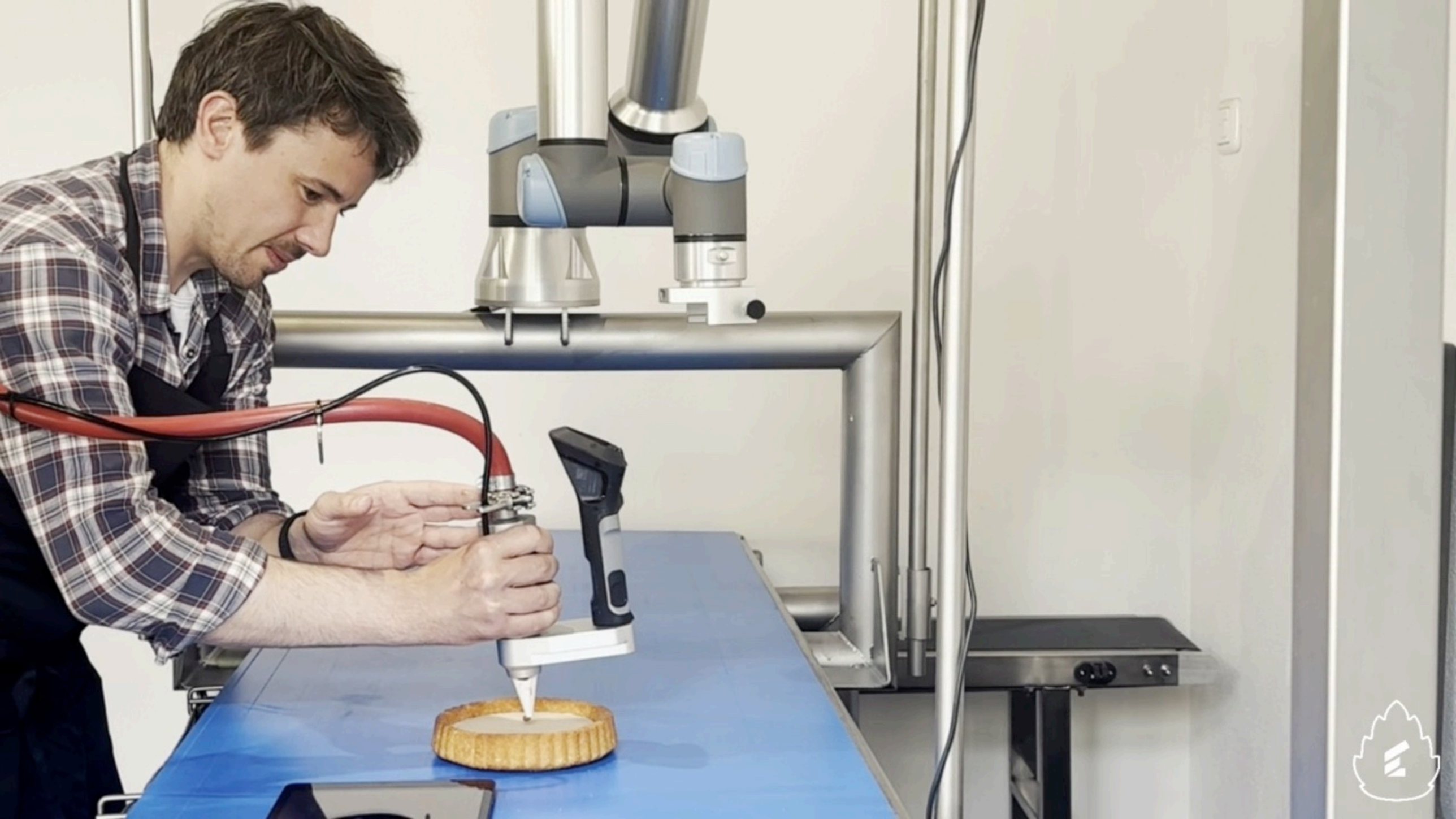
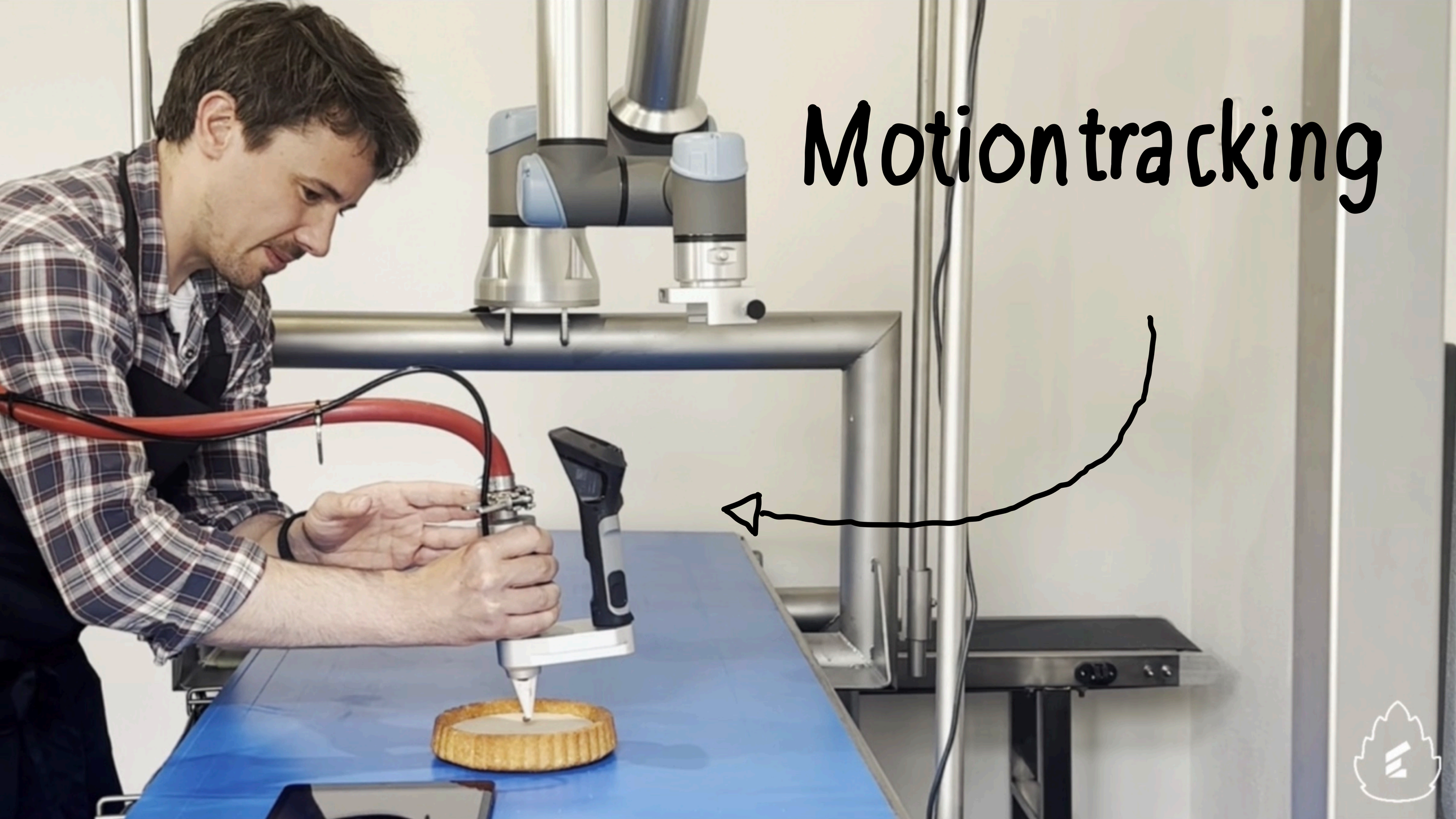
Declarative 3D

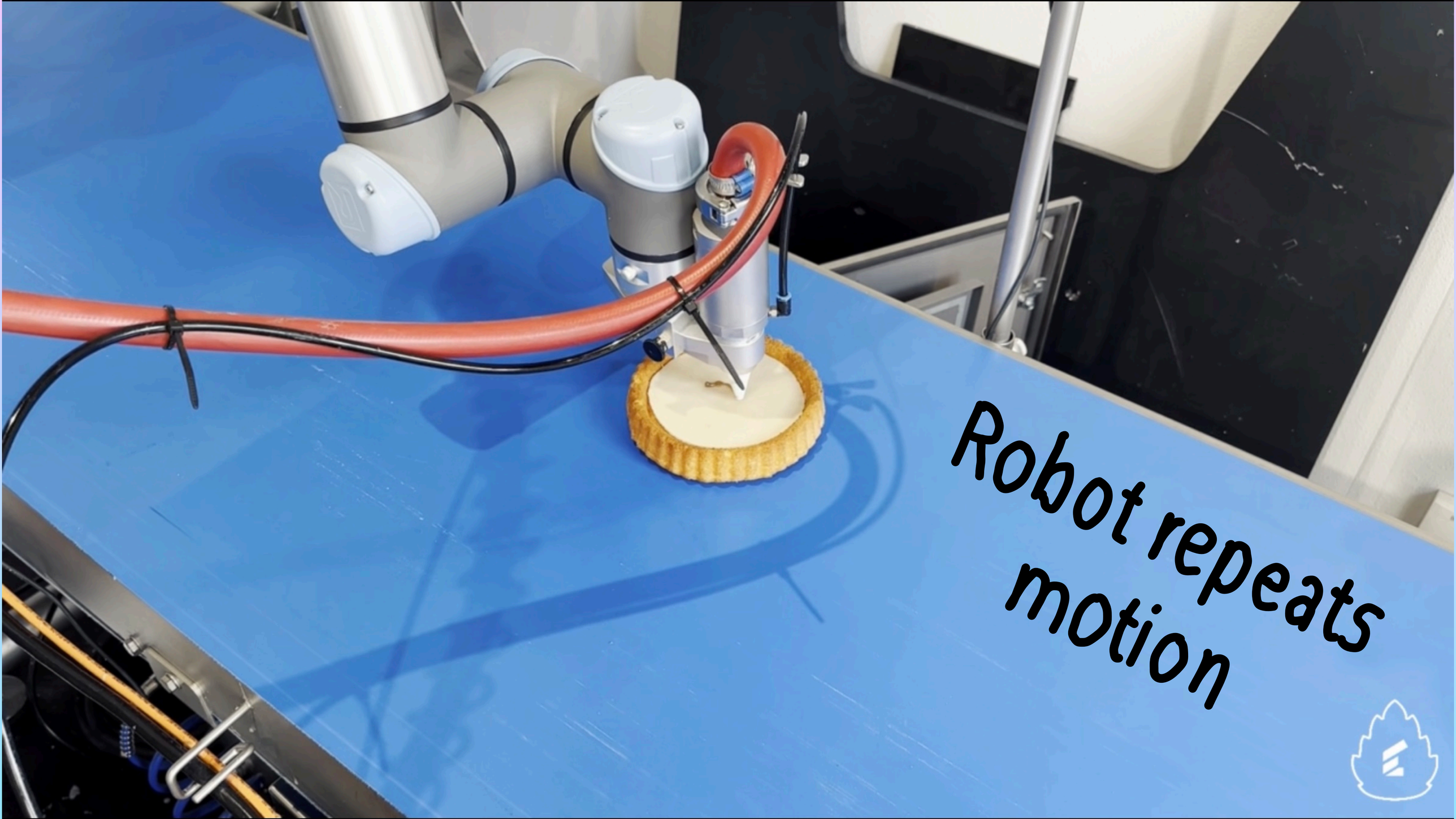
Achieve seemless integration
between SwiftUI and 3D


Hi,
I'm Dominik




Motion tracking


iPad guides through the process
Press button on TracePen to stop
Create
Help


Robot repeats
motion

11:53 Mon 4. Sep

92 %

Back

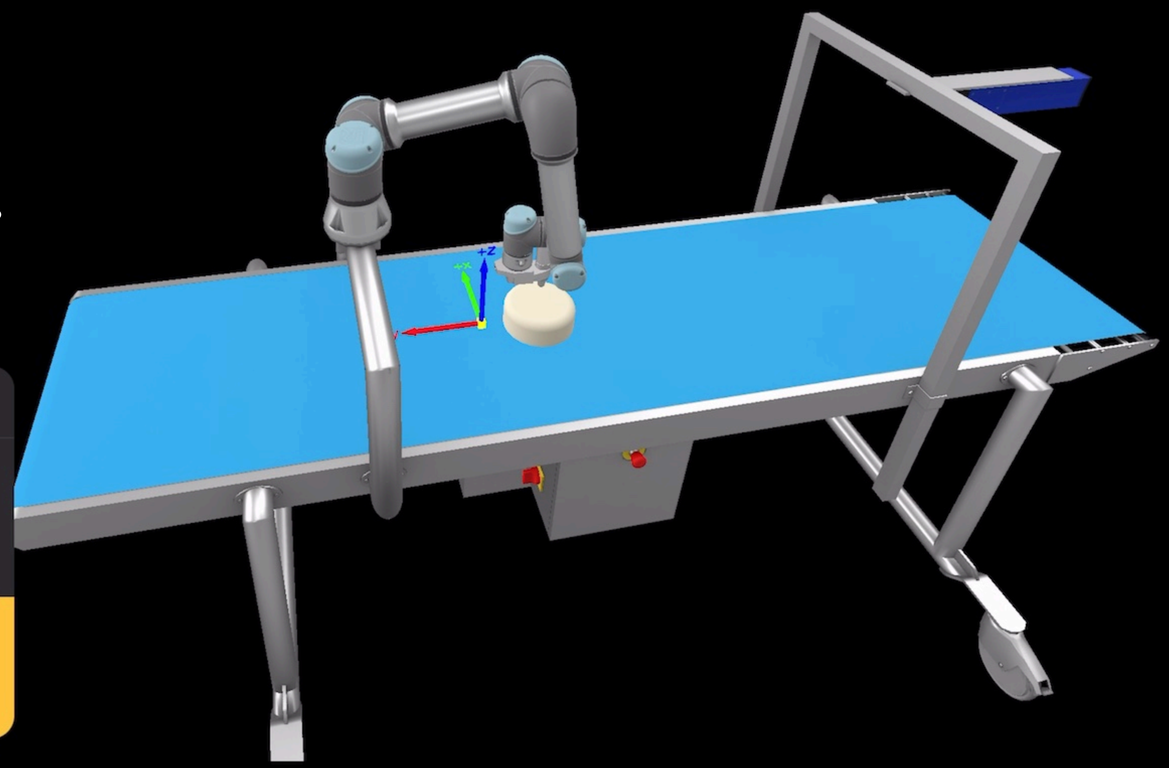
Flower

Finish

Conveyor Speed

~ 0 m/min

3D Visualisation



Hints

Dispensing

Conveyor

Stop

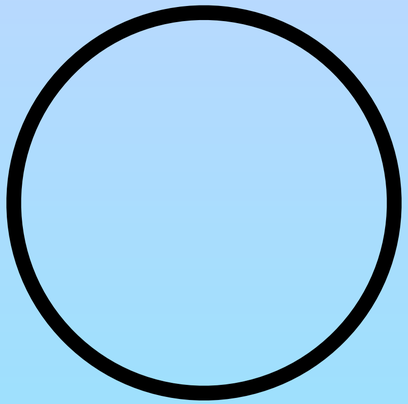
Production

Create & Edit

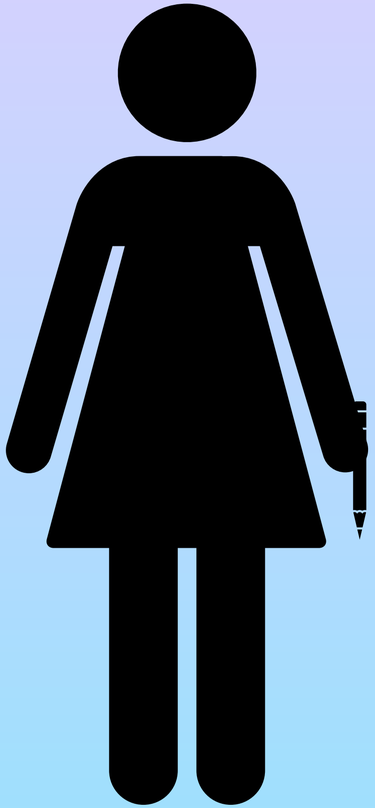
Help

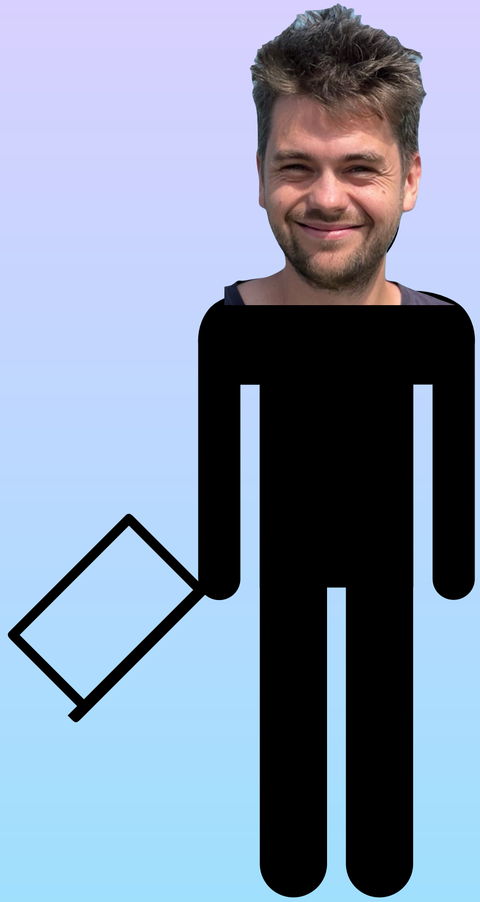
Settings

We started simple and

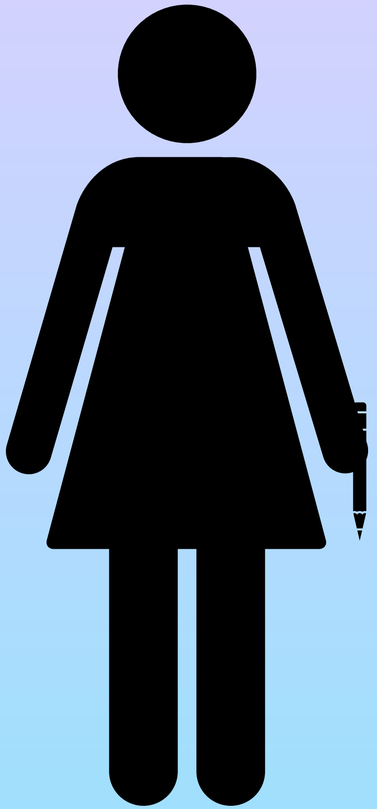


turned it quickly
into a mess

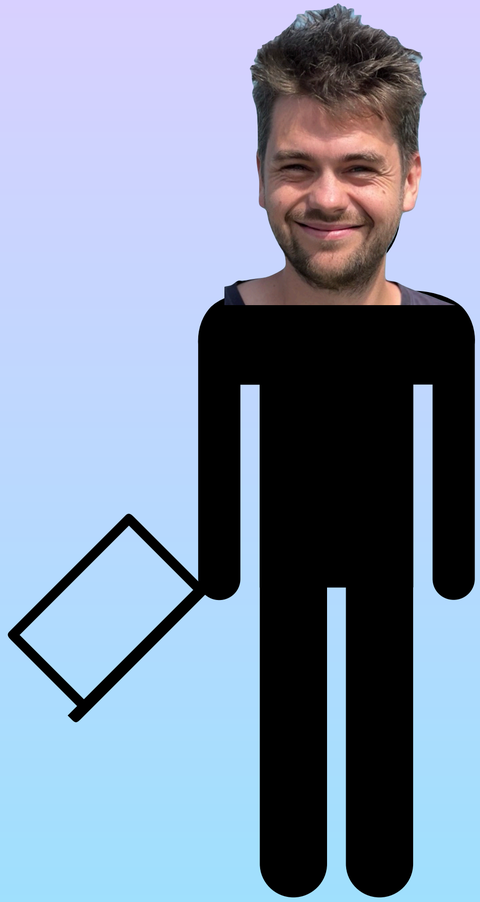




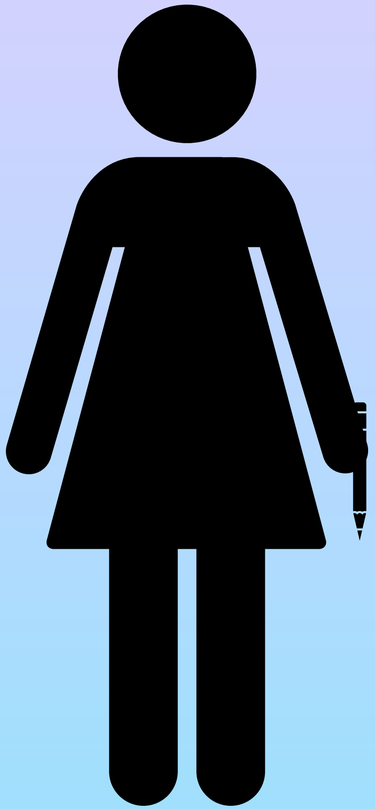
Product Owner



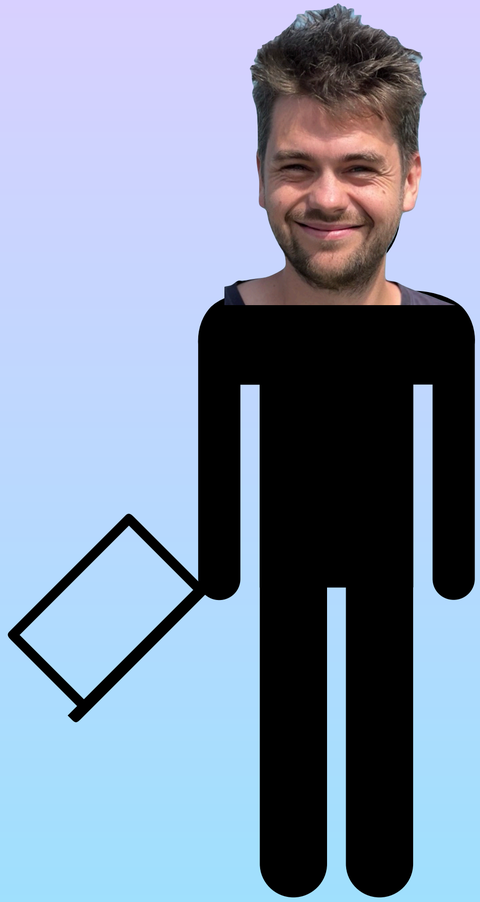
Mastermind Engineer



PO



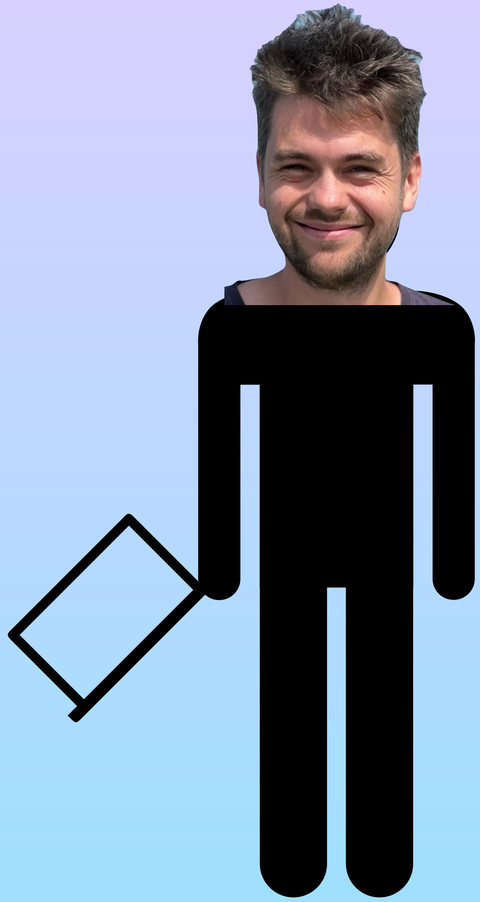
ME

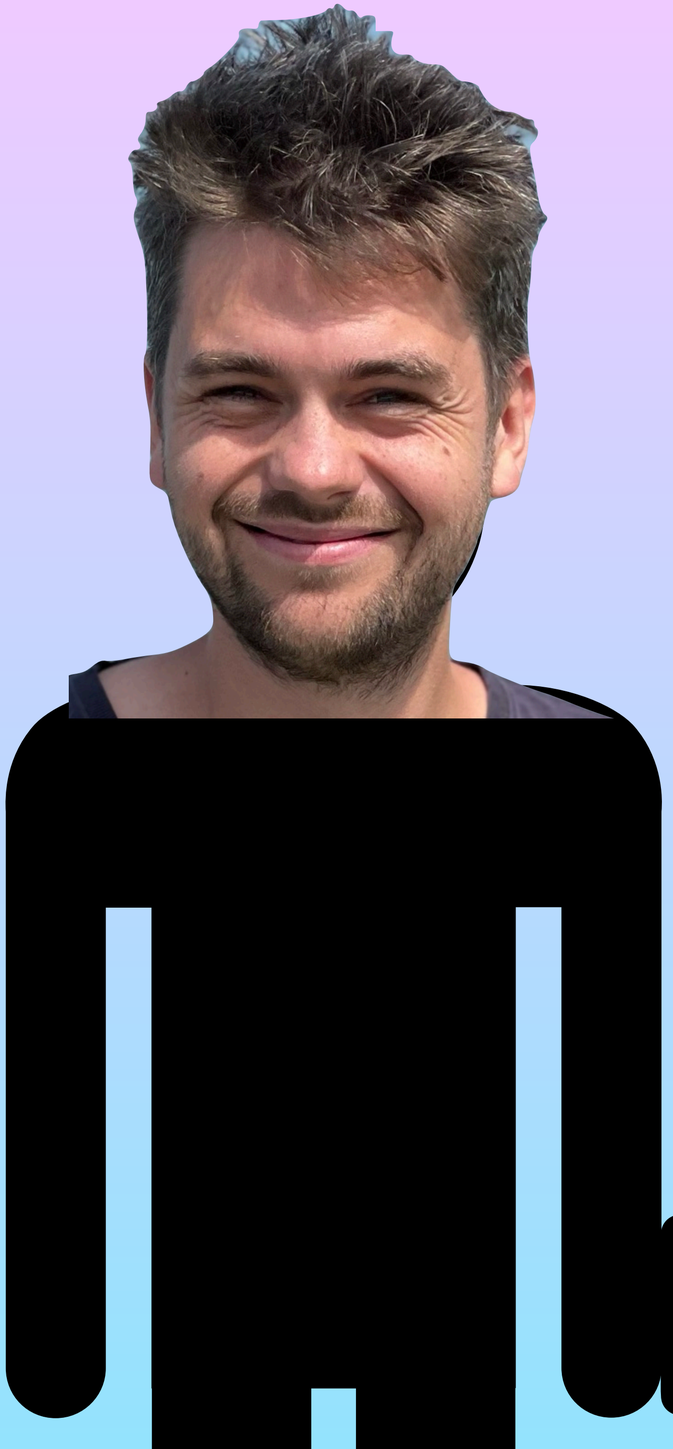


PO

User needs xy

ME





```
struct Scene3D: View {

    static var shared = Scene3D()

    func setup() {}
    func updateRobot() {}
}
```

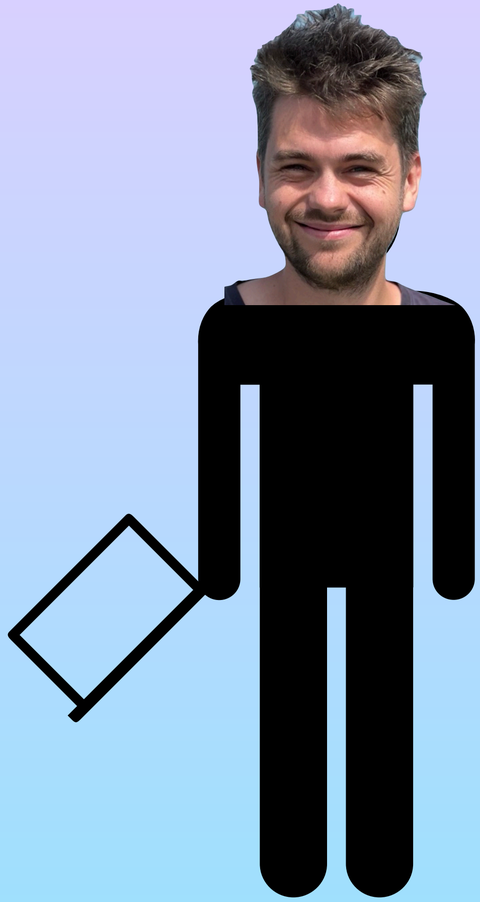
func xyC)

A FEW MOMENTS LATER

PO

User needs AB

ME



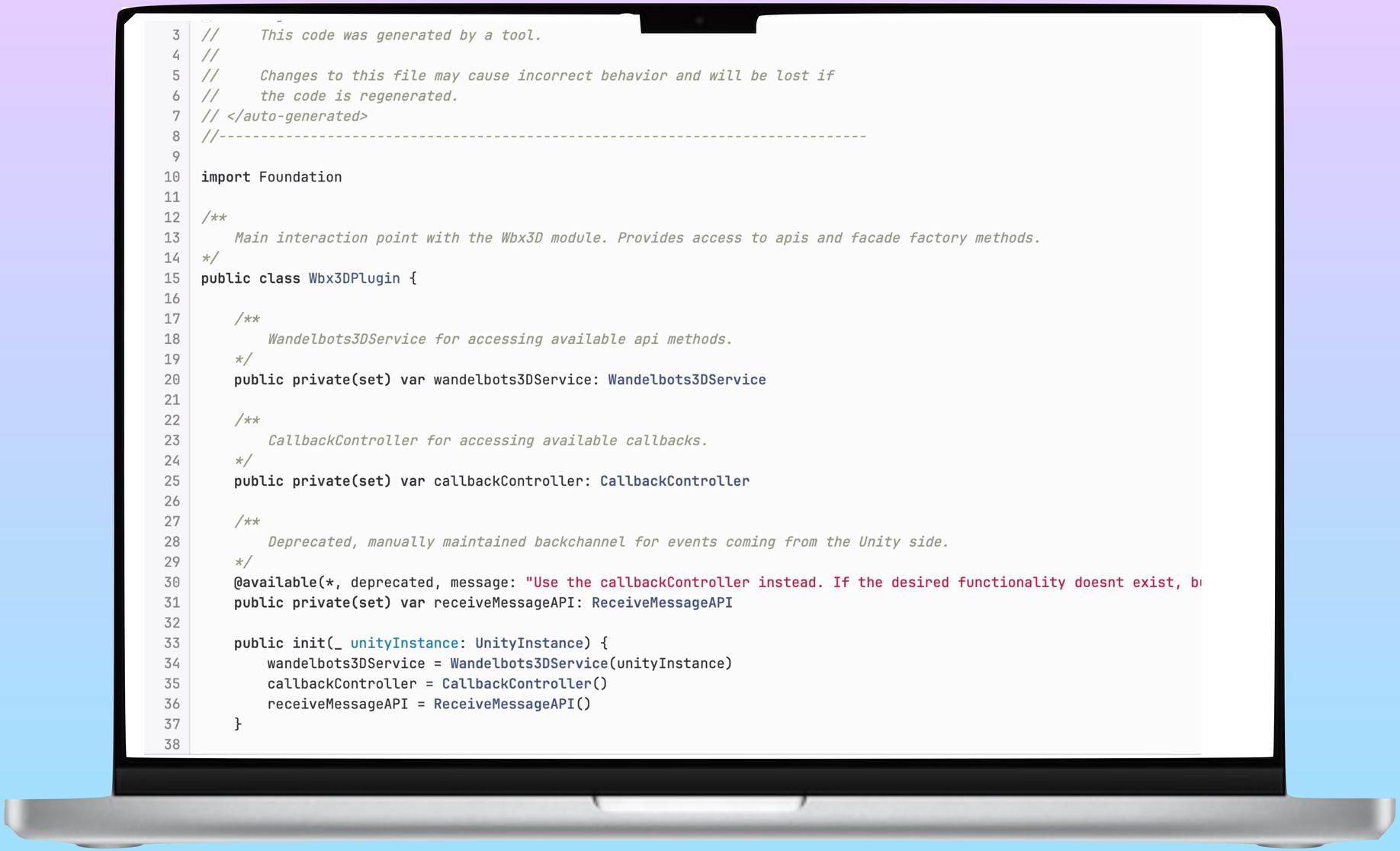
```
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

import Foundation

/**
    Main interaction point with the Wbx3D module. Provides access to apis and facade factory methods.
*/
public class Wbx3DPlugin {

    /**
        Wandelbots3DService for accessing available api methods.
    */
    public private(set) var wandelbots3DService: Wandelbots3DService

    /**
        CallbackController for accessing available callbacks.
    */
    public private(set) var callbackController: CallbackController

    /**
        Deprecated, manually maintained backchannel for events coming from the Unity side.
    */
    @available(*, deprecated, message: "Use the callbackController instead. If the desired functionality doesnt exist, b
    public private(set) var receiveMessageAPI: ReceiveMessageAPI

    public init(_ unityInstance: UnityInstance) {
        wandelbots3DService = Wandelbots3DService(unityInstance)
        callbackController = CallbackController()
        receiveMessageAPI = ReceiveMessageAPI()
    }
```

func ABC()

ME

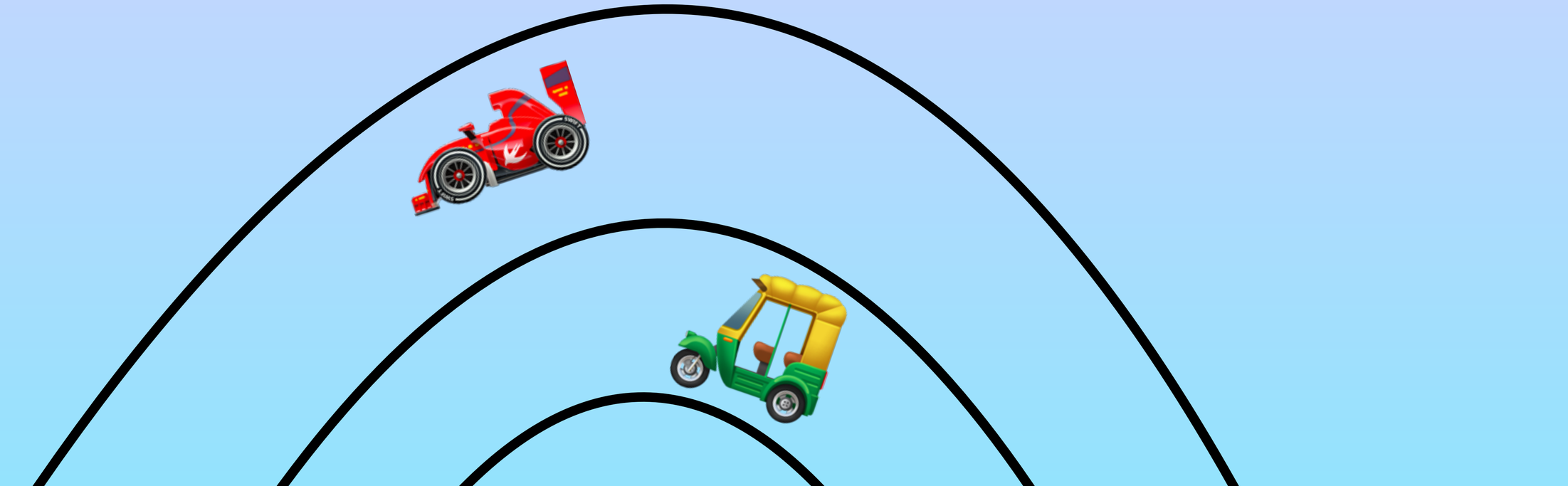
~~Mastermind~~
~~Engineer~~

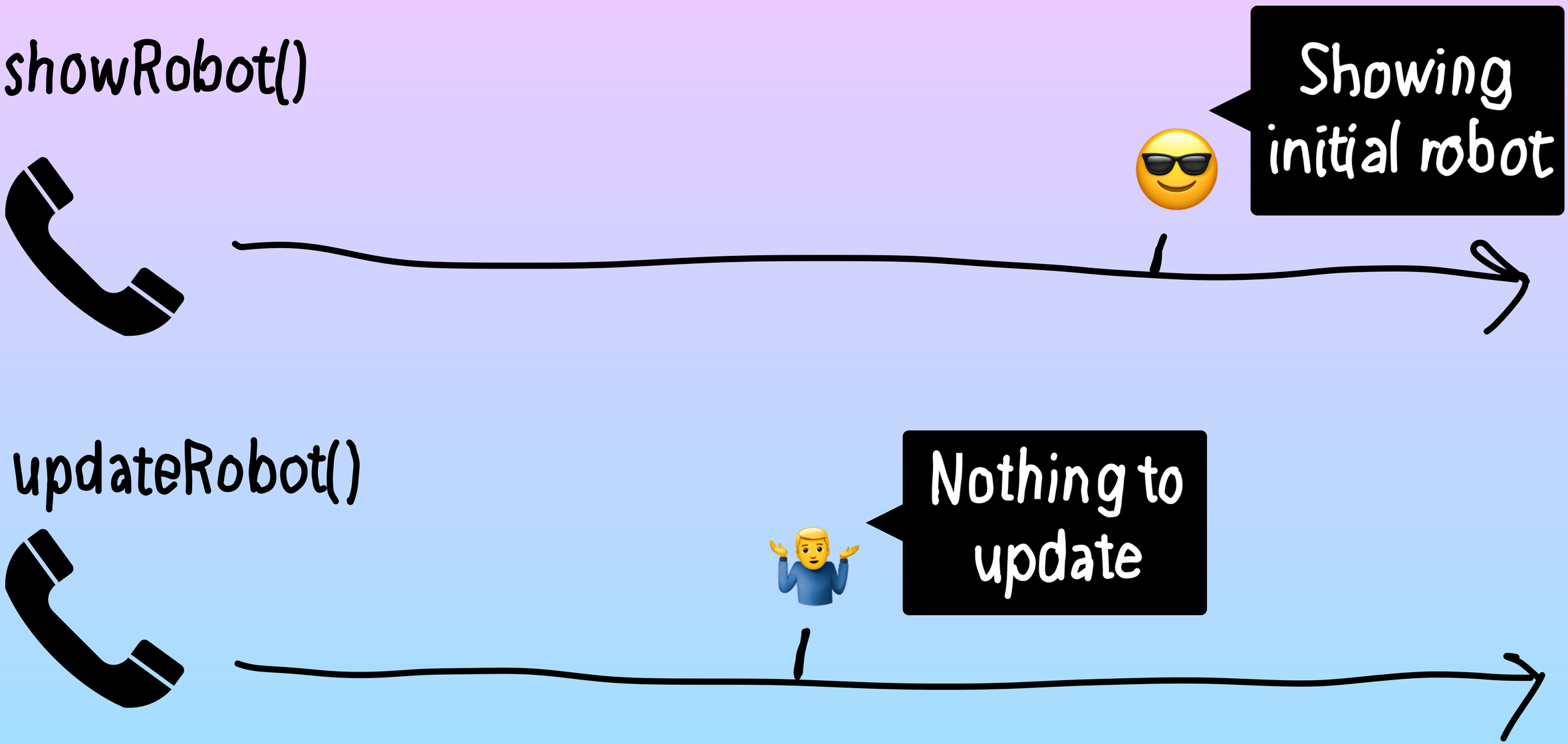
Mediocre
Engineer



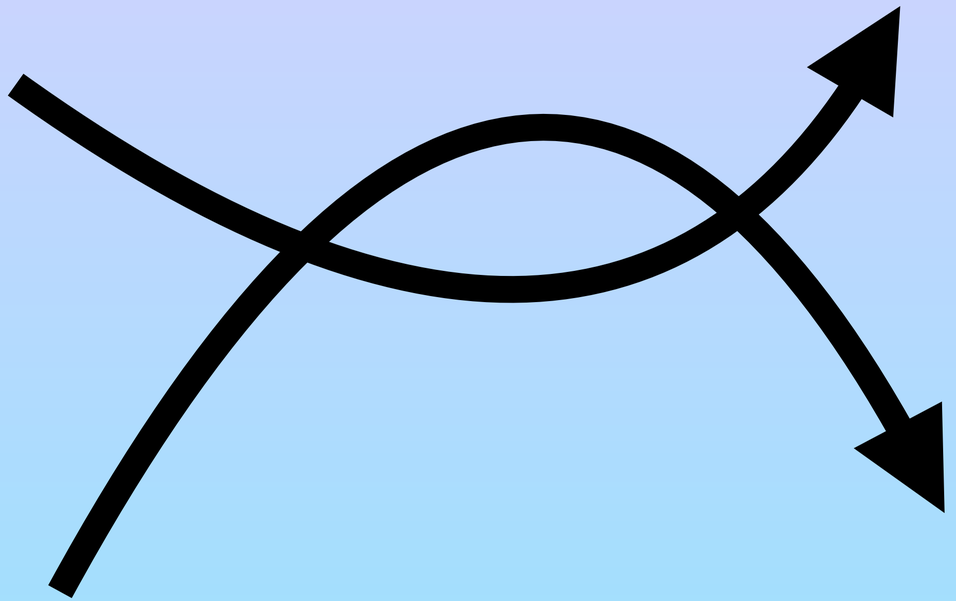
```
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

import Foundation

/**
    Main interaction point with the Wbx3D module. Provides access to apis and facade factory methods.
*/
public class Wbx3DPlugin {

    /**
        Wandelbots3DService for accessing available api methods.
    */
    public private(set) var wandelbots3DService: Wandelbots3DService

    /**
        CallbackController for accessing available callbacks.
    */
    public private(set) var callbackController: CallbackController

    /**
        Deprecated, manually maintained backchannel for events coming from the Unity side.
    */
    @available(*, deprecated, message: "Use the callbackController instead. If the desired functionality doesnt exist, b
    public private(set) var receiveMessageAPI: ReceiveMessageAPI

    public init(_ unityInstance: UnityInstance) {
        wandelbots3DService = Wandelbots3DService(unityInstance)
        callbackController = CallbackController()
        receiveMessageAPI = ReceiveMessageAPI()
    }

```

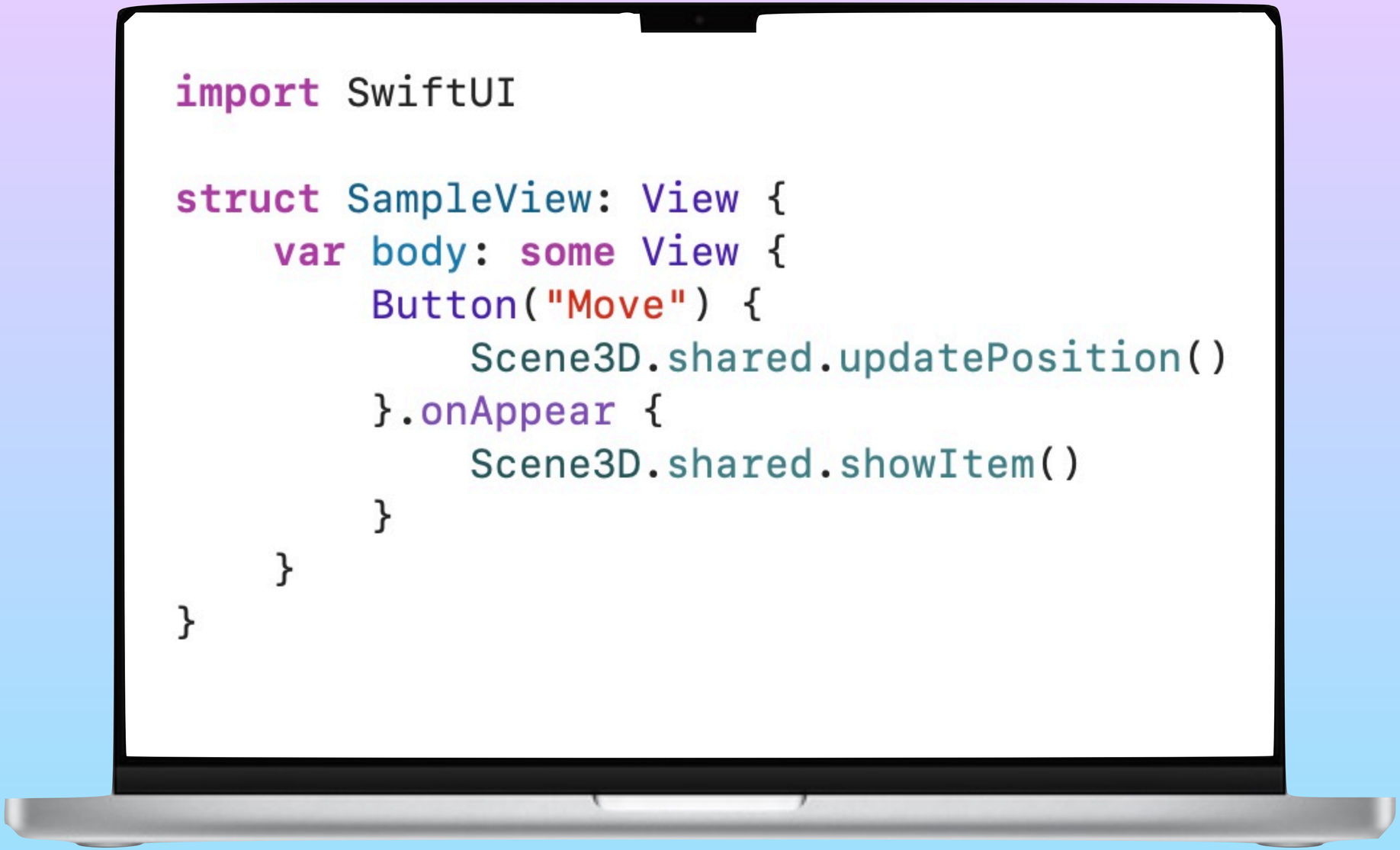
Race conditions

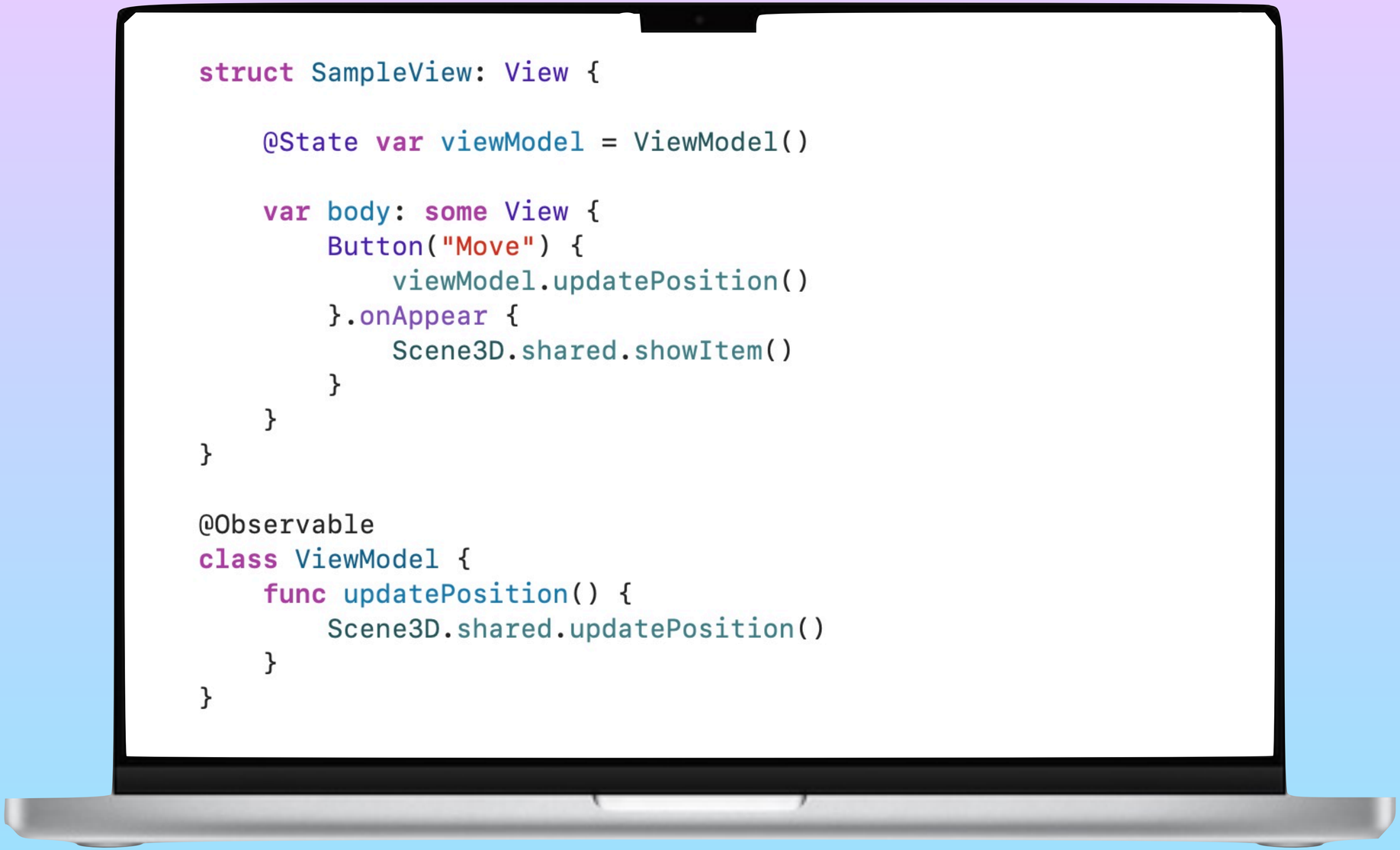



showRobot()
Showing initial robot
updateRobot()
Nothing to update

No consistency




import SwiftUI
struct SampleView: View {
var body: some View {
Button("Move") {
Scene3D.shared.updatePosition()
}.onAppear {
Scene3D.shared.showItem()
}
}
}

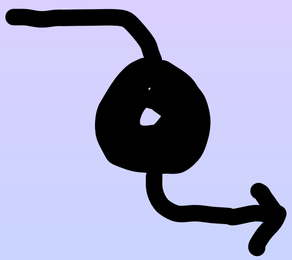

struct SampleView: View {
@State var viewModel = ViewModel()
var body: some View {
Button("Move") {
viewModel.updatePosition()
}.onAppear {
Scene3D.shared.showItem()
}
}
}
@Observable
class ViewModel {
func updatePosition() {
Scene3D.shared.updatePosition()
}
}

Imperative

vs

Declaritive

UIKit

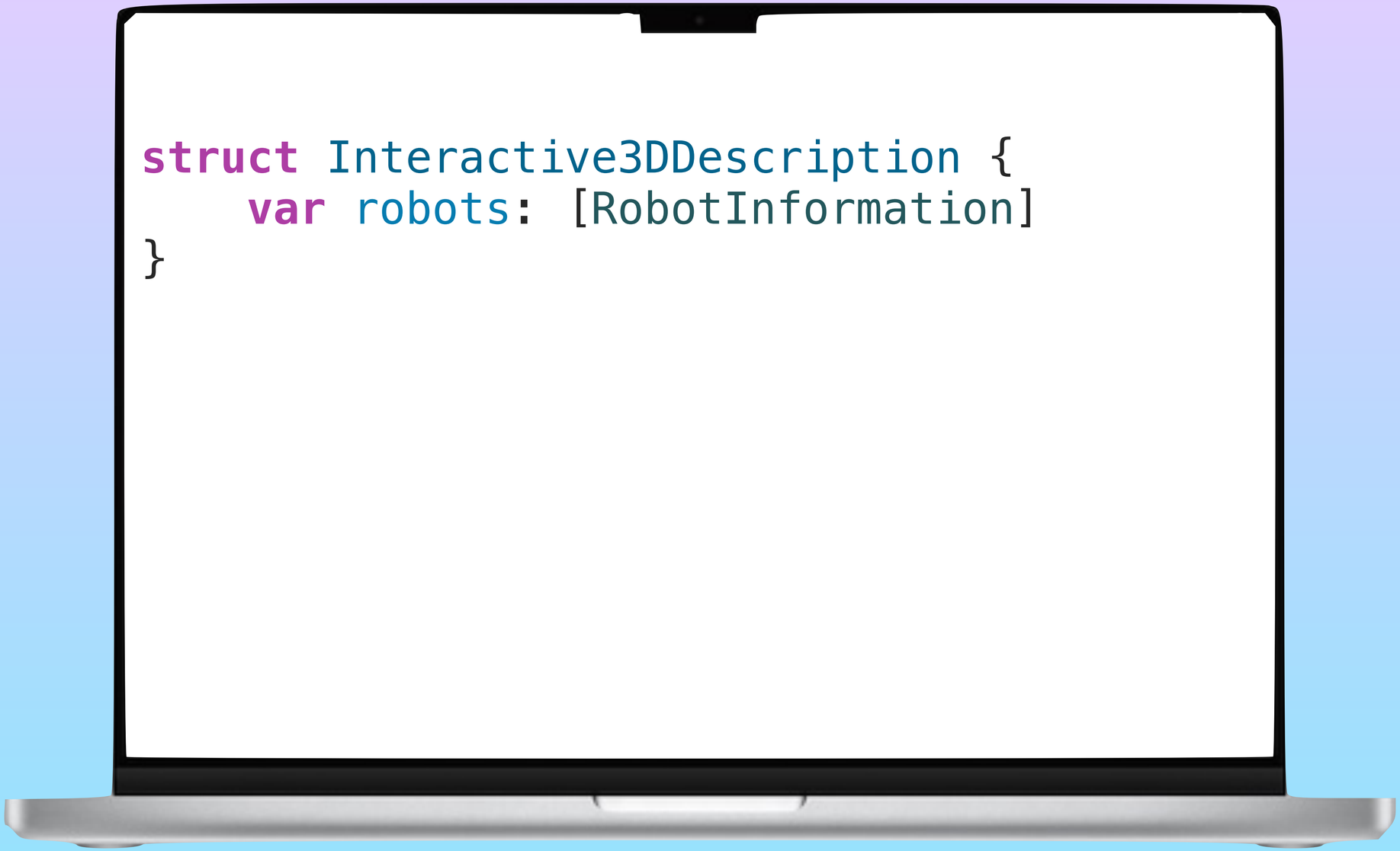


```
button.text = model.prompt
```

SwiftUI

```
Button {

} label: {
    Text(model.prompt)
}
```

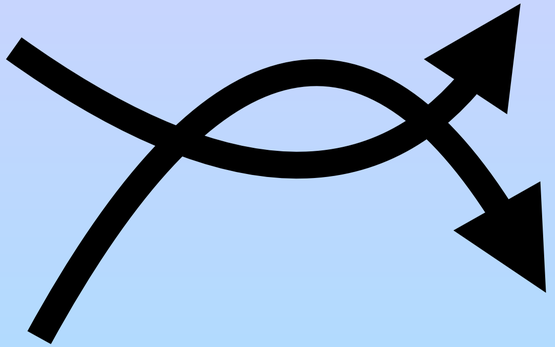
State object



State object

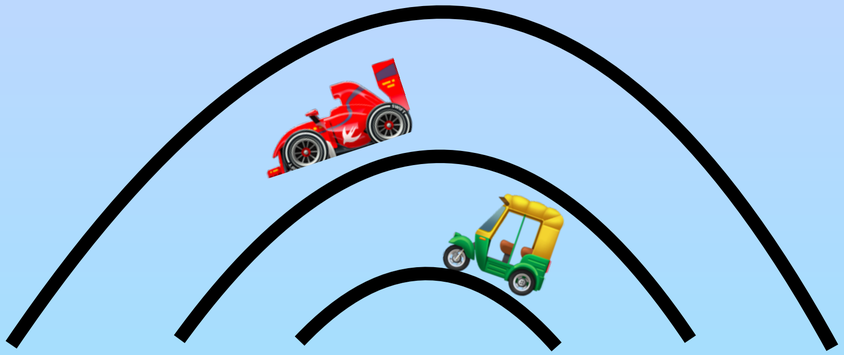
```
struct ContentView: View {

    var body: some View {
        SceneKit3DView(.init(
            robots: [
                .init(type: .ur3, joints: [0,-1.2,1.7-1.9,1.5,0]),
                .init(type: .ur5e, joints: [0,-1.2,1.7-1.9,1.5,0])
            ]
        ))
    }
}
```

No consistency



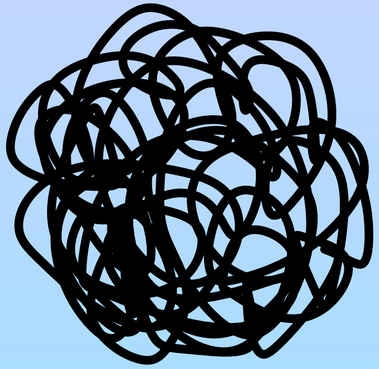
✓

Race conditions



✓

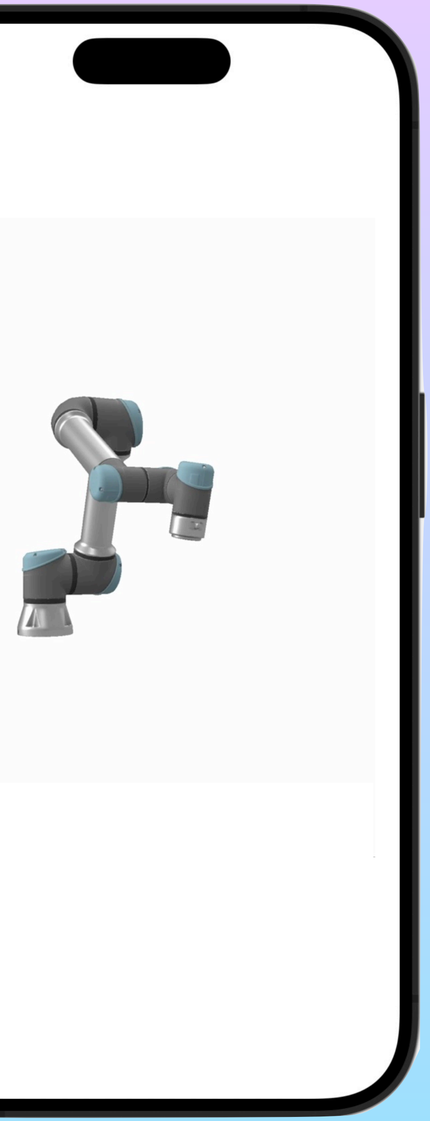
Mess



✗

Potential problem: array

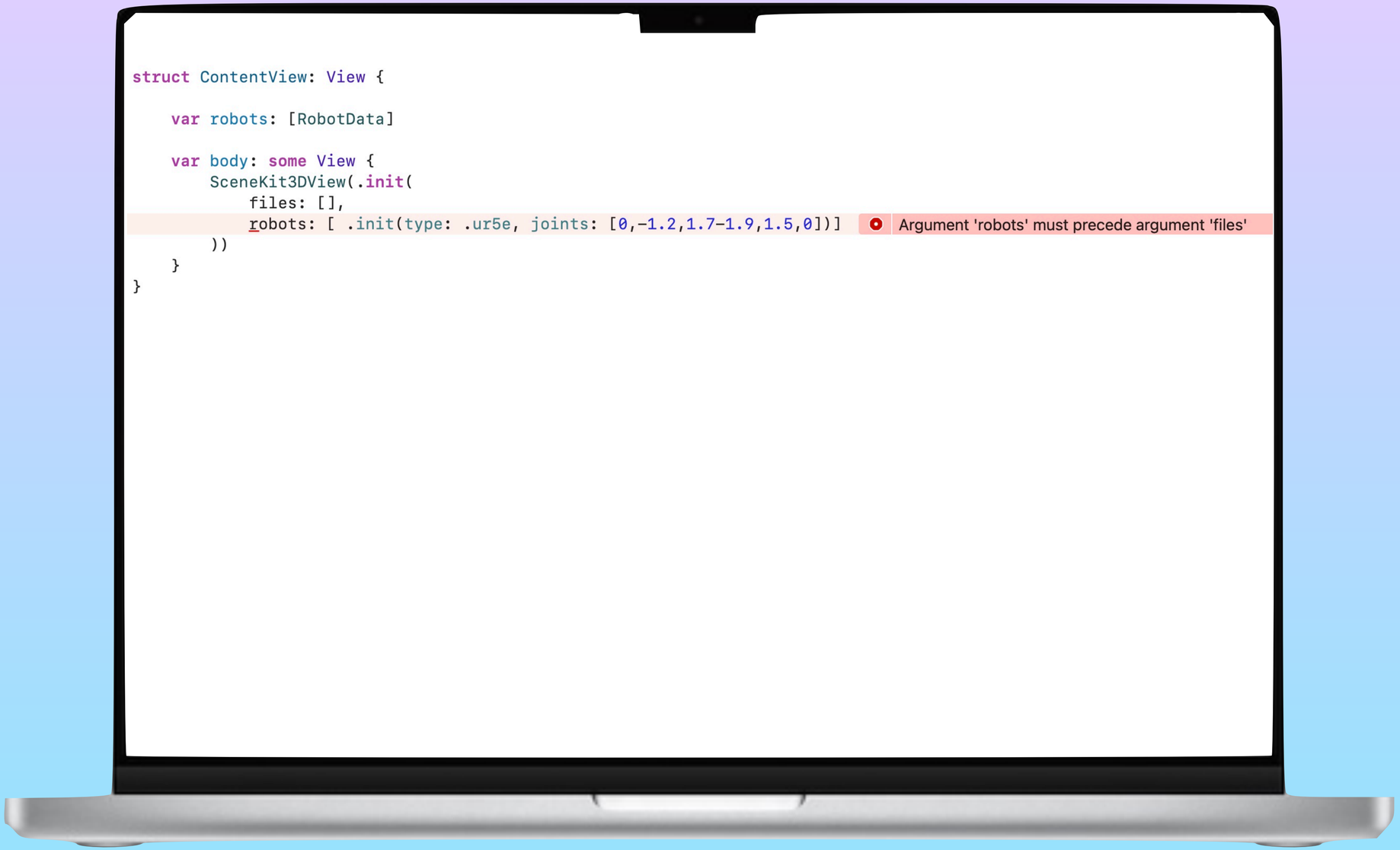
```
struct ContentView: View {

    var robots: [RobotData]

    var body: some View {
        SceneKit3DView(.init(
            robots: robots
                .filter { $0.type != .ur3 }
                .map { RobotInformation(type: $0.type, joints: $0.joints) }
        ))
    }
}
```



Potential problem: conditions

```
struct ContentView: View {

    var isSmallRobot: Bool

    var body: some View {
        SceneKit3DView(.init(
            robots: [
                conditionalRobot
            ]
        ))
    }

    var conditionalRobot: RobotInformation {
        if isSmallRobot {
            .init(type: .ur3, joints: [0,-1.2,1.7-1.9,1.5,0])
        } else {
            .init(type: .ur5e, joints: [0,-1.2,1.7-1.9,1.5,0])
        }
    }
}
```

Potential problem: right order



Swift has a solution
for this problem:

@resultBuilder

Is my problem worthy?

Dave Verwer

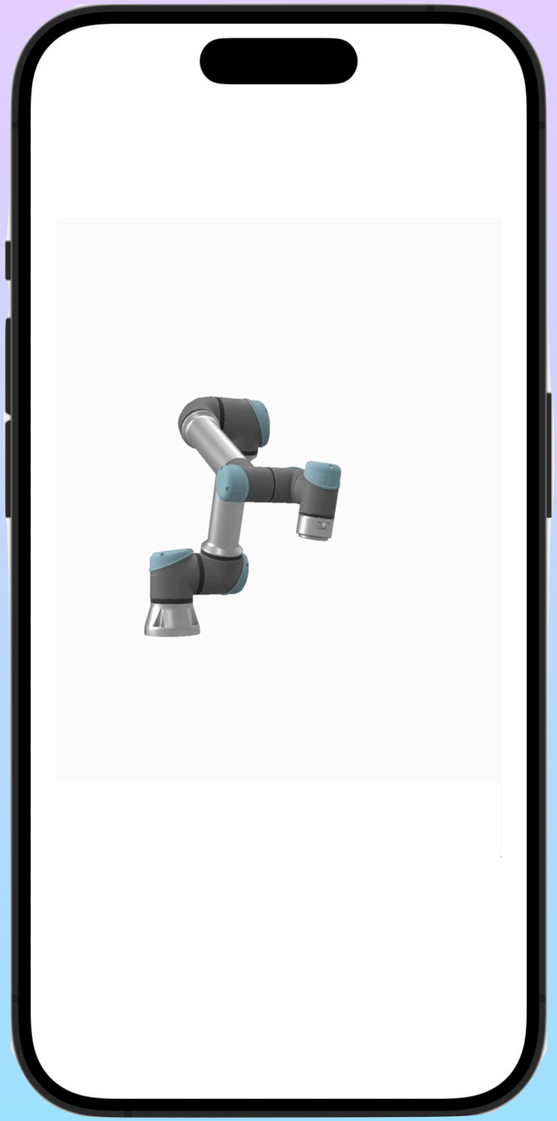
RequestDL

I keep saying that every problem does not need a `ResultBuilder`, and yet I keep coming across example after example where they make sense, like this new package from Brenno De Moura for making network requests more readable!

swiftpackageindex.com

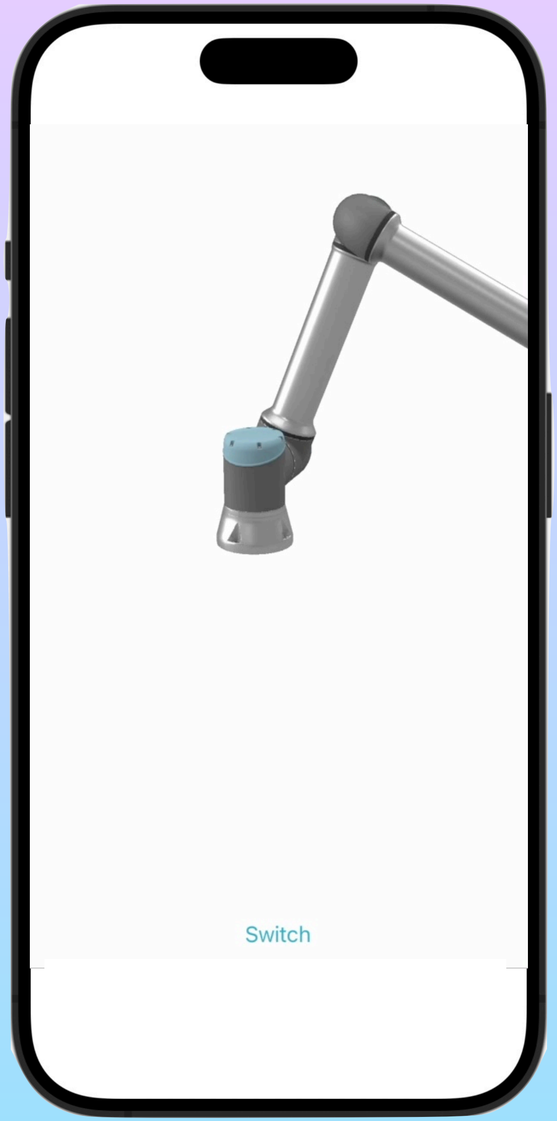
Single robot

```
struct ContentView: View {
    var body: some View {
        Interactive3DView {
            Robot3D(type: .ur5e, joints: [0,-1.2,1.7,-1.9,-1.5,0])
        }
    }
}
```



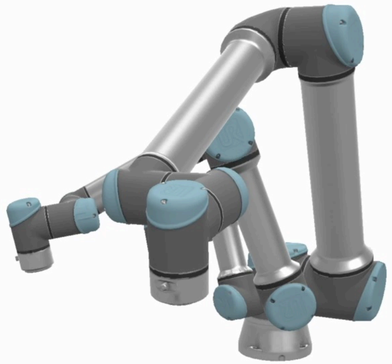
Single robot

```
struct ContentView: View {

    @State var isSmallRobot = false

    var body: some View {
        ZStack(alignment: .bottom) {
            Interactive3DView {
                Robot3D(
                    type: isSmallRobot ? .ur3 : .ur10e,
                    joints: [1,-1.2,1.7,-1.9,-1.5,0]
                )
            }
            Button("Switch") {
                isSmallRobot.toggle()
            }.padding(50)
        }

    }
}
```



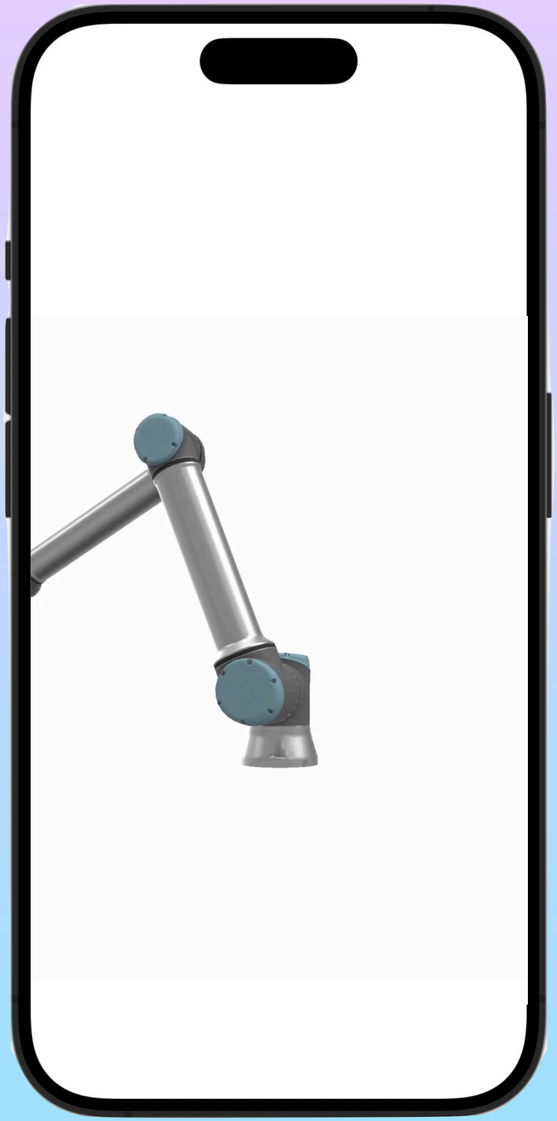
Multiple robots

```
struct ContentView: View {
    var body: some View {
        Interactive3DView {
            Robot3D(type: .ur3, joints: [-1,-1.2,1.7,-1.9,-1.5,0])
            Robot3D(type: .ur5e, joints: [0,-1.2,1.7,-1.9,-1.5,0])
            Robot3D(type: .ur10e, joints: [1,-1.2,1.7,-1.9,-1.5,0])
        }
    }
}
```



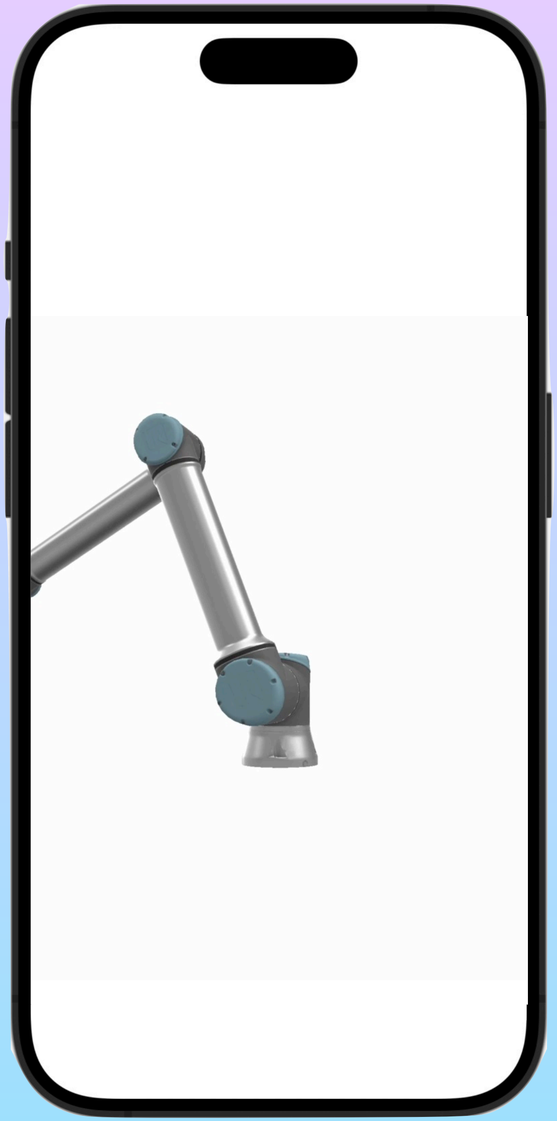
Conditional robot

```
struct ContentView: View {

    var isSmallRobot: Bool

    var body: some View {
        Interactive3DView {
            if isSmallRobot {
                Robot3D(type: .ur3, joints: [-1,-1.2,1.7,-1.9,-1.5,0])
            } else {
                Robot3D(type: .ur10e, joints: [-1,-1.2,1.7,-1.9,-1.5,0]
            }
        }
    }
}
```



Conditional robot

```
struct ContentView: View {

    var isRobot: Bool

    var body: some View {
        Interactive3DView {
            if isRobot {
                Robot3D(
                    type:.ur5e,
                    joints: [1,-1.2,1.7,-1.9,-1.5,0]
                )
            } else {
                File3D(name: "cake")
            }
        }
    }
}
```

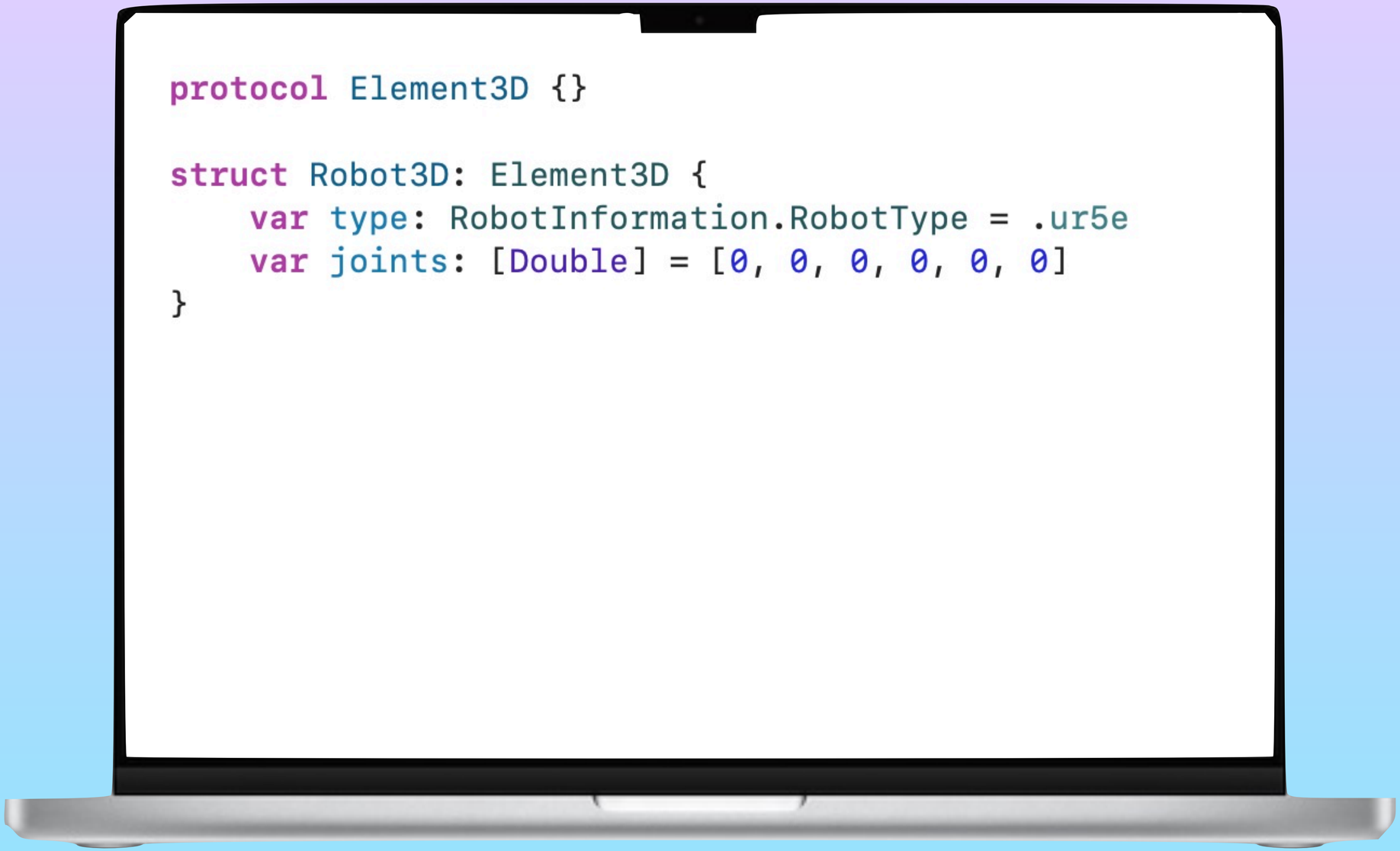


Looping robots

```swift
import SwiftUI
import Robot3D

struct ContentView: View {

    var robotList: [RobotData]

    var body: some View {
        Interactive3DView {
            for robot in robotList {
                if robot.type != .ur3 {
                    Robot3D(type: robot.type, joints: robot.joints)
                }
            }
        }
    }
}
```

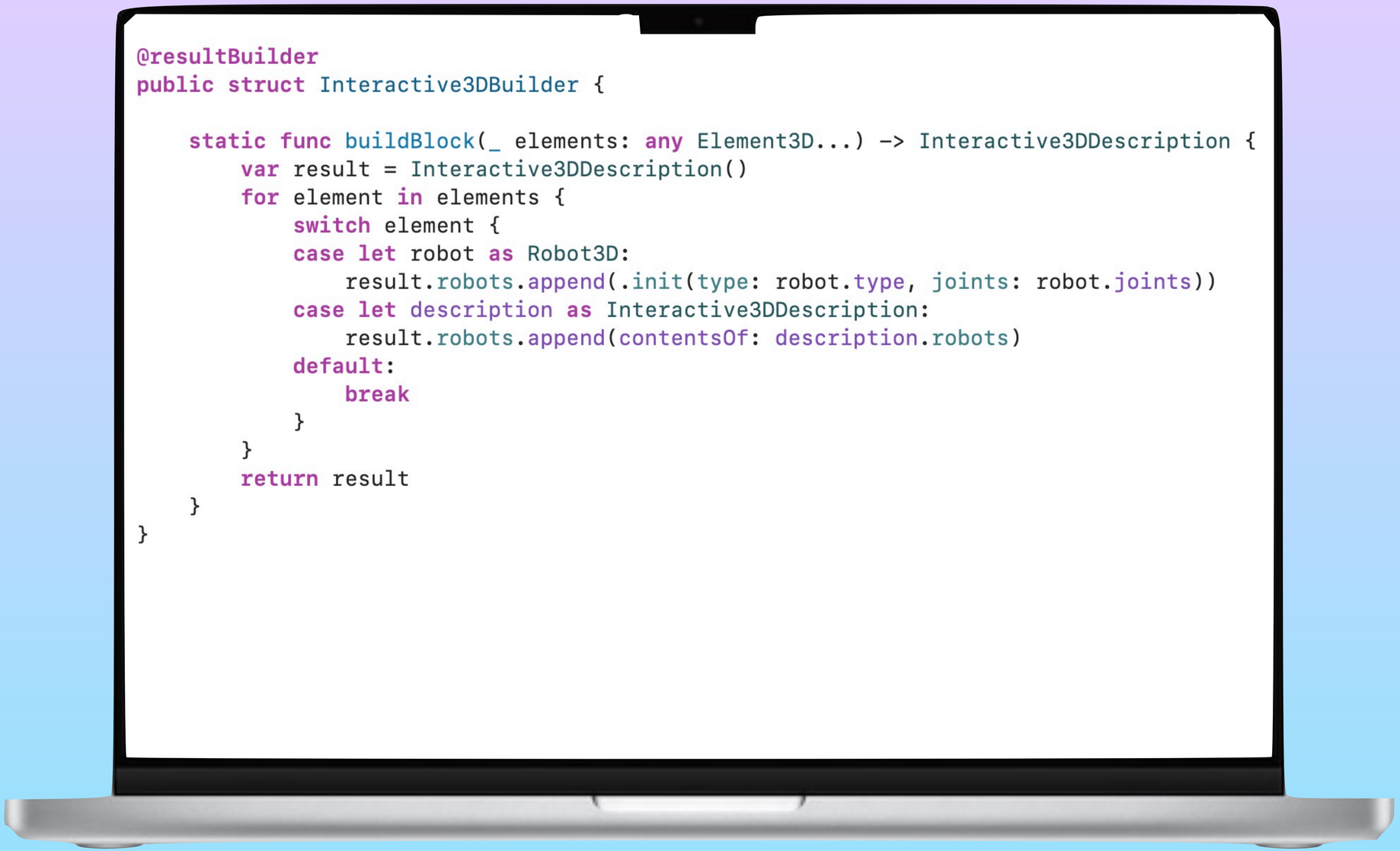
Look under the hood

Base protocol



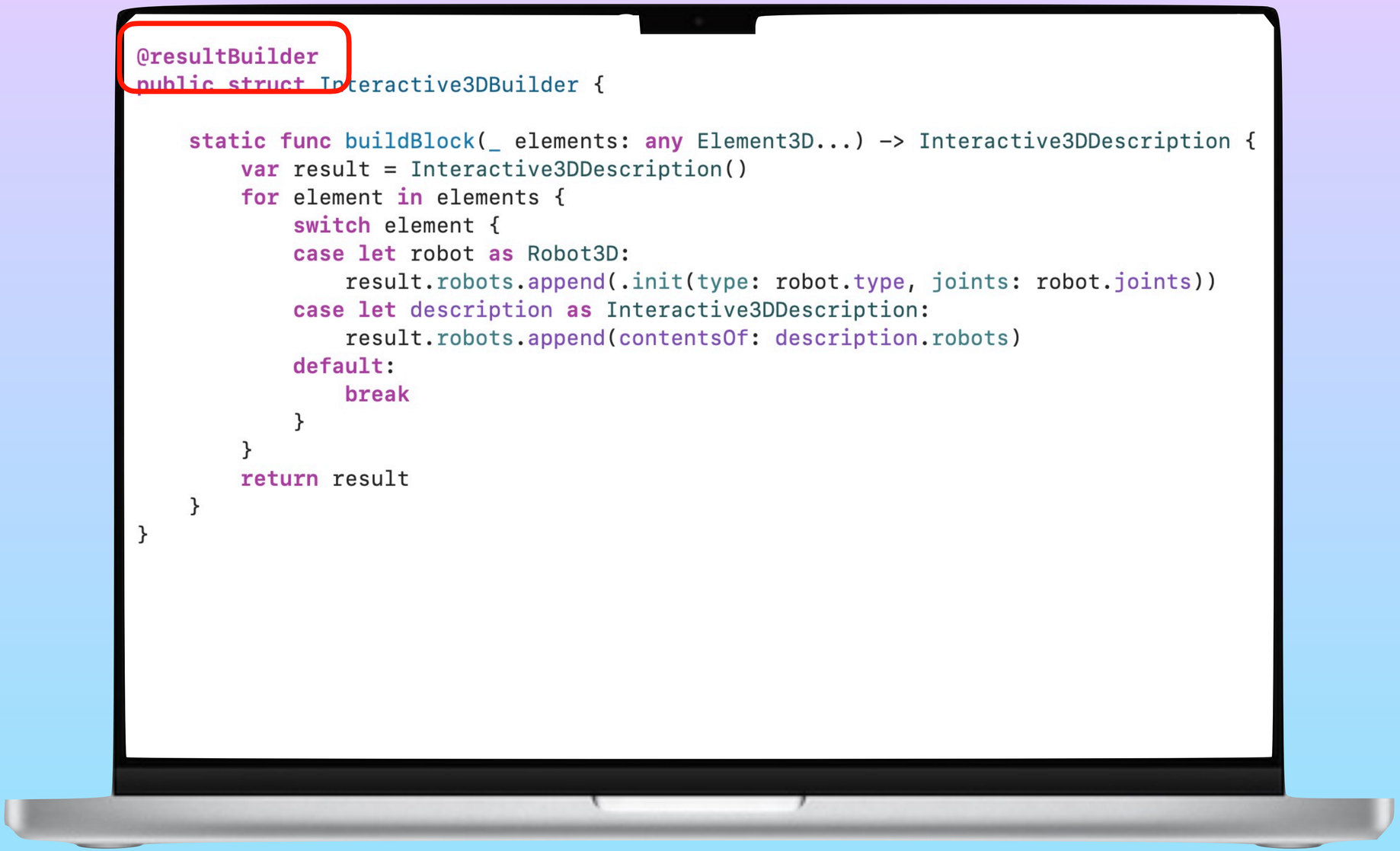
```
protocol Element3D {}

struct Robot3D: Element3D {
    var type: RobotInformation.RobotType = .ur5e
    var joints: [Double] = [0, 0, 0, 0, 0, 0]
}
```

buildBlock



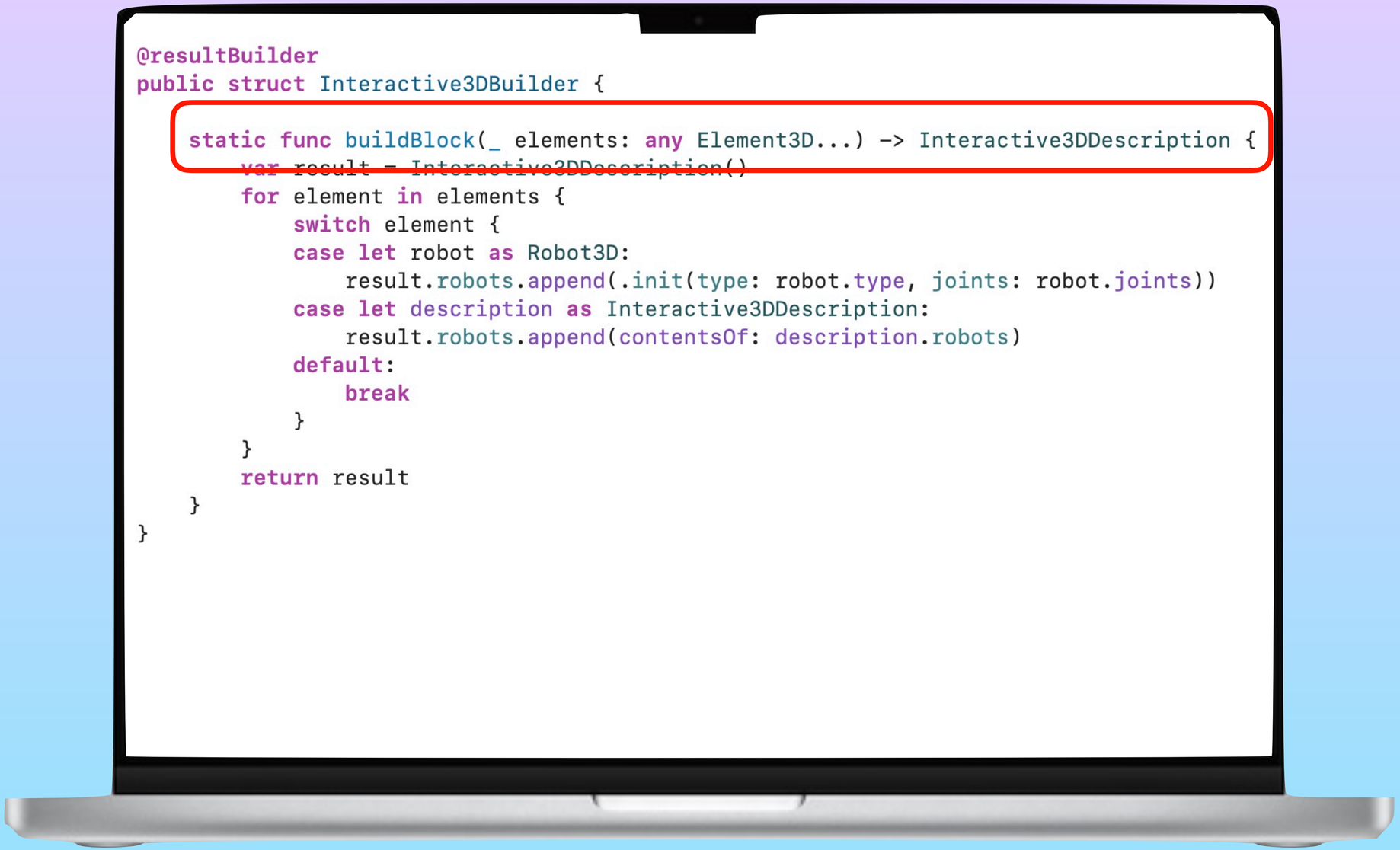
```swift
@resultBuilder
public struct Interactive3DBuilder {

    static func buildBlock(_ elements: any Element3D...) -> Interactive3DDescription {
        var result = Interactive3DDescription()
        for element in elements {
            switch element {
            case let robot as Robot3D:
                result.robots.append(.init(type: robot.type, joints: robot.joints))
            case let description as Interactive3DDescription:
                result.robots.append(contentsOf: description.robots)
            default:
                break
            }
        }
        return result
    }
}
```

buildBlock



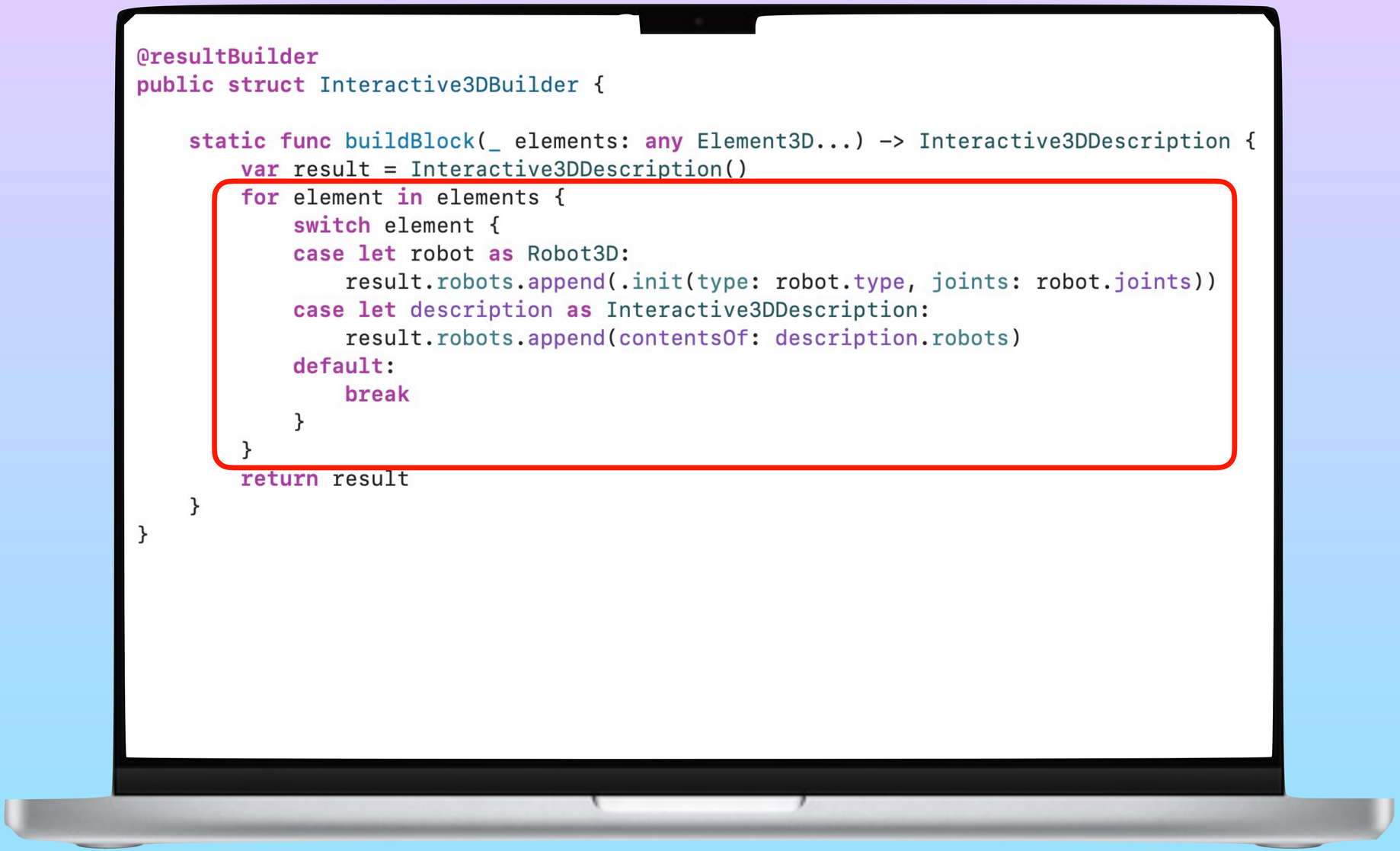
```
@resultBuilder
public struct Interactive3DBuilder {

    static func buildBlock(_ elements: any Element3D...) -> Interactive3DDescription {
        var result = Interactive3DDescription()
        for element in elements {
            switch element {
            case let robot as Robot3D:
                result.robots.append(.init(type: robot.type, joints: robot.joints))
            case let description as Interactive3DDescription:
                result.robots.append(contentsOf: description.robots)
            default:
                break
            }
        }
        return result
    }
}
```

buildBlock



```swift
@resultBuilder
public struct Interactive3DBuilder {

    static func buildBlock(_ elements: any Element3D...) -> Interactive3DDescription {
        var result = Interactive3DDescription()
        for element in elements {
            switch element {
            case let robot as Robot3D:
                result.robots.append(.init(type: robot.type, joints: robot.joints))
            case let description as Interactive3DDescription:
                result.robots.append(contentsOf: description.robots)
            default:
                break
            }
        }
        return result
    }
}
```

buildBlock

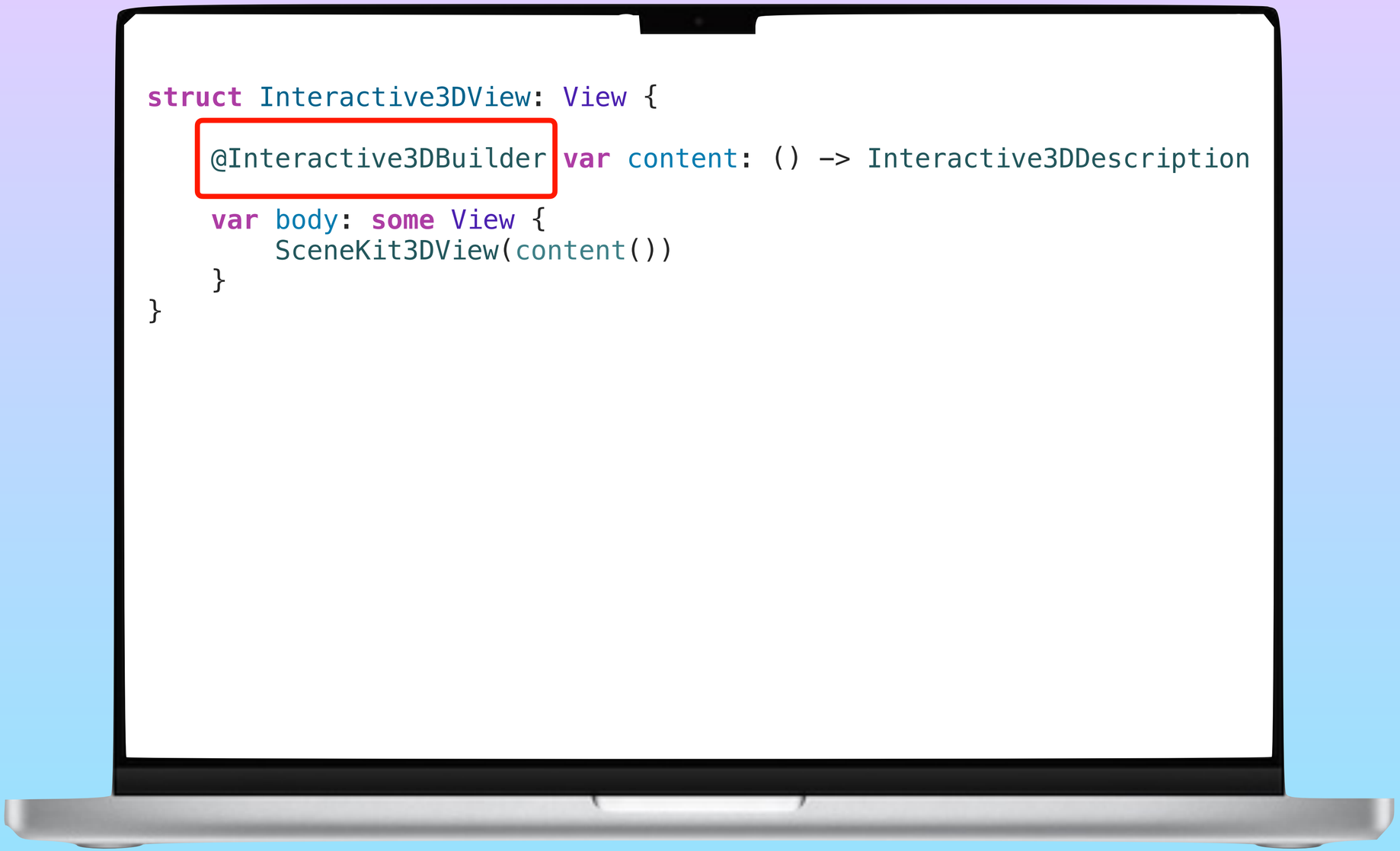


```swift
@resultBuilder
public struct Interactive3DBuilder {

    static func buildBlock(_ elements: any Element3D...) -> Interactive3DDescription {
        var result = Interactive3DDescription()
        for element in elements {
            switch element {
            case let robot as Robot3D:
                result.robots.append(.init(type: robot.type, joints: robot.joints))
            case let description as Interactive3DDescription:
                result.robots.append(contentsOf: description.robots)
            default:
                break
            }
        }
        return result
    }
}
```

buildBlock

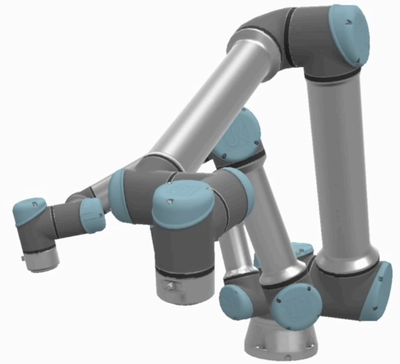
```
struct Interactive3DView: View {

    @Interactive3DBuilder var content: () -> Interactive3DDescription

    var body: some View {
        SceneKit3DView(content())
    }
}
```

buildBlock

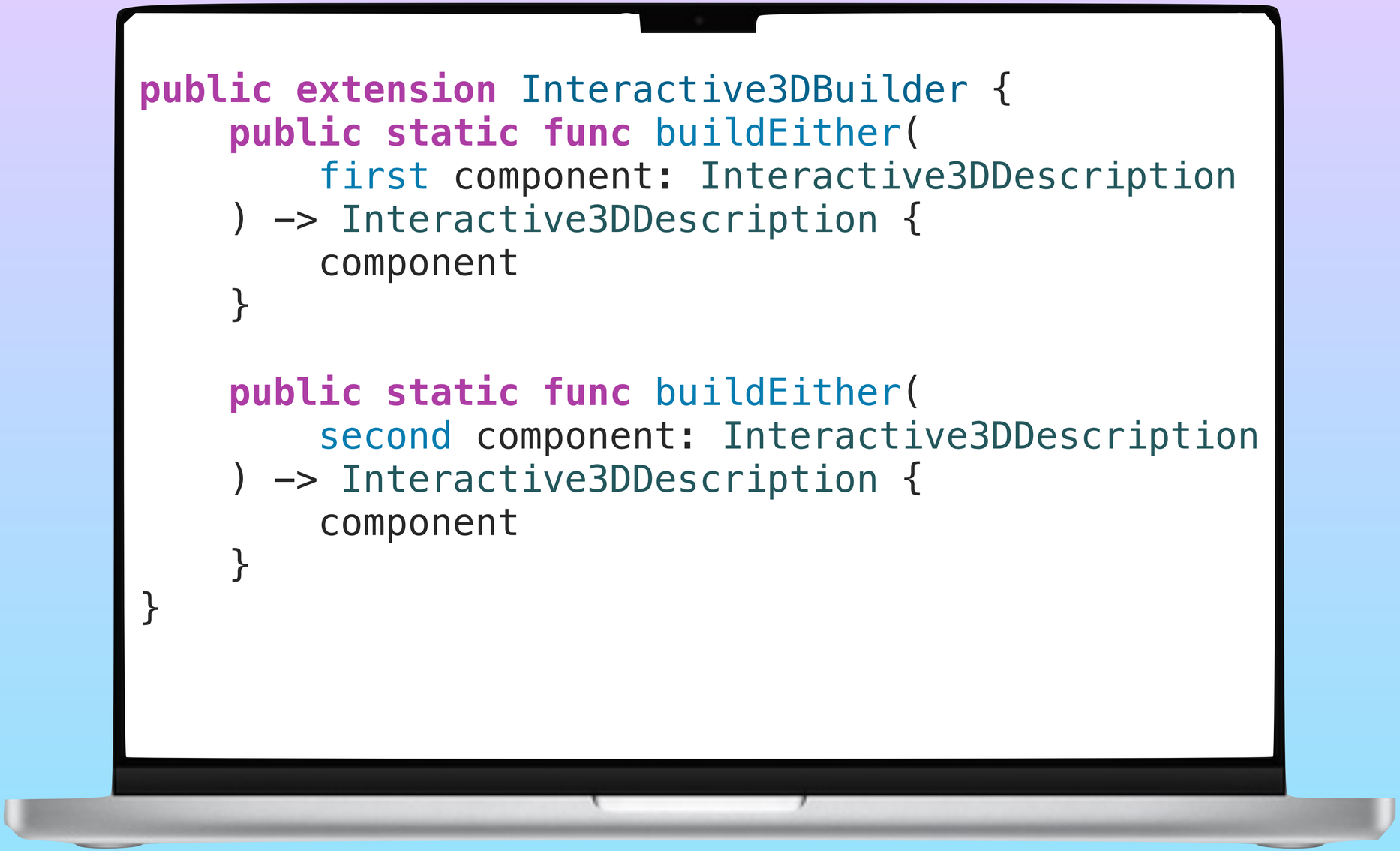

struct Interactive3DView: View {
@Interactive3DBuilder var content: () -> Interactive3DDescription
var body: some View {
SceneKit3DView(content())
}
}

buildBlock

```swift
import SwiftUI
import Robot3D

struct ContentView: View {
    var body: some View {
        Interactive3DView {
            Robot3D(type: .ur3, joints: [-1,-1.2,1.7,-1.9,-1.5,0])
            Robot3D(type: .ur5e, joints: [0,-1.2,1.7,-1.9,-1.5,0])
            Robot3D(type: .ur10e, joints: [1,-1.2,1.7,-1.9,-1.5,0])
        }
    }
}
```



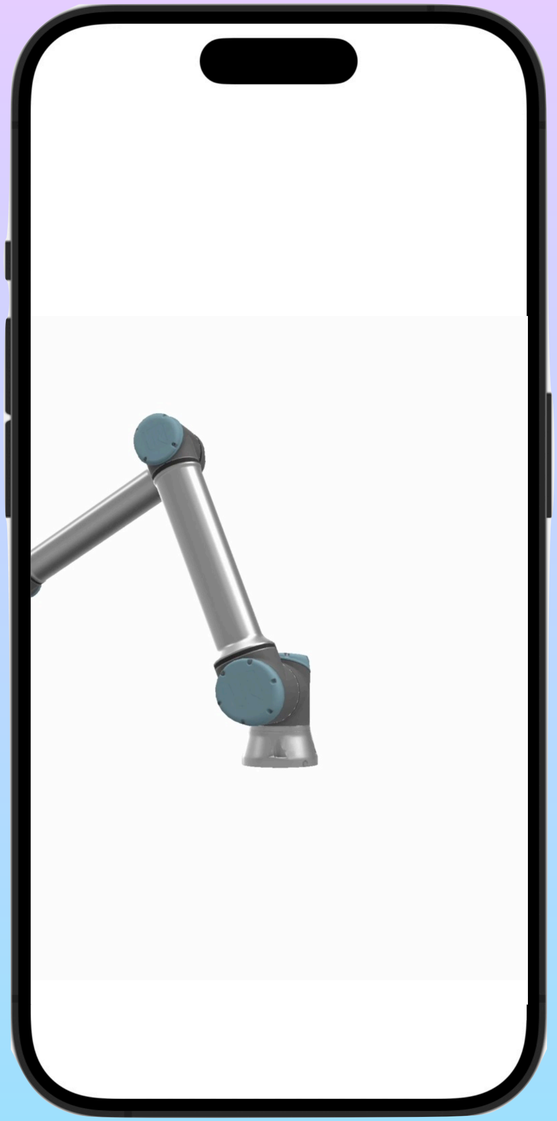
buildEither



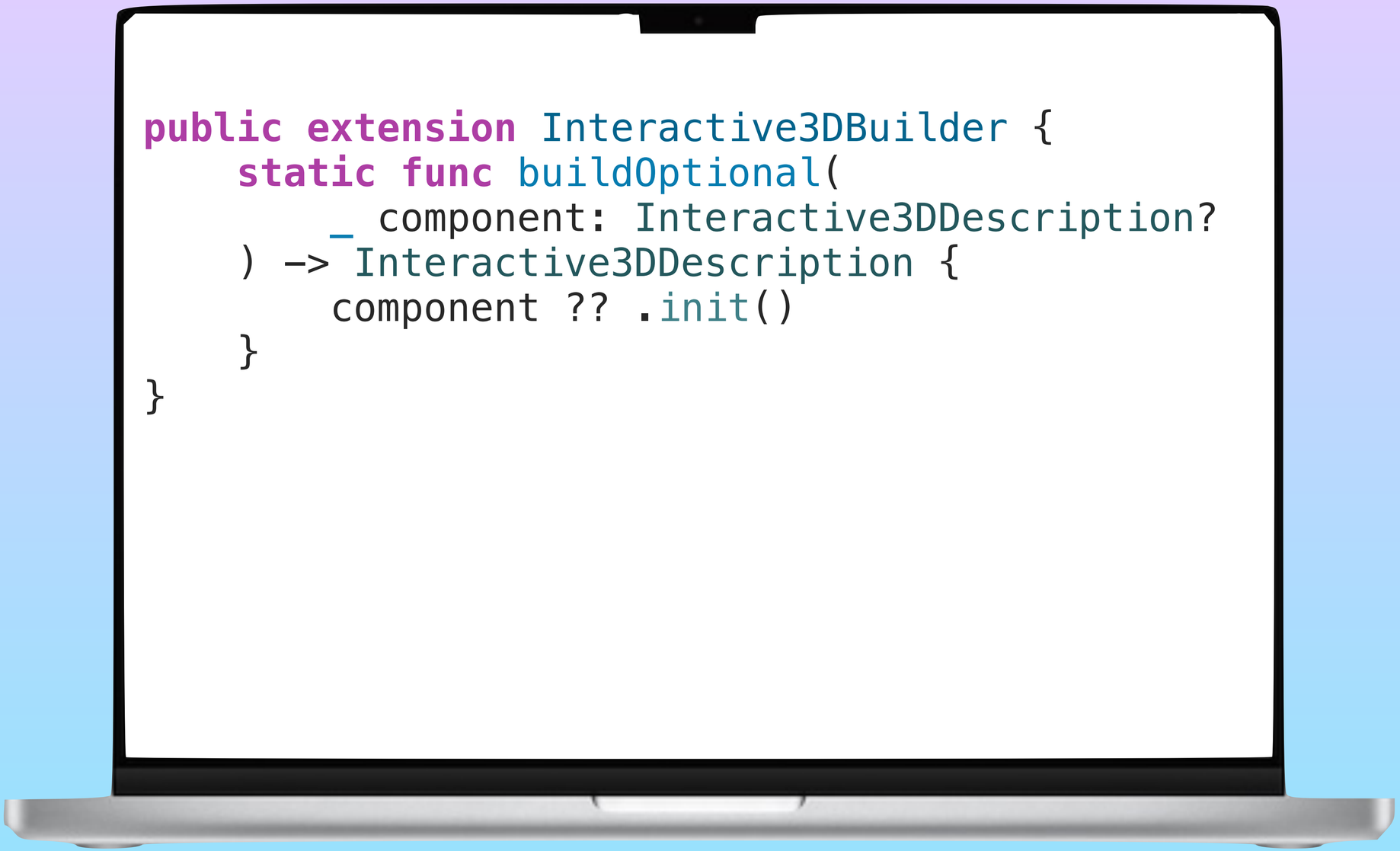
```
public extension Interactive3DBuilder {
    public static func buildEither(
        first component: Interactive3DDescription
    ) -> Interactive3DDescription {
        component
    }

    public static func buildEither(
        second component: Interactive3DDescription
    ) -> Interactive3DDescription {
        component
    }
}
```

buildEither

```swift
import SwiftUI
import Robot3D

struct ContentView: View {

    var isSmallRobot: Bool

    var body: some View {
        Interactive3DView {
            if isSmallRobot {
                Robot3D(type: .ur3, joints: [-1,-1.2,1.7,-1.9,-1.5,0])
            } else {
                Robot3D(type: .ur10e, joints: [-1,-1.2,1.7,-1.9,-1.5,0])
            }
        }
    }
}
```



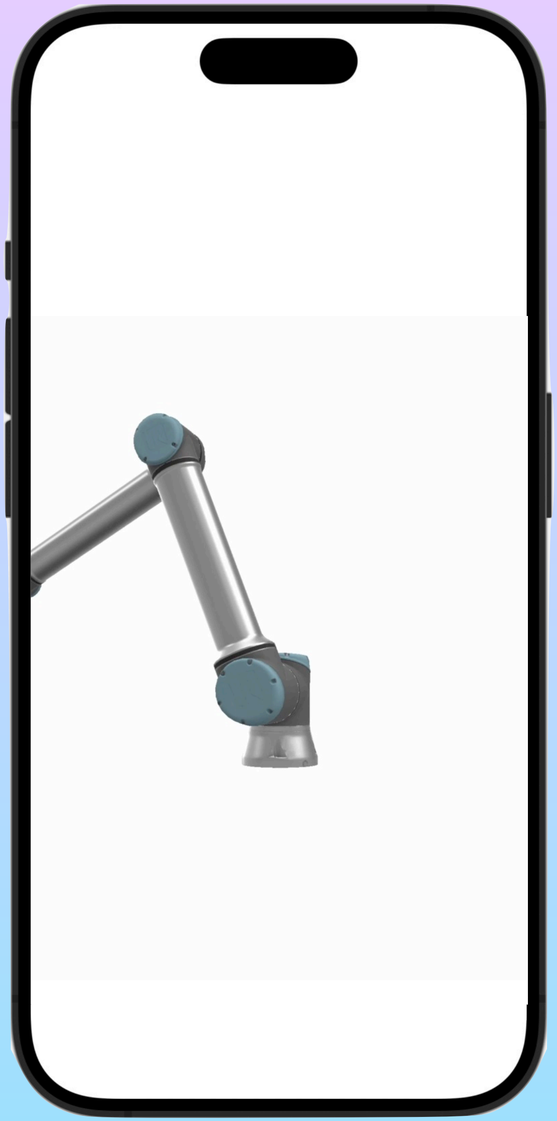
buildOptional



```
public extension Interactive3DBuilder {
    static func buildOptional(
        _ component: Interactive3DDescription?
    ) -> Interactive3DDescription {
        component ?? .init()
    }
}
```

buildOptional

```swift
import SwiftUI
import Robot3D

struct ContentView: View {

    var robot: RobotData?

    var body: some View {
        Interactive3DView {
            if let robot {
                Robot3D(type: robot.type, joints: robot.joints)
            }
        }
    }
}
```



buildArray

```
public extension Interactive3DBuilder {
    static func buildArray(
        _ components: [Interactive3DDescription]
    ) -> Interactive3DDescription {
        var result = Interactive3DDescription()
        for component in components {
            result.robots.append(contentsOf: component.robots)
        }
        return result
    }
}
```

buildArray

```swift
import SwiftUI
import Robot3D

struct ContentView: View {

    var robotList: [RobotData]

    var body: some View {
        Interactive3DView {
            for robot in robotList {
                if robot.type != .ur3 {
                    Robot3D(type: robot.type, joints: robot.joints)
                }
            }
        }
    }
}
```

11:53 Mon 4. Sep

92 %

Back

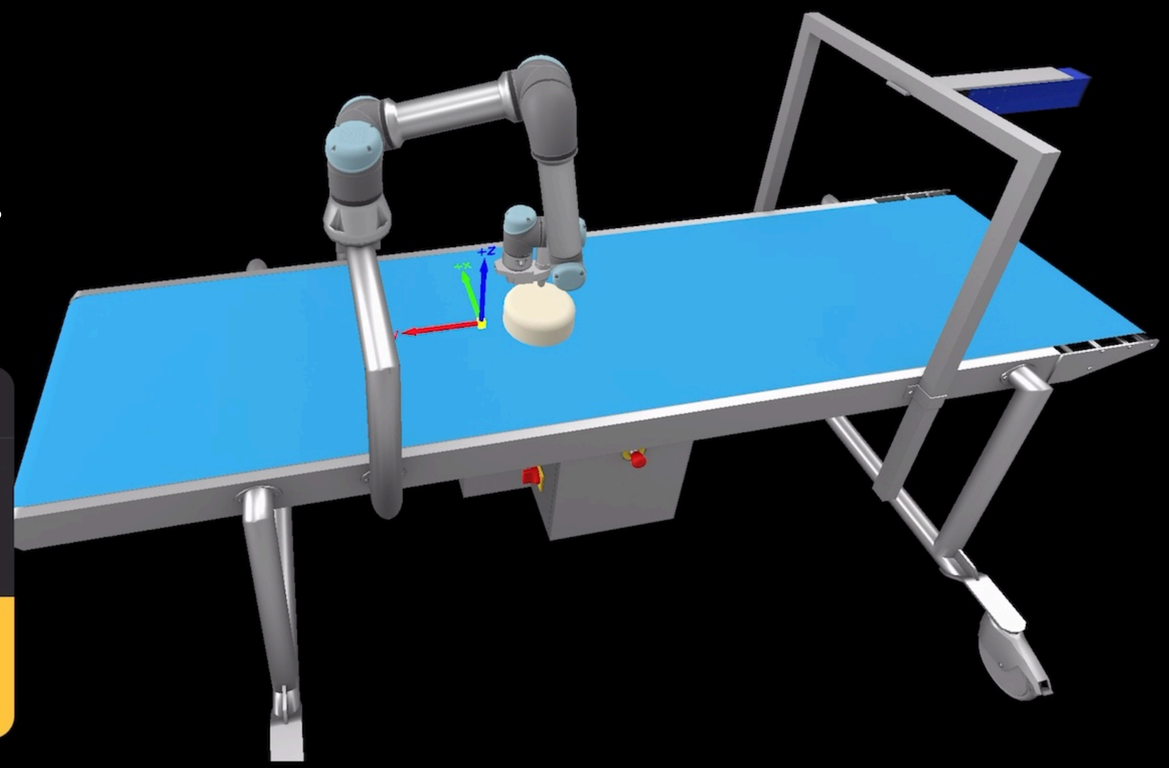
Flower

Finish

Conveyor Speed

~ 0 m/min

3D Visualisation



Hints

Dispensing

Conveyor

Stop

Production

Create & Edit

Help

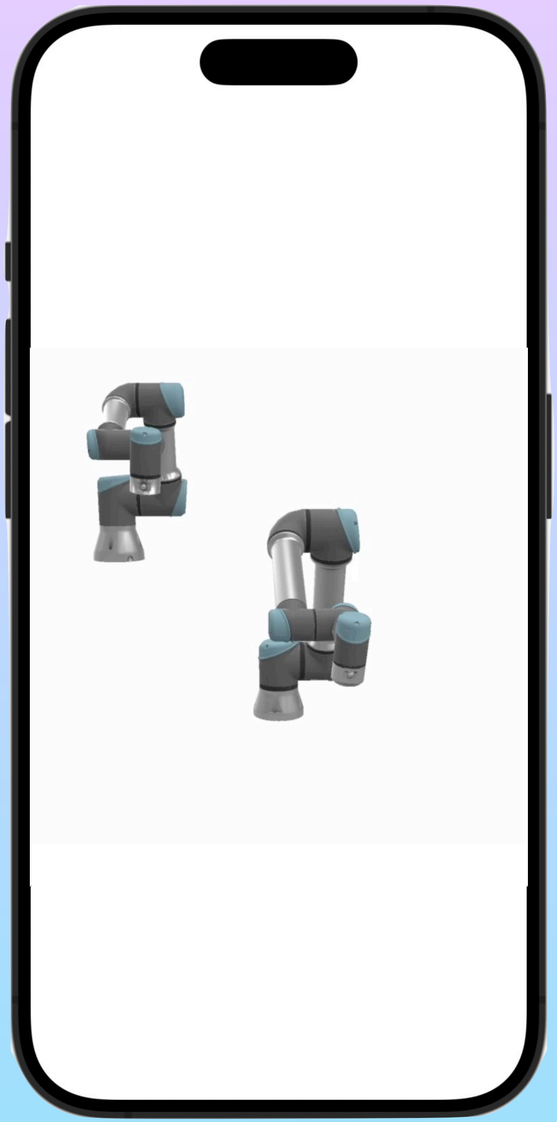
Settings

Modifiers

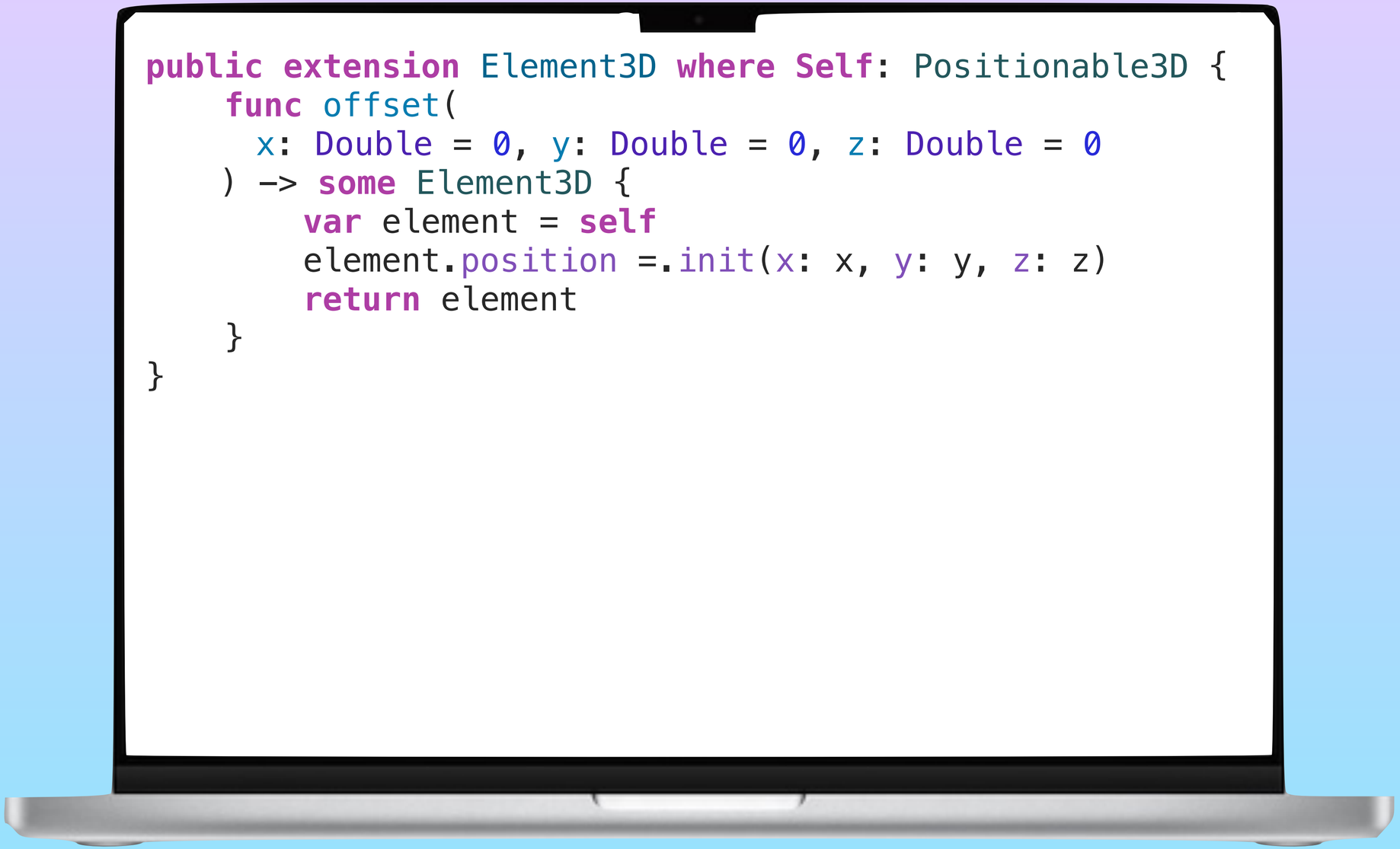
```
Interactive3DView {
   Robot3D(type: .ur5e, joints: robotState?.jointConfiguration) {
      Nozzle3D(showTracepen: recordingStep == .calibration)
   }

   if teachingState.finishedRecording {
     File3D(name: "Axis3D")
       .offset(x: 0, y: -300, z: -350)
   }

   if recordingStep.isCloseUpCamera {
      Camera3D()
        .moveTo(node: "Conveyor")
   }

   Conveyor3D()

   if recordingStep.showCake {
     File3D(name: "marcipaneCake", type: .usdz)
       .offset(y: cakeOffset, z: -375)
   }
}
```

Modifiers

```
struct ContentView: View {

    var body: some View {
        Interactive3DView {
            Robot3D(
                type: .ur5e,
                joints: [1,-1.2,1.7,-1.9,-1.5,0]
            )
                .offset(x: 500)
            Robot3D(
                type: .ur3,
                joints: [1,-1.2,1.7,-1.9,-1.5,0]
            )
                .offset(z: 500)
        }
    }
}
```



Modifiers

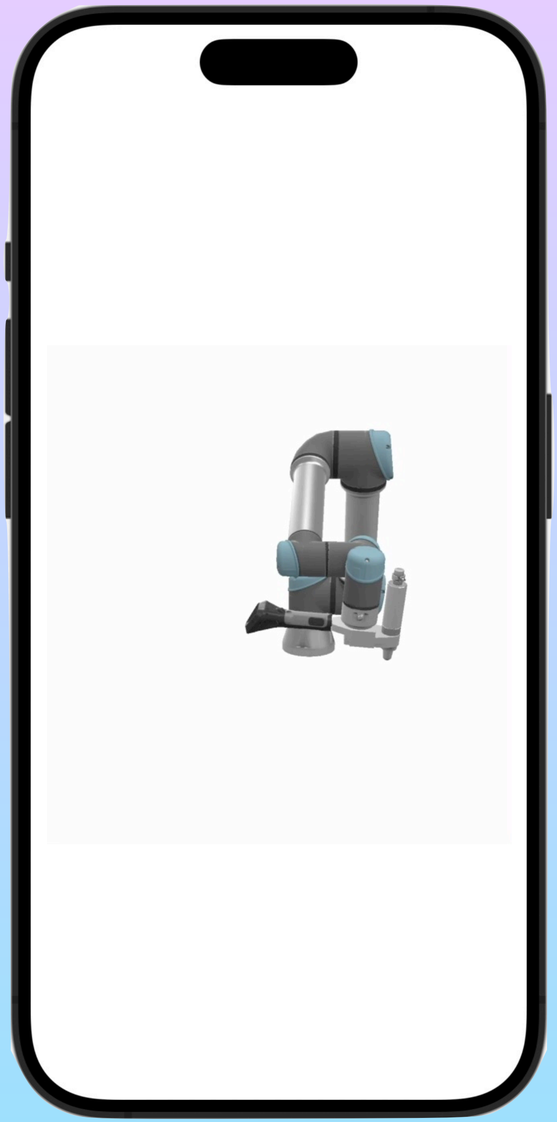

public extension Element3D where Self: Positionable3D {
func offset(
x: Double = 0, y: Double = 0, z: Double = 0
) -> some Element3D {
var element = self
element.position =.init(x: x, y: y, z: z)
return element
}
}

Closure Attachment

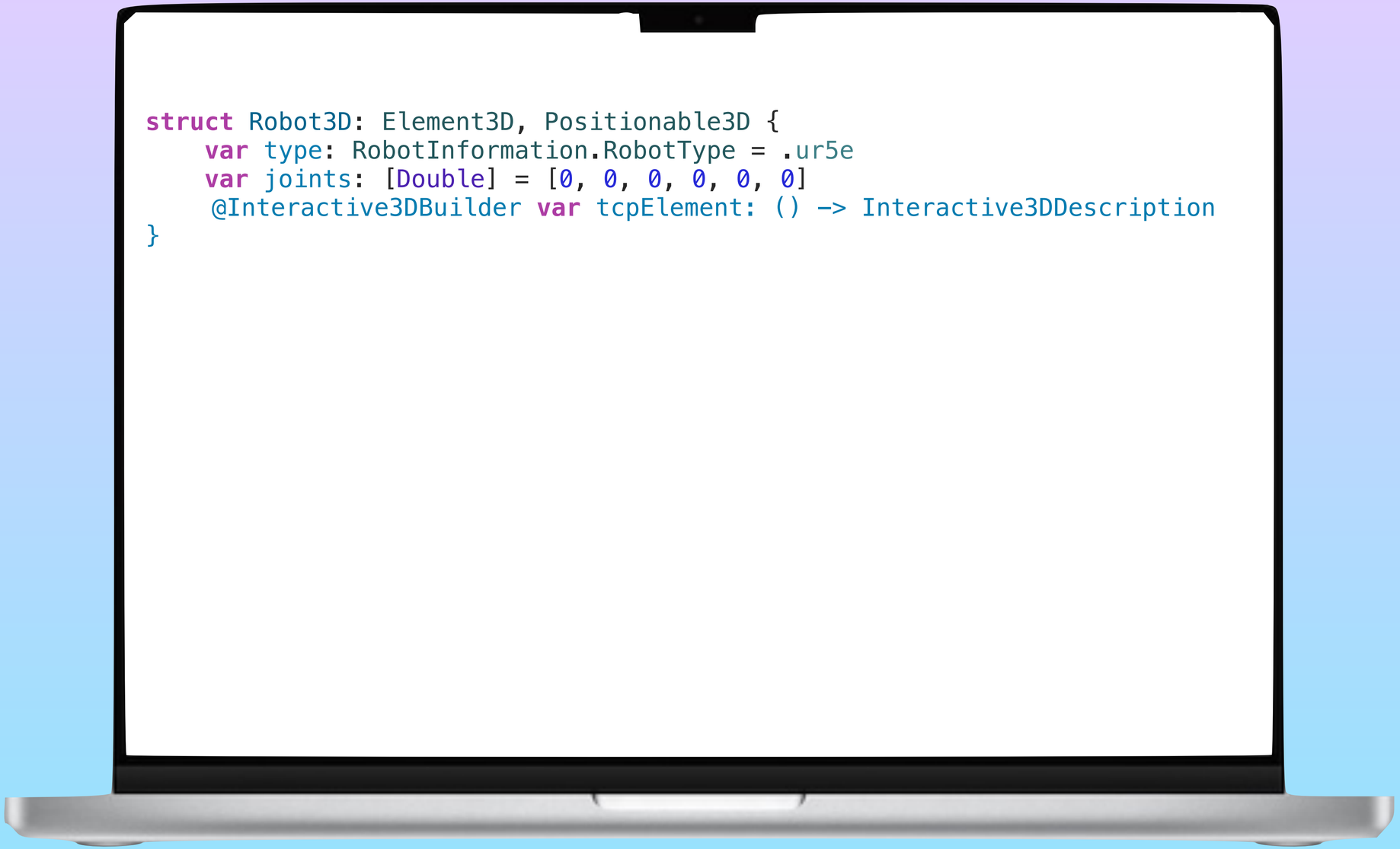
```
Interactive3DView {
   Robot3D(type: .ur5e, joints: robotState?.jointConfiguration) {
      Nozzle3D(showTracepen: recordingStep == .calibration)
   }

   if teachingState.finishedRecording {
     File3D(name: "Axis3D")
       .offset(x: 0, y: -300, z: -350)
   }

   if recordingStep.isCloseUpCamera {
      Camera3D()
        .moveTo(node: "Conveyor")
   }

   Conveyor3D()

   if recordingStep.showCake {
     File3D(name: "marcipaneCake", type: .usdz)
       .offset(y: cakeOffset, z: -375)
   }
}
```

Closure Attachment

```
struct ContentView: View {

    var body: some View {
        Interactive3DView {
            Robot3D(
                type: .ur5e,
                joints: [1,-1.2,1.7,-1.9,-1.5,0]
            ) {
                File3D(name: "Tooltip", type: .usdz)
            }
        }
    }
}
```

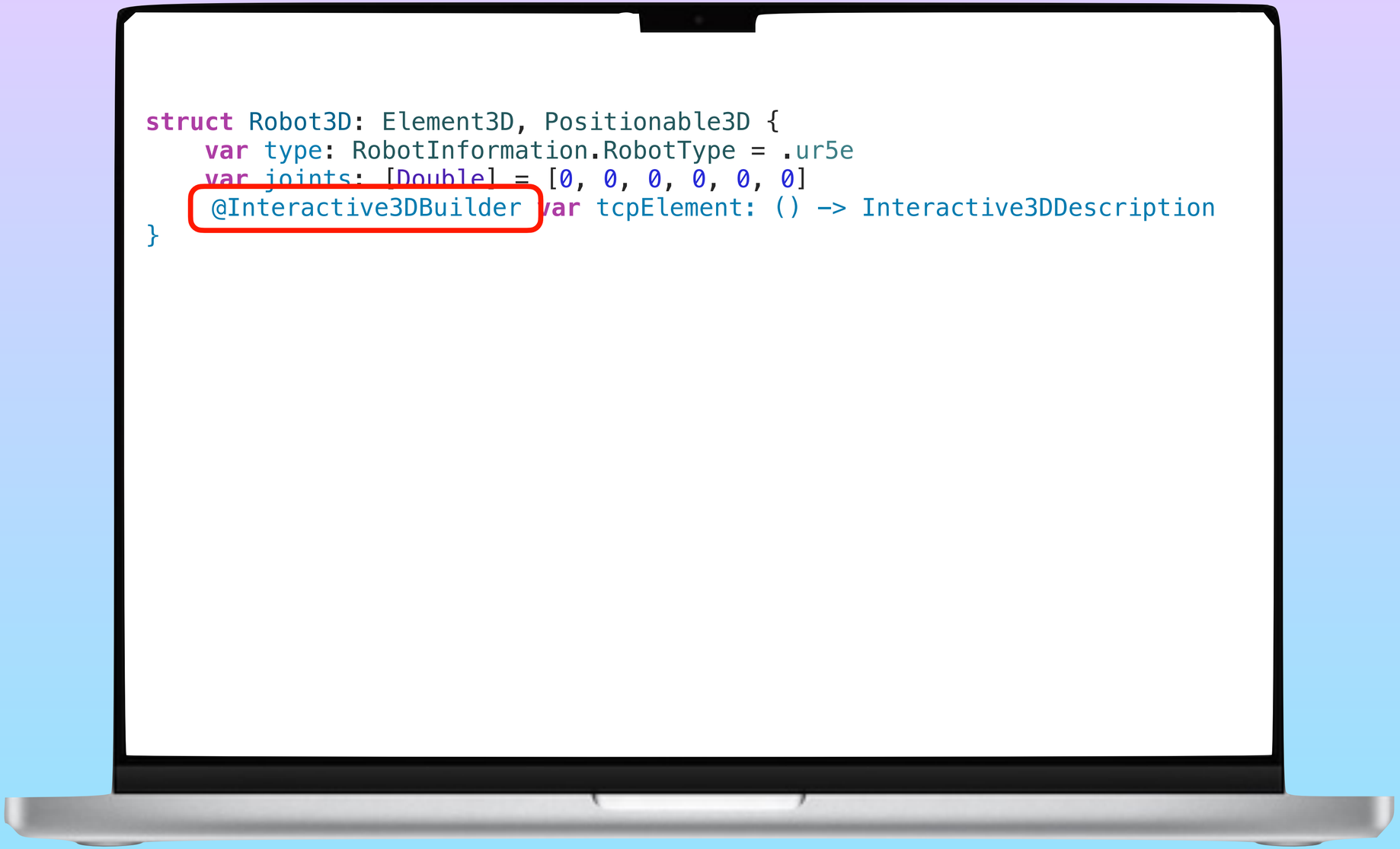


Closure Attachment



```
struct Robot3D: Element3D, Positionable3D {
    var type: RobotInformation.RobotType = .ur5e
    var joints: [Double] = [0, 0, 0, 0, 0, 0]
    @Interactive3DBuilder var tcpElement: () -> Interactive3DDescription
}
```

Closure Attachment



Summary

we started with imperative calls

moved to declarative structs

and ended with @resultBuilder

Thank you

dominik@riegger.tech

riegger-tech

@domi@troet.cafe