// turbosquid .dae prefre the root to hold the pictures

// MB ProgressHUD writtly in object C
header .h
implmation .m → those two needed
create folder reference
↓ drag into the xcode and create Bridging header
↓ in the Bridging header
#import "MB ProgressHUD.h"

// ViewController.swift
import UIKit
import SceneKit
import ARKit

```
class ViewController: UIViewController, ARSCNViewDelegate {
    var sceneView: ARSCNView!

    private var hud : MBProgressHUD!
    private var newAngleY: Float = 0.0
    private var currentAngleY: Float = 0.0
    private var localTranslatePosition: CGPoint!

    override func viewDidLoad(){
      super.viewDidLoad()

      DispatchQueue.main.async{ // optional to dispatch in viewDidLoad
        self.hud = MBProgressHUD.showAdded(to: self.view, animated: true)
        self.hud.label.text = "detecting plane..."
      }

      self.sceneView = ARSCNView(frame: self.view.frame)
      self.view.addSubview(self.sceneView)
      self.sceneView.autoenablesDefaultLighting = true  // make difference
      sceneView.delegate = self
      let scene = SCNScene()
      sceneView.scene = scene

      registerGestureRecognizers()
    }

    private func registerGestureRecognizers()
        let tapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(tapped))
        self.sceneView.addGestureRecognizer(tapGestureRecognizer)

        let pinchGestureRecognizer = UIPinchGestureRecognizer(target: self, action: #selector(pinched))
        self.sceneView.addGestureRecognizer(pinchGestureRecognizer)

        let panGestureRecognizer = UIPanGestureRecognizer(target: self, action: #selector(panned))
        self.sceneView.addGestureRecognizer(panGestureRecognizer)

        let longPressGestureRecognizer = UILongPressGestureRecognizer(target: self, action: #selector(longPressed))
        self.sceneView.addGestureRecognizer(longPressGestureRecognizer)

    }

    @objc func longPressed(recognizer: UILongPressGestureRecognizer){
```

```
        guard let sceneView = recognizer.view as? ARSCNView else {
            return
        }
        let touch = recognizer.location(in: sceneView)
        let hitTestResults = self.sceneView.hitTest(touch, options: nil)

        if let hitTest = hitTestResults.first {
            if let parentNode = hitTest.node.parent {
                if recognizer.state == .began {
                    self.localTranslatePosition = touch        // defined by user
                } else if recognizer.state == .changed {
                    let deltaX = Float(touch.x - self.localTranslatePosition.x) / 700   // larger smoother
                    let deltaY = Float(touch.y - self.localTranslatePosition.y) / 700
                    parentNode.localTranslate(by: SCNVector3(deltaX, 0.0, deltaY))
                    self.localTranslatePosition = touch
                }
            }
        }
    }

    objc func panned(recognizer: UIPanGestureRecognizer) {
        if recognizer.state == .changed {
            guard let sceneView = recognizer.view as? ARSCNView else {
                return
            }
            let touch = recognizer.location(in: sceneView)
            let translation = recognizer.translation(in: sceneView)   // .translation exists in Pan recognizer not all others
            let hitTestResults = self.sceneView.hitTest(touch, options: nil)
            if let hitTest = hitTestResults.first {
                if let parentNode = hitTest.node.parent {
                    self.newAngleY = Float(translation.x) * (Float)(Double.pi) / 180   // on the screen
                    self.newAngleY += self.currentAngleY
                    parentNode.eulerAngles.y = self.newAngleY
                }
            }
        } else if recognizer.state == .ended {
            self.currentAngleY = self.newAngleY
        }
    }

    @objc func pinched(recognizer: UIPinchGestureRecognizer) {
        if recognizer.state == .changed {
            guard let sceneView = recognizer.view as? ARSCNView else {
                return
            }
            let touch = recognizer.location(in: sceneView)
            let hitTestResults = self.sceneView.hitTest(touch, options: nil)
            if let hitTest = hitTestResults.first {
```

// fix the wrong center
Create a parent ⇒ move the chairNode center will change the center of the parentNode

```
            let chairNode = hitTest.node
            let pinchScaleX = Float(recognizer.scale) * chairNode.scale.x
            let pinchScaleY = Float(recognizer.scale) * chairNode.scale.y
            let pinchScaleZ = Float(recognizer.scale) * chairNode.scale.z
            chairNode.scale = SCNVector3(pinchScaleX, pinchScaleY, pinchScaleZ)
            recognizer.scale = 1
        }
    }
}

@objc func tapped(recognizer: UIGestureRecognizer){
    let sceneView = recognizer.view as! ARSCNView
    let touchLocation = recognizer.location(in: sceneView)  // 2D location CGPoint

    let hitTestResult = sceneView.hitTest(touchLocation, types: .existingPlaneUsingExtent)
    if let hitTest = hitTestResult.first { // if the hitTestRest has a first one
        let chairScene = SCNScene(named: "chair.dae")!
        guard let chairNode = chairScene.rootNode.childNode(withName: "parentNode", recursively: true) else{
            return
        }
        chairNode.position = SCNVector3(hitTest.worldTransform.columns.3.x,
                                        hitTest.worldTransform.columns.3.y,
                                        hitTest.worldTransform.columns.3.z)
        self.sceneView.scene.rootNode.addChildNode(chairNode)
    }
}

func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor){
    if anchor is ARPlaneAnchor{
        DispatchQueue.main.async{
            self.hud.label.text = "Plane Detected"
            self.hud.hide(animated: true, afterDelay: 1.0)
        }
    }

}

override func viewWillAppear(_ animated: Bool){
    super.viewWillAppear(animated)
    let configuration = ARWorldTrackingConfiguration()
    configuration.planeDetection = .horizontal    //enable plane detection

    sceneView.session.run(configuration)
}
```

later: guard let sceneView = recognizer.view as? ARSCNView else{ return }

// check target membership

⚠ important

called every time find an anchor