```
// ViewController.swift
import UIKit
import SceneKit
import ARKit

enum BodyType : Int {
    case box = 1
    case plane = 2
}

class viewController: UIViewController, ARSCNViewDelegate {
    var sceneView: ARSCNView!
    var planes = [OverlayPlane]()

    override func viewDidLoad(){
        super.viewDidLoad()
        self.sceneView = ARSCNView(frame: self.view.frame)
        self.sceneView.debugOptions = [ARSCNDebugOptions.showFeaturePoints, ARSCNDebugOptions.showWorldOrigin]
        self.view.addSubview(self.sceneView)

        sceneView.delegate = self
        sceneView.showsStatistics = true
        let scene = SCNScene()
        sceneView.scene = scene

        registerGestureRecognizers()
    }

    private func registerGestureRecognizers(){
        let tapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(tapped))
        tapGestureRecognizer.numberOfTapsRequired = 1
        let doubleTappedGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(doubleTapped))
        doubleTappedGestureRecognizer.numberOfTapsRequired = 2
        tapGestureRecognizer.require(toFail: doubleTappedGestureRecognizer) // single tap will fail if double tap performed

        self.sceneView.addGestureRecognizer(tapGestureRecognizer)
        self.sceneView.addGestureRecognizer(doubleTappedGestureRecognizer)
    }

    @objc func doubleTapped(recognizer: UIGestureRecognizer){
        let sceneView = recognizer.view as! ARSCNView
        let touchLocation = recognizer.location(in: sceneView)
        let hitTestResult = sceneView.hitTest(touchLocation,
        if !hitTestResult.isEmpty{
            guard let hitResult = hitTestResult.first else{
                return
            }
            let node = hitResult.node
            node.physicsBody?.applyForce(SCNVector3(hitResult.worldCoordinates.x, 2.0, hitResult.worldCoordinates.z), asImpulse: true)
        }
    }
}
```

large force

```
@objc func tapped(recognizer: UIGestureRecognizer){
    let sceneView = recognizer.view as! ARSCNView
    let touchLocation = recognizer.location(in: sceneView)
    let hitTestResult = sceneView.hitTest(touchLocation, types: .existingPlaneUsingExtent)

    if !hitTestResult.isEmpty{
        guard let hitResult = hitTestResult.first else{
            return
        }
        addBox(hitResult: hitResult)

    }
}

private func addBox(hitResult: ARHitTestResult){
    let box = SCNBox(width:0.2, height:0.2, length:0.2, chamferRadius:0)
    let material = SCNMaterial()
    material.diffuse.contents = UIColor.red
    box.materials = [material]
    let boxNode = SCNNode(geometry: box)
    boxNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)
    boxNode.physicsBody?.categoryBitMask = BodyType.box.rawValue

    boxNode.position = SCNVector3(hitResult.worldTransform.columns.3.x, hitResult.worldTransform.columns.3.y +
                                  Float(box.height/2), hitResult.worldTransform.column.3.z)

    self.sceneView.scene.rootNode.addChildNode(boxNode)

}

private func addTable(hitResult:ARHitTestResult){
    let tableScene = SCNScene(named: "art.scnassets/bench.dae")
    let tableNode = tableScene?.rootNode.childNode(withName:"SketchUp", recursively:true)

    tableNode.position = SCNVector3(hitResult.worldTransform.columns.3.x, hitResult.worldTransform.columns.3.y
                                    ,hitResult.worldTransform.column.3.z)

    self.sceneView.scene.rootNode.addChildNode(tableNode)
}

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

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

// dynamic ⇒ can be moving, can be affected by forces
kinematic ⇒ unaffected by forces but can cause collisions when moved
static ⇒ can not moving ⇒ like a wall, can not move
⇒ automatic find the best shape

//sketch up ⇒ 3D warehouse
// TurboSquid.com

download ⇒ 3DS or DAE
export as ⇒ collada file .dae
Bounding Box ⇒ in meters

```
                                                    ⇒ called every frame
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor){
        if !(anchor is ARPlaneAnchor){
            return
        }
        DispatchQueue.main.async{
            self.label.text = "Plane Detected"
            UIView.animate(withDuration: 3.0, animations: {
                self.label.alpha = 1.0
            }){ (completion: Bool) in
                self.label.alpha = 0.0
            }
        }
        let plane = OverlayPlane(anchor: anchor as! ARPlaneAnchor)
        self.planes.append(plane)

        node.addChildNode(plane)

    }

    func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor){
                                                    ↗ call every frame
        let plane = self.planes.filter{ plane in
            return plane.anchor.identifier == anchor.identifier
            }.first

        if plane == nil{
            return
        }
        plane?.update(anchor: anchor as! ARPlaneAnchor)
    }

}
```

// OverlayPlane.swift

```
import: ARKit

class OverlayPlane: SCNNode{
    var anchor: ARPlaneAnchor
    var planeGeometry: SCNPlane!

    init(anchor: ARPlaneAnchor){
        self.anchor = anchor
        super.init()
        setup()
    }
```

```
func update(anchor: ARPlaneAnchor){
    self.planeGeometry.width = CGFloat(anchor.extent.x);
    self.planeGeometry.height = CGFloat(anchor.extent.y);
    self.position = SCNVector3Make(anchor.center.x, 0, anchor.center.z);
    let planeNode = self.childNode.first
    planeNode.physicsBody = SCNPhysicsBody(type.static, shape: SCNPhysicsShape(geometry: self.planeGeometry, options: nil))

}

private func setup(){
    self.planeGeometry = SCNPlane(width: CGFloat(self.anchor.extent.x), height: CGFloat(self.anchor.extent.z))
    let material = SCNMaterial()
    material.diffuse.contents = UIImage(named: "overlay_grid.png")
    self.planeGeometry.materials = [material]
    let planeNode = SCNNode(geometry: self.planeGeometry)
    planeNode.physicsBody = SCNPhysicsBody(type.static, shape: SCNPhysicsShape(geometry: self.planeGeometry, options))
    planeNode.physicsBody?.categoryBitMask = BodyType.plane.rawValue
    planeNode.position = SCNVector3Make(anchor.center.x, 0, anchor.center.z)
    planeNode.transform = SCNMatrix4MakeRotation(Float(-Double.pi/2.0), 1.0, 0.0, 0.0)

    self.addChildNode(planeNode)
}

required init?(coder aDecoder: NSCoder){
    fatalError("init(coder:) has not been implemented")
}
}
```