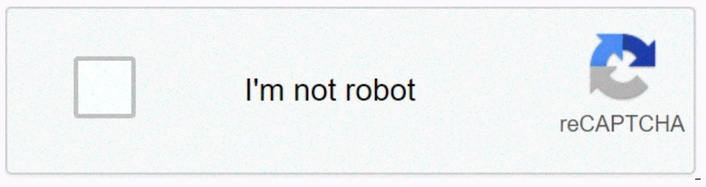

Continue

Augmented reality android apps tutorial

Update note: Zahidur Rahman Faisal updated this tutorial for Android 7.0, Android Studio 3.5 and Kotlin 1.3. Joe Howard wrote the original. Nowadays, most people who own a smartphone have played augmented reality-based games like Pokémon GO or Harry Potter: Wizards Unite. Have you ever wondered how those apps complement a wonderful world around you and keep your engagement? Well, find out by making your own augmented reality (AR) program. You can build augmented reality programs using OpenGL, Unity, and Unreal. In this tutorial, you will get started with Android ARCore by building on top of a modified version of the OpenGL sample app that Google provides. Note: This tutorial assesses that you are familiar with Kotlin. If you just start with Kotlin development, please check out our tutorial, Kotlin For Android: An Introduction, first. Introduction to ARCore On WWDC June 2017, Apple announced ARKit, its foray in the world of AR development. Two months later, Google announced ARCore, which withdrew it from the Tango indoor mapping project. Tango only works on specific devices that have a depth sensor, while ARCore is available on most modern Android devices. ARCore relies on three key mechanics to complement the real world around you: Movement track: ARCore determines both the position and orientation of a virtual – simulated – object in the real world using the phone's camera and sensor data. This is called the object's attitude. ARCore follows the attitude of the virtual objects in a scene while your phone moves. This allows it to give the objects out of the correct perspective according to your position. Environmental combat: ARCore can detect horizontal or vertical surfaces such as tables, floors or walls while processing input from camera from your device. Those detected are called aircraft. ARCore attaches virtual objects to an aircraft at a fixed location and orientation. Those fixed points are called anchors. Mild estimate: ARCore understands the lighting of the environment. It can then adjust the average intensity and color of virtual objects to place them under similar circumstances as the environment surrounding them. This makes the virtual objects look more realistic. The race to worsen this emerging technology is on, now more than ever. Get ready to explore the brave new — complement — world like a Viking! At the beginning, Vikings have cannons in fact? Whether they did or not, there is no reason why Vikings can't have cannons in augmented reality! :] Your goal for this tutorial is to supplement a scene - a Viking shows a cannon on a target - and projects the scene around you using your Android device. Start by downloading the project material by clicking on the Download Material button at the top or bottom of this Open the starting project in Android Studio 3.5 or later. The project will not be compiled at this time. Don't worry, it will be a while. Setting up the environment before you start building your scene, you need a device capable of run ARCore. Here are two options; you can choose the one that is right for you. Using a Real Device to run ARCore apps on a real device, you must check if your device has ARCore support. Google offers a full list of ARCore-backed devices. If you're lucky and find your device on this list, you're good to go! Open Google Play Store from your device and install Google Play services for AR. This service contains support libraries you need to run ARCore-based apps. If your current device doesn't support ARCore, don't worry. You can still run it on the Android Emulator! Using an Emulator to use ArCore on an emulator, you need to create an Android Virtual Device (AVD) that can run ARCore apps. Give it the following configuration: When you create your AVD, you must select a Pixel, Pixel 2 or Pixel 3 hardware profile. Set the system image to Oreo: API Level 27: x86: Android 8.1 (Google APIs) or a higher APIS level. Now set the rear camera to VirtualScene door to Verify Configuration · Show Advanced Settings · Camera Back and pick VirtualScene from the drop down list. Download next Google Play services for AR for your AVD from Github. Start your AVD and drag the downloaded APK on the running emulator. You can also install the APK using the following command while running the Virtual Device: $adb install -r Google_Play_Services_for_AR_1.14_x86_for_emulator.apk Adding ARCore Dependencies and Permissions Before you build and run the app, you must add the dependencies and permissions ARCore needs. Open the program level building.gradle file and add the following line to dependencies {...}: implementation 'com.google.ar:core:1.14.0' Click Synchronize Now in the upper-right corner to synchronize your project and your dependencies will be in place. Next, open AndroidManifest.xml and add the following to the manifest tag: <uses-permission android:name=android.permission.CAMERA></uses-permission > <uses-feature android:name=android.hardware.camera.ar android:required=true></uses-feature > <uses-feature android:glesversion=0x000200000 android:required=true></uses-feature> This code adds the necessary permission and feature requests for ARCore. Also add the following just before the closing application tag: <meta-data android:name=com.google.ar.core android:value=required></meta-data> It adds the ARCore metadata. Now you're all set to launch your first ARCore app! Build and run the project. You'll see a prompt to provide camera permissions. After you've given permission, you'll see a radio group at the top. You will use it later to choose the type of object to insert into the scene. You'll see a snack bar at the bottom indicating looking for surfaces.... You can also highlight a few points which means that youth are tracking them. If you target the device on a flat surface, you start detecting aircraft: Once the application detects the first aircraft, the snackbar disappears and you'll see the plane on the screen. Note: ARCore uses clusses function points to detect the surface's angle. So, you has problems detecting flat surfaces without texture or light-colored aircraft such as a white wall. At this stage the utility doesn't do much, but take time to check its code to get your bearings... especially before you have a Viking with a cannon! Behind the ARCore scene The 3D models you will use are in main/assets/models. Here you can find models for a Viking, a cannon and a target. The OpenGL shearers are in main/assets/shafts. The shades are from the Google ARCore sample program. You will see a package named generally. Inside there are two folders: Delivery: This folder contains all the classes associated with OpenGL delivery. Helper: Here you will find utilizers such as CameraPermissionHelper and SnackbarHelper, so you don't have to write boiler code. Aircraft, Anchors and Poses You have a PlaneAttachment class within delivery to quality your work. PlaneAttachment uses an aircraft and an anchor as input. It constructs an attitude of that information. For a quick repetition: An aircraft is a real planar surface. It consists of a group of function points that appear to lie on a horizontal or vertical surface, such as a floor or walls. An anchor points to a fixed location and orientation in physical space to describe the exact position of a virtual object in the real world. An attitude describes a coordinate transforming a virtual object's local frame to frame the real world coordinate. Note: All three classes are part of the ARCore SDK. You can read more about each of them in the official documentation. Imagine the real world around you is an ocean and the aircraft are ports. A port can anchor many ships, or virtual objects, each with their specific attitude. Thus, PlaneAttachment allows you to attach an anchor to an aircraft and pick up the corresponding attitude. ARCore uses the attitude as you move around the anchor point. ARCore Session The starting project includes an ARCore session object in MainActivity.kt. The session describes the entire AR state. You will use it to attach anchors to aircraft when the user taps on the screen. In setupSession (), you can check from onResume (...), the app checks whether the device supports ARCore. If not, it displays a toast and the activity ends. Enlarge your first scene Nowit that your app is running on a supported device or emulator, it's time to set up certain objects to get into the scene! Add objects Open MainActivity.kt and add the following features: private fall vikingObject = ObjectRenderer() private fall cannonObject = ObjectRenderer() privately drop targetObject = ObjectRenderer () Here you define each property as an ObjectRender of the ARCore sample app. Also add three PlaneAttachment properties just below those objects: private var vikingAttachment: PlaneAttachment? = Null private var cannonAttachment: AircraftAttachment? = zero private fresh targetAttachment: AircraftAttachment? = zero It is nulleables initialized as zeros You will create them later, when the user taps on the screen. Now you need to set the objects, which do in onSurfaceCreated (...). Add the following inside the triple block, just below the//TODO comment: //1 vikingObject.createOnGlThread (this@MainActivity, getString (R.string.model_viking_obj), getString (R.string.model_viking_png)) cannonObject.createOnGlThread (this@MainActivity, getString(R.string.model_cannon_obj), getString(R.string.model_cannon_png)) targetObject.createOnGlThread (this@MainActivity, getString (R.string.model_target_obj), getString(R.string.model_target_png)) /2 targetObject.setMaterialProperties (0.0f, 3.5f, 1.0f, 6.0f) vikingObject.setMaterialProperties(0.0f, 3.5f, 1.0f, 6.0f) cannonObject.setMaterialProperties(0.0f, 3.5f, 1.0f, 6.0f) What you're doing here is: You use the 3D model files from the beginning to set up each of the three objects. You set values for amending, diffuse, specular and specular power on each object. These material properties are the surface properties of the delivered model. Changing these values changes the way you see the surface of the object. Here's a closer look at what each of the light values do: Amending: The intensity of non-directional surface relief. Diffuse: The reflectivity of the diffuse, or carpet, surface. Specular: How reflective the specular, or shiny, surface is. Specular force: The surface shines. Larger values lead to a smaller, sharper speculative peak. Play around with these values to see how your object changes. Attach Anchors to the Session Your next step is to give the user the ability to attach an anchor to the session when typing on the screen. To get started, find handleTap (...) in MainActivity.kt. Insert the following inside axis statement, just above the/TODO comment before the breaking statement: when (mode) { Mode.VIKING -> vikingAttachment = addSessionAnchment ofAttachment (vikingAttachment, Hit) Mode.CANNON -> cannonAttachment = addsessionAnchorFromAttachment (cannonAttachment, hit) Mode.TARGET -> targetAttachment = addSessionAnchorFromAttachment (targetAttachment, hit) } You will see an error because you still don't addSessionAnchorFromAttachment (...). You will address this error for a while. The radio buttons at the top of the screen control the value of mode. It is a Kotlin enumeration class that includes a scale factor driving value for each mode. The scale factor agrees the size of the corresponding 3D model in the scene. For each mode, you will introduce a new value for the corresponding PlaneAttachment in the when statement. You use the old attachment and the hit value for the crane, which is an ARCore Heatresult defines the intersection of the 3D ray for the crane and an aircraft. Now add toSessionAnchorFromAttachment (...) at the bottom of MainActivity.kt: private fun addIngSsionAnchorFromAttachment(previousAttachment: PlaneAttachment?, heatResult): PlaneAttachment? {/1 previousAttachment?. Anchor?. LOOSE () / 2 fall plane = as Aircraft fall anchor = session!!. createAnchor (hit.hitPose) //3 return AircraftAttachment (aircraft, anchor) } } } you are doing here: If the previous Amedication is not zero, you remove its anchor from the session. You take the HeatResult aircraft and create the anchor of the HitResult hold. You then add the anchor to the session. Finally, with the above information about the aircraft and the anchor, you return the PlaneAttachment. You are almost ready to see your Viking do some target practice! :] Draw the objects Your last step is to draw the objects on the screen. You have created aircraft attachments when the user taps, but now you need to draw the objects as part of the screen delivery. To do this, go to unDrawFrame (...) and add the following calls to the bottom of the three block: jogger (vikingObject, VikingAttachment, Mode.VIKING.scaleFactor, projectionMatrix, viewMatrix, lightIntensity) drawObject(cannonObject, cannonAttachment, Mode.CANNON.scaleFactor, projectionMatrix, viewMatrix, lightIntensity) joGObject (targetObject, targetAttachment, Mode.TARGET.scaleFactor, projectionMatrix, viewMatrix, lightIntensity) Here, you name the pre-existing draw It takes the object, the corresponding attachment, the scale factor and the matrics and values required for OpenGL to record the object. The app calculates the matrics using these already-present helper features: //1 private fun computerProjectionMatrix (camera: Camera:Camera): FloatArray { fall projectionMatrix = FloatArray (maxAllocationSize) camera.getProjectionMatrix (projectionMatrix, 0, 0.1f, 100.0f) return projectionMatrix}/2 private fun computer ViewMatrix (camera: Camera): FloatArray { fall viewMatrix = FloatArray (maxAllocationSize) camera.getViewMatrix (viewMatrix, 0) return viewMatrix }/3 private fun computer LightIntensity (frame: Frame): FloatArray {fall lightIntensity = FloatArray(4) frame.lightEstimate.getColorCorrection (lightIntensity, 0) show lightIntensity } Here what's going on in the code above: ARCore uses the current session's camera input to calculate projection Matrix. It also uses those inputs to calculate viewMatrix. Finally, it uses the frame, which the AR state describes at a specified time, to calculate the light intensity. Build and run, then select a radio button at the top to select an object mode. Find an aircraft with your camera and tap to place an object. The angle that has an object when you place it depends on your device's orientation and trend. Move your device around and place your object with the corner you prefer. Once you've placed all the objects, if you turn your phone, you'll see a scene as follows: Move around the scene and see how your Viking is preparing to fire. There is no stopping your Viking now! Where to go from here? Check out the end project by clicking the Download Material button at the top or bottom of this tutorial. You have reached the banks of ARCore's world with OpenGL and Android Studio like a Viking! To discover more, check out the official ARCore Plot these resources to enrich your ARCore skill stock: You can also use with Unity, Unreal and Web. Since a good deal of your development with ARCore is likely to state on Unity, look at our Unit Content. Finally, you can find some cool demons made with ARCore on the Google Experiments website. I hope you enjoy this short intro to ARCore with Kotlin. If you have any questions or comments, join the forum discussion below. AR/VR Android & Kotlin Tutorials The raywenderlich.com newsletter is the easiest way to keep track of everything you need to know as a mobile developer. Get a weekly consumed from our tutorials and courses, and receive a free in-depth email course as a bonus! 4.6/5 ratings