AR Class 436 f21

Transform Matrixes

All of this works because transform matrixes hold so much information, and are amenable to math.

- transform matrix (change position, orientation, shape)
- point of view (rear camera)
- *transform* represents difference between where the camera is now, vs at start
- we want *loc* row, location of camera relative, *fwd* is aiming of camera relative to start

$$\begin{bmatrix} right_x & up_x & -fwd_x & loc_x \\ right_y & up_y & -fwd_y & loc_y \\ right_z & up_z & -fwd_z & loc_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

436kea

436kea does a little less, and a little more, than Ikea

- not as many items (1)
- find vertical surfaces to put posters

Real world vs Virtual world

- not out of whole cloth like VR

- virtual world uses same coordinates

Objects can get added in two ways:
- programmatically
- automatically.
  - feature points!
    - more useful to ARkit than developers
      - if point moves, probably means that the phone moved
  - axis!
  - horizontal planes (ARkit 1.0)
  - vertical planes (ARkit 1.5)

ARkit
- Adds anchors when finding planes
- location, orientation (normal)

This app, in response, will add:
- posters on vertical planes
- furniture on horizontal

Step 1: initialization:

- **x**, **z** specified by orientation of phone, **y** by gravity
- want both horizontal and vertical planes
- ambient light detector, used to decide how “bright” to make virtual objects

```
func createARConfiguration() -> ARConfiguration {
    let config = ARWorldTrackingConfiguration()
    config.worldAlignment = .gravity
    config.planeDetection = [.horizontal, .vertical]
    config.isLightEstimationEnabled = true
    return config
  }
```

Step 2: anchors

Once configured, ARkit will create anchors whenever planes detected. Can *respond* by defining:

- `renderer(_: didAdd: for)`
- gets called on every anchor found. Eventually we will use them to place objects...

```
guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

let planeType: String

    if planeAnchor.alignment == .horizontal {
      planeType = "horizontal"
    } else {
      planeType = "vertical"
    }
    print(String(format: "%@ %.2g %.2g %.2g\n", planeType, planeAnchor.center.x, planeAnchor.center.y, planeAnchor.center.z))
```

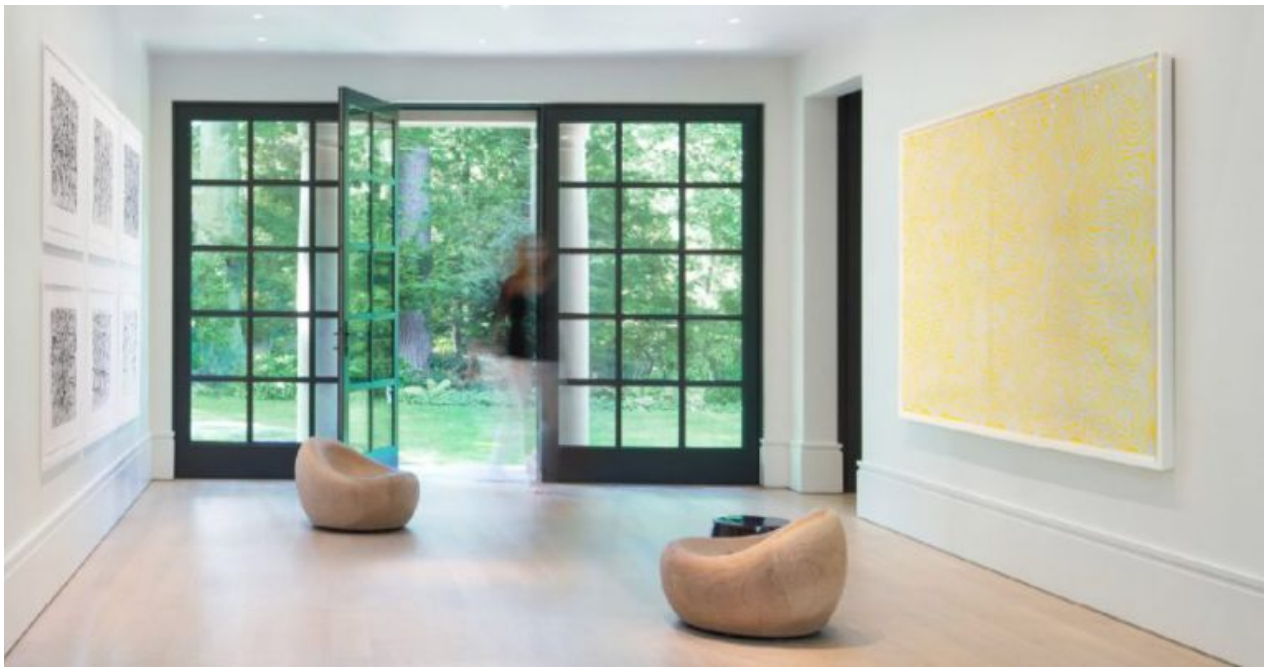
RUN
(notice **horizontal** and **vertical** prints in console)

Detecting Planes

ARKit uses sensors to find planes: camera and accelerometers
Plane needs two points to define:

- point on the plane
- vector normal to the surface.
  - easy for horizontal: gravity (the accelerometer)
  - hard for vertical
    - why can't we just use a vector perpendicular to gravity?
      - ?

    i. look for rectangular high-contrast borders (pictures)



- use known horizontal plane, trace upwards using normal
- ray-tracing:
  - trace line of the edge of a known horizontal plane into the distance
  - trace some lines parallel and above that
  - look for planes that intersect

Called by `renderer(didAdd: For:)` method. Width/height in anchor `extent`.

```
func drawPlaneNode(on node: SCNNode, for planeAnchor: ARPlaneAnchor) {
  // Create a plane node with the same position and size
  // as the detected plane.
  let planeNode = SCNNode(geometry: SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z)))
  planeNode.position = SCNVector3(planeAnchor.center.x,planeAnchor.center.y,planeAnchor.center.z)
  planeNode.geometry?.firstMaterial?.isDoubleSided = true
  // Align the plane with the anchor.
  //  pjk: SNCplanes are perp to anchors by default, need to rotate 90 deg clockwise aroundx axis
```

```
  planeNode.eulerAngles = SCNVector3(-Double.pi / 2,0,0)
  if planeAnchor.alignment == .horizontal {
    planeNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "grid")
    planeNode.name = "horizontal"
  } else {
    planeNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "opusa")
    planeNode.name = "vertical"
  }
  // Give the plane node the appropriate surface.
  node.addChildNode(planeNode)
  appState = .readyToFurnish
}
```

Go to `renderer. didAdd` and add:

- `drawPlaneNode(on: node, for: planeAnchor)`

RUN

- point at floor: grid “Place furniture here”
- vertical plane: “Should see Opus.

Adapting

- Think open-world games: more stuff pops up when you travel
- Analogy here is:
  - ARkit changes mind about a surface as the camera moves:
    - plane position or size might be different than first estimate
    - remove when camera moves too far away

- case one

```
  func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor) {
```

```
    guard let planeAnchor = anchor as? ARPlaneAnchor else {return}
    node.enumerateChildNodes { (childNode, _) in
      childNode.removeFromParentNode()
    }
    drawPlaneNode(on: node, for: planeAnchor)
  }
```

- case two

```
  func renderer(_ renderer: SCNSceneRenderer, didRemove node: SCNNode, for anchor: ARAnchor) {
    // We only want to deal with plane anchors.
    guard anchor is ARPlaneAnchor else { return }
    // Remove any children this node may have.
    node.enumerateChildNodes { (childNode, _) in
      childNode.removeFromParentNode()
  }
```

Hit Testing

Need to answer two tasks:

- Verify that user looking at a horizontal surface right now.
- Add furniture when user taps on plane.

Hit Testing:

> *Do these 2D coords in the view correspond a real OR virtual object in the AR scene?*

- We are really trying to figure out whether whether the user is actually indicating a detected plane.
- ARKit gives you back a vector of detected objects in the right direction, in order of distance from camera, each containing coords, size, orientation.

Or:

- need to tell which way user looking
- need to see if detected planes along that line

When to tell the user “Click on something to place furniture”

Given our app, this translates into “let’s find all detected planes in view”:

- collect planes in view
  - can’t look *everywhere*, (millions of possibilities)
  - this app picks 25 points and tries each
  - just for the message up top

```
  func isAnyPlaneInView() -> Bool {
    let screenDivisions = 5 - 1
    let viewWidth = view.bounds.size.width
    let viewHeight = view.bounds.size.height

    for y in 0...screenDivisions {
      let yCoord = CGFloat(y) / CGFloat(screenDivisions) * viewHeight
      for x in 0...screenDivisions {
        let xCoord = CGFloat(x) / CGFloat(screenDivisions) * viewWidth
        let point = CGPoint(x: xCoord, y: yCoord)

        // Perform hit test for planes.
        let hitTest = sceneView.hitTest(point, types: .estimatedHorizontalPlane)
        if !hitTest.isEmpty {
          return true
        }
      }
    }
```

```
    return false
  }
```

RUN: status area on top of screen changes according to whether looking at a horizontal plane or not.

See if User Clicked on One

```
  @objc func handleScreenTap(sender: UITapGestureRecognizer) {
    // Find out where the user tapped on the screen.
    let tappedSceneView = sender.view as! ARSCNView
    let tapLocation = sender.location(in: tappedSceneView)

    // Find all the detected planes that would intersect with
    // a line extending from where the user tapped the screen.
    let planeIntersections = tappedSceneView.hitTest(tapLocatio
n, types: [.estimatedHorizontalPlane])

    // If the closest of those planes is horizontal,
    // put the current furniture item on it.
    if !planeIntersections.isEmpty {
      addFurniture(hitTestResult: planeIntersections.first!)
    }
  }
```

```
  func addFurniture(hitTestResult: ARHitTestResult) {
    // Get the real-world position corresponding to
    // where the user tapped on the screen.
    let transform = hitTestResult.worldTransform
    let positionColumn = transform.columns.3
```

```
    let initialPosition = SCNVector3(positionColumn.x,
                                     positionColumn.y, positionC
olumn.z)

    // Get the current furniture item, correct its position if n
ecessary,
    // and add it to the scene.
    let node = bookshelfNode()
    node.position = initialPosition
    sceneView.scene.rootNode.addChildNode(node)
  }
```

```
func bookshelfNode() -> SCNNode {
  let scene = SCNScene(named: "SceneAssets.scnassets/furniture.s
cn")
  let node = (scene?.rootNode.childNode(withName: "bookcase", re
cursively: false))!
  return node
}
```

RUN

Conclusions

- this stuff is not that hard (at least to get a prototype running)
-