Wyatt Davis and Sarah Rockow

CIS 357.01

10 December 2021

Final Project

# Table of Contents

## Overview

The topic of this project is ARCore on the Android platform. The goal of the project was to build an entertainment app inspired by Tamagotchis. Using ARCore, a 3d model can be placed on a surface and react to the user's actions. To support ARCore, Sceneform is extremely helpful as it provides a variety of functions, such as model rendering and animations. The app supports simple actions, which are limited to changing the models, moving the models, and animating the models to dance. The user can choose between two models - a halloween-themed pumpkin man and a tiger - and the user can also choose to animate the models. Currently, the app is constrained to only allow two models to be present at the same time, but there is an easy way to allow for more models in one session.

ARCore supports augmented reality features for Android Studio. Using only ARCore, developers can create apps to place virtual 3D models into the physical world; however, there are other platform features that support and enhance ARCore, notably Sceneform. Sceneform supports ARCore through animations and allows new developers to bypass OpenGL for Android, which can be challenging to learn.

The base code for this app was provided by Scene View Open Community on Github (https://github.com/SceneView). Sceneform SDK for android has been archived with version 1.16.0. The Scene View Open Community is maintaining support for Sceneform until Google releases an alternative. While the changes made to the initial repository are in Kotlin, many of the original files in Sceneform were written in Java.

## Getting Started

| | |
|---|---|
| **IDE Version** | Android Studio 2020.3.1 (or updated) |
| **Minimum SDK** | 27 - Android 8.1 (Oreo) |
| **Dependencies** | com.google.ar:core:1.28.0 |

When running the project using an Android device, developers may need to disable antivirus software as it can affect Android Studio's exports. An example is MalwareBytes, which blocks Android Studio from installing the app on an Android device.

## Coding Instructions

The app can be run using either the Android Studio emulator with an AR compatible phone, or with an AR compatible Android. A complete list of compatible Android devices can be found at https://developers.google.com/ar/devices.

Using the forked Sceneform repository, download the core, sceneform, and ux folders as they are necessary to support ARCore.

To run this project, clone https://github.com/daviswygvsu/CIS357ARProject/tree/master. Open in Android Studio, and using a compatible device, run the project from the samples/ar-model-viewer folder.

---

## Core

### Google

This folder contains the original sceneform folders and files that were archived by Google. The original sceneform supported animations, collisions, rendering, and placement. The classes and helper classes within the folder were written in Java, not Kotlin.

### Gorisse.thomas.sceneform

The core folder contains the maintained support for sceneform. In the 'gorisse.thomas.sceneform' folder, the dependencies for environment, light, materials, scenes, and ARCore are maintained together. This folder contains the files that allow the virtual models to interact with the real world by recognizing the environment and helping to anchor them.

---

## Sceneform

This folder contains a Manifest file listing the project package and a BuildConfig file listing the project package again. This folder is necessary as it contains a global debugger and allows access to the other files supporting sceneform.

---

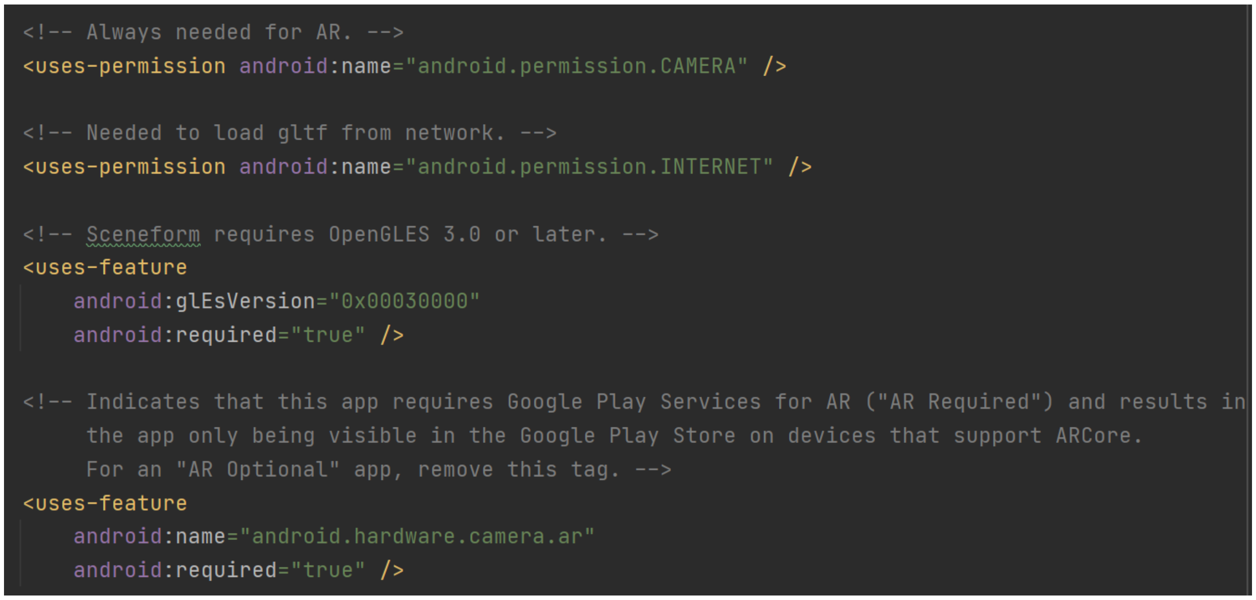
## Ux

The "ux" folder for this project contains some unnecessary components as it was also meant to support other features of ARCore such as face recognition and rendering. The focus for this project was 3D models. The ux folder contains relevant support for ArFragments, Gestures, and Nodes for placement. The features in this folder help make it intuitive for users to place objects by providing them with a grid where they can place models.

## Application Files

### Manifest

The Manifest file has the default activities and intents, but it also includes some features and permissions which are essential for ARCore. The following permissions are for using the camera to place objects and recognize the surroundings, the Internet to load models, and for OpenGLES to function properly.



```
<!-- Always needed for AR. -->
<uses-permission android:name="android.permission.CAMERA" />

<!-- Needed to load gltf from network. -->
<uses-permission android:name="android.permission.INTERNET" />

<!-- Sceneform requires OpenGLES 3.0 or later. -->
<uses-feature
    android:glEsVersion="0x00030000"
    android:required="true" />

<!-- Indicates that this app requires Google Play Services for AR ("AR Required") and results in
     the app only being visible in the Google Play Store on devices that support ARCore.
     For an "AR Optional" app, remove this tag. -->
<uses-feature
    android:name="android.hardware.camera.ar"
    android:required="true" />
```

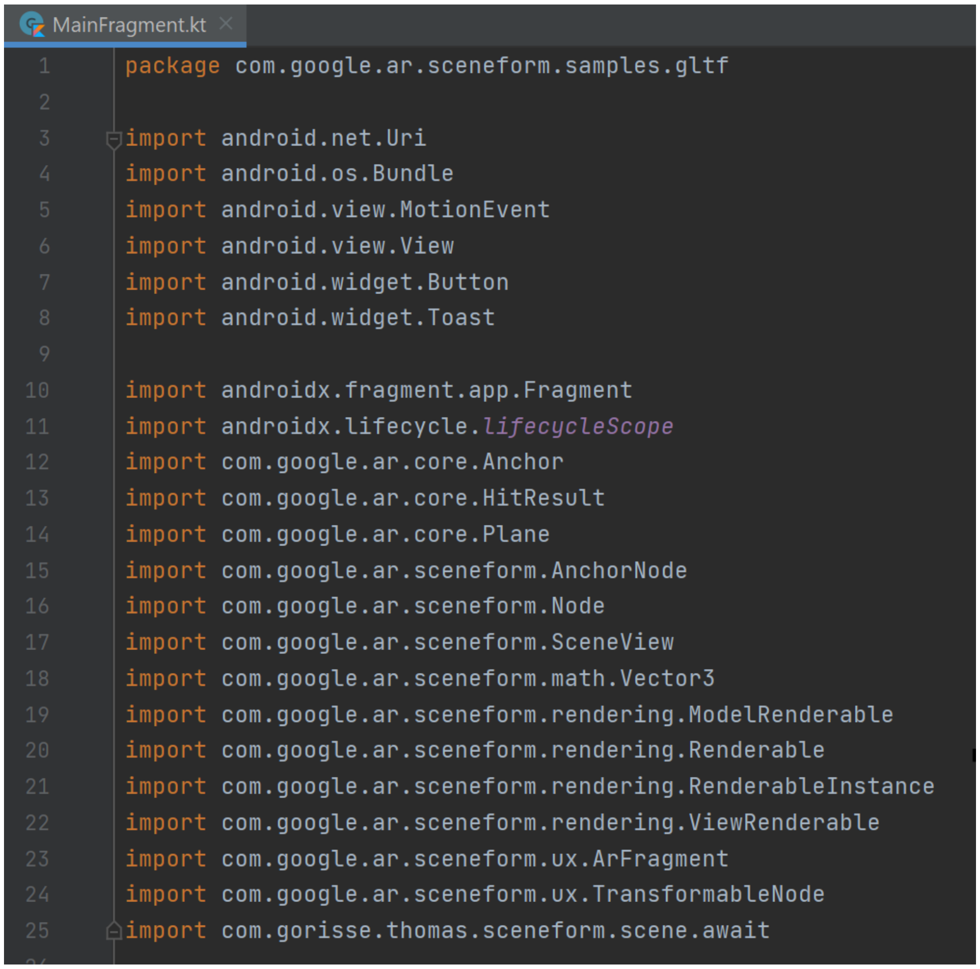
### Activity

The Activity file contains the opening scene which gives ARCore time to load the actual surroundings and start recognizing surfaces. Its main purpose is to serve as a buffer.

### MainFragment

The MainFragment file holds the actionable code. It holds the renderers, the view, and the activity.

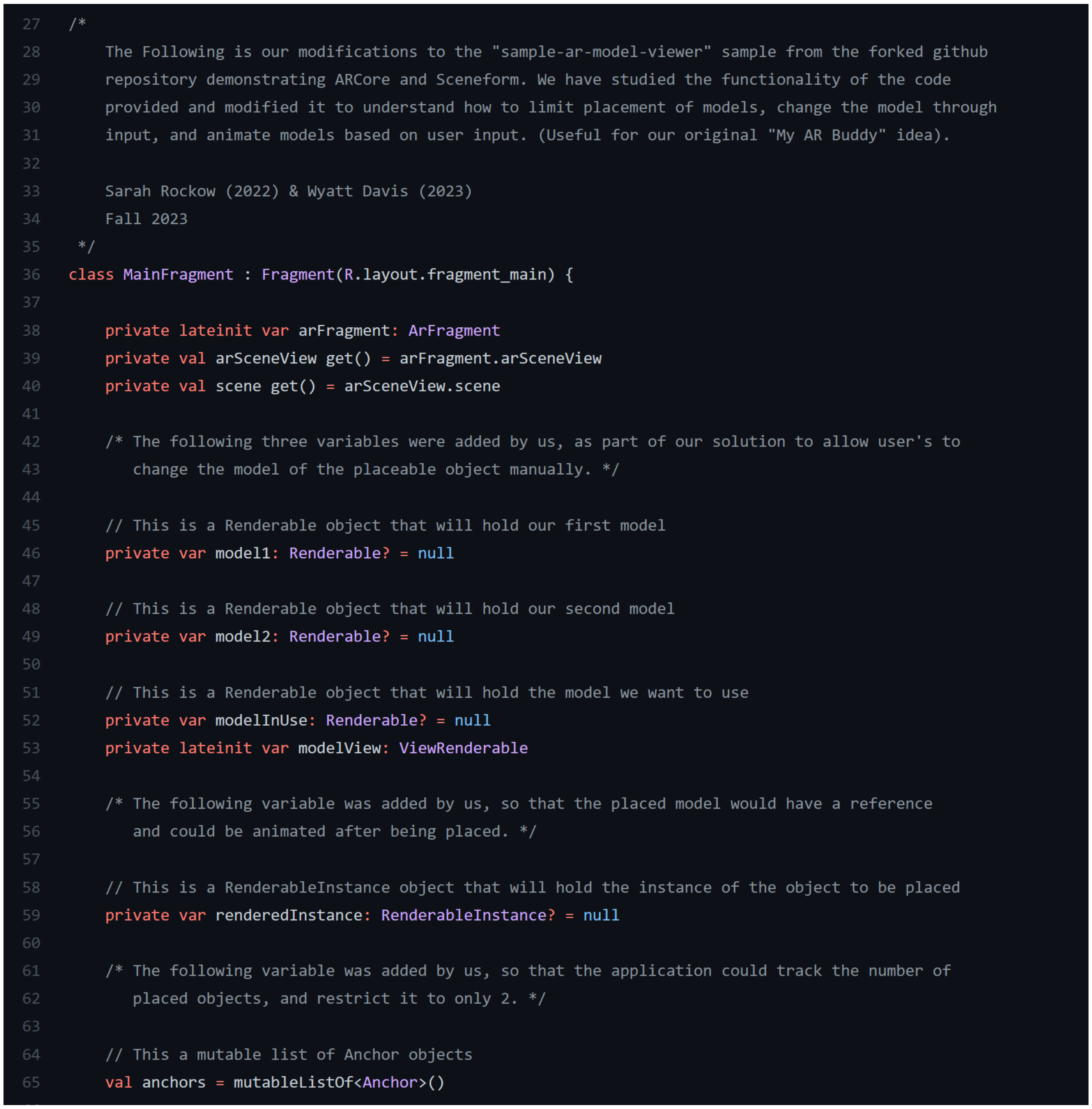
These are the imports for the MainFragment file. Except for 'com.google.ar.core.Anchor,' all of the imports were included in the supported Scene View repository.



MainFragment.kt

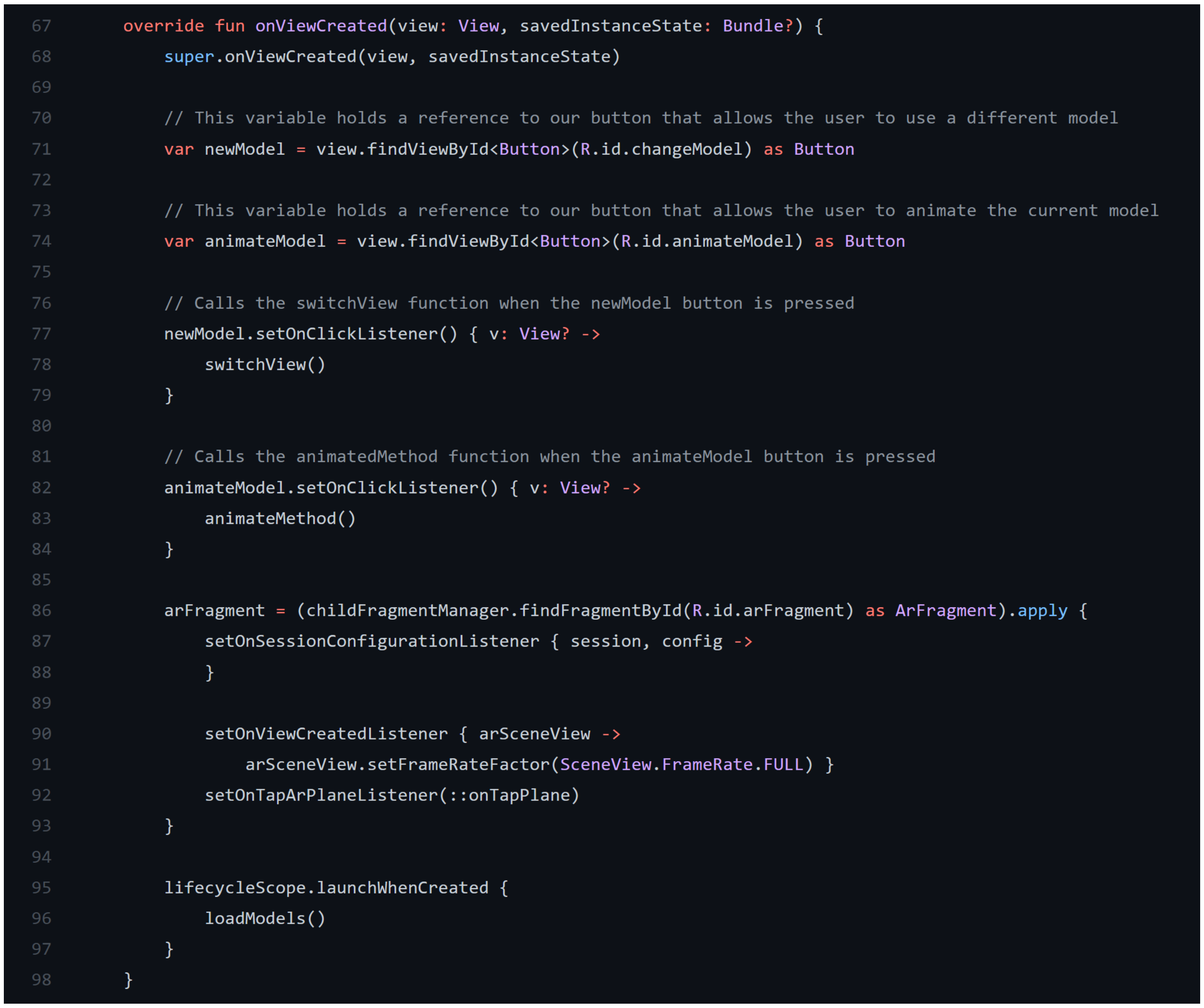
```
package com.google.ar.sceneform.samples.gltf

import android.net.Uri
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.widget.Button
import android.widget.Toast

import androidx.fragment.app.Fragment
import androidx.lifecycle.lifecycleScope
import com.google.ar.core.Anchor
import com.google.ar.core.HitResult
import com.google.ar.core.Plane
import com.google.ar.sceneform.AnchorNode
import com.google.ar.sceneform.Node
import com.google.ar.sceneform.SceneView
import com.google.ar.sceneform.math.Vector3
import com.google.ar.sceneform.rendering.ModelRenderable
import com.google.ar.sceneform.rendering.Renderable
import com.google.ar.sceneform.rendering.RenderableInstance
import com.google.ar.sceneform.rendering.ViewRenderable
import com.google.ar.sceneform.ux.ArFragment
import com.google.ar.sceneform.ux.TransformableNode
import com.gorisse.thomas.sceneform.scene.await
```

Private variables



```kotlin
/*
    The Following is our modifications to the "sample-ar-model-viewer" sample from the forked github
    repository demonstrating ARCore and Sceneform. We have studied the functionality of the code
    provided and modified it to understand how to limit placement of models, change the model through
    input, and animate models based on user input. (Useful for our original "My AR Buddy" idea).

    Sarah Rockow (2022) & Wyatt Davis (2023)
    Fall 2023
 */
class MainFragment : Fragment(R.layout.fragment_main) {

    private lateinit var arFragment: ArFragment
    private val arSceneView get() = arFragment.arSceneView
    private val scene get() = arSceneView.scene

    /* The following three variables were added by us, as part of our solution to allow user's to
       change the model of the placeable object manually. */

    // This is a Renderable object that will hold our first model
    private var model1: Renderable? = null

    // This is a Renderable object that will hold our second model
    private var model2: Renderable? = null

    // This is a Renderable object that will hold the model we want to use
    private var modelInUse: Renderable? = null
    private lateinit var modelView: ViewRenderable

    /* The following variable was added by us, so that the placed model would have a reference
       and could be animated after being placed. */

    // This is a RenderableInstance object that will hold the instance of the object to be placed
    private var renderedInstance: RenderableInstance? = null

    /* The following variable was added by us, so that the application could track the number of
       placed objects, and restrict it to only 2. */

    // This a mutable list of Anchor objects
    val anchors = mutableListOf<Anchor>()
```

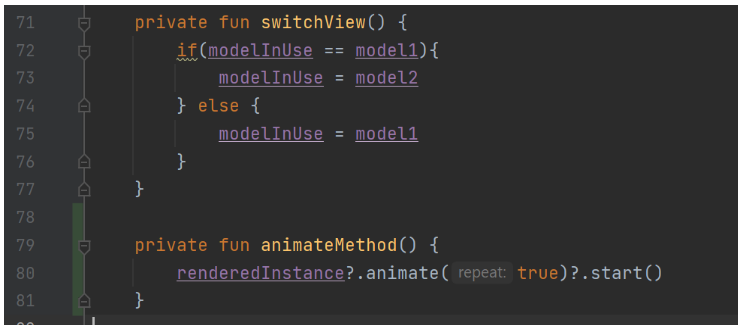
## OnViewCreated()



```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // This variable holds a reference to our button that allows the user to use a different model
        var newModel = view.findViewById<Button>(R.id.changeModel) as Button

        // This variable holds a reference to our button that allows the user to animate the current model
        var animateModel = view.findViewById<Button>(R.id.animateModel) as Button

        // Calls the switchView function when the newModel button is pressed
        newModel.setOnClickListener() { v: View? ->
            switchView()
        }

        // Calls the animatedMethod function when the animateModel button is pressed
        animateModel.setOnClickListener() { v: View? ->
            animateMethod()
        }

        arFragment = (childFragmentManager.findFragmentById(R.id.arFragment) as ArFragment).apply {
            setOnSessionConfigurationListener { session, config ->
            }

            setOnViewCreatedListener { arSceneView ->
                arSceneView.setFrameRateFactor(SceneView.FrameRate.FULL) }
            setOnTapArPlaneListener(::onTapPlane)
        }

        lifecycleScope.launchWhenCreated {
            loadModels()
        }
    }
```

In the OnViewCreated method, the main fragment, arFragment, is created. This fragment contains the scene where the models will be placed. Two buttons, newModel and animateModel, are also declared and referenced in the code. These buttons use two private methods that will allow the user to switch the models being used or to animate the models.

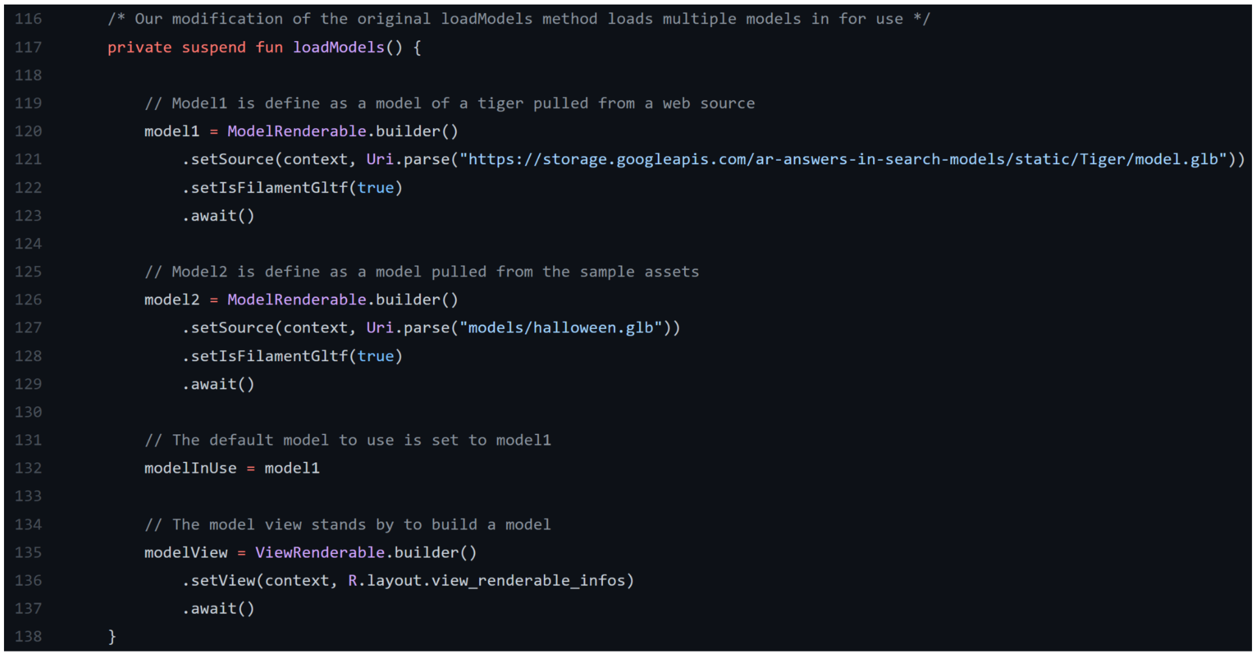
## Actions



```
    private fun switchView() {
        if(modelInUse == model1){
            modelInUse = model2
        } else {
            modelInUse = model1
        }
    }

    private fun animateMethod() {
        renderedInstance?.animate( repeat: true)?.start()
    }
```

These two functions are used to switch the models and animate them.

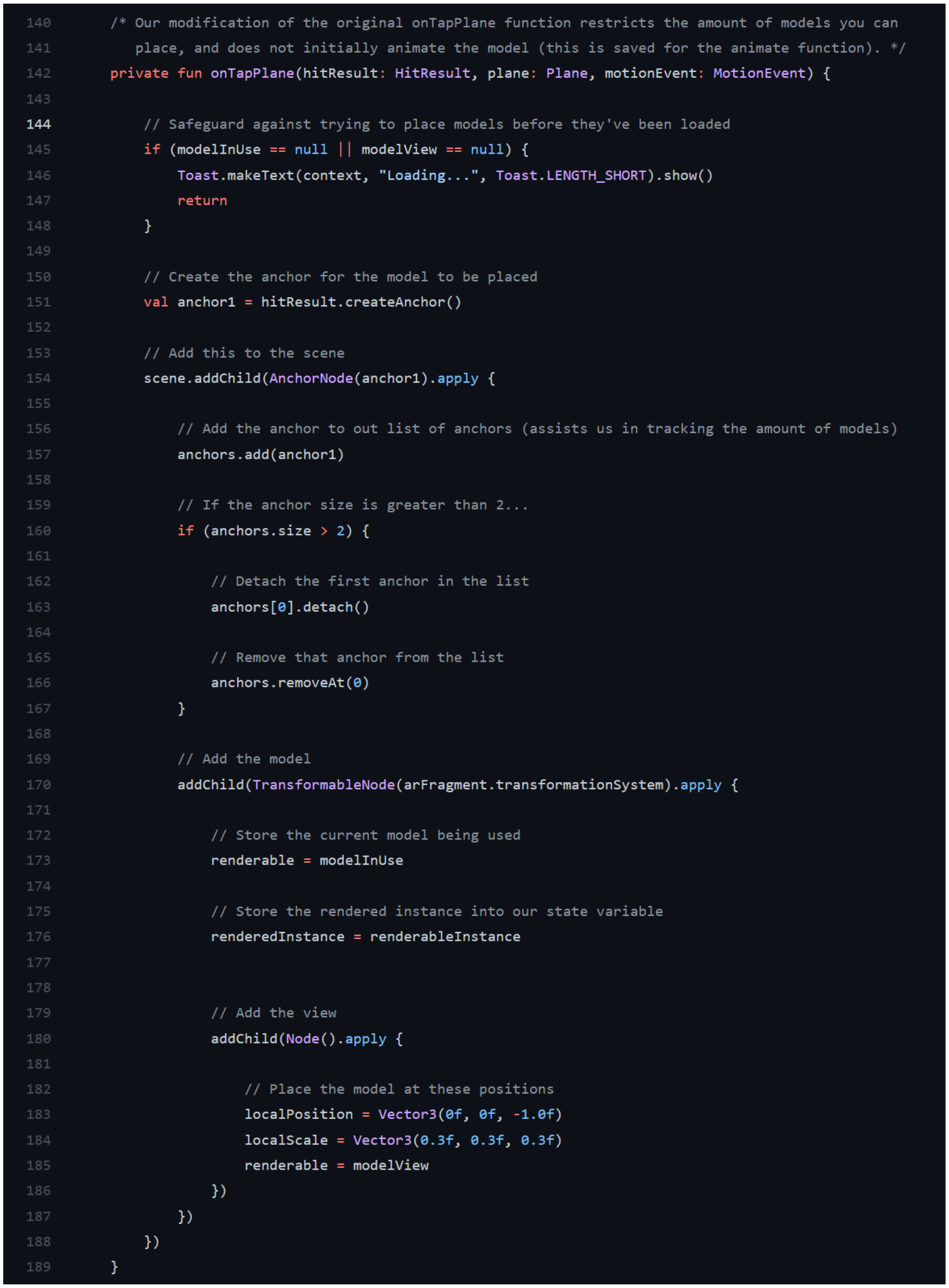
## loadModels()



```
    /* Our modification of the original loadModels method loads multiple models in for use */
    private suspend fun loadModels() {

        // Model1 is define as a model of a tiger pulled from a web source
        model1 = ModelRenderable.builder()
            .setSource(context, Uri.parse("https://storage.googleapis.com/ar-answers-in-search-models/static/Tiger/model.glb"))
            .setIsFilamentGltf(true)
            .await()

        // Model2 is define as a model pulled from the sample assets
        model2 = ModelRenderable.builder()
            .setSource(context, Uri.parse("models/halloween.glb"))
            .setIsFilamentGltf(true)
            .await()

        // The default model to use is set to model1
        modelInUse = model1

        // The model view stands by to build a model
        modelView = ViewRenderable.builder()
            .setView(context, R.layout.view_renderable_infos)
            .await()
    }
```

This loadModels() function creates two models - the tiger model and the halloween model. After creating the models, the function sets the modelInUse variable as model1 to be used outside of the function. To add a different model, create a var of type Renderable? outside of the function and then set it equal to ModelRenderable.builder(). Set the model source by either entering the hyperlink where the glb file can be downloaded or by calling it from the models file.

onTapPlane()



```kotlin
/* Our modification of the original onTapPlane function restricts the amount of models you can
   place, and does not initially animate the model (this is saved for the animate function). */
private fun onTapPlane(hitResult: HitResult, plane: Plane, motionEvent: MotionEvent) {

    // Safeguard against trying to place models before they've been loaded
    if (modelInUse == null || modelView == null) {
        Toast.makeText(context, "Loading...", Toast.LENGTH_SHORT).show()
        return
    }

    // Create the anchor for the model to be placed
    val anchor1 = hitResult.createAnchor()

    // Add this to the scene
    scene.addChild(AnchorNode(anchor1).apply {

        // Add the anchor to out list of anchors (assists us in tracking the amount of models)
        anchors.add(anchor1)

        // If the anchor size is greater than 2...
        if (anchors.size > 2) {

            // Detach the first anchor in the list
            anchors[0].detach()

            // Remove that anchor from the list
            anchors.removeAt(0)
        }

        // Add the model
        addChild(TransformableNode(arFragment.transformationSystem).apply {

            // Store the current model being used
            renderable = modelInUse

            // Store the rendered instance into our state variable
            renderedInstance = renderableInstance


            // Add the view
            addChild(Node().apply {

                // Place the model at these positions
                localPosition = Vector3(0f, 0f, -1.0f)
                localScale = Vector3(0.3f, 0.3f, 0.3f)
                renderable = modelView
            })
        })
    })
}
```

Inside of the last function, onTapPlane, the function checks to make sure that a model is available and that it has been rendered successfully. Once the function finds the model and view, an anchor is created and that anchor is added to a list of anchors. Anchors are used to ground the models so that they are in a fixed location. The Anchor class had to be imported into the repository in order to access the list of anchors, which was necessary to limit the amount of anchors.

Inside the first addChild method, an anchor is created and then an if statement is used to limit the number of anchors, and models by extension. If the limit is exceeded, then a least recently used (LRU) policy is enacted by removing the oldest instance of an anchor. Inside the second addChild method, the model is set using the Renderable class. Afterwards, a relative position is created for the model. Because of this position, when the user walks closer or farther away from the model, the model will have a corresponding adjustment in size.

---

### Models

Sceneform will support animations with .gfb, .gltf, and .fbk extensions. In this project, .gfb files were used. The first .gfb was imported into the project, while the second was loaded from Google's apis. If the .gfb file is corrupted, then it will not render correctly when it is loaded into the app. Online converters, such as AnyConv.com, will not accurately convert OBJ files to .gfb.

The models folder should hold all of the .gfb files used in the project. These files are then referenced and used in the MainActivity code when loading models.

---

# Further Discussion/Conclusions

While we based our app on a forked repository, another approach to AR for Android is OpenGL. Based on our research, OpenGL is harder to implement than Sceneform as it is less intuitive for new developers. It seems that OpenGL would include more customization than was included in this project. Given more time, we would have liked to have been able to compare OpenGL functionality with the archived Sceneform.

Earlier in the project, we were focusing on implementing InstantPlacement and the Depth API. However, the phone we used to test the app, a motorola z4, was not capable of running the Depth API. While InstantPlacement was useful in our first iteration, we did not include it as the Anchor class and subclasses were sufficient for our demo app.

To run our project, a user can clone our repository and run it using an Android compatible phone. This tutorial was based on the Scene View repository and the major changes we made included

(1) adding new models, (2) disabling and enabling animations, (3) limiting the amount of models in a session, and (4) allowing the user to switch between models in the same session.

While we are currently not planning on updating our project, our completed project can be found at https://github.com/daviswygvsu/CIS357ARProject/tree/master.