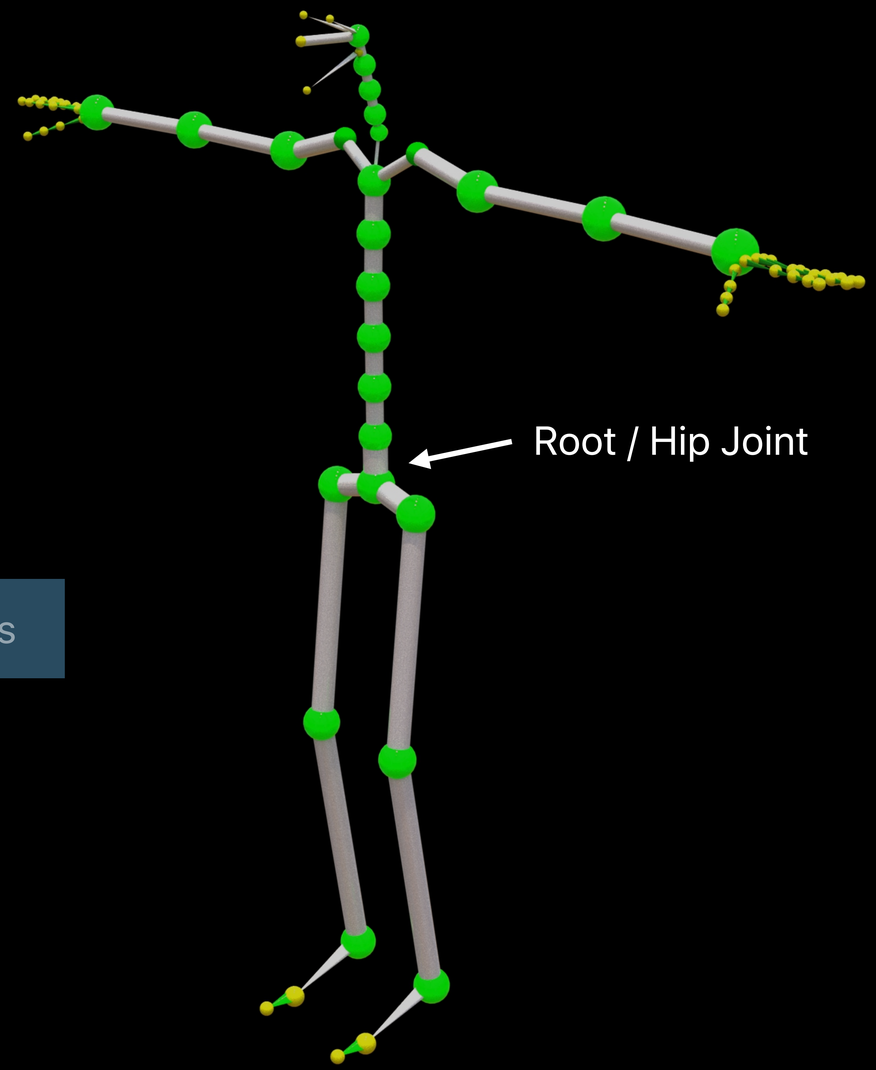

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

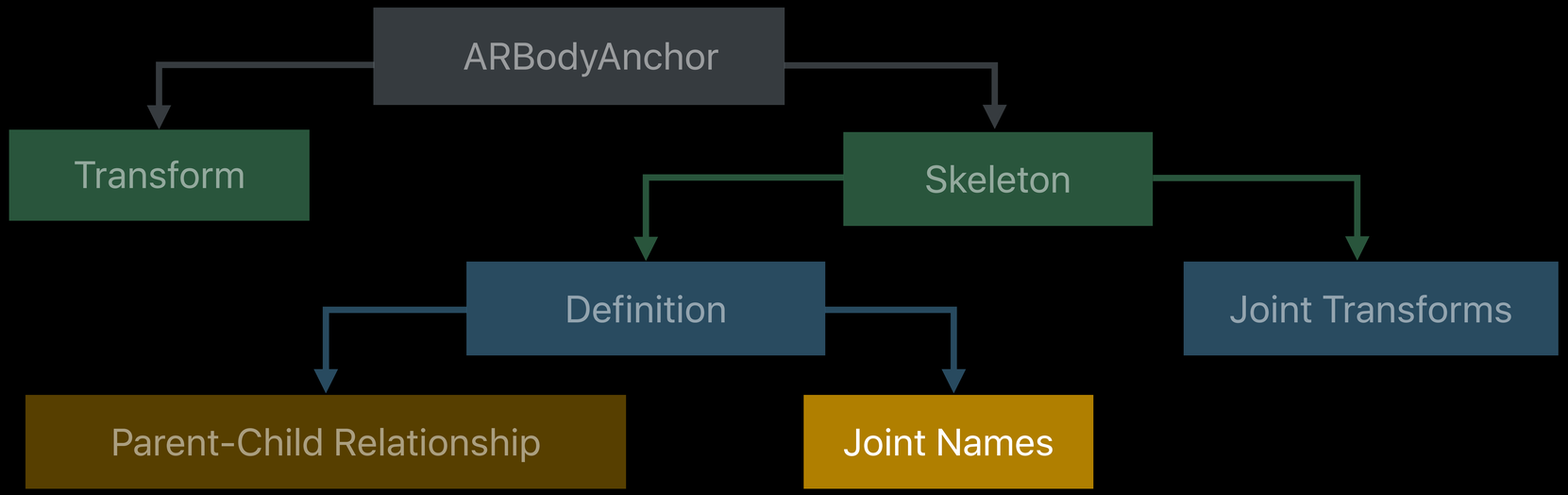


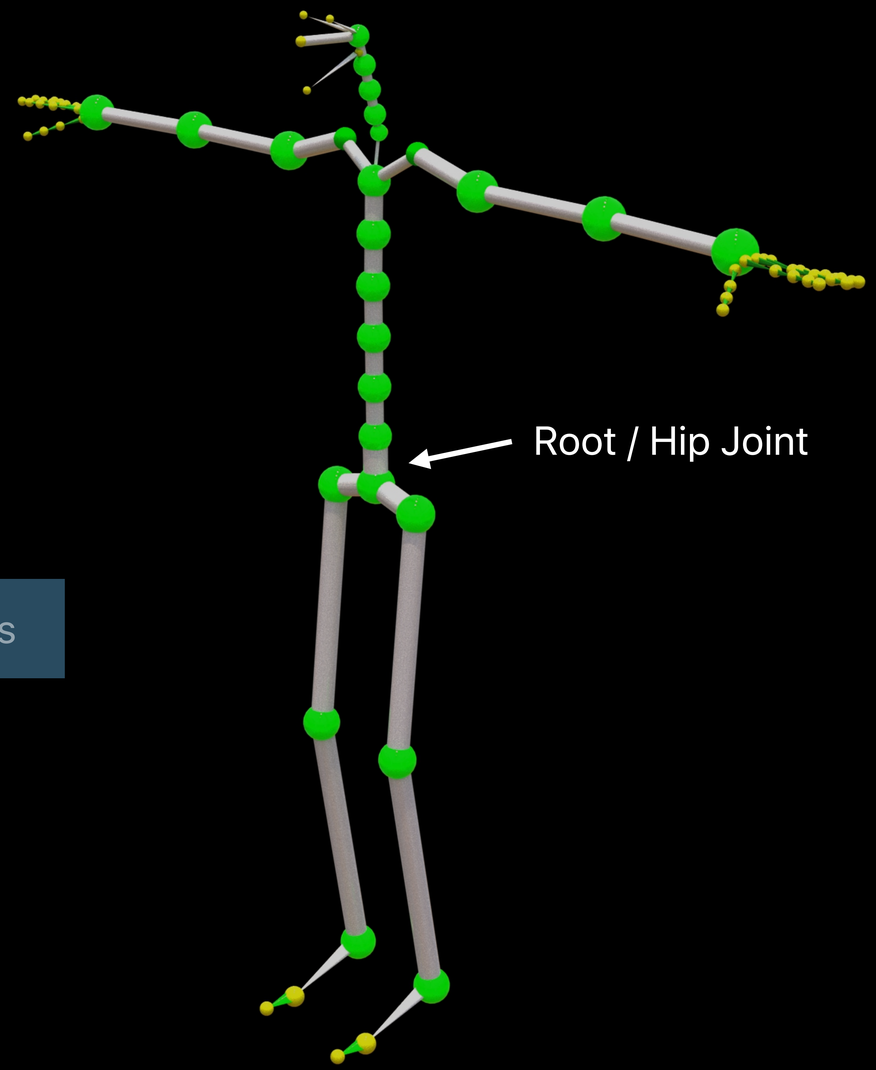

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

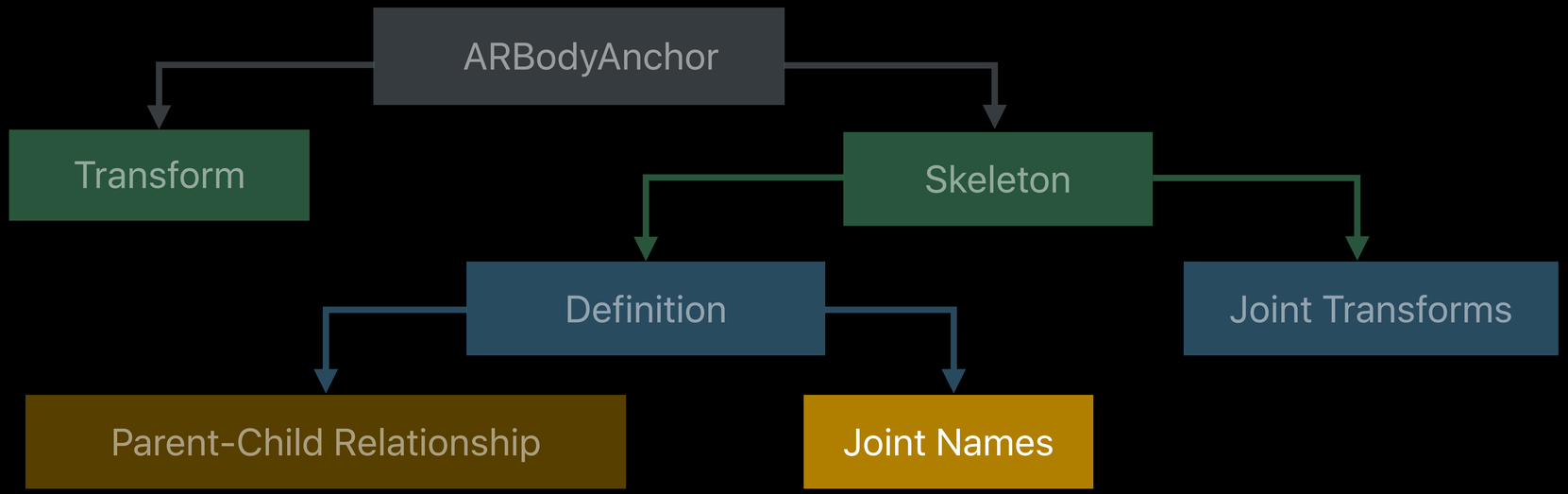

Root / Hip Joint

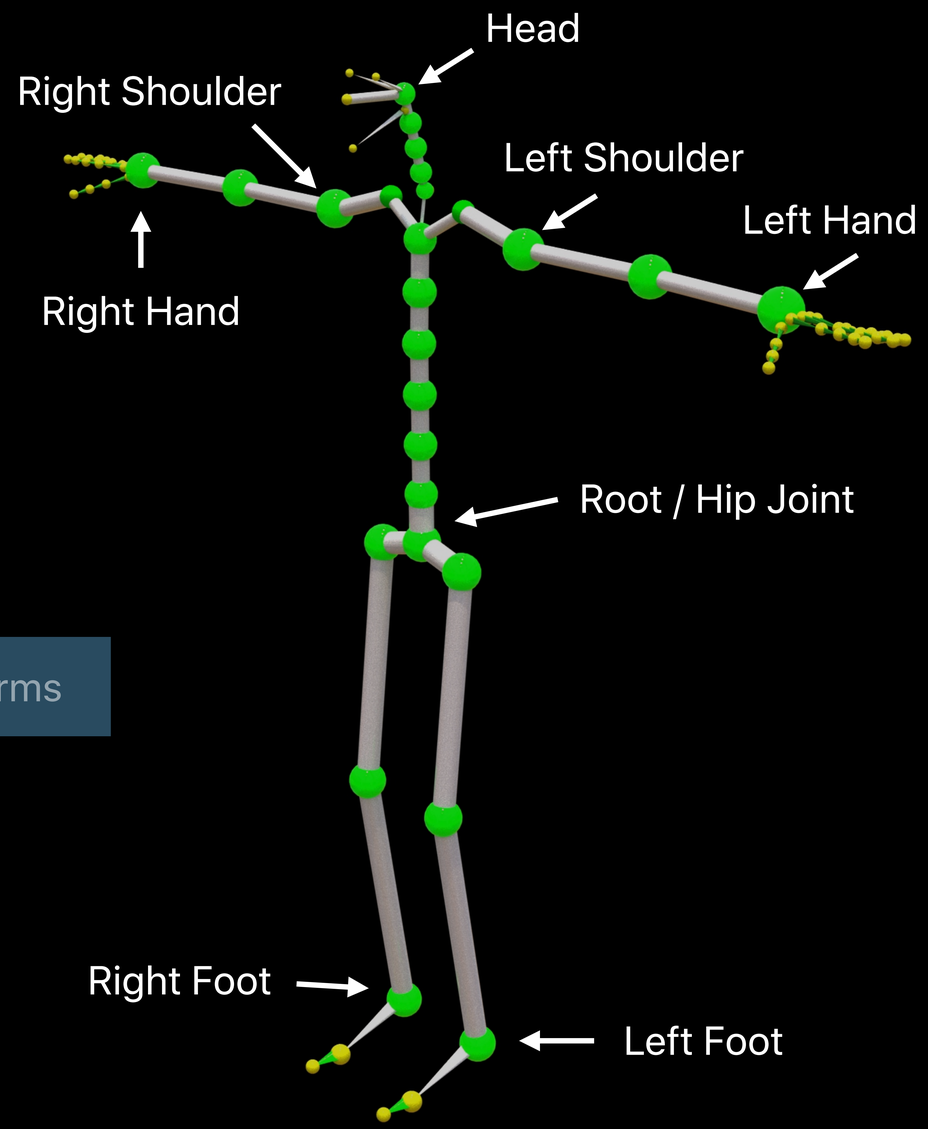

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

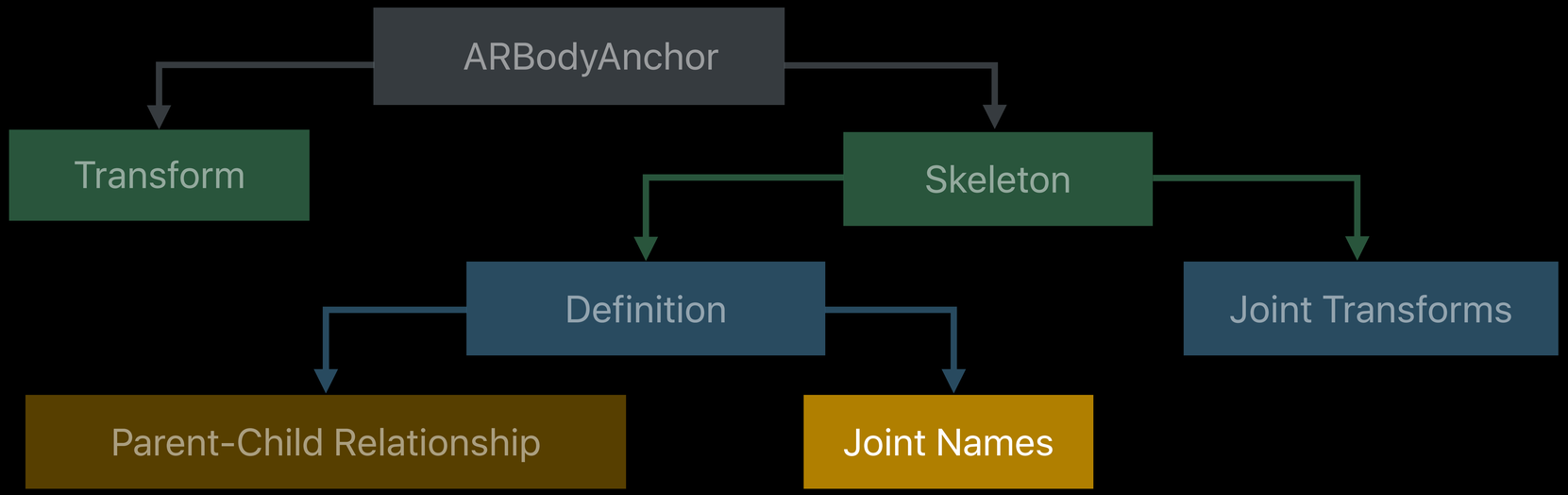

Root / Hip Joint

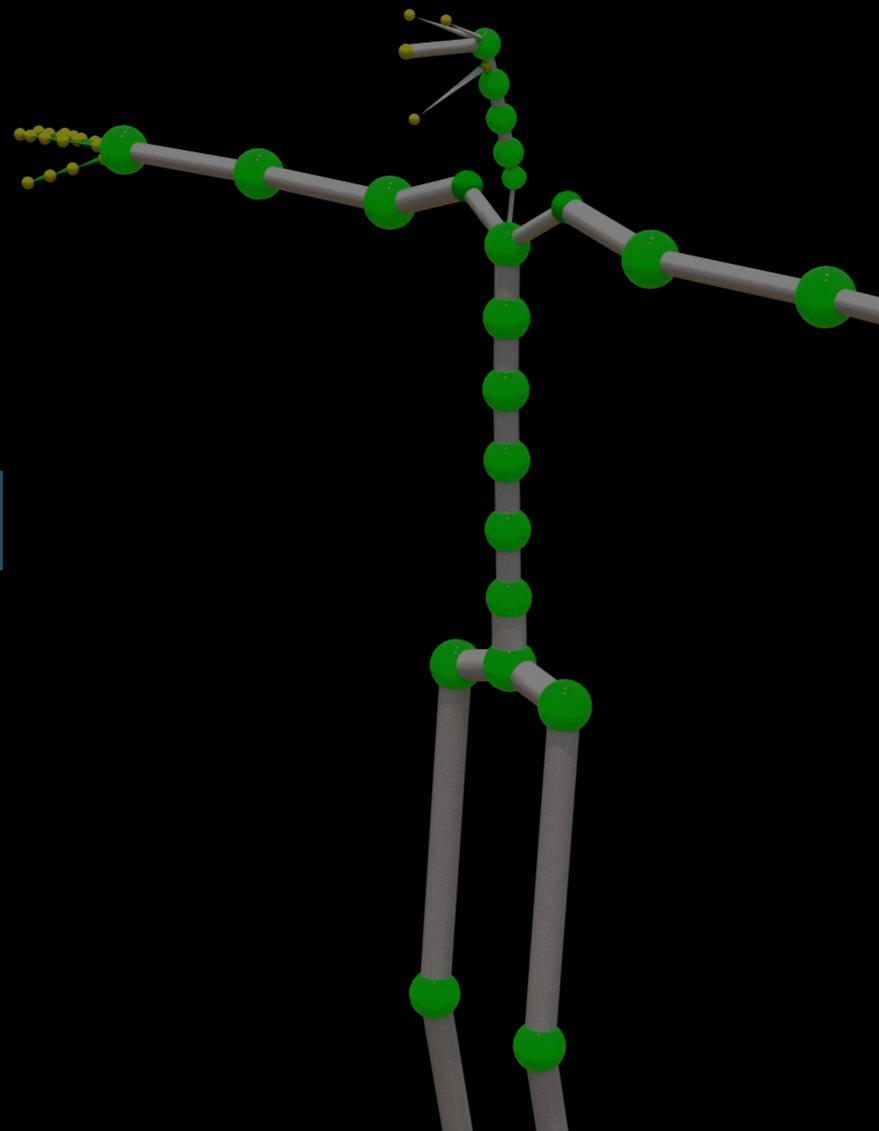

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

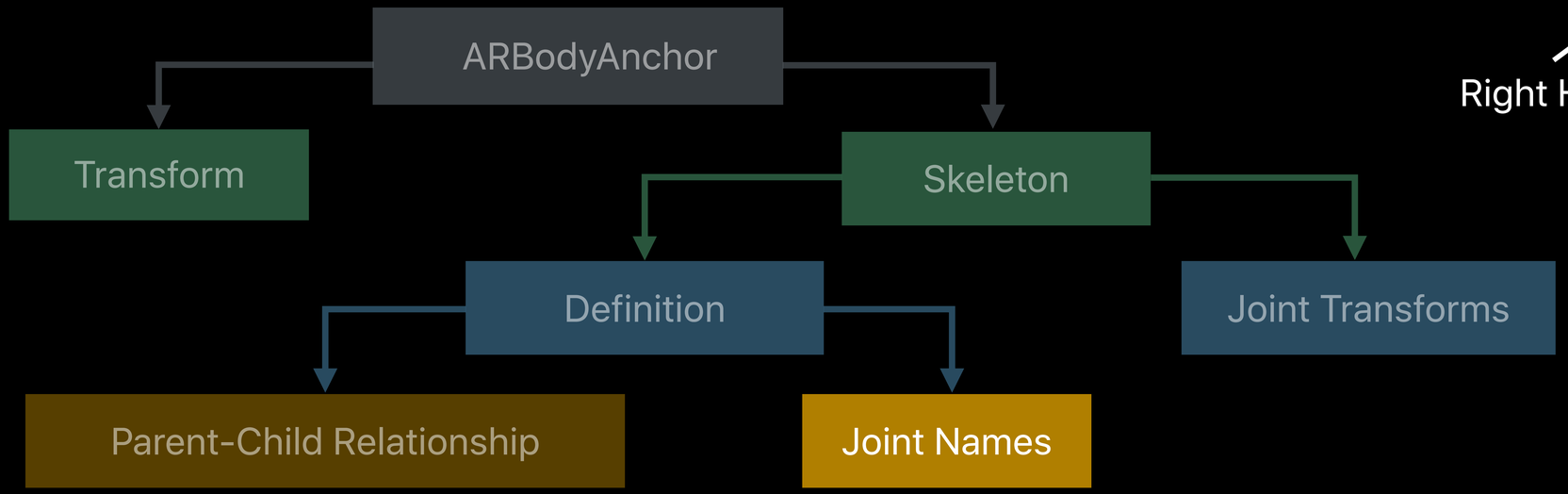

Root / Hip Joint

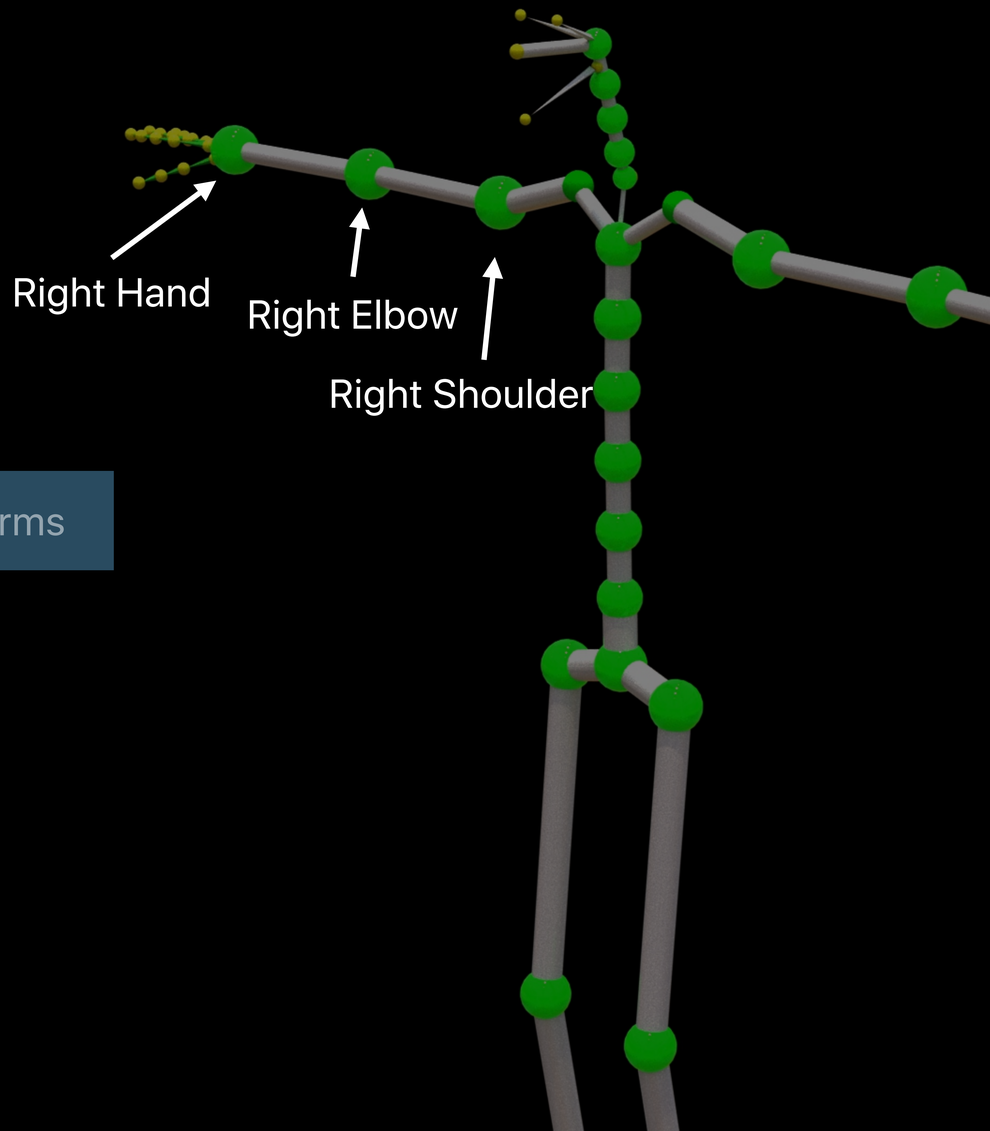

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

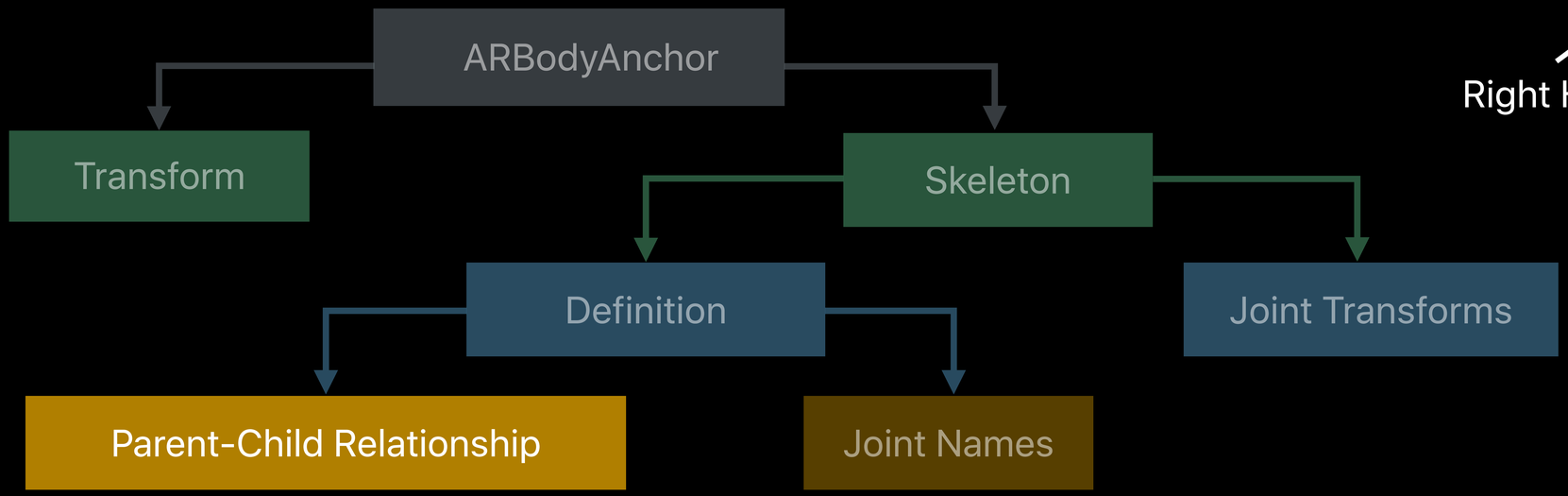

Head
Right Shoulder
Left Shoulder
Left Hand
Right Hand
Root / Hip Joint
Right Foot
Left Foot

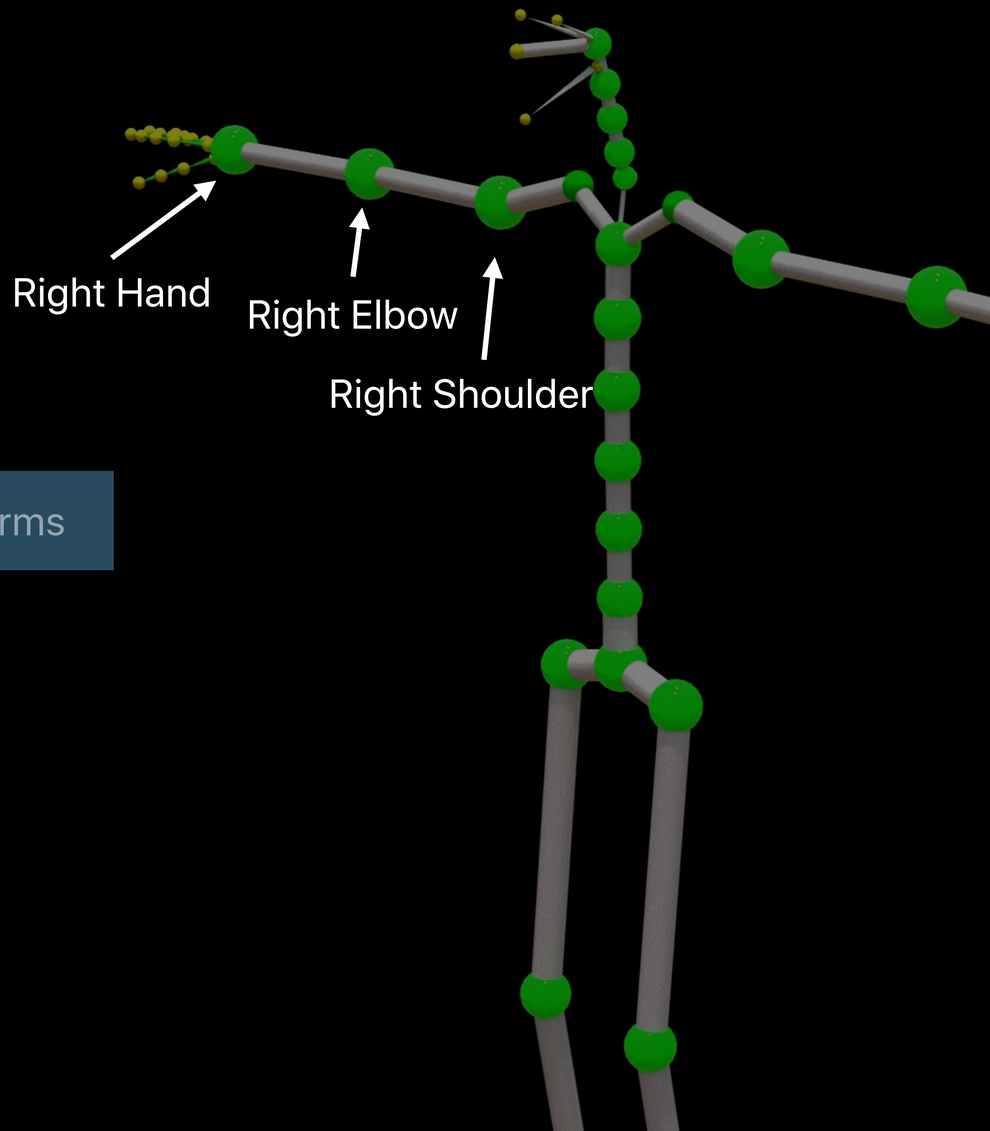

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

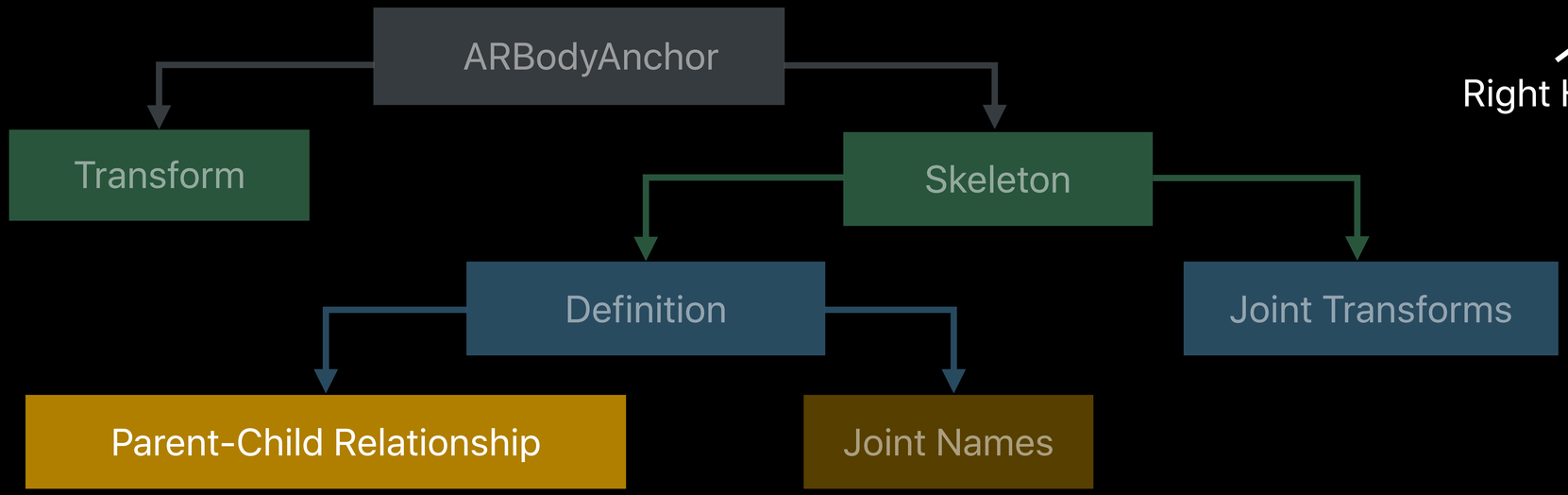



ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

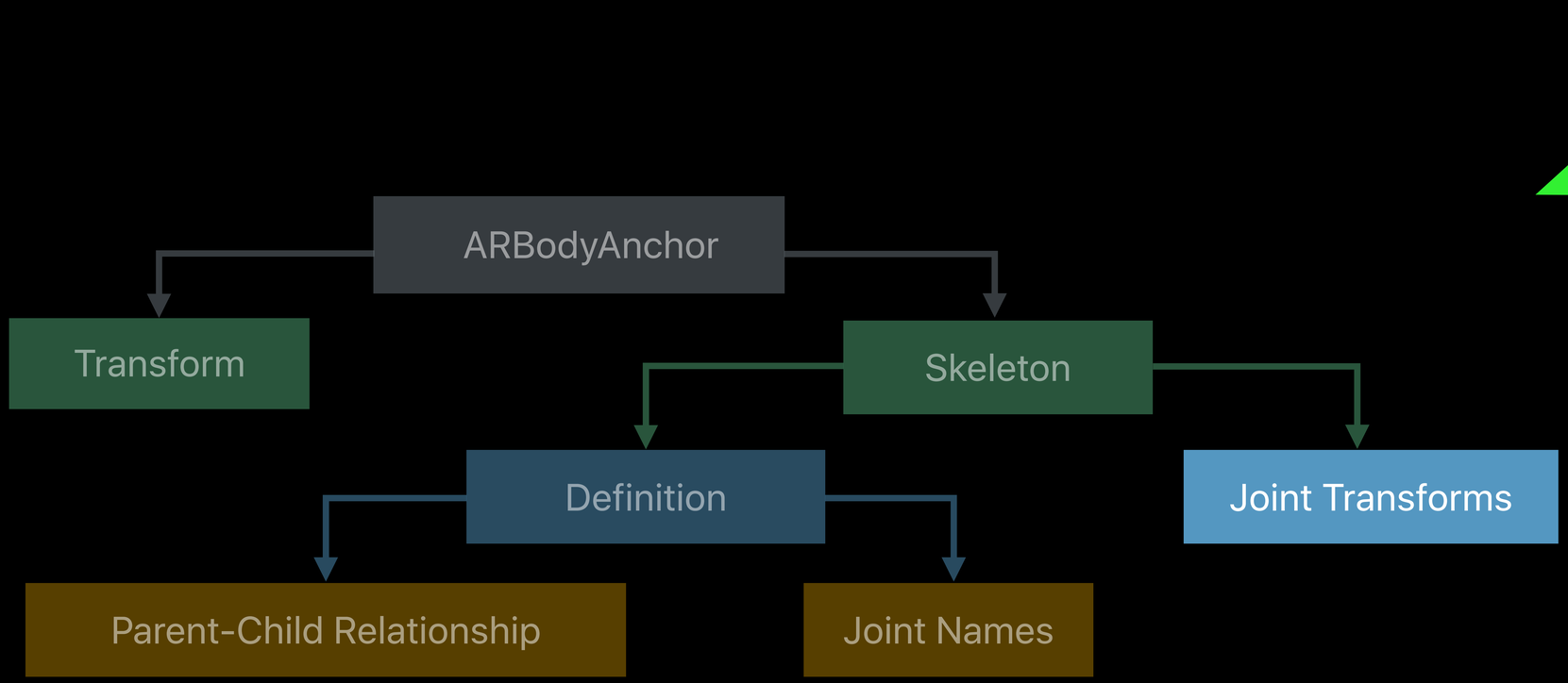

Right Hand
Right Elbow
Right Shoulder

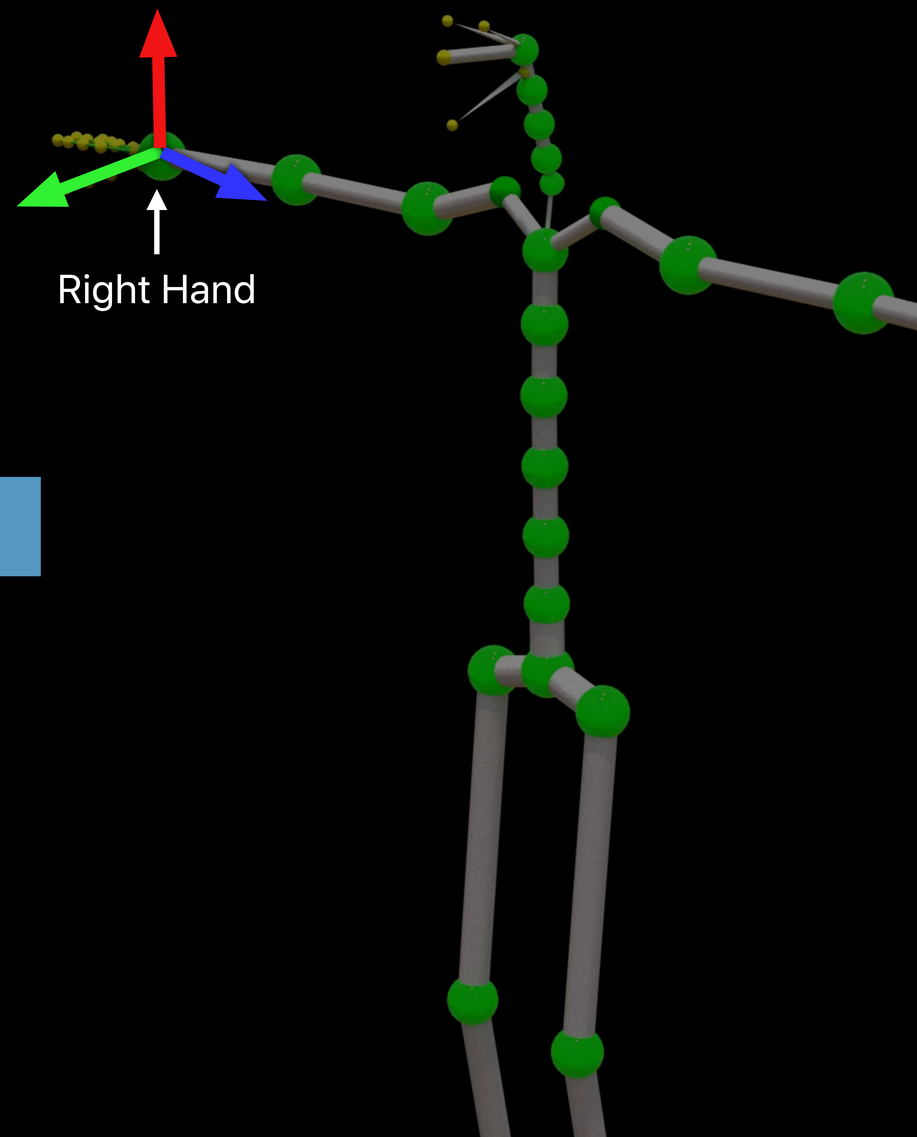

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names


Right Hand
Right Elbow
Right Shoulder


ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names


Right Hand
Right Elbow
Right Shoulder

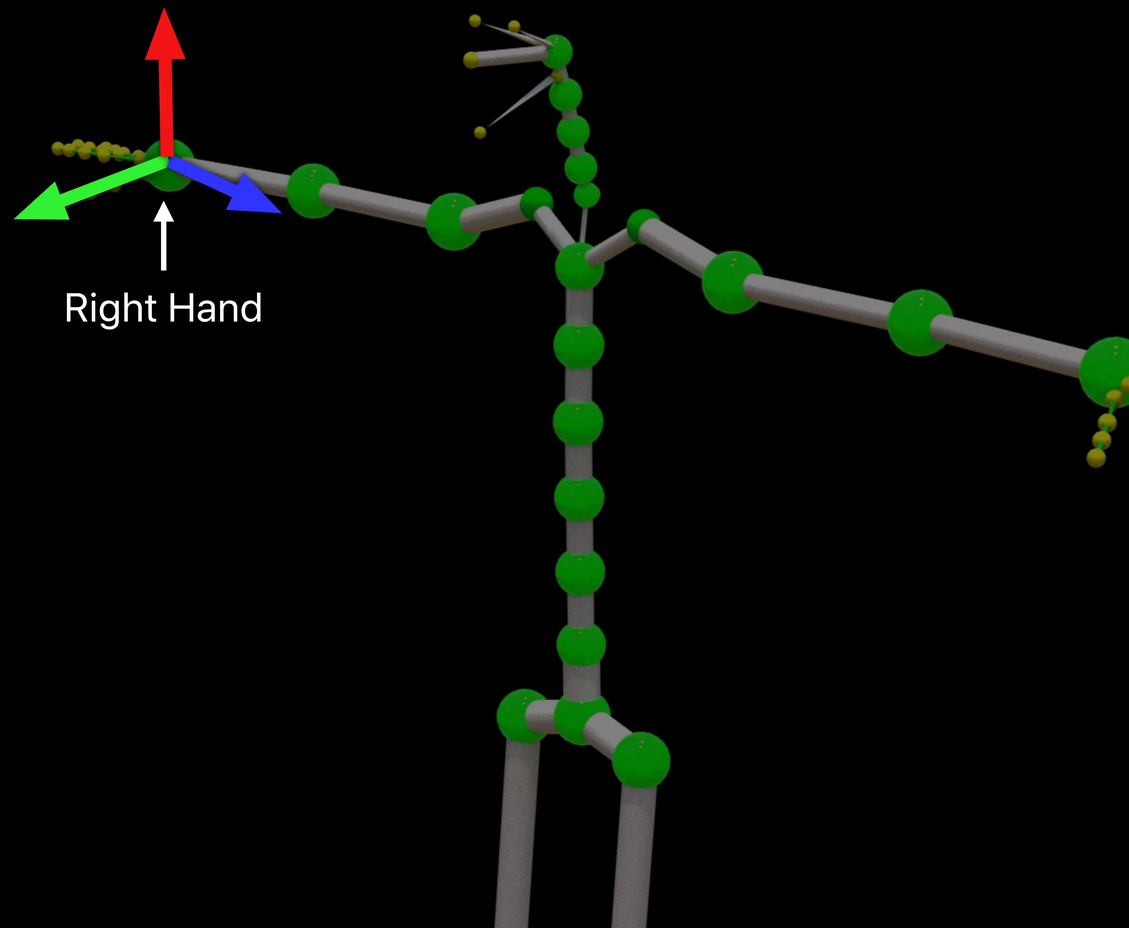

ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names


Right Hand

Accessing Joint Transforms

Relative to parent

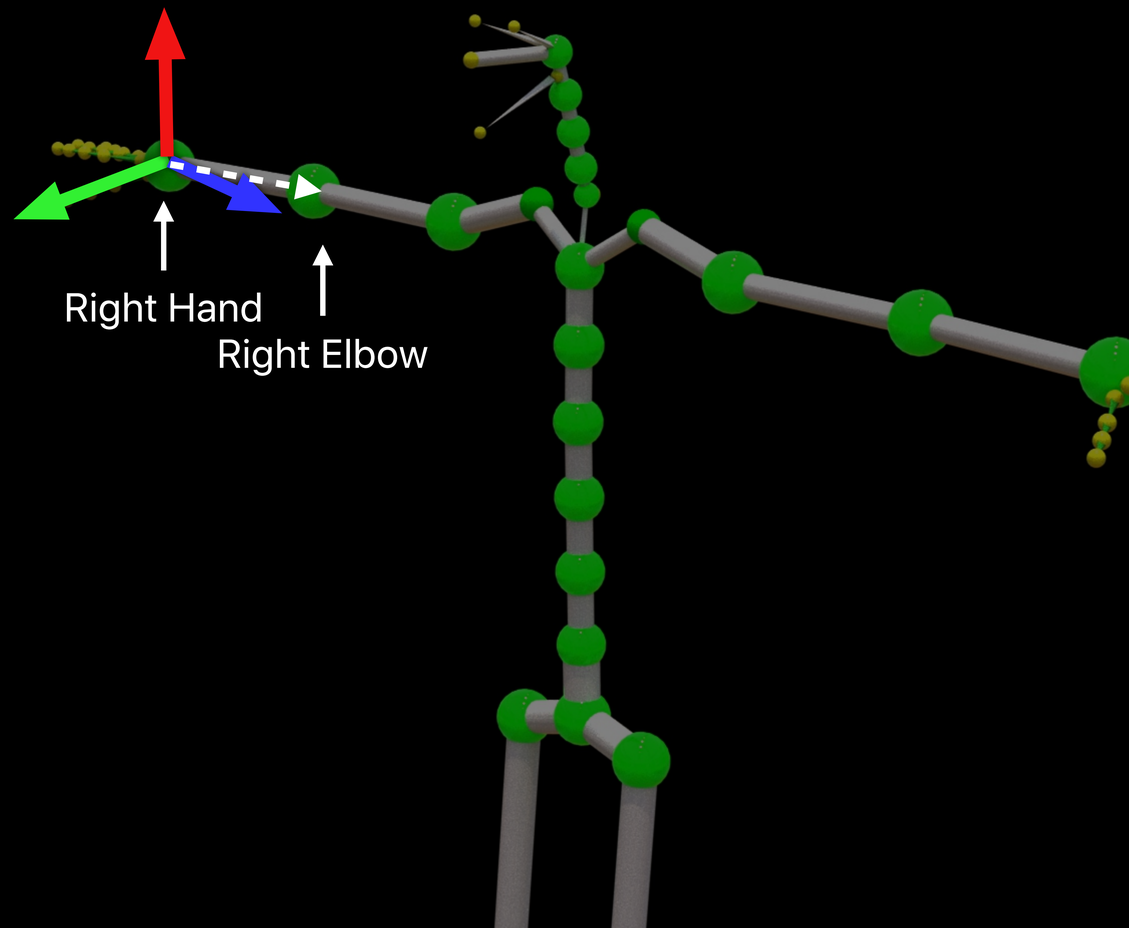
```
func localTransform (for JointName:
.rightHand)
```



Accessing Joint Transforms

Relative to parent

```
func localTransform (for JointName:
.rightHand)
```



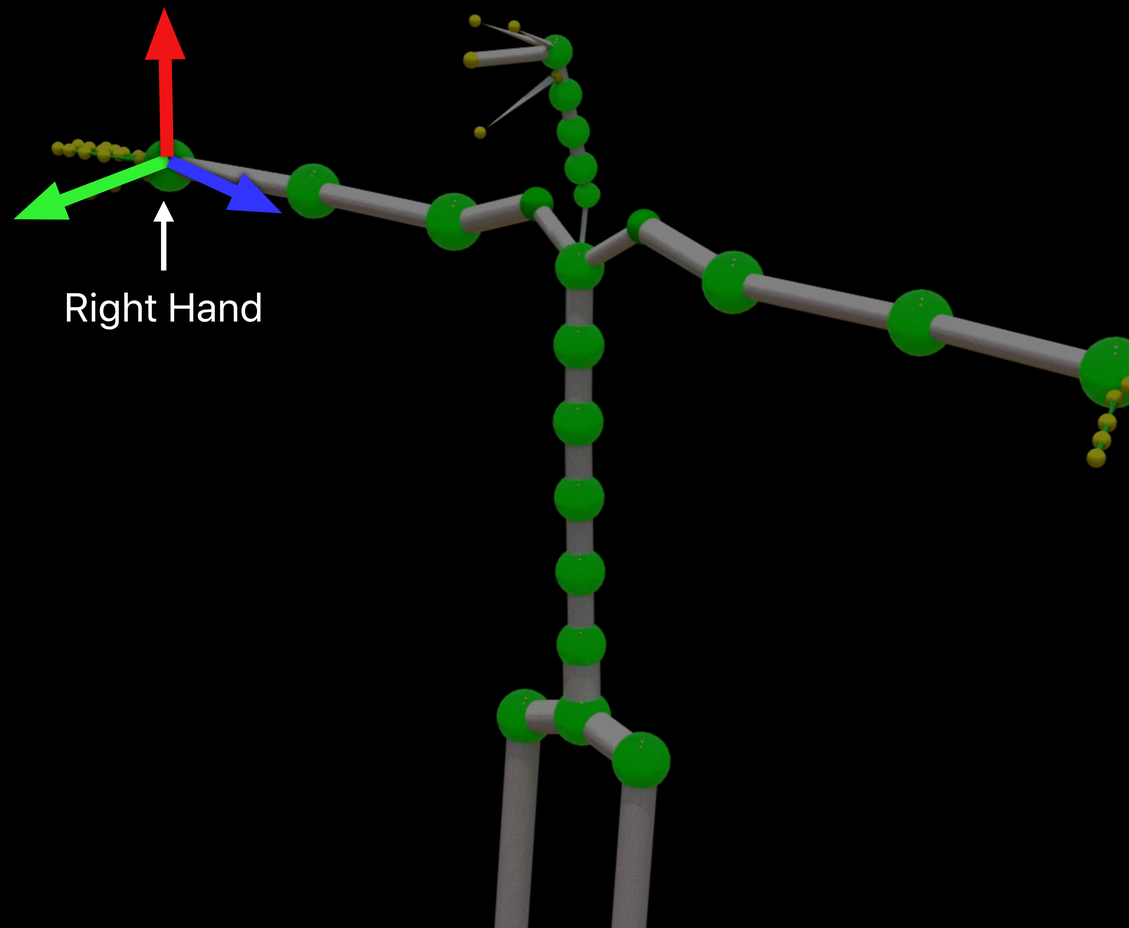
Accessing Joint Transforms

Relative to parent

```
func localTransform (for JointName:
.rightHand)
```

Relative to root

```
func modelTransform (for JointName:
.rightHand)
```



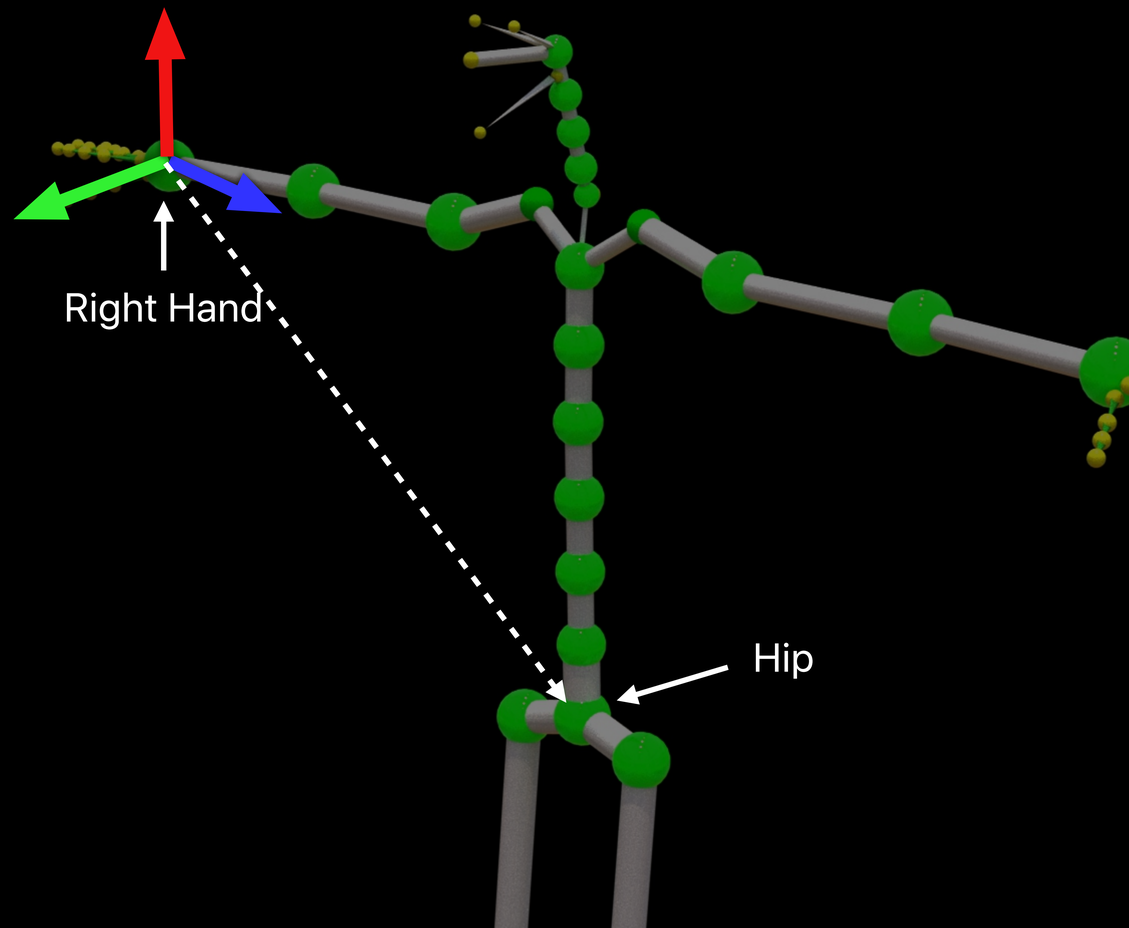
Accessing Joint Transforms

Relative to parent

```
func localTransform (for JointName:
.rightHand)
```

Relative to root

```
func modelTransform (for JointName:
.rightHand)
```



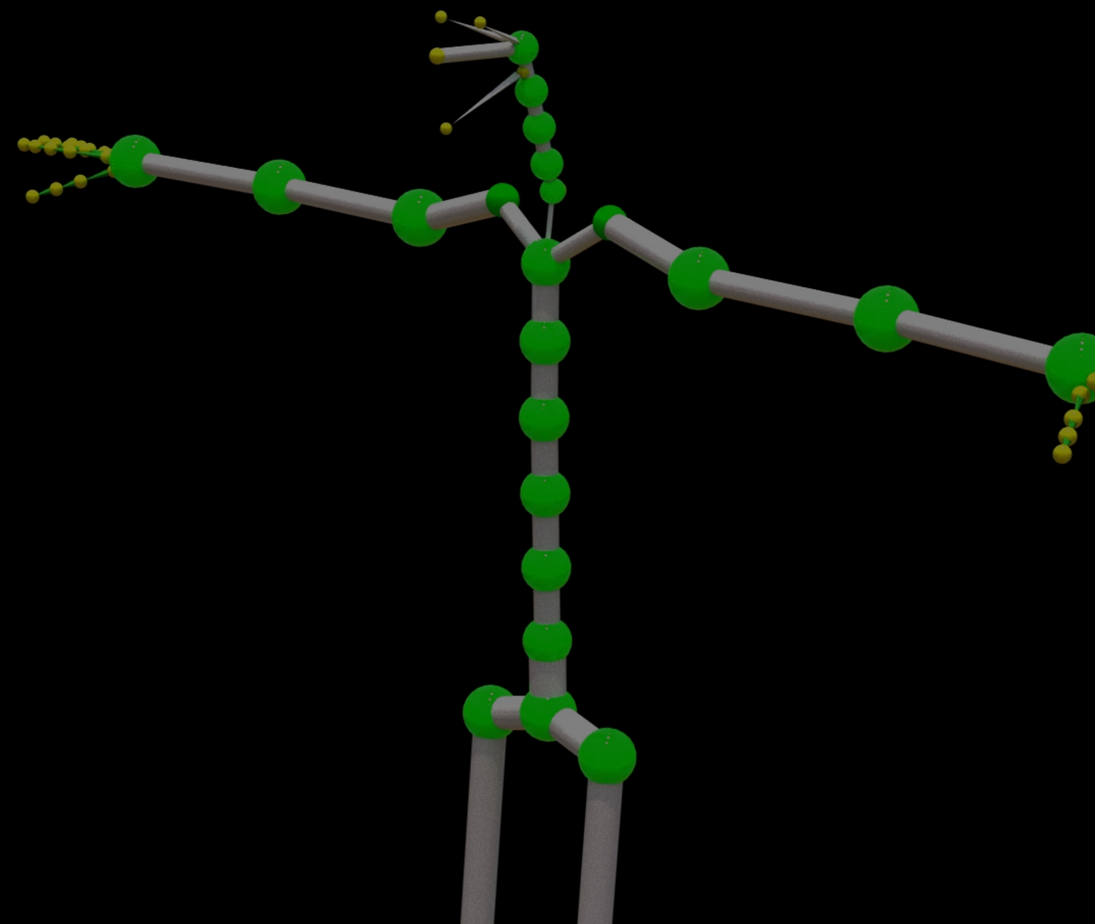
Accessing Joint Transforms

Relative to parent

```
func localTransforms
```

Relative to root

```
func modelTransforms
```



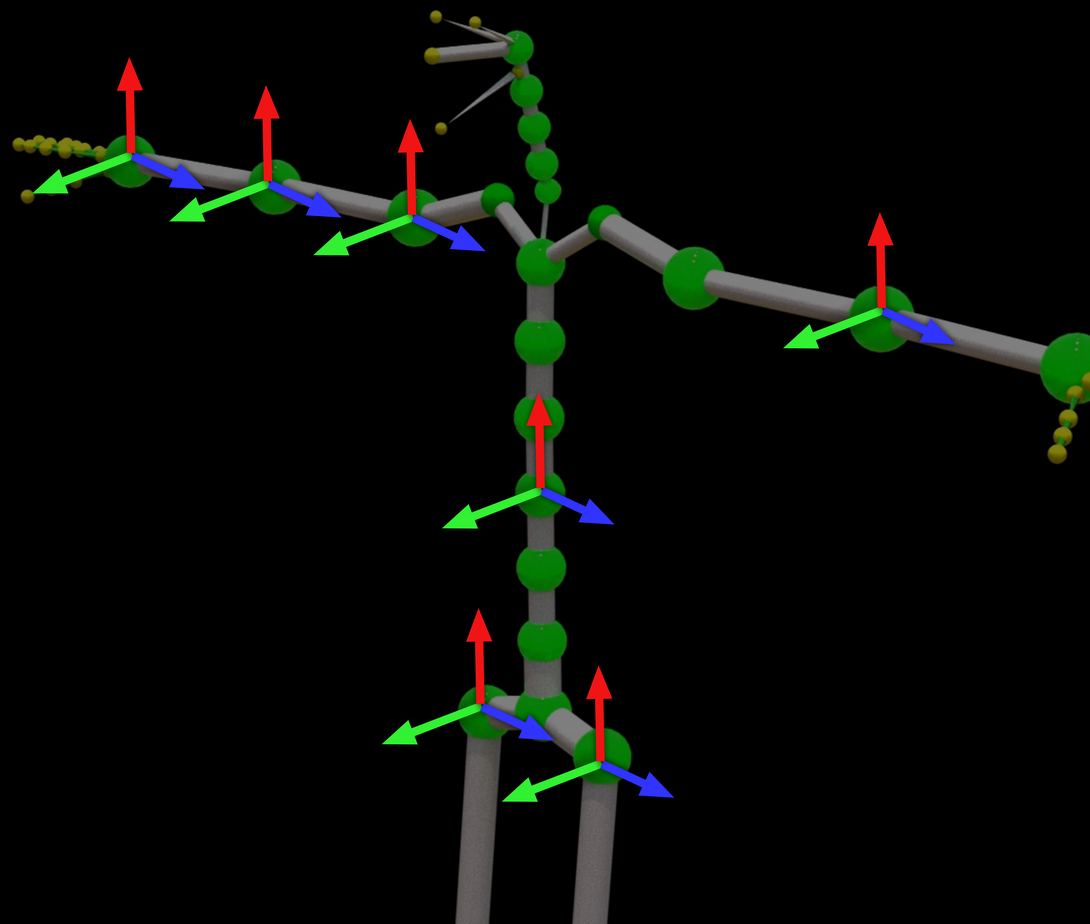
Accessing Joint Transforms

Relative to parent

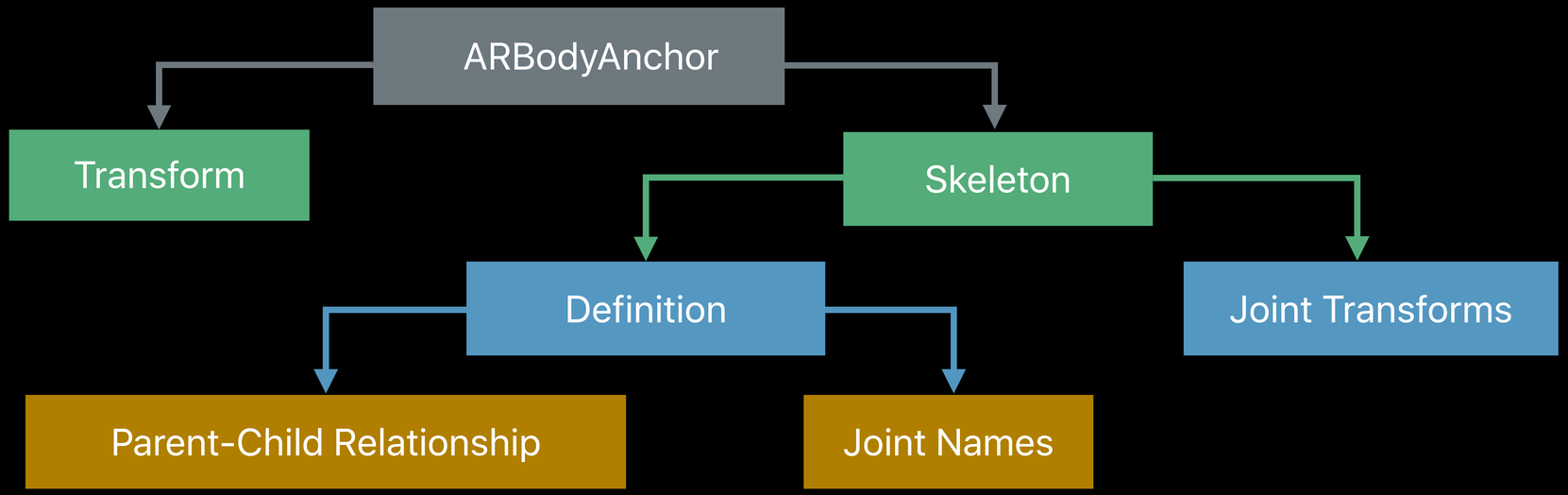
```
func localTransforms
```

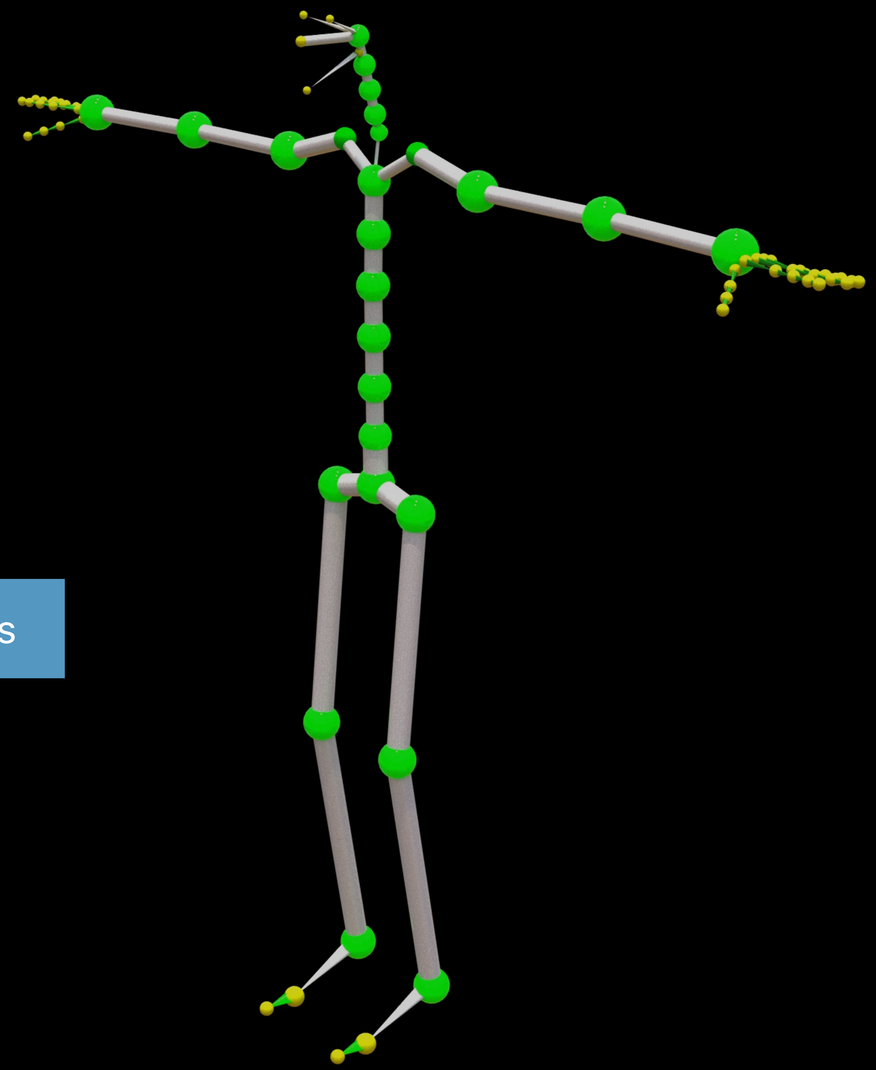
Relative to root

```
func modelTransforms
```



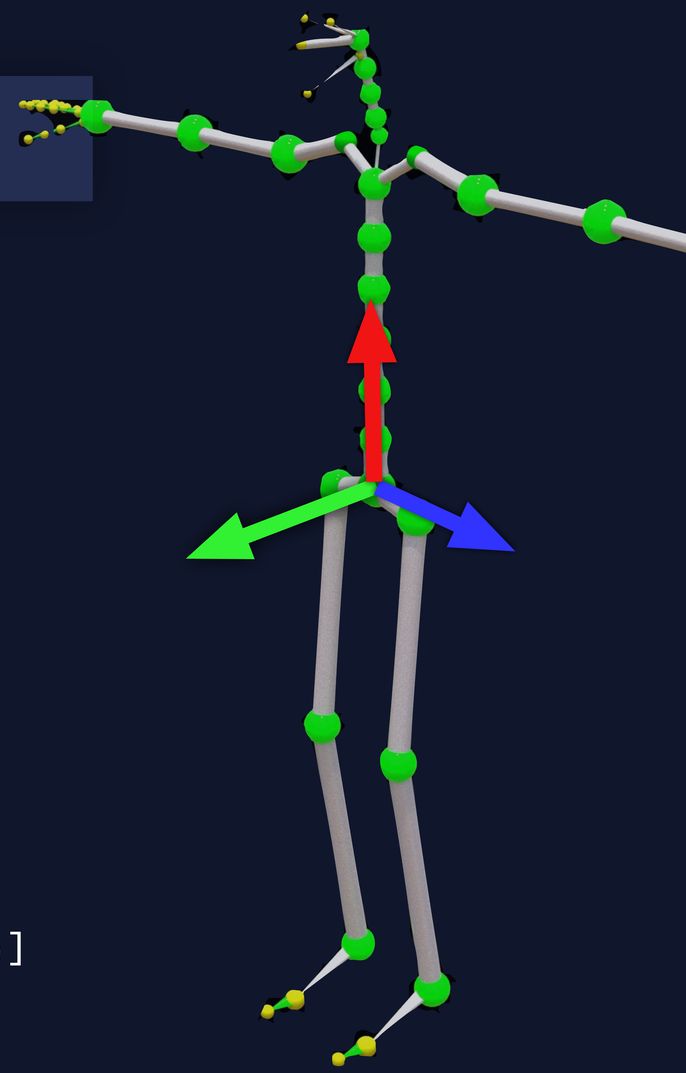
ARSkeleton


ARBodyAnchor
Transform
Skeleton
Definition
Joint Transforms
Parent-Child Relationship
Joint Names

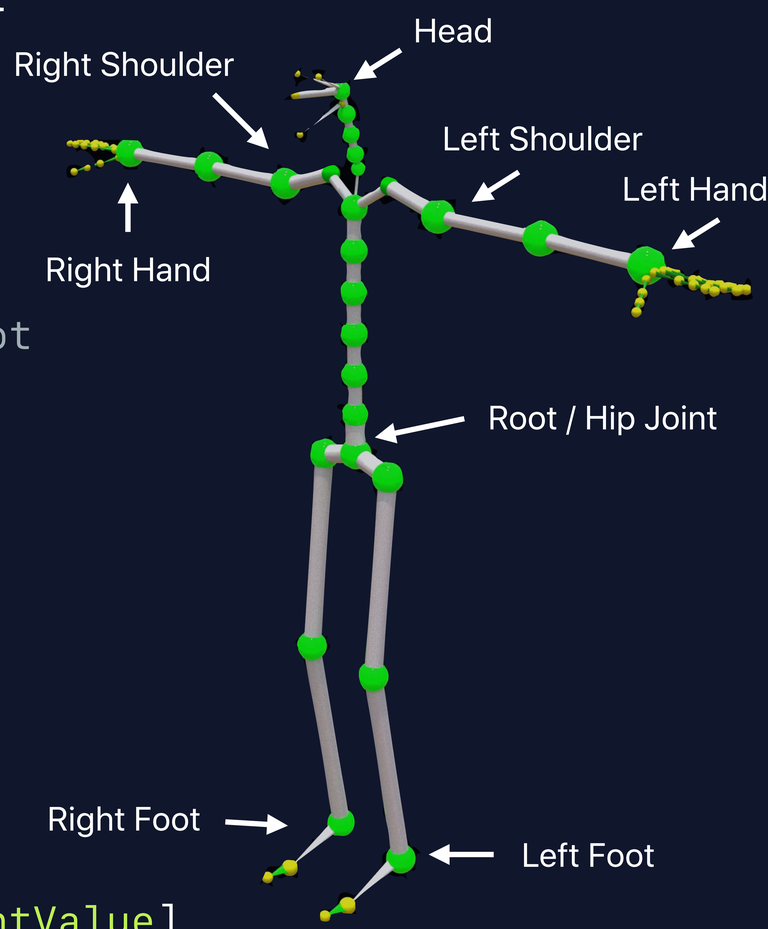


```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
        // Access to the Position of Root Node
        let hipWorldPosition = bodyAnchor.transform
        // Accessing the Skeleton Geometry
        let skeleton = bodyAnchor.skeleton
        // Accessing List of Transforms of all Joints Relative to Root
        let jointTransforms = skeleton.jointModelTransforms
        // Iterating over All Joints
        for (i, jointTransform) in jointTransforms.enumerated() {
            // Extract Parent Index from Definition
            let parentIndex = skeleton.definition.parentIndices[ i ]
            // Check If It's Not Root
            guard parentIndex != -1 else { continue }
            // Find Position of Parent Joint
            let parentJointTransform = jointTransforms[parentIndex.intValue]
            …
        }
    }
```

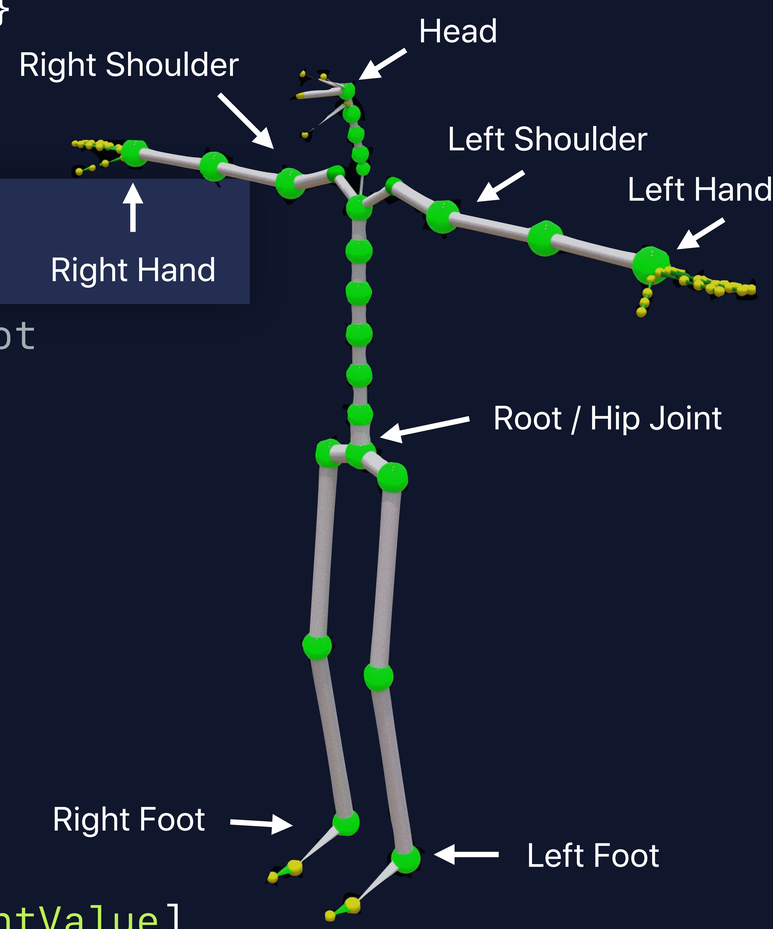
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



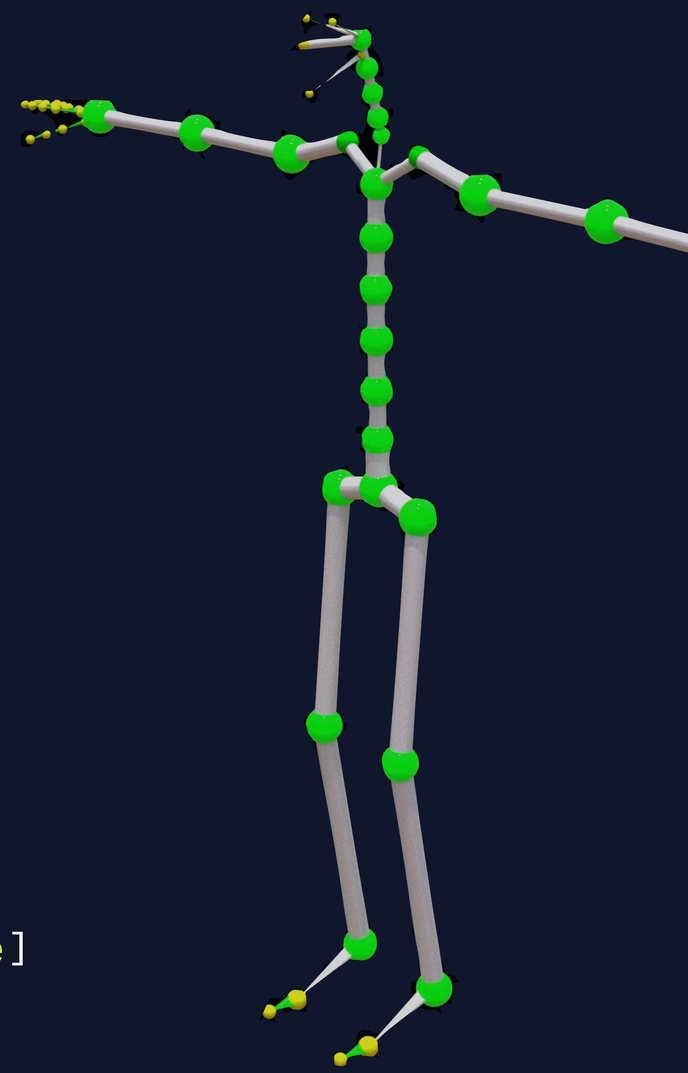
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



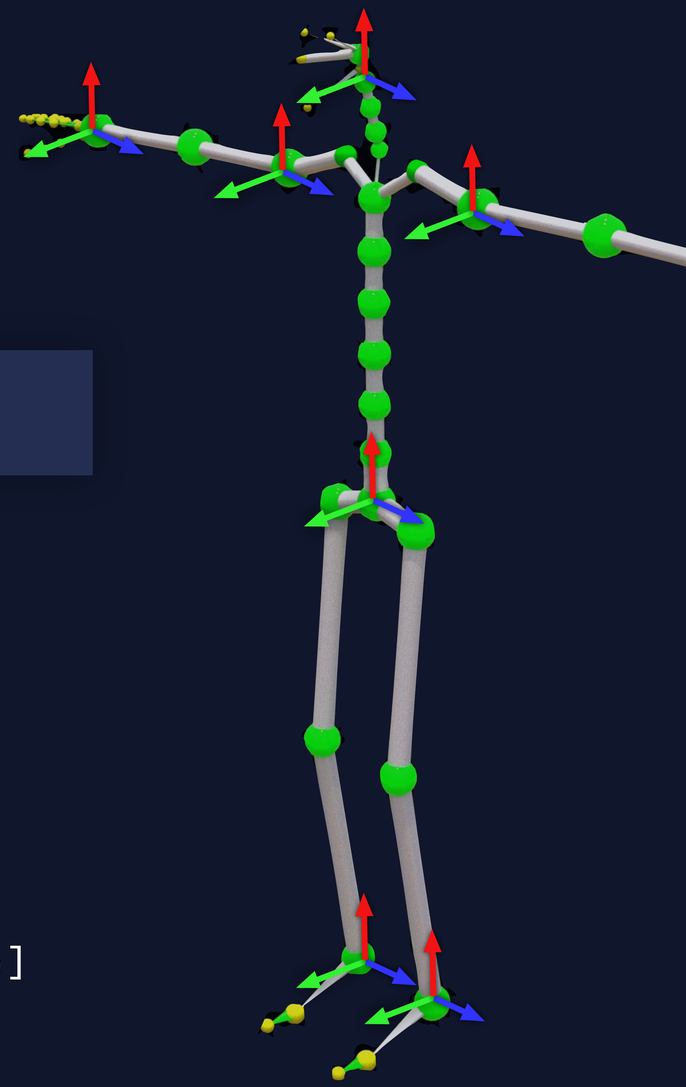
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



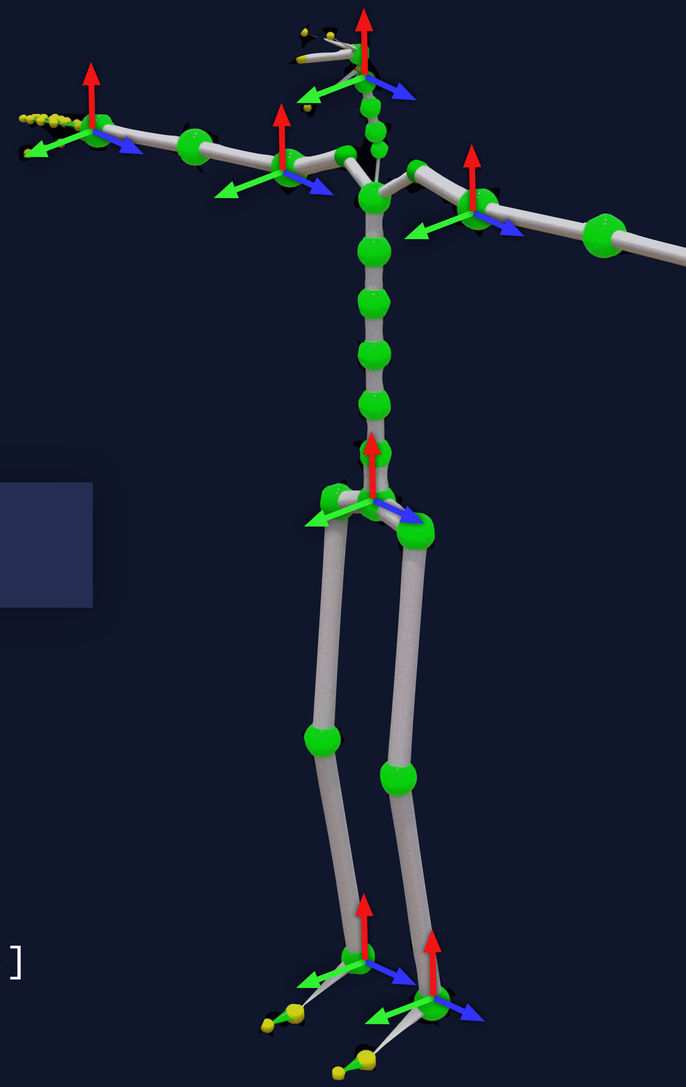
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



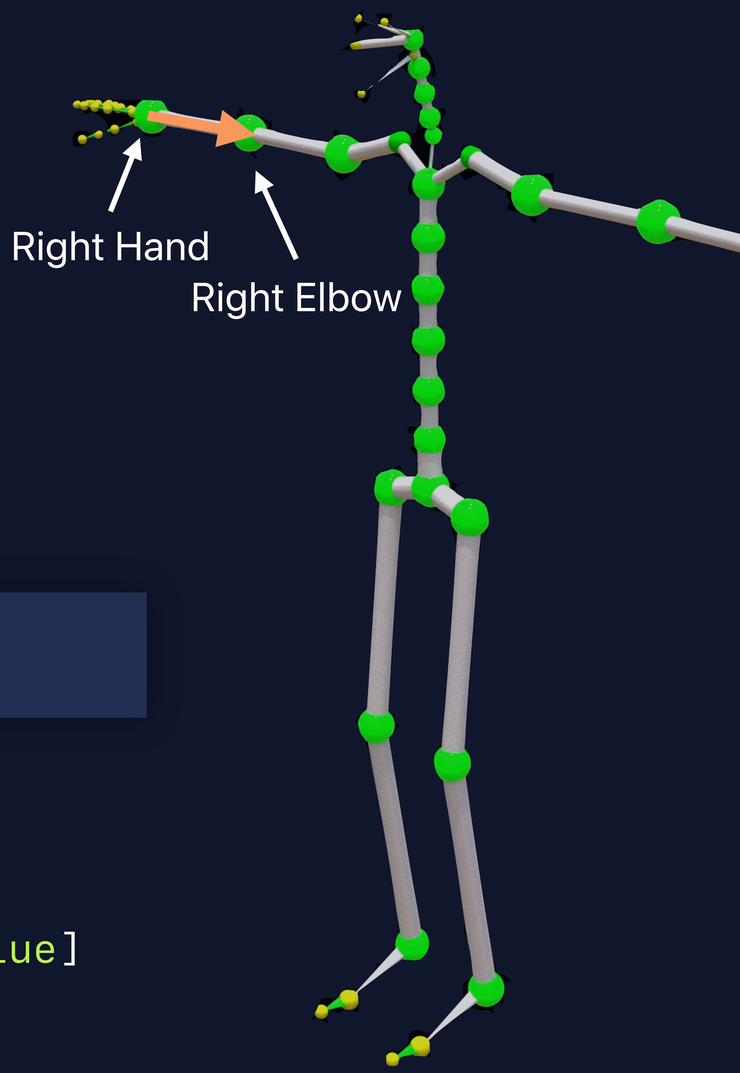
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



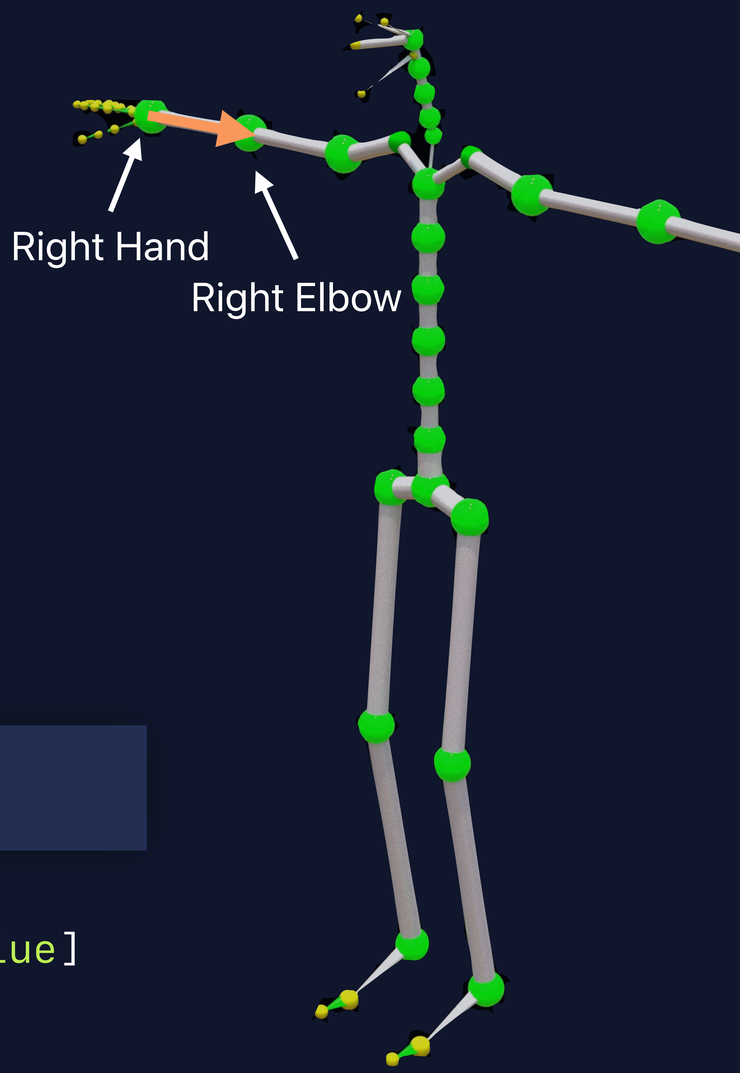
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



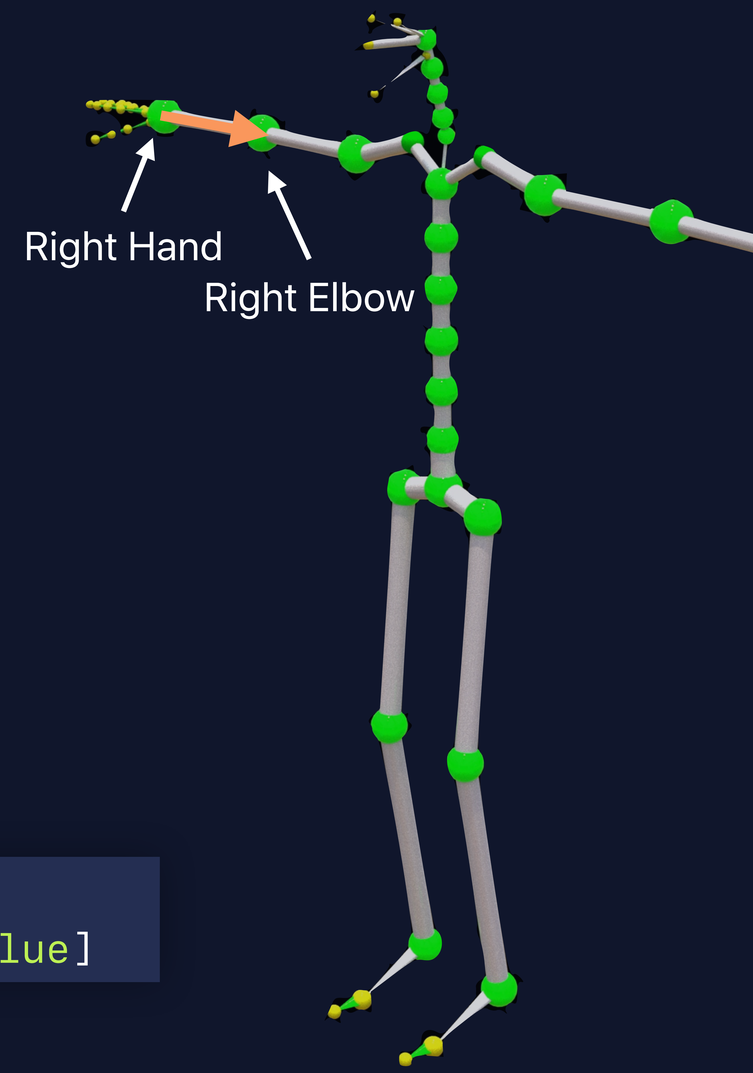
```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



```swift
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



```
// Look for the bodyAnchor
for anchor in anchors {
      guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
       // Access to the Position of Root Node
       let hipWorldPosition = bodyAnchor.transform
       // Accessing the Skeleton Geometry
       let skeleton = bodyAnchor.skeleton
       // Accessing List of Transforms of all Joints Relative to Root
       let jointTransforms = skeleton.jointModelTransforms
       // Iterating over All Joints
       for (i, jointTransform) in jointTransforms.enumerated() {
           // Extract Parent Index from Definition
           let parentIndex = skeleton.definition.parentIndices[ i ]
           // Check If It's Not Root
           guard parentIndex != -1 else { continue }
           // Find Position of Parent Joint
           let parentJointTransform = jointTransforms[parentIndex.intValue]
           …
       }
   }
```



Extracting Data from Skeleton in 2D

Extracting Data from 2D Skeleton

NEW

Detailed access to 2D skeleton elements

Extracting Data from 2D Skeleton

NEW

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates

Extracting Data from 2D Skeleton

NEW

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates

Easy to use API

Extracting Data from 2D Skeleton

NEW

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates

Easy to use API

Can be used for image analysis

Extracting Data from 2D Skeleton

NEW

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates

Easy to use API

Can be used for image analysis

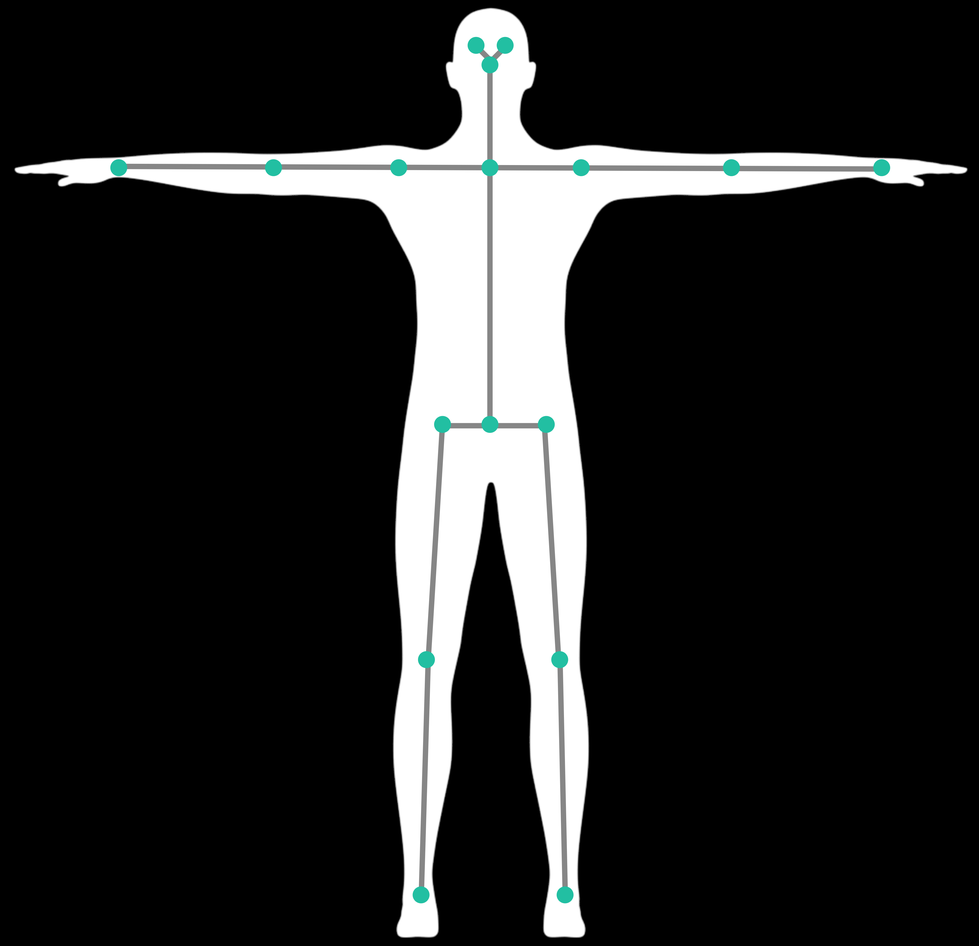
Interfaced through ARBody2D

Extracting Data from 2D Skeleton

NEW

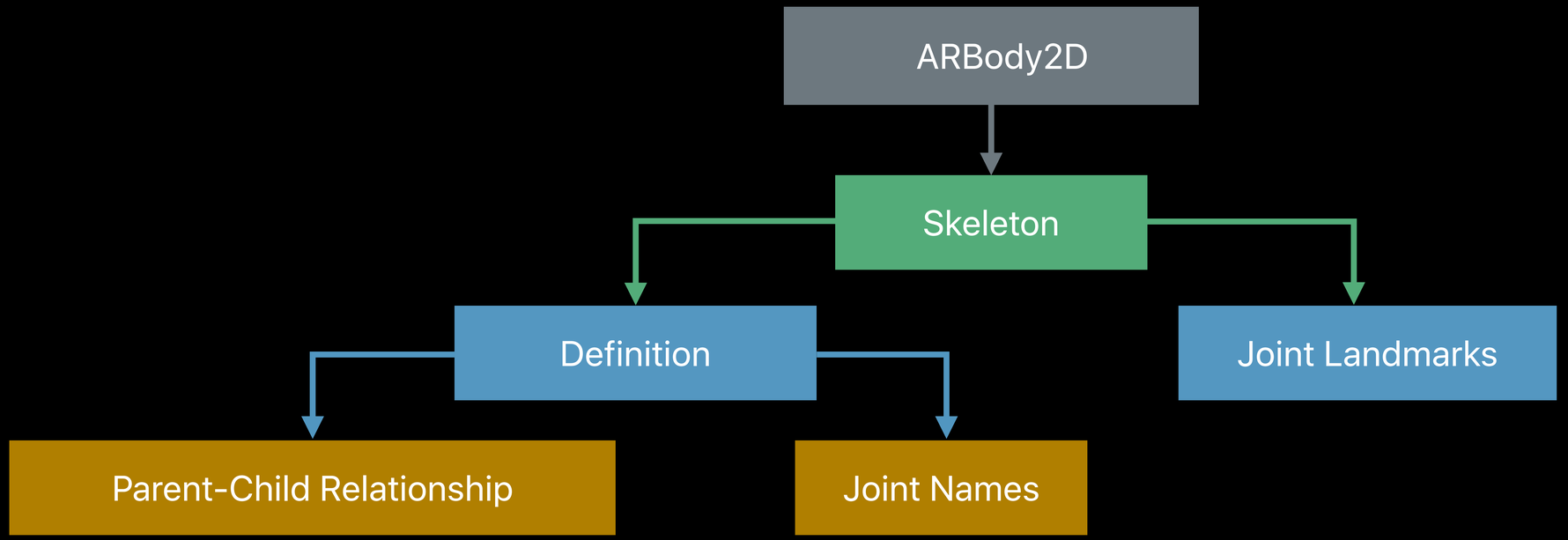
ARBody2D object

ARBody2D contains 2D skeletal structure

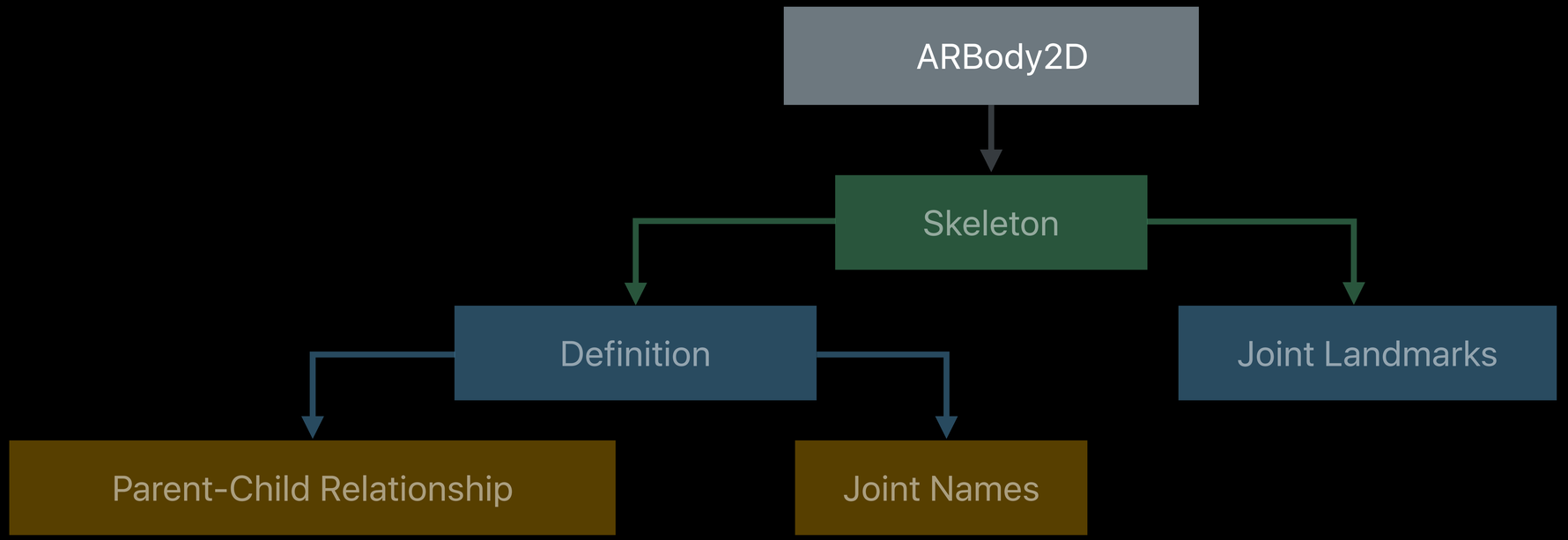


Extracting Data from 2D Skeleton

Extracting Data from 2D Skeleton



Extracting Data from 2D Skeleton



Accessing ARBody2D Object

NEW

ARFrame

```
func session(_ session ARSession, didUpdate frame: ARFrame){
      // Accessing ARBody2D Object from ARFrame
      let person = frame.detectedBody
}
```

Accessing ARBody2D Object

NEW

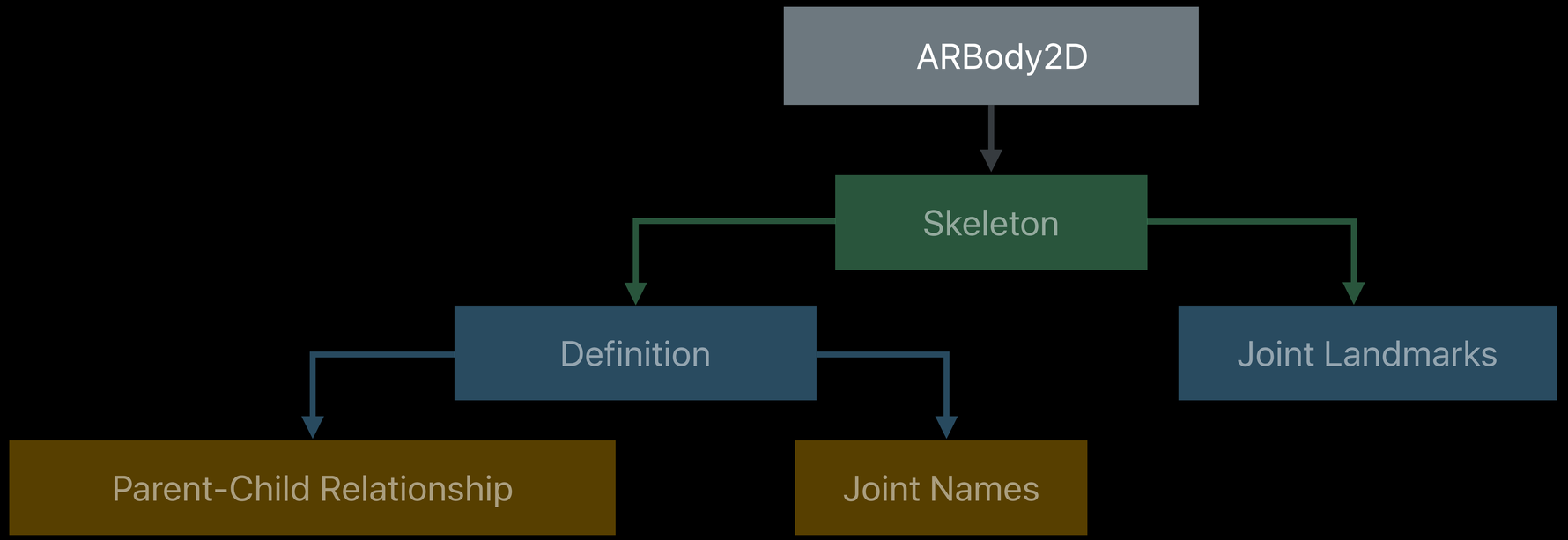
ARFrame

```
func session(_ session ARSession, didUpdate frame: ARFrame){
      // Accessing ARBody2D Object from ARFrame
      let person = frame.detectedBody
}
```

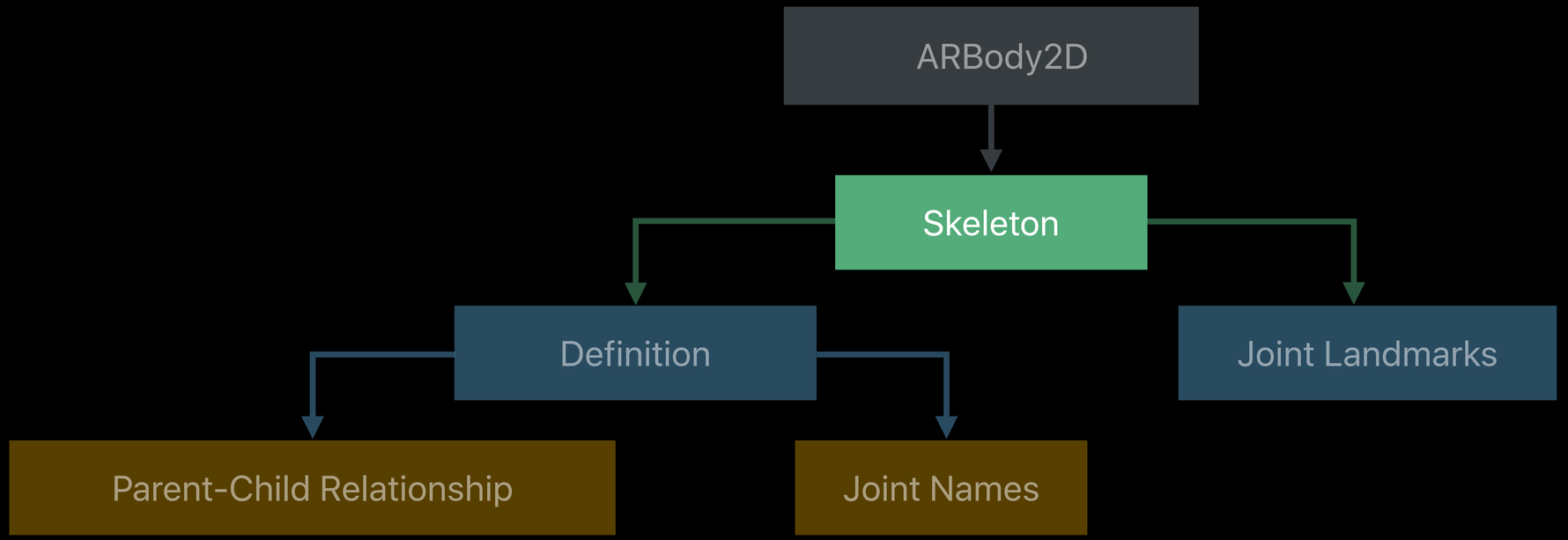
ARBodyAnchor

```
guard let bodyAnchor = anchor as? ARBodyAnchor else {continue}
      // Accessing ARBody2D Object from referenceBody Property
      let body2D = bodyAnchor.referenceBody
```

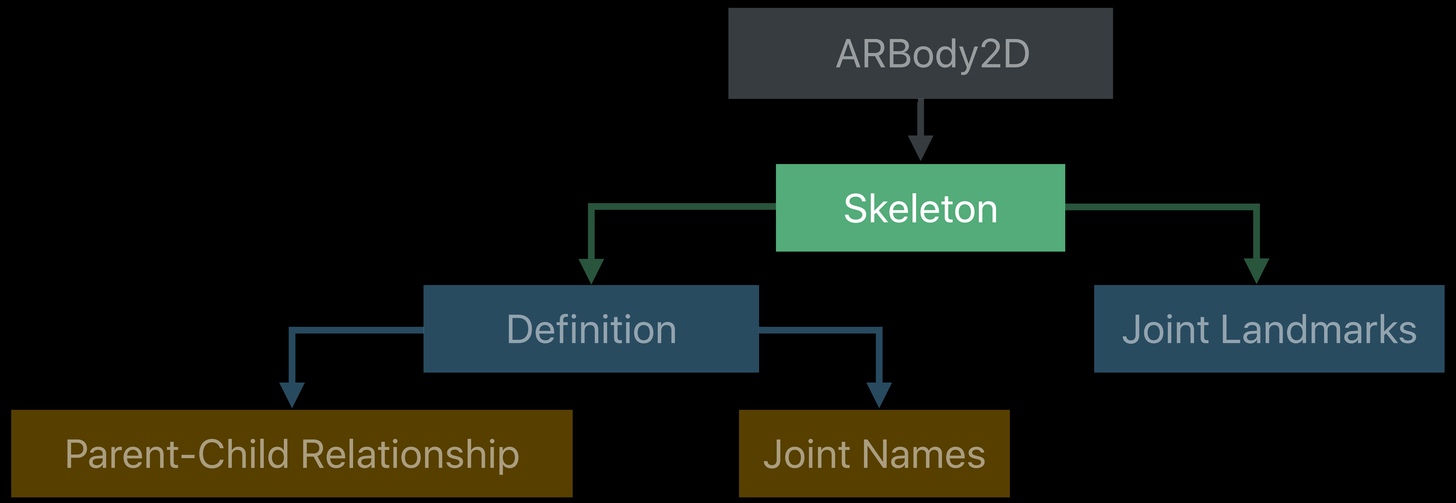
Extracting Data from 2D Skeleton

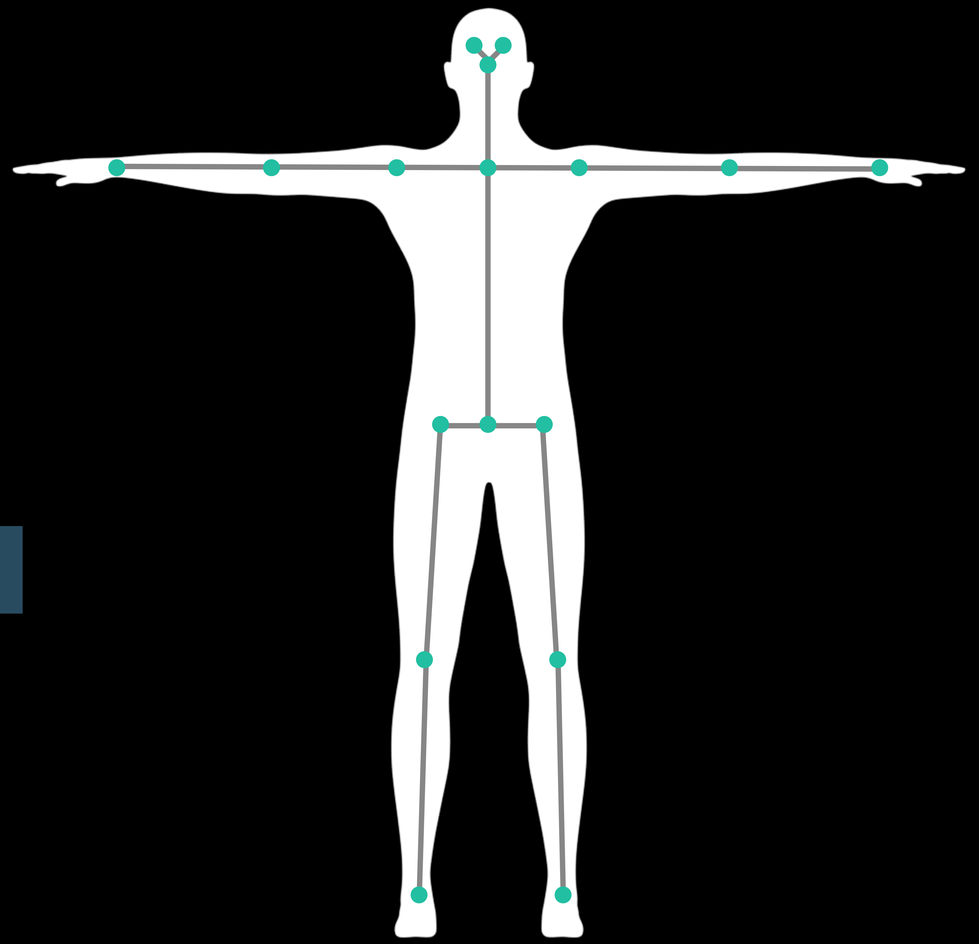


Extracting Data from 2D Skeleton

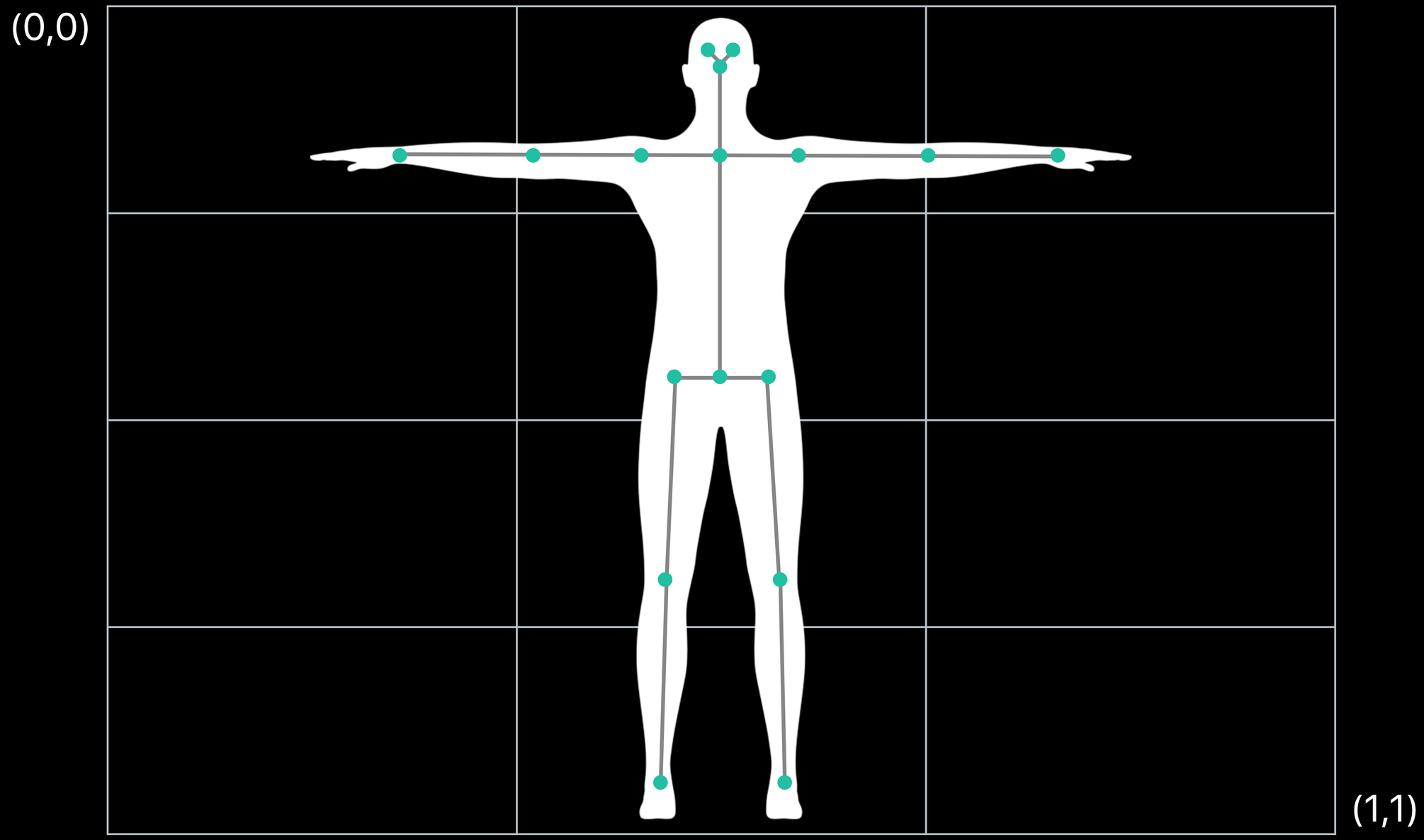


Extracting Data from 2D Skeleton

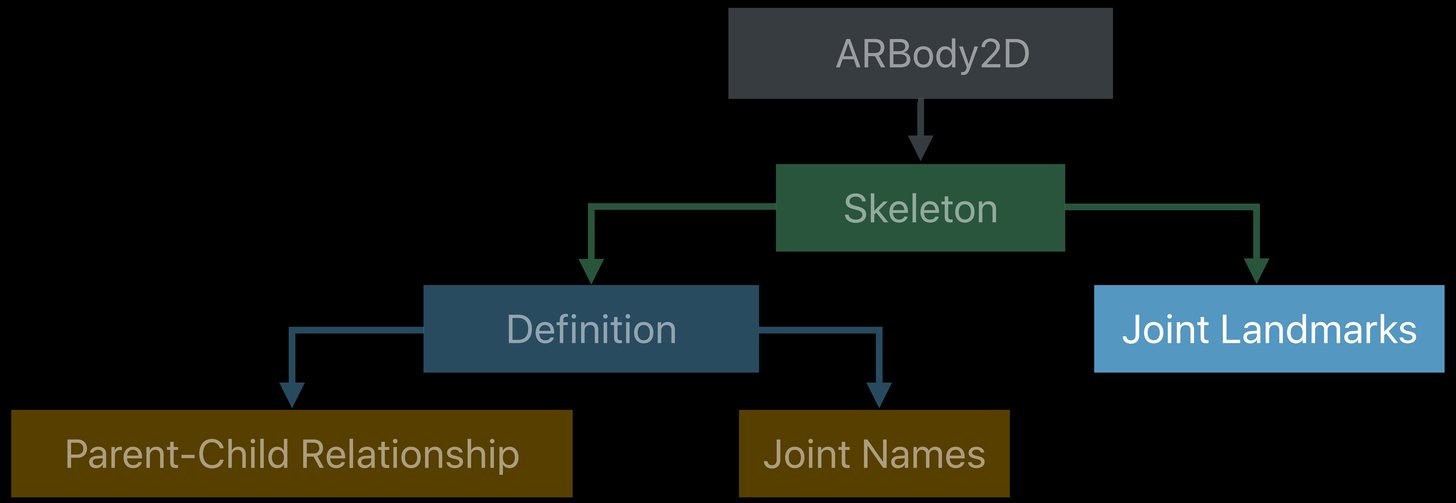


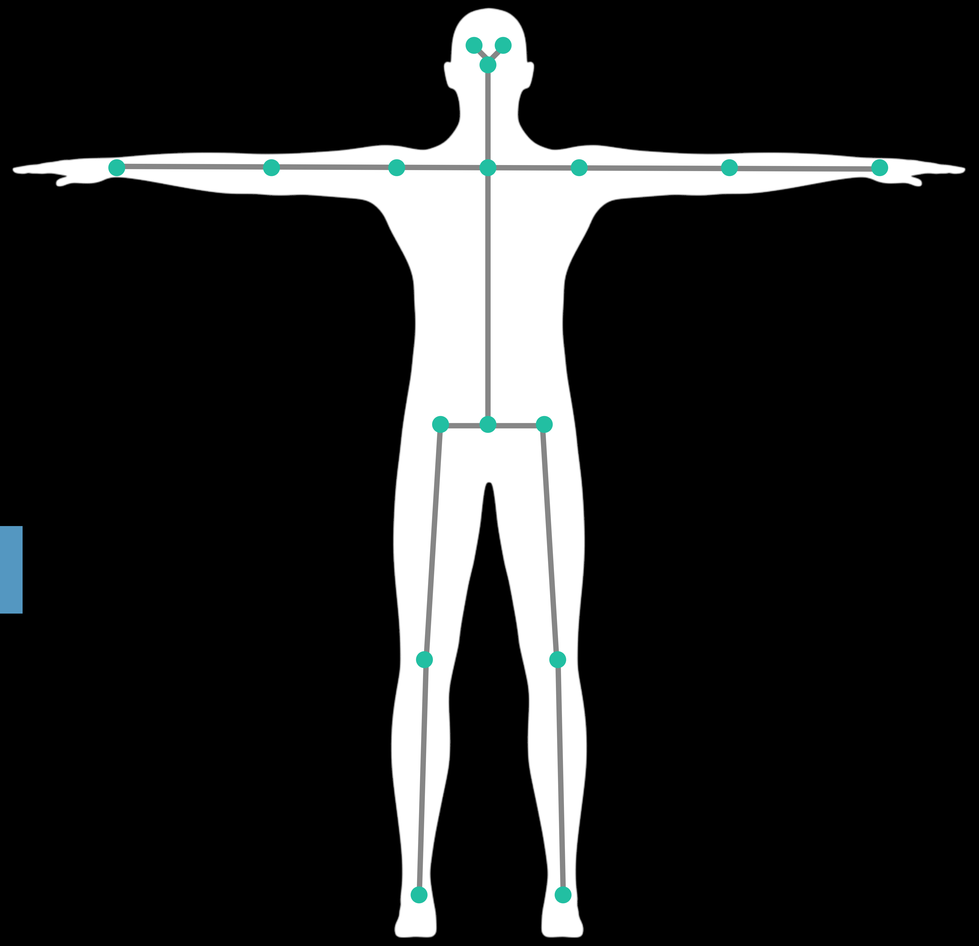


Extracting Data from 2D Skeleton

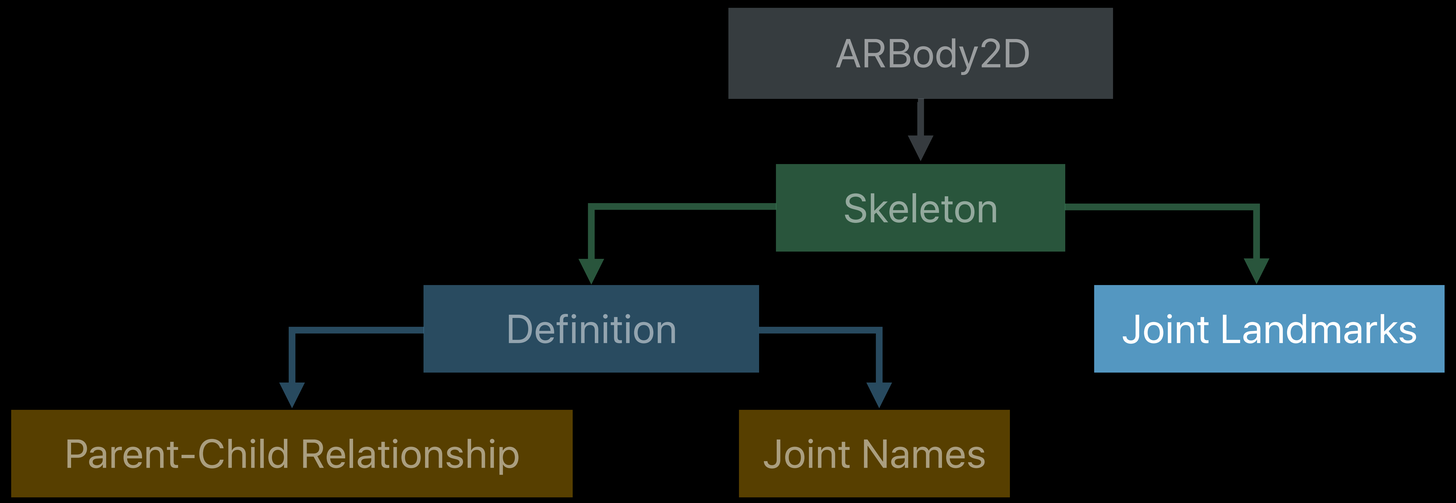


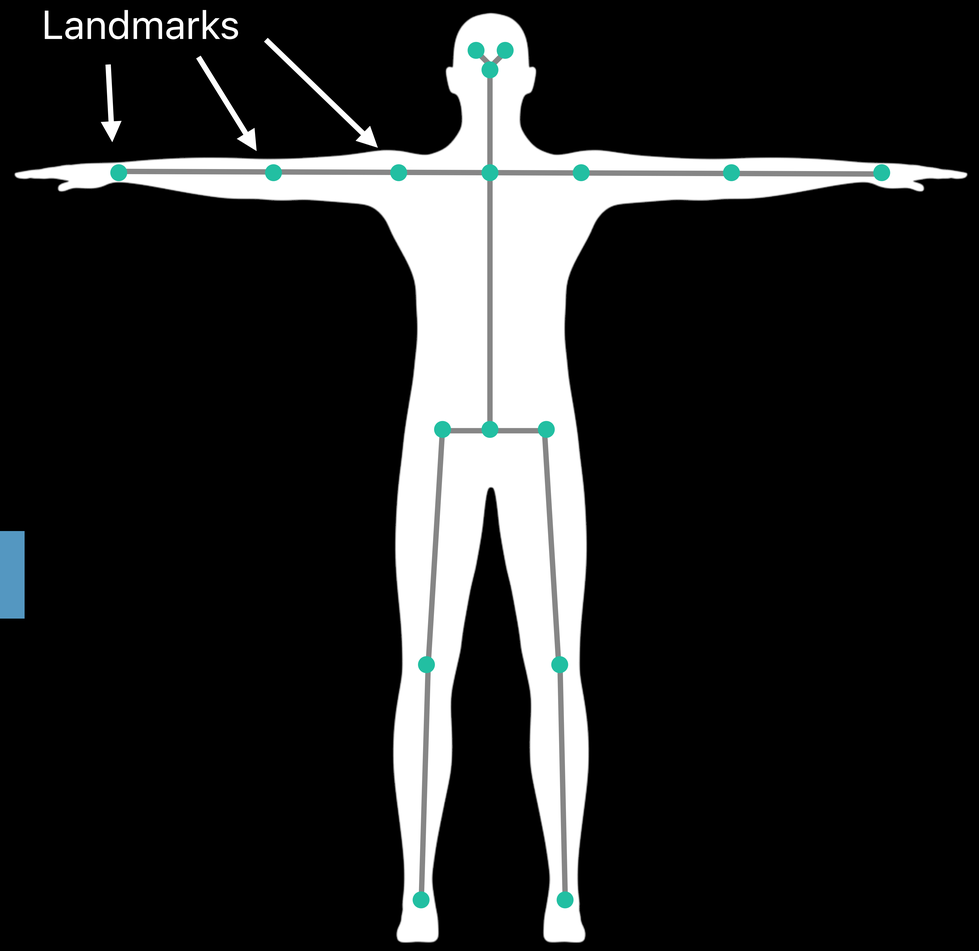
Extracting Data from 2D Skeleton



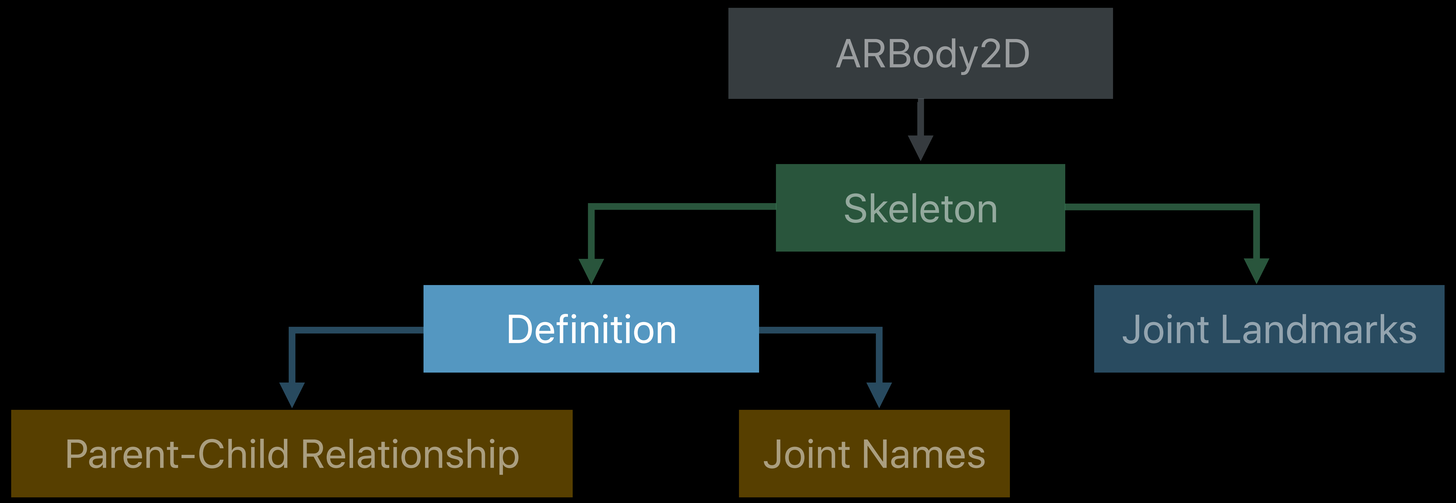


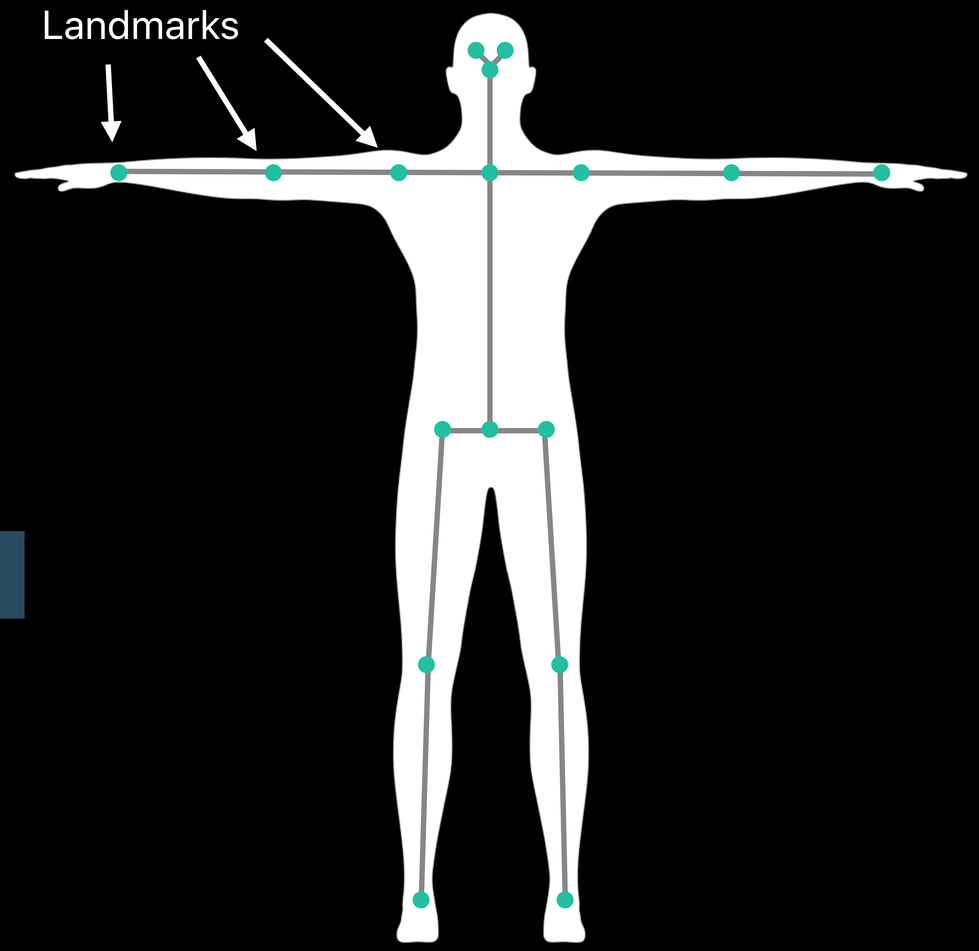
Extracting Data from 2D Skeleton



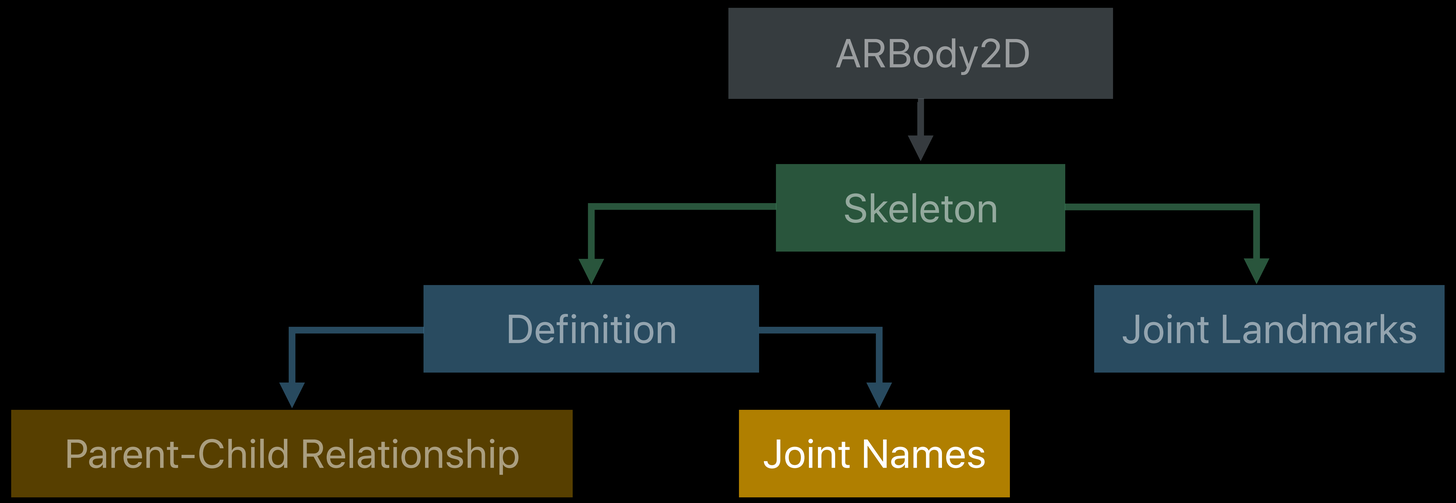


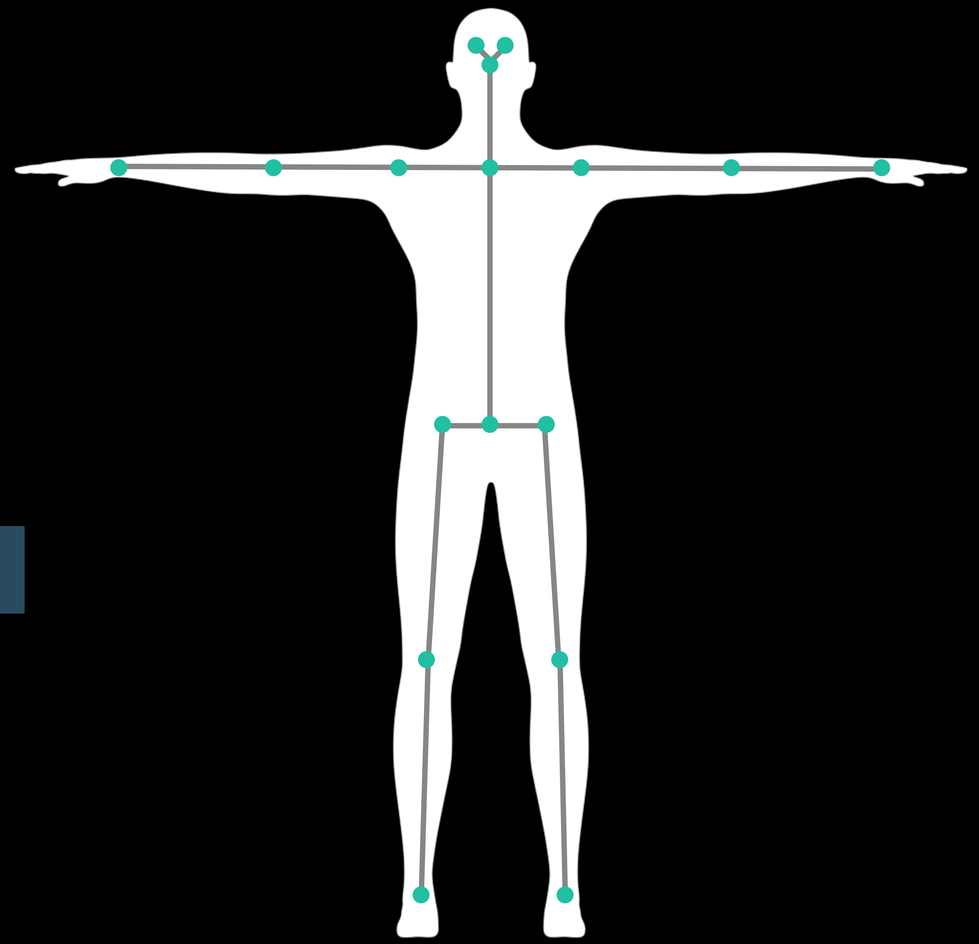
Extracting Data from 2D Skeleton



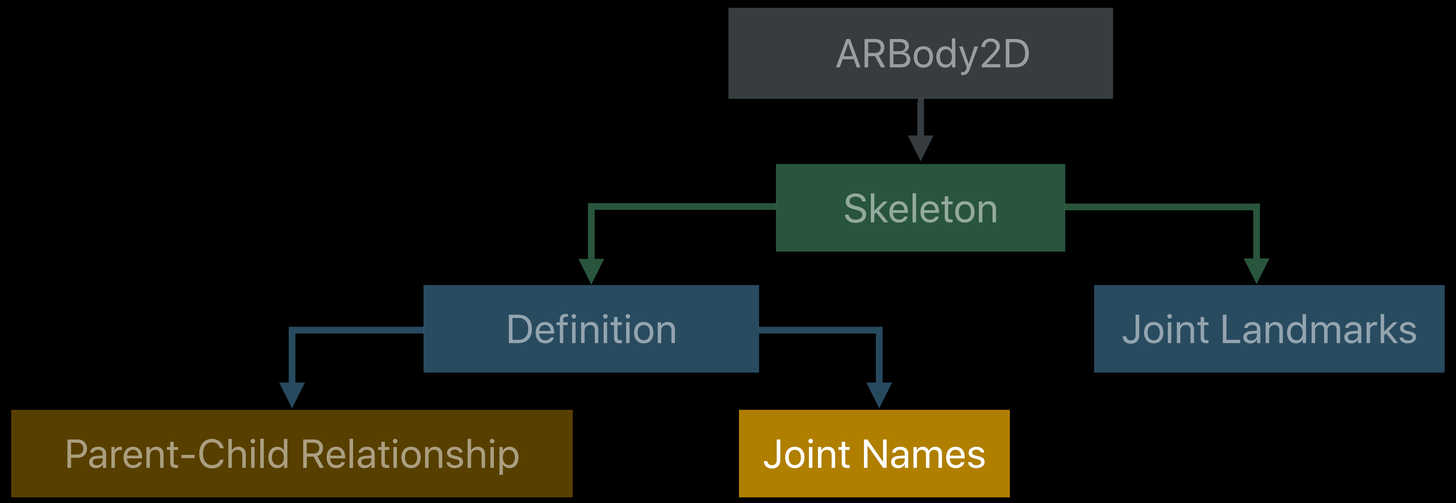


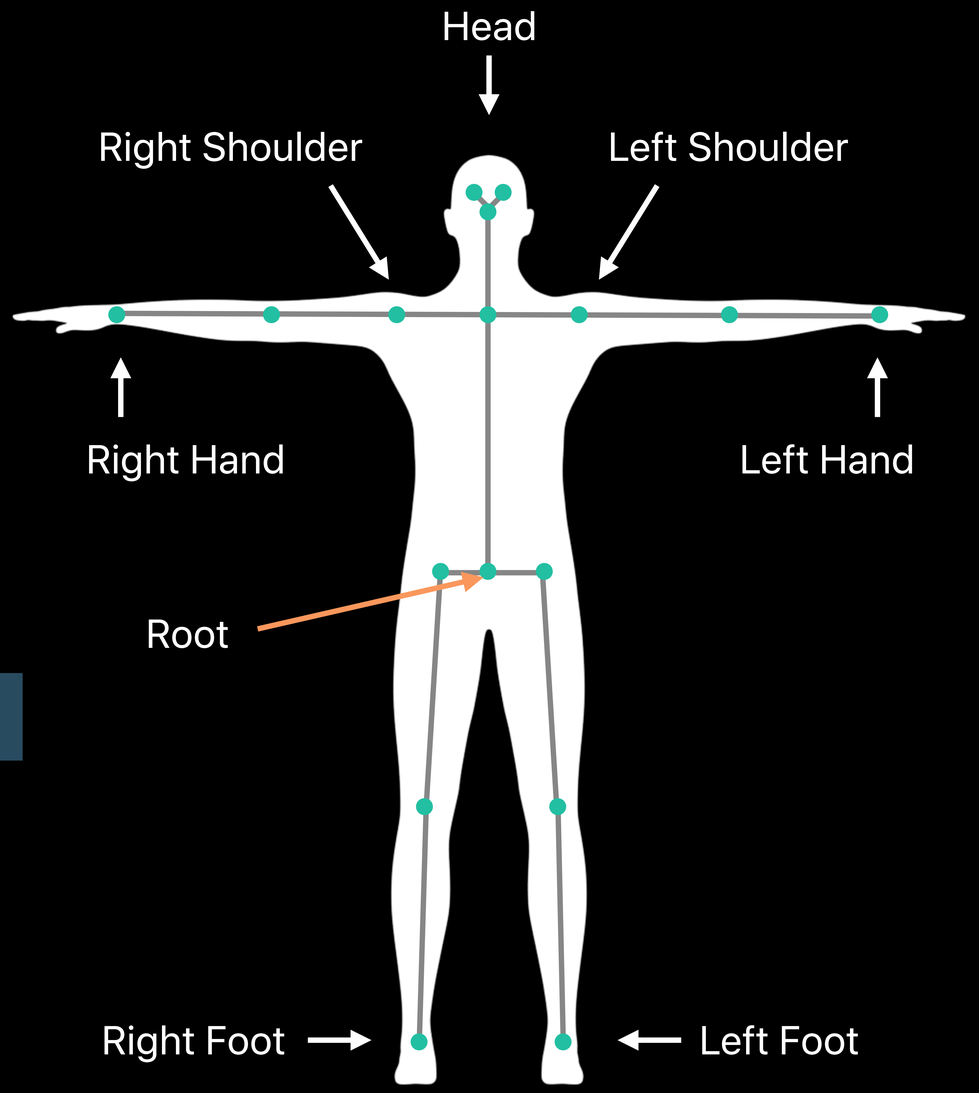
Extracting Data from 2D Skeleton



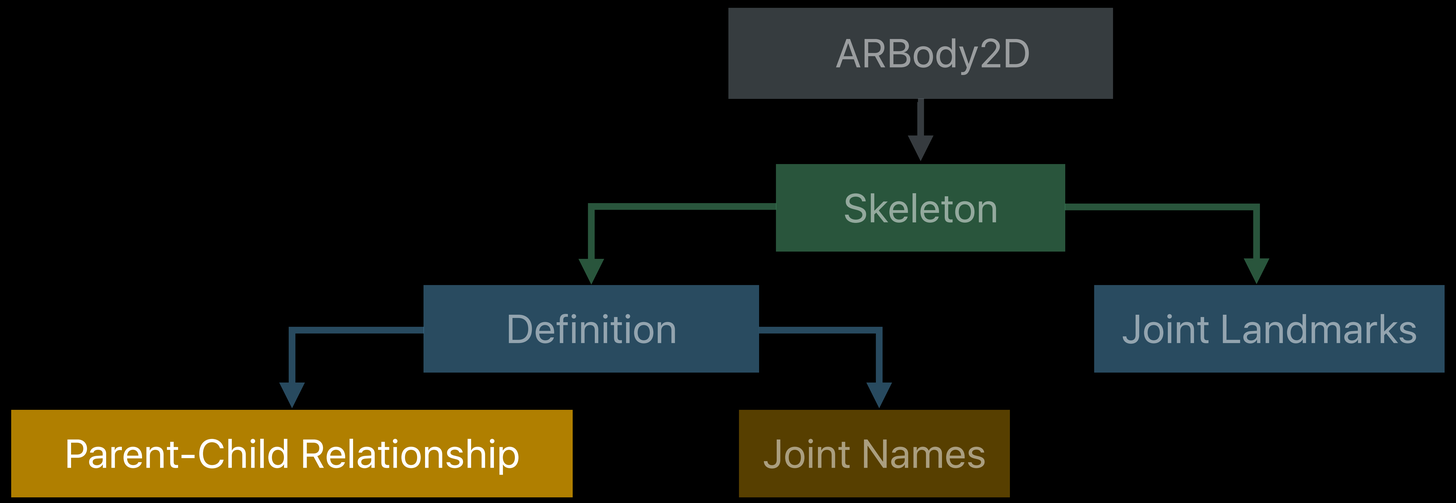


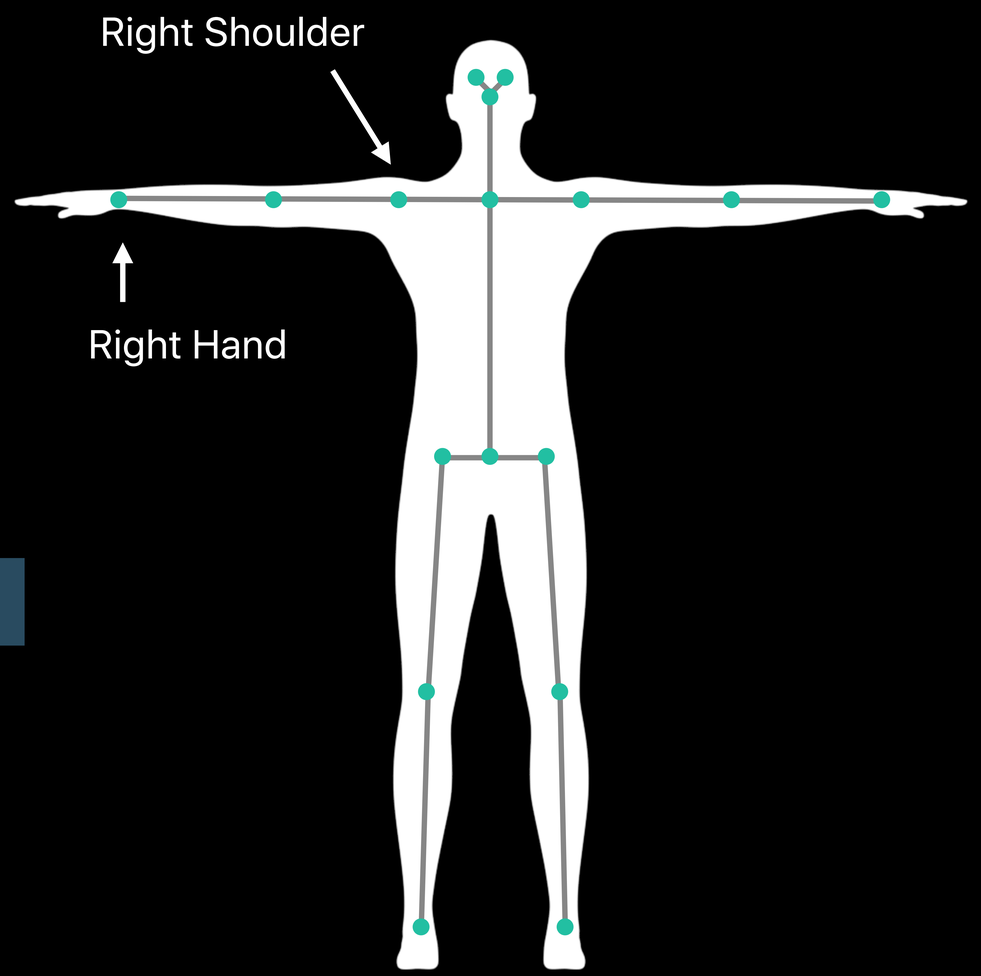
Extracting Data from 2D Skeleton



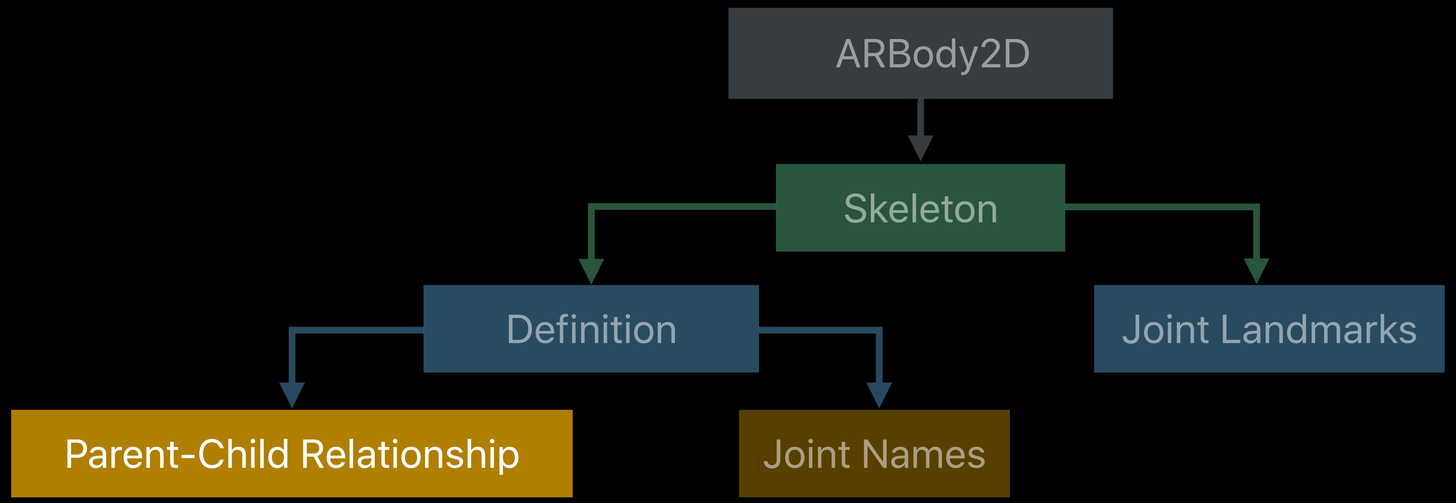


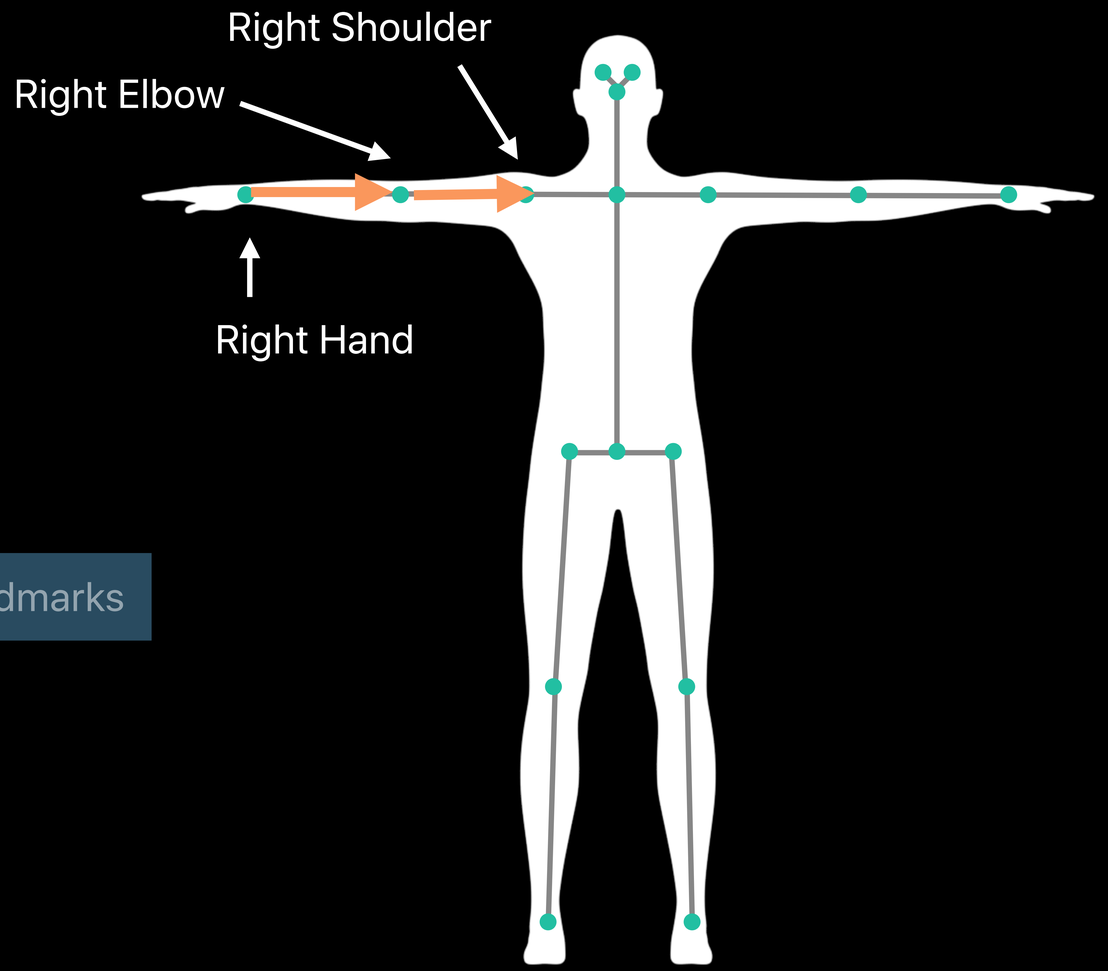
Extracting Data from 2D Skeleton



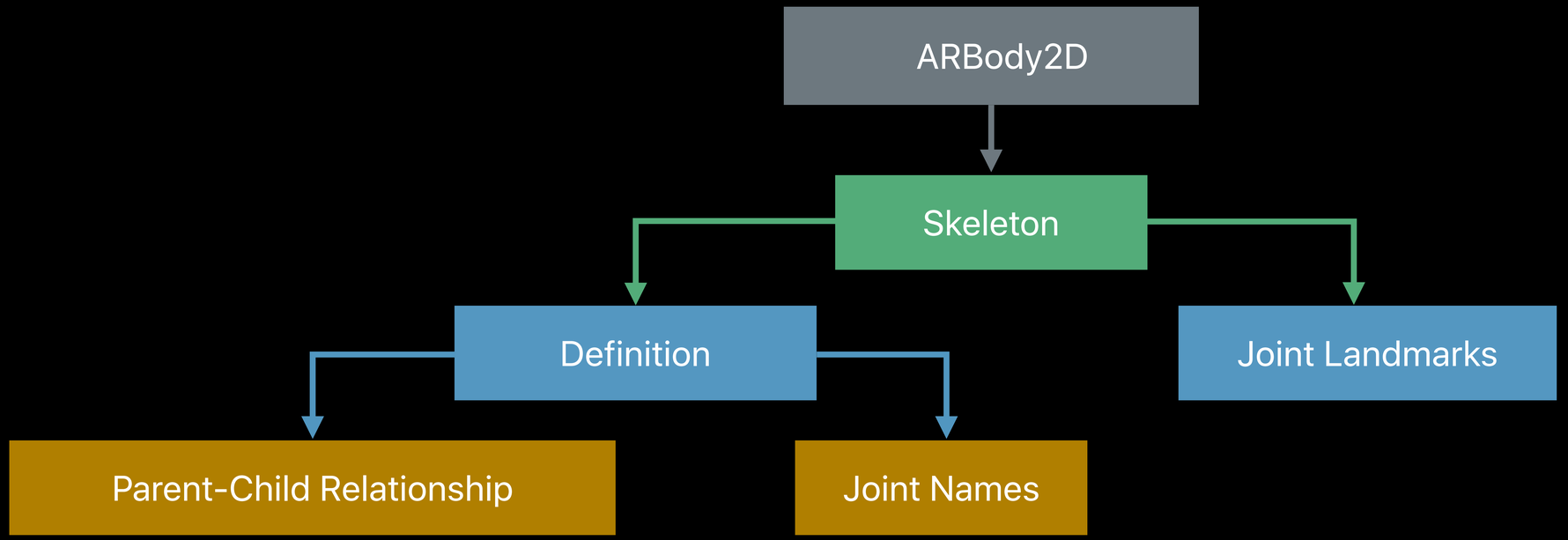


Extracting Data from 2D Skeleton



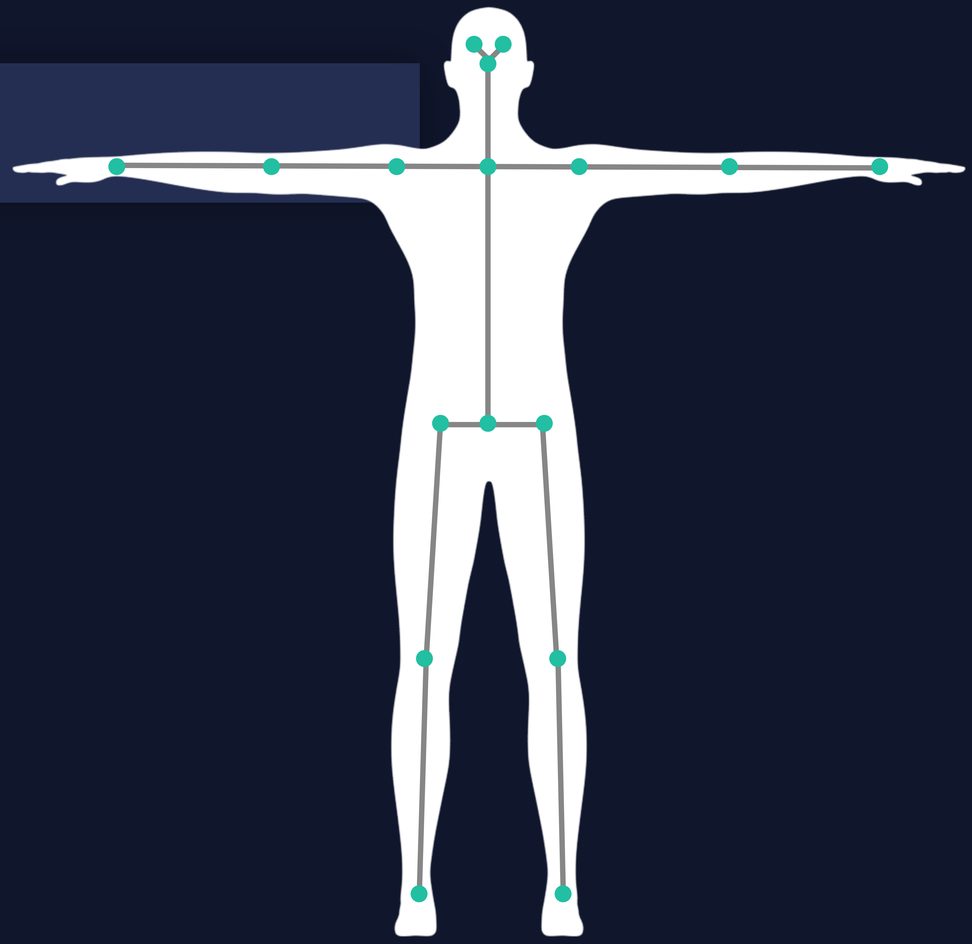


Extracting Data from 2D Skeleton

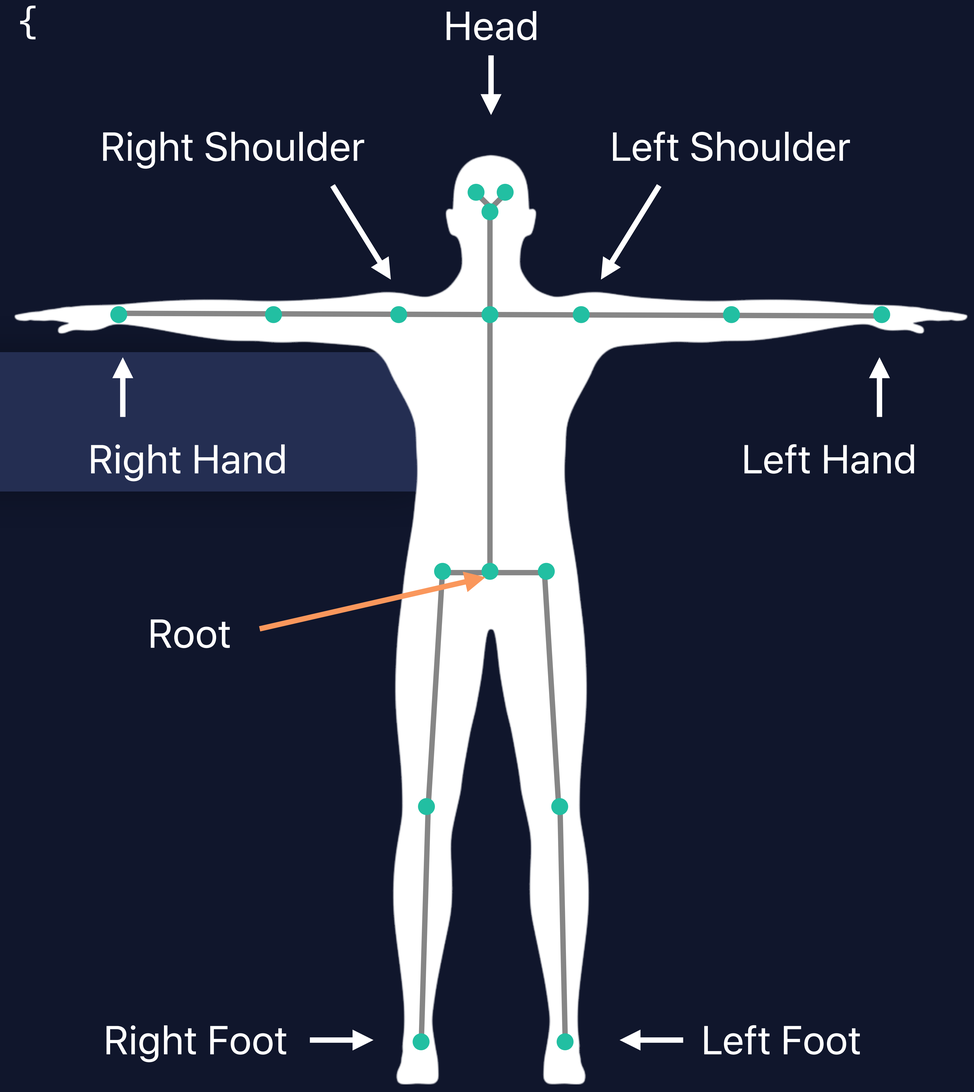


```
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```

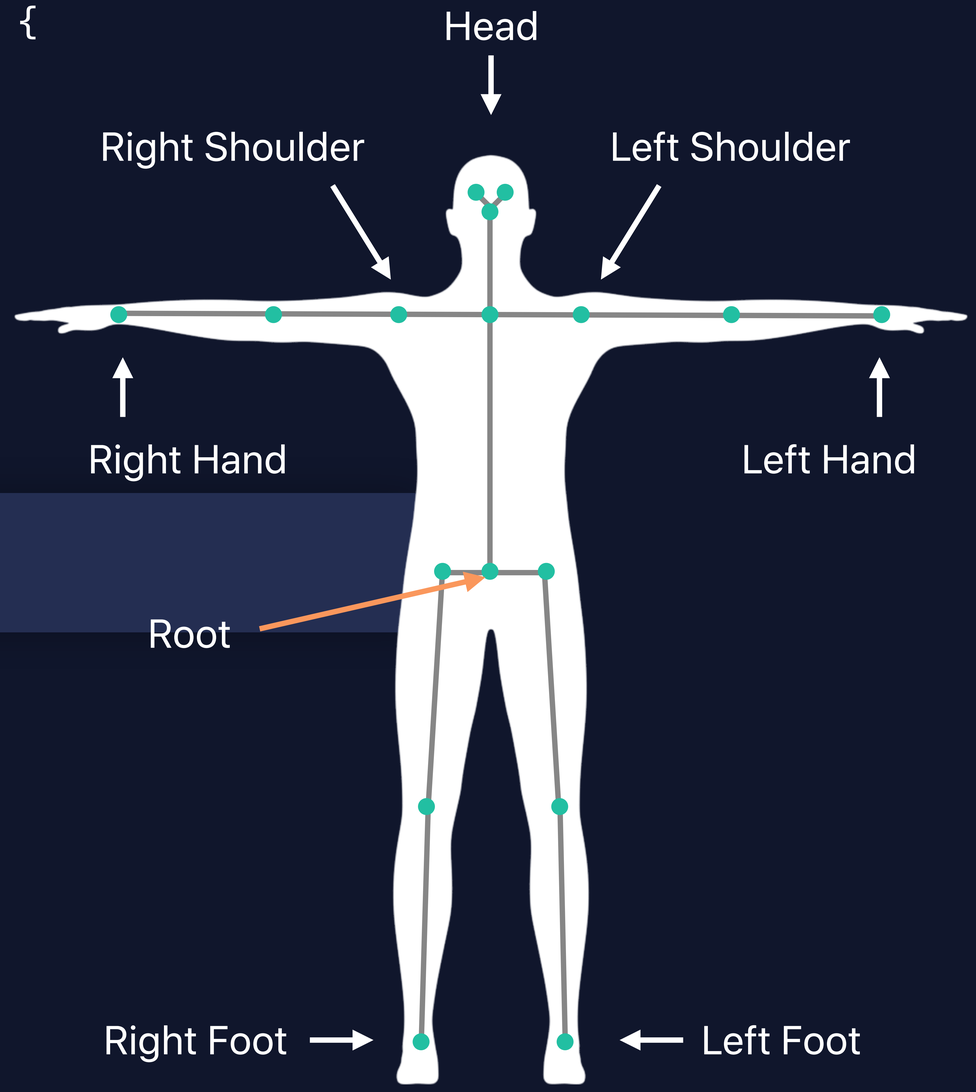
```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



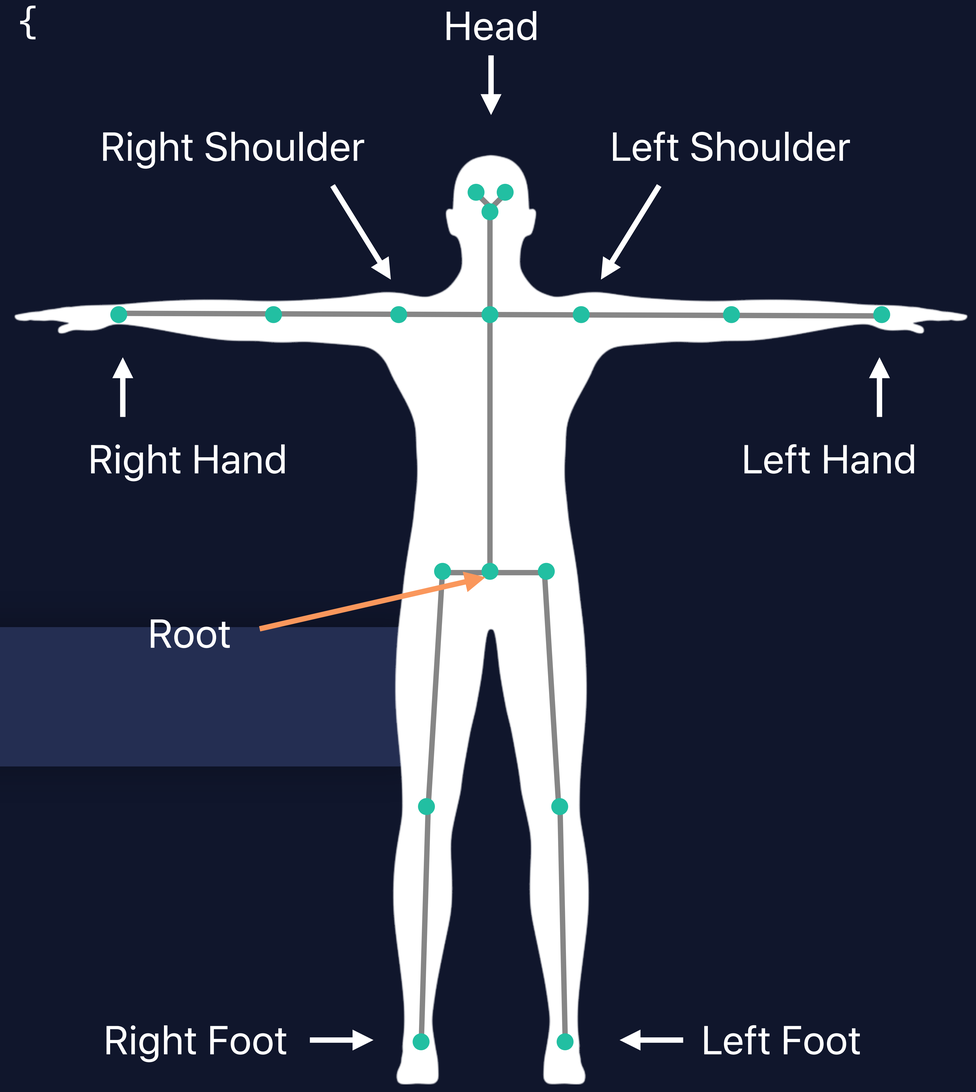
```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



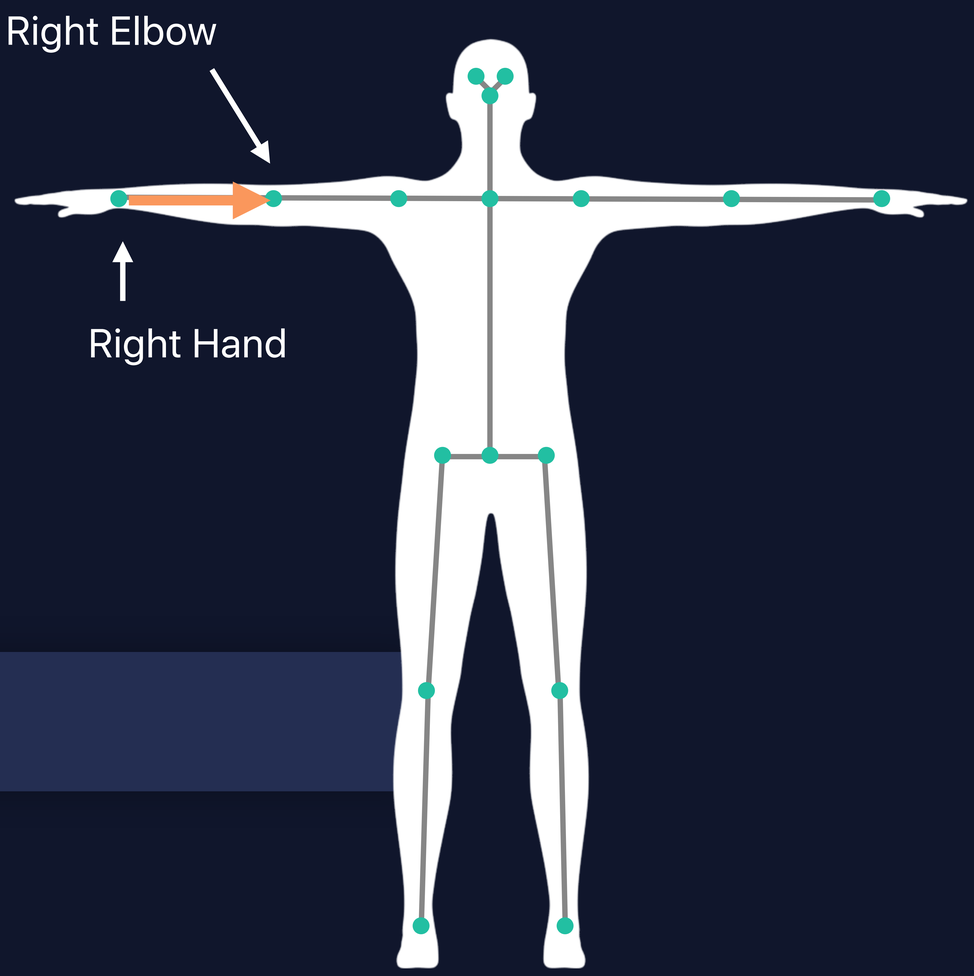
```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



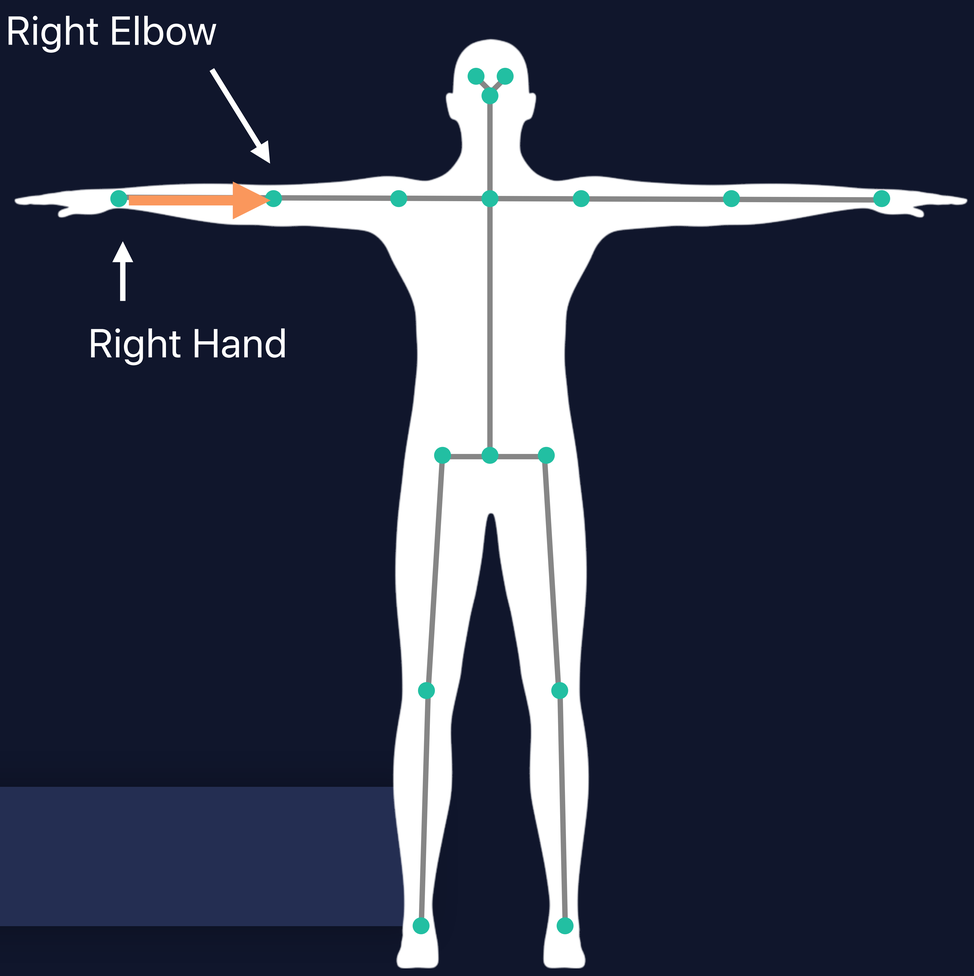
```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



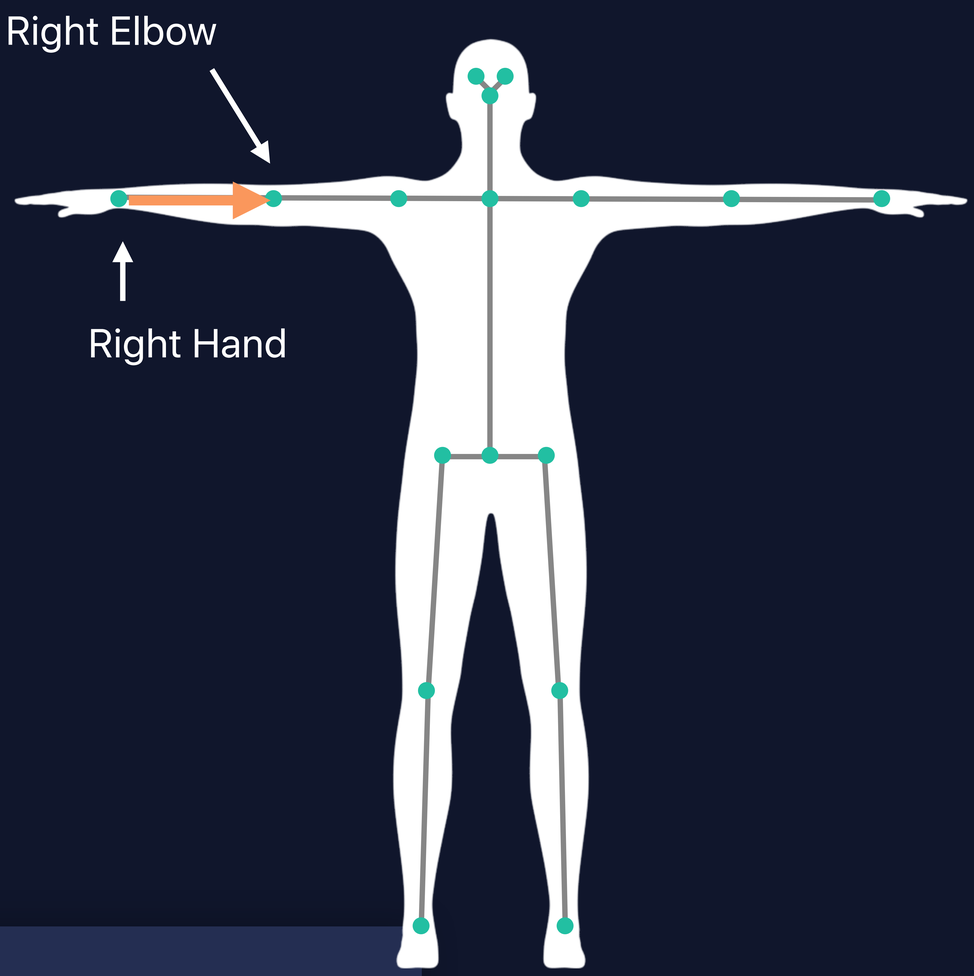
```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



```swift
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



```
func session(_ session ARSession, didUpdate frame: ARFrame) {

        // Accessing ARBody2D Object from ARFrame
        let person = frame.detectedBody

        // Use Skeleton Property to Access the Skeleton
        let skeleton2D = person.skeleton

        // Access Definition Object Containing Structure
        let definition = skeleton2D.definition

        // List of Joint Landmarks
        let jointLandmarks = skeleton2D.jointLandmarks

        // Iterate over All the Landmarks
        for (i, joint) in jointLandmarks.enumerated() {

            // Find Index of Parent
            let parentIndex = definition.parentIndices[i]

            // Check If It's Not the Root
            guard parentIndex != -1 else { continue }

            // Find Position of Parent Index
            let parentJoint = jointLandmarks [parentIndex.intValue]
            …
        }
}
```



Motion Capture in AR

Motion Capture in AR

Access to tracked person

Motion Capture in AR

Access to tracked person

Provides 3D and 2D skeleton

Motion Capture in AR

Access to tracked person

Provides 3D and 2D skeleton

Enables character animation

Motion Capture in AR

Access to tracked person

Provides 3D and 2D skeleton

Enables character animation

RealityKit API for quickly animating a character

Motion Capture in AR

Access to tracked person

Provides 3D and 2D skeleton

Enables character animation

RealityKit API for quickly animating a character

ARKit API for advanced use cases

Bringing People into AR

People Occlusion

Motion Capture

More Information

developer.apple.com/wwdc19/607

ARKit Lab	Thursday, 3:00
RealityKit and Reality Composer Lab	Thursday, 3:00


WWDC19