Bringing People into AR

Adrian Lindberg
Tanmay Batra
RealityKit
People Occlusion
Occlusion between people and rendered content
Overview
Overview
Overview
Overview
Overview

Camera
How It Works
How It Works

Segmentation | Depth | People
Machine Learning

Segmentation

Depth
var segmentationBuffer : CVPixelBuffer? { get }
var estimatedDepthData : CVPixelBuffer? { get }
How It Works

Segmentation  Depth  People
How It Works
How It Works
Matting

Desired Matted
Matting
Composition

RealityKit

SceneKit

Metal
Provides new ARView
RealityKit

Provides new ARView

Easy to use API for photorealism in AR
RealityKit

Provides new ARView

Easy to use API for photorealism in AR

Built-in support for People Occlusion
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
}
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

Added support to ARSCNView
SceneKit

Added support to ARSCNView

Just enable frame semantics
SceneKit

Added support to ARSCNView

Just enable frame semantics

Does composition as a post-process
SceneKit

Added support to ARSCNView

Just enable frame semantics

Does composition as a post-process

May not work well with transparency
Custom Composition
Custom Composition

Integrate into your own renderer

Metal
Custom Composition

Integrate into your own renderer

Complete control of composition
Custom Composition

Integrate into your own renderer

Complete control of composition

Provide simple access to required functionality
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()
}
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
Custom Composition

Matte and depth resolution mismatch
Can’t be solved by alpha only
Custom Composition

Matte and depth resolution mismatch

Can’t be solved by alpha only

Need to modify estimated depth buffer
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()}

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
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
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

Quickly animate characters
Motion Capture in RealityKit

Quickly animate characters

Simple and easy to use API
Motion Capture in RealityKit

- Quickly animate characters
- Simple and easy to use API
- Add your custom rigged characters
Motion Capture in RealityKit

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

ARBodyTrackingConfiguration()

ARView
Motion Capture in RealityKit

ARBodyTrackingConfiguration() → ARView
Motion Capture in RealityKit

ARBodyTrackingConfiguration() → ARView → [Icon]
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)
// Load Rigged Mesh and Tracked Person

```swift
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)
// 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)
3D Skeleton Joints
Extracting Data from Skeleton in 3D
Extracting Data from Skeleton

Detailed access to skeleton elements
Extracting Data from Skeleton

Detailed access to skeleton elements

Easy to use API
Extracting Data from Skeleton

Detailed access to skeleton elements

Easy to use API

Enable advanced use cases
Extracting Data from Skeleton

Detailed access to skeleton elements

Easy to use API

Enable advanced use cases

Provided via ARBodyAnchor
class ARBodyAnchor : ARAnchor

Transform

Skeleton

Definition

Joint Transforms

Parent-Child Relationship

Joint Names
class ARSkeleton

Transform

ARBodyAnchor

Skeleton

Definition

Parent-Child Relationship

Joint Names

Joint Transforms

class ARSkeleton
var transform: matrix_float4x4
var transform: matrix_float4x4
Transform

ARBodyAnchor

Transform

Skeleton

Definition

Joint Transforms

Parent-Child Relationship

Joint Names
Root / Hip Joint

Transform

ARBodyAnchor

Skeleton

Definition

Joint Transforms

Parent-Child Relationship

Joint Names
Root / Hip Joint

Transform

ARBodyAnchor

Skeleton

Definition

Joint Transforms

Parent-Child Relationship

Joint Names

Root / Hip Joint
Transform

Definition

Parent-Child Relationship

Joint Names

Skeleton

Joint Transforms

Right Hand
Accessing Joint Transforms

Relative to parent

```swift
func localTransform (for JointName: .rightHand)
```
Accessing Joint Transforms

Relative to parent

```python
func localTransform (for JointName: .rightHand)
```
Accessing Joint Transforms

Relative to parent

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

Relative to root

```swift
func modelTransform (for JointName: .rightHand)
```
Accessing Joint Transforms

Relative to parent

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

Relative to root

```swift
func modelTransform (for JointName: .rightHand)
```
Accessing Joint Transforms

Relative to parent

```go
func localTransforms
```

Relative to root

```go
func modelTransforms
```
Accessing Joint Transforms

Relative to parent

`func localTransforms`

Relative to root

`func modelTransforms`
// 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]
        ...
    }
}
// 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]
        ...
    }
}
for anchor in anchors {
    guard let bodyAnchor = anchor as? ARBodyAnchor else { return }
    let hipWorldPosition = bodyAnchor.transform
    let skeleton = bodyAnchor.skeleton
    let jointTransforms = skeleton.jointModelTransforms
    for (i, jointTransform) in jointTransforms.enumerated() {
        let parentIndex = skeleton.definition.parentIndices[i]
        guard parentIndex != -1 else { continue }
        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]
        ...
    }
}
// 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]
        ...
    }
}
// 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]
        ...
    }
}
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

Detailed access to 2D skeleton elements
Extracting Data from 2D Skeleton

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates
Extracting Data from 2D Skeleton

Detailed access to 2D skeleton elements

Skeleton joints provided as normalized image coordinates

Easy to use API
Extracting Data from 2D Skeleton

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

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

ARBody2D object

ARBody2D contains 2D skeletal structure
Extracting Data from 2D Skeleton
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
    - Parent-Child Relationship
  - Joint Landmarks
    - Joint Names
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
  - Joint Landmarks
    - Parent-Child Relationship
    - Joint Names
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    // Accessing ARBody2D Object from ARFrame
    let person = frame.detectedBody
}

Accessing ARBody2D Object
Accessing ARBody2D Object

ARFrame

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

ARBodyAnchor

```swift
guard let bodyAnchor = anchor as? ARBodyAnchor else {continue}
    // Accessing ARBody2D Object from referenceBody Property
    let body2D = bodyAnchor.referenceBody
```
Extracting Data from 2D Skeleton

ARBody2D

Skeleton

Definition

Joint Landmarks

Parent-Child Relationship

Joint Names
Extracting Data from 2D Skeleton

- ARBody2D
  - Skeleton
    - Definition
    - Joint Landmarks
      - Parent-Child Relationship
      - Joint Names
Extracting Data from 2D Skeleton

- ARBody2D
  - Skeleton
    - Definition
      - Parent-Child Relationship
    - Joint Landmarks
      - Joint Names
Extracting Data from 2D Skeleton
Extracting Data from 2D Skeleton

- **ARBody2D**
  - **Skeleton**
    - **Definition**
    - **Joint Landmarks**
    - **Parent-Child Relationship**
    - **Joint Names**
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
- Definition
  - Parent-Child Relationship
  - Joint Names
- Joint Landmarks

Landmarks
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
    - Parent-Child Relationship
    - Joint Names
  - Joint Landmarks
- Landmarks
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
  - Joint Landmarks
    - Parent-Child Relationship
    - Joint Names

Diagram showing a human figure with joint landmarks.
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
  - Joint Landmarks
  - Parent-Child Relationship
  - Joint Names

Joint Names:
- Right Shoulder
- Left Shoulder
- Right Hand
- Left Hand
- Right Foot
- Left Foot

Joint Landmarks:
- Head
- Right Shoulder
- Left Shoulder
- Right Hand
- Left Hand
- Root
- Right Foot
- Left Foot
Extracting Data from 2D Skeleton

- ARBody2D
  - Skeleton
    - Definition
    - Joint Landmarks
      - Parent-Child Relationship
      - Joint Names

- Right Shoulder
- Right Hand
Extracting Data from 2D Skeleton

- ARBody2D
- Skeleton
  - Definition
  - Joint Landmarks
  - Parent-Child Relationship
  - Joint Names

- Right Shoulder
- Right Elbow
- Right Hand
Extracting Data from 2D Skeleton

- ARBody2D
  - Skeleton
    - Definition
    - Joint Landmarks
      - Parent-Child Relationship
      - Joint Names
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[proxy.Int(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]

        ...
    }
}
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ARKit Lab</td>
<td>Thursday, 3:00</td>
</tr>
<tr>
<td>RealityKit and Reality Composer Lab</td>
<td>Thursday, 3:00</td>
</tr>
</tbody>
</table>