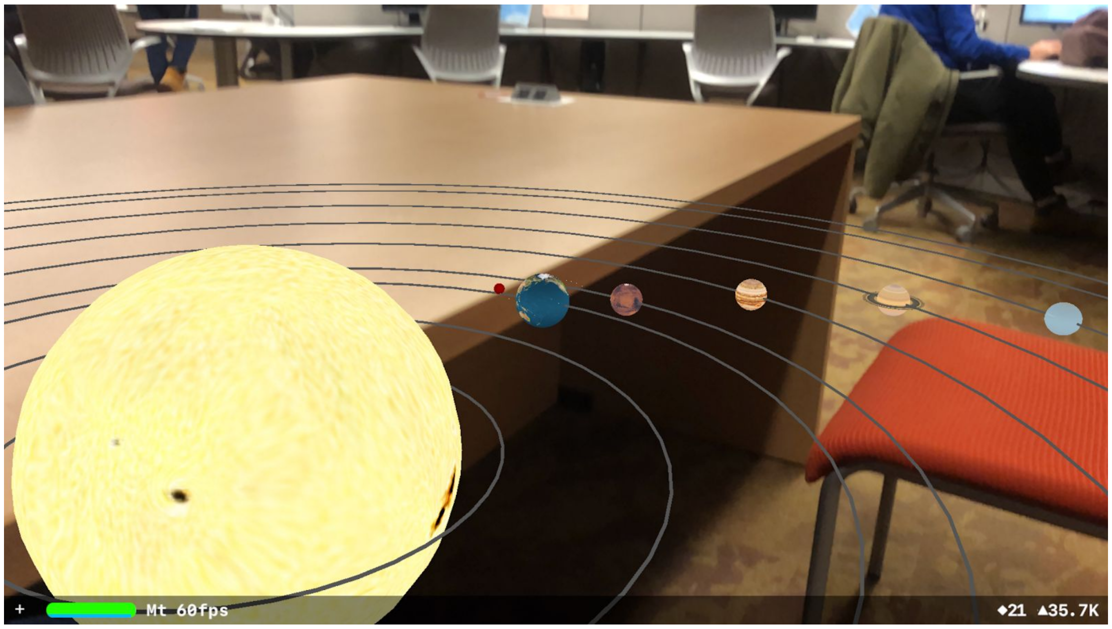
Solar System SceneKit

A Demo App by: Cole Sellers + Clay Negen

Overview

SceneKit is a high-level 3D graphics framework that helps you create 3D animated scenes and effects in your apps. It incorporates a physics engine, a particle generator, and easy ways to script the actions of 3D objects so you can describe your scene in terms of its content — geometry, materials, lights, and cameras — then animate it by describing changes to those objects. We have created a tutorial that will teach you how to start learning the SceneKit and all it incorporates all in one app! Our demo app is a solar system comprised of the plants and star found in the milky way galaxy displayed as Augmented Reality through the camera on an IPhone. The solar system is fully equipped with motion, texture mapping (to make the planets look real) and **add new features.** We utilize Xcode to create our IOS project and Swift to program the application. We will be taking your through step by step on how to create the project, the planets, the motion, and even making your solar system look like our very own.

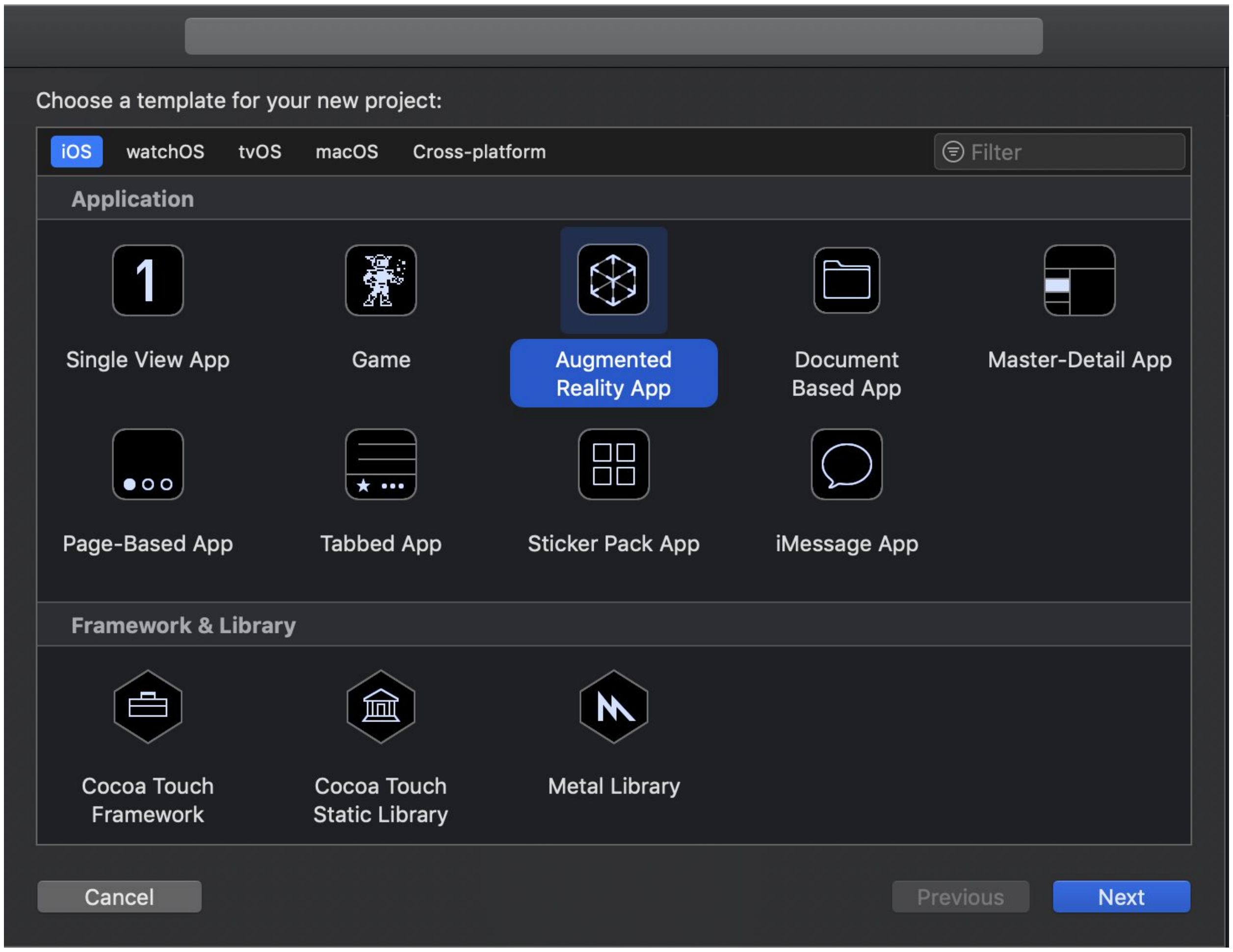
We will guide you all the way to creating a Solar System:



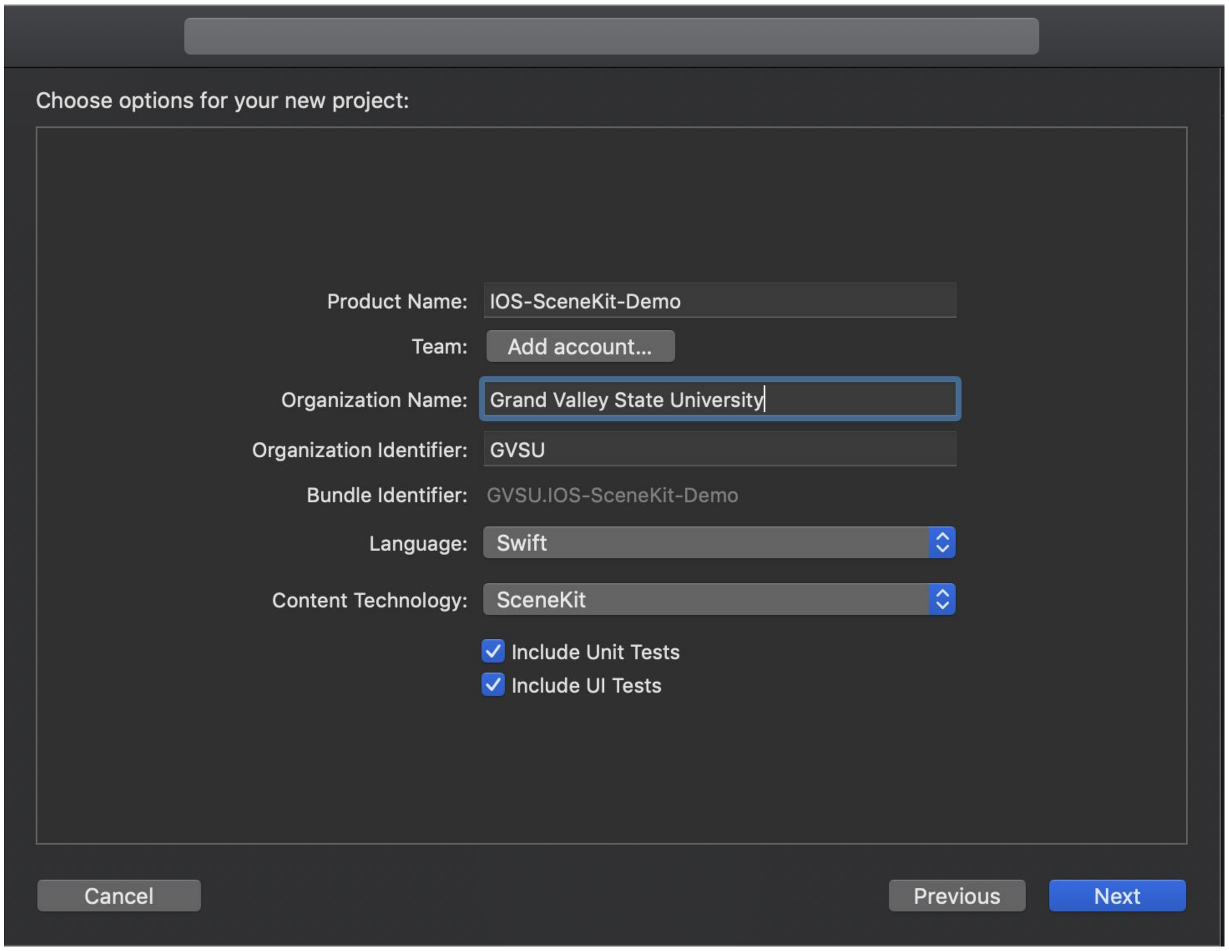
Getting Started

First you will need to download XCode from the apple website. This can be found at https://developer.apple.com/xcode/. We recommend XCode 10 or above, but we will be doing this tutorial on the newest version, XCode 11.

Once you have XCode downloaded open it up and select the “Create New Project” option, this will open up XCode’s main page. You will be faced with several templates to choose from, select the Augmented Reality App and click next.



From there, a project options panel will load, make sure SceneKit is currently selected under *Content Technology*.


Choose options for your new project:

Product Name: IOS-SceneKit-Demo

Team: Add account...

Organization Name: Grand Valley State University

Organization Identifier: GVSU

Bundle Identifier: GVSU.IOS-SceneKit-Demo

Language: Swift

Content Technology: SceneKit

✓ Include Unit Tests

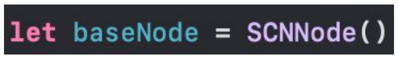
✓ Include UI Tests

Then we are ready to get started building our Solar System!

Coding Instructions

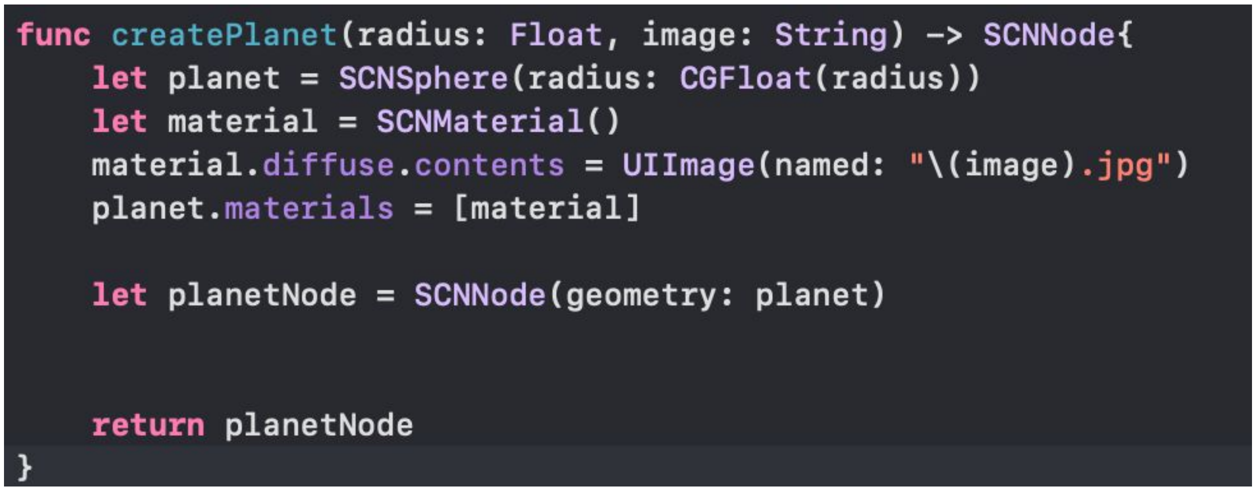
After you have completed the above steps to get an empty SceneKit application created you should be ready to begin.

The first step is to create a global base node using SCNNode() method. This method will create a point on your display in order to attach object, lights, or other displayable content to.



```swift
let baseNode = SCNNode()
```

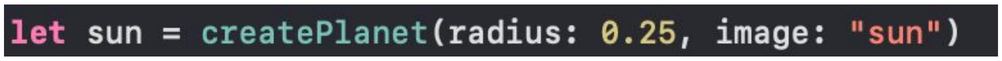
Next we will enter our viewDidLoad method. This is where we will create our solar system initially. IN order to create a planet, let's say the sun, you will need to create individual nodes and attach them to the initial base node we created above. In order to make this code more modular we created a method called createPlanet() it takes in 2 parameters; a radius and an image. The methods code snippet is attached below.



```swift
func createPlanet(radius: Float, image: String) -> SCNNode{
    let planet = SCNSphere(radius: CGFloat(radius))
    let material = SCNMaterial()
    material.diffuse.contents = UIImage(named: "\(image).jpg")
    planet.materials = [material]

    let planetNode = SCNNode(geometry: planet)


    return planetNode
}
```

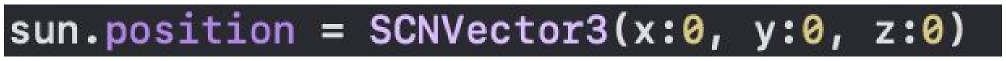
As you can see, this method takes in a Float value for the radius and a String for the image, it then returns a SSNNode. We then create a planet using the SCNSphere() method. This method takes in a CGFloat value and then returns a Sphere shaped node. We then create the "materials for this node using the SCNMaterial() function. This is a built in function that creates a set of shading attributes that define the appearance of a geometry's surface when rendered. We then added the UIImage from our assets folder of the planets terrain. After we add the materials to the planet variable we create the actual node for the specified planet and return that back to the viewDidLoad() method.

Once you return this node to your variable, in your case, the sun. you should have something very similar to this then to create the planet.



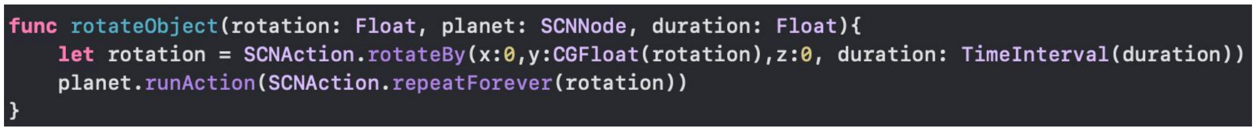
```swift
let sun = createPlanet(radius: 0.25, image: "sun")
```

Now that you have your planet node created, you need to name it. The details on why will be explained a bit later in this tutorial. In order to do this you will do something like sun.name = “sun”. Finally we have come to the point of positioning this node in relation to our basenode. As we will be adding the sun to it here in a little bit. The sun will be positioned in the center point of our screen so we will use the following code to place it there. (Note: the positions of x,y,x are in meters)



```swift
sun.position = SCNVector3(x:0, y:0, z:0)
```

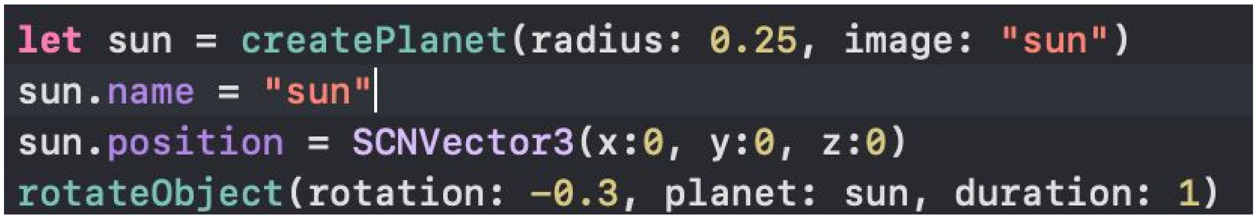
Once you have positioned your sun at the center of your screen, it's time to make it rotate. As we did above with the planet creation. This will happen on every planet so it is best to make it its own method.



```swift
func rotateObject(rotation: Float, planet: SCNNode, duration: Float){
    let rotation = SCNAction.rotateBy(x:0,y:CGFloat(rotation),z:0, duration: TimeInterval(duration))
    planet.runAction(SCNAction.repeatForever(rotation))
}
```

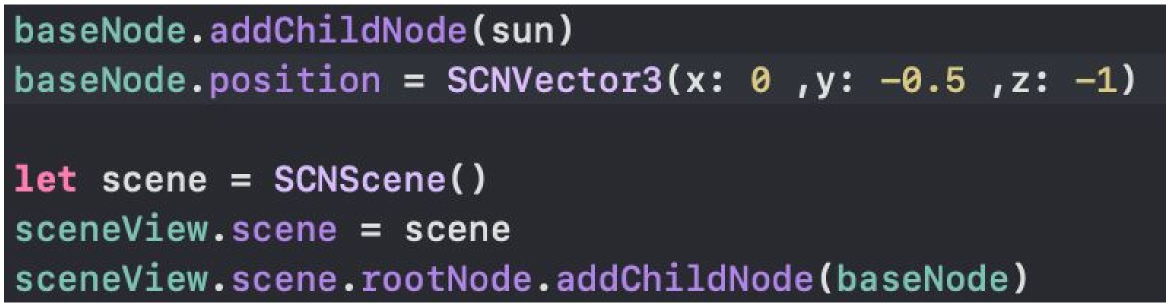
The above code snippet will handle the rotations of the planets. It takes three parameters, the rotation, the planet node, and the duration of the rotation. It creates and SCNAction that will rotate the following node by the given parameters and then it will add it to the planet node and make this action repeat forever.

At this point in your viewDidLoad() you should have something very similar to this in order to create your sun.



```swift
let sun = createPlanet(radius: 0.25, image: "sun")
sun.name = "sun"
sun.position = SCNVector3(x:0, y:0, z:0)
rotateObject(rotation: -0.3, planet: sun, duration: 1)
```

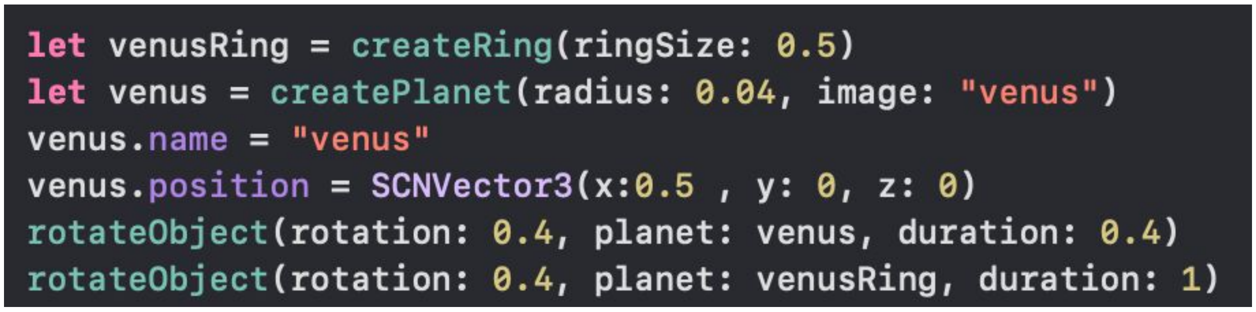
Now it's time to actually add this node to the basenode so it will display on the screen.
In order to do this we will have to add the following code snippets.



```
baseNode.addChildNode(sun)
baseNode.position = SCNVector3(x: 0 ,y: -0.5 ,z: -1)

let scene = SCNScene()
sceneView.scene = scene
sceneView.scene.rootNode.addChildNode(baseNode)
```

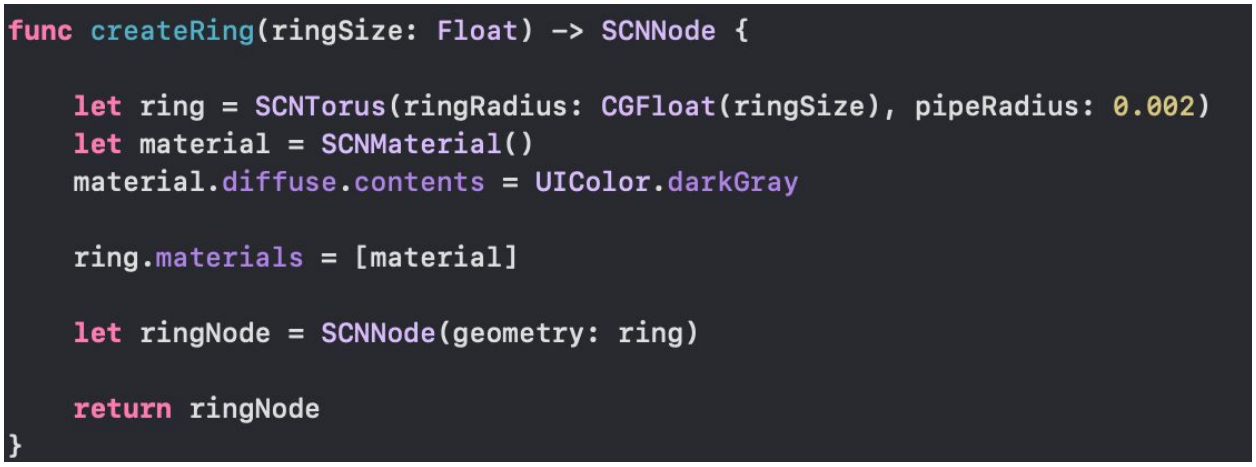
The above lines will add the sun to the basenode, position the basenode at the following location on your screen, create a new scene, add it to your sceneView, then add the basenode to the sceneView.

The next step is adding a planet on a ring orbiting the sun. Since we made our code modular when creating the sun this step should be pretty similar. Below is a screenshot it takes in order for a planet to be created orbiting the sun.



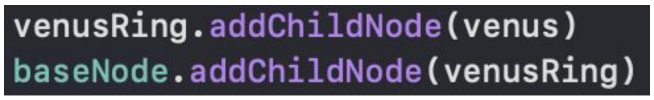
```
let venusRing = createRing(ringSize: 0.5)
let venus = createPlanet(radius: 0.04, image: "venus")
venus.name = "venus"
venus.position = SCNVector3(x:0.5 , y: 0, z: 0)
rotateObject(rotation: 0.4, planet: venus, duration: 0.4)
rotateObject(rotation: 0.4, planet: venusRing, duration: 1)
```

Notice that the only real main difference is that we are now running the method createRing() with a parameter of the ring size and then running the rotateObject on this newly created ring. See below for a description for what the createRing() contains.



```
func createRing(ringSize: Float) -> SCNNode {

    let ring = SCNTorus(ringRadius: CGFloat(ringSize), pipeRadius: 0.002)
    let material = SCNMaterial()
    material.diffuse.contents = UIColor.darkGray

    ring.materials = [material]

    let ringNode = SCNNode(geometry: ring)

    return ringNode
}
```

As you can see this is very similar to creating a planet. The difference is that instead of creating a SCNSphere we are now creating a SCNTorus and then setting its parameters to the radius and the pipeRadius, or the size of the ring itself. After you have the above code implemented you will have the planet venus ready to be added to the baseNode we created earlier. In order to do this we will repeat the process of adding to the baseNode with one extra step.



```
venusRing.addChildNode(venus)
baseNode.addChildNode(venusRing)
```

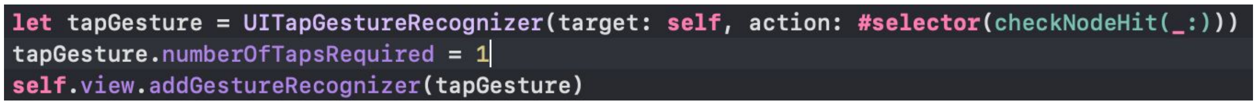
As you can see we first need to actually add venus to the ring we created for it, then we add that ring node to the basenode. Once this step is completed you will have a sun node and venus will be orbiting it.

Now if you want to add the remaining planets in the solar system you would repeat the above step that we did for venus for the others. (NOTE: you need to make sure that the planet.positions x-axis is the same as the ring radius it belongs to.)

After you have all of the planets added you will notice there are a few things missing, like the moon and Saturn's ring. Lets go ahead and add those now. In order to add the moon you will need to treat it similar to how you added a planet to the basenode. You will need to repeat the ring steps and creation of a planet, add the planet to the ring, then add the ring to the earth.

Now we need to add the feature where if you click on any of the planets it will zoom in on the planet and make a transparent image with text display a bit above and behind the planet.

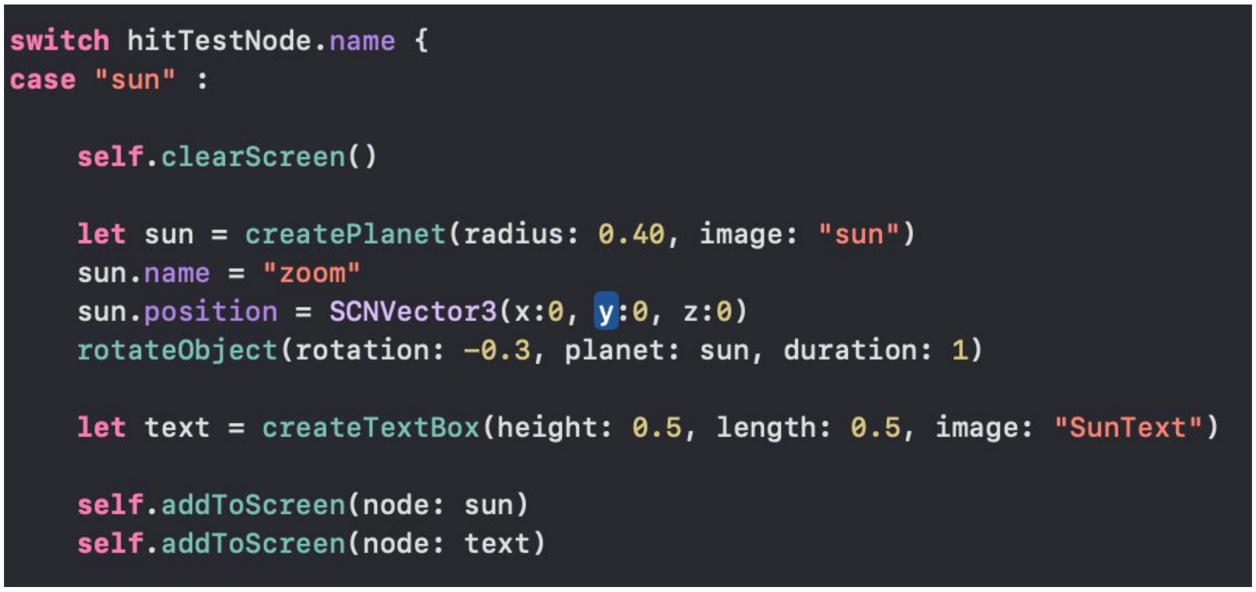
In order to do this we will need to set up a UITapGestureRecognizer. The following snippet of code should be placed at the bottom of you viewDidLoad() method.



```
let tapGesture = UITapGestureRecognizer(target: self, action: #selector(checkNodeHit(_:)))
tapGesture.numberOfTapsRequired = 1
self.view.addGestureRecognizer(tapGesture)
```

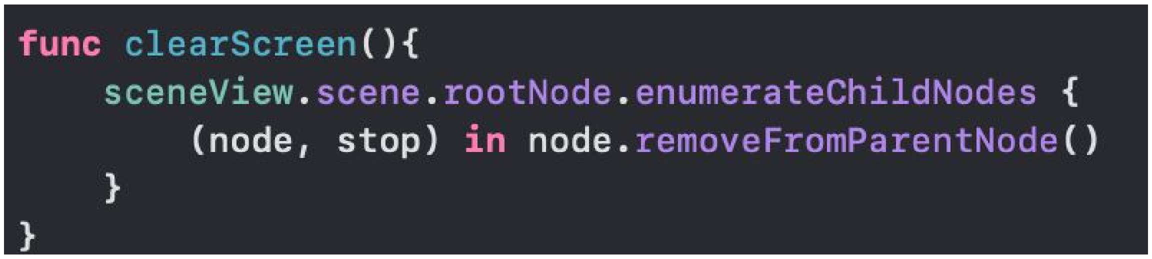
This snippet will create the UITapGestureRecognizer and make it check if a node was hit checkNodeHit is a built in function in the sceneKit and does exactly what the name suggests. It detects if a node was tapped on and if so fires off the UITapGestureRecognizer. You will also want to set the number of taps to one and add it to your view.

This is where the naming of the initial nodes came in handy because in the checkNodeHit() method we will run a switch case to determine what planet needs to be enlarged. For this example we will show you the sun. The same steps can be repeated for every planet. The following code snippet will fire if the sun node was clicked on.



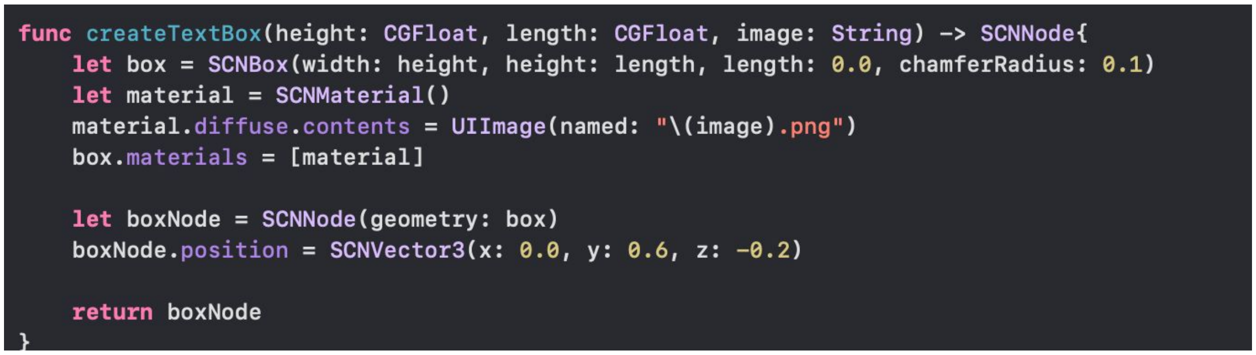
```
switch hitTestNode.name {
case "sun" :

    self.clearScreen()

    let sun = createPlanet(radius: 0.40, image: "sun")
    sun.name = "zoom"
    sun.position = SCNVector3(x:0, y:0, z:0)
    rotateObject(rotation: -0.3, planet: sun, duration: 1)

    let text = createTextBox(height: 0.5, length: 0.5, image: "SunText")

    self.addToScreen(node: sun)
    self.addToScreen(node: text)
```

As you can see the switch case is being run on the node name. Once the case enters the sun portion it will run the clearScreen() method.



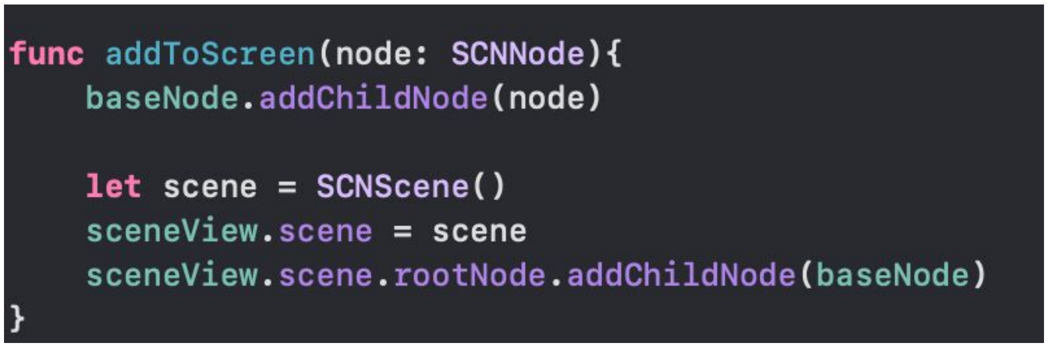
```
func clearScreen(){
    sceneView.scene.rootNode.enumerateChildNodes {
        (node, stop) in node.removeFromParentNode()
    }
}
```

This method does exactly what the name suggests it takes every node that is added to the parent and removes them from the scene. The next step is to do the same step we used to create the sun node initially and make it rotate. The only difference between adding the sun to the solar system and adding it individually is that it is a larger size. Next we want to add the text image above this. In order to do this we created a createTextBox() method that took the height, length, and image as parameters. Attached is the code snippet for the createTextBox() method.



```
func createTextBox(height: CGFloat, length: CGFloat, image: String) -> SCNNode{
    let box = SCNBox(width: height, height: length, length: 0.0, chamferRadius: 0.1)
    let material = SCNMaterial()
    material.diffuse.contents = UIImage(named: "\(image).png")
    box.materials = [material]

    let boxNode = SCNNode(geometry: box)
    boxNode.position = SCNVector3(x: 0.0, y: 0.6, z: -0.2)

    return boxNode
}
```

As you can tell it very similar to adding the other nodes except you are doing a SCNBox this time. Once that is returned you can see we are adding both of the newly created nodes to the screen and to do so we created a method called addToScreen() taking in one parameter, the node being added. This method as pictured below is the same as in the viewDidLoad() when it comes to adding a node to the screen.



```swift
func addToScreen(node: SCNNode){
    baseNode.addChildNode(node)

    let scene = SCNScene()
    sceneView.scene = scene
    sceneView.scene.rootNode.addChildNode(baseNode)
}
```

After the above step is completed for every planet you should have a fully functional solar system with the ability to enlarge any planet. The code for the following project is located in a repo here: https://github.com/ClayNegen/IOS-AR-Solar-System.

Sources:

https://developer.apple.com/scenekit/

https://developer.apple.com/xcode/