Task 1 : Create a COVID-19 Tracker Android App to See Details of any City and State in India

Approach

Step 1: Create a new project
Step 2: Now add some files before writing Java and XML code

- Go to **Gradle Scripts -> build.gradle (Module: app)** section and import the following dependencies and click the "**Sync Now**" button to Sync the APP.
- To Learn More About Volley Library Click Here.

```
implementation 'com.android.volley:volley:1.1.1'
```

Step 3: Use JSON Parsing to fetch data from the website

- Click on this URL- https://api.covid19india.org/state_district_wise.json
- It will Show Data of the whole Country and in this, We Are Going to fetch data from here Only.

Step 4: Design the Layout of the activity_main.xml

1. Go to **app -> res -> layout -> activity_main.xml**
2. Add A TextView To the layout: To display LGM at the top of the screen.
3. Add A ListView To the Layout: To display the list of cities tracking details on the screen.

Step 5: Design the Layout for The ListView

1. Create A new Layout by right click on the layout folder inside res Folder
2. Then Click Layout Resource File and Set the name **testing.xml**
3. Click **Finish** To save
4. Click testing.xml and Start Design UI Layout for the ListView
5. The Drawable file used in **testing.xml** is "**arrow upward**". Go to **drawable -> New -> Vector Asset** and search for **"arrow upward"** and add it to your file**.**

Step 6: Create a New JAVA Class to fetch the data we want to fetch from the Website

1. Create a new JAVA class name it as **Model.java**
2. Use getters and setters functions to create a function for the data you want to fetch from the website.

Step 7: Create an Adapter Class

- Now create a new JAVA Adapter Class to put the data that have fetched into a ListView that has created before.

Step 8: Working with MainActivity.java file

- In this file use volley to fetch the data from the provided url.

Task 2: Create a Face Detection Android App using Machine Learning KIT on Firebase.

Pre-requisites:

- Firebase Machine Learning kit
- Adding Firebase to Android App

Approach

Step 1: Create a New Project

1. Open a **new project** in android studio with whatever name you want.
2. We are gonna work with **empty activity** for the particular project.
3. The minimum **SDK** required for this particular project is **23**, so choose any API of 23 or above.
4. The language used for this project would be **JAVA**.
5. Leave all the options other than those mentioned above, untouched.
6. Click on **FINISH**.

Step 2: Connect with ML KIT on Firebase.

1. Login or signup on Firebase.
2. In Firebase console, create a **new project** or if you wanna work with an existing project then open that.
3. **Name** the project as per your choice.
4. Go to **Docs**.

5. Click on Firebase ML, and in the left space, choose '**recognize text**' under Vision.
6. Go through the steps mentioned for better understanding.
7. Come back to Android Studio.
8. Go to **Tools -> Firebase -> Analytics -> Connect with Firebase -> Choose your project from the dialog box appeared -> Click Connect.** (This step connects your android app to the Firebase)

Step 3: Custom Assets and Gradle

- For enhancing the GUI either choose an image of .png format and add it in the res folder and set it as the background of main .xml file, or set a background color by going to the design view of the layout and customizing background under **Declared Attributes.**
- To, include the ML KIT dependencies, in the app, go to Gradle Script -> build.gradle(Module:app) and add an implementation mentioned below:

implementation 'com.google.firebase:firebase-ml-vision:17.0.0'

- Now copy the below-mentioned text, and paste it at the very end of the app level Gradle, outside all the brackets.

apply plugin: 'com.google.gms.google-services'

- Next, go to bulid.gradle (project) and copy the below-mentioned text, and paste it in 'dependencies' classpath.

classpath 'com.google.gms:google-services:4.2.0'

- Click on sync now.

Step 4: Designing the UI

- Add a Button to open the camera option.
- Now go to **layout -> new -> layout resource file -> Name: fragment_resultdialog.xml.** This file has been created to customize the output

screen, which will display a dialog box called Result Dialog box with a text view called Result Text with all the attributes of the detected image.

Step 5: Firebase App Initializer

- Create a new java class by **java -> new -> class -> Name: LCOFaceDetection.java -> superclass: Application(android.app.Application).**

Step 6: Inflating the Result Dialog Box

- Create a new java class namely, **ResultDialog.java and superclass, DialogFragment**, which is the java file for the **fragment_resultdialog.xml**. Below is the example code for java file.

Step 7: Open Camera on a Real Device and Enabling Face Detection

- Below is the example code for the **main java file**.
- There is a need of **FirebaseVision** and **FirebaseVisionFaceDetector** classes for this.
- Here's a list of all the settings you can configure in your face detection model.

Setting	**Description**
Performance mode	FAST (default) | ACCURATE Favor speed or accuracy when detecting faces.

Detect landmarks	NO_LANDMARKS (default) | ALL_LANDMARKS Whether to attempt to identify facial “landmarks”: eyes, ears, nose, cheeks, mouth, and so on.
Detect contours	NO_CONTOURS (default) | ALL_CONTOURS Whether to detect the contours of facial features. Contours are detected for only the most prominent face in an image.
Classify faces	NO_CLASSIFICATIONS (default) | ALL_CLASSIFICATIONS Whether or not to classify faces into categories such as “smiling”, and “eyes open”.
Minimum face size	float (default: 0.1f) The minimum size, relative to the image, of faces to detect.

Enable face tracking	false (default) | true Whether or not to assign faces an ID, which can be used to track faces across images. Note that when contour detection is enabled, only one face is detected, so face tracking doesn't produce useful results. For this reason, and to improve detection speed, don't enable both contour detection and face tracking.

- It is suggested to read a detailed analysis of these classes and work on the code at the Firebase ML docs for text recognition.

Task 3: Augmented Faces with ARCore in Android

Augmented Faces permit the application to naturally distinguish various regions of an individual's face, and utilize those areas to overlay resources, for example, surfaces and models in a way that appropriately matches the contours and regions of an individual face. ARCore is a stage for building Augmented reality applications on Android. Augmented Face is a subsystem of ARCore that permits your application to:

- Naturally, recognize various areas of any individual's identified face, and utilize those regions to overlay resources, for example, surfaces and models in a way that appropriately matches the contours and regions of an individual face.
- Utilize the 468-point face mesh that is given by ARCore to apply a custom texture over a distinguished face.

For example, we can create effects like animated masks, glasses, virtual hats, perform skin retouching, or the next Snapchat App.

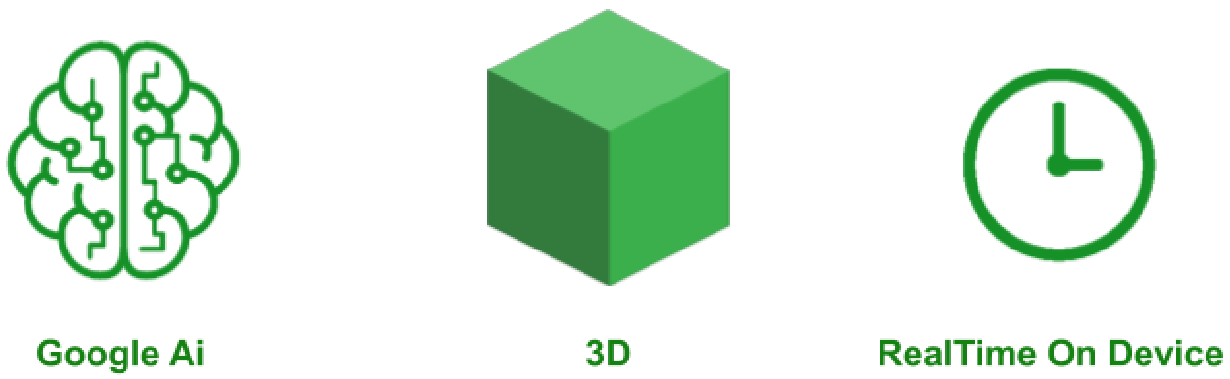
How Does it All Work?

Augmented faces don't require uncommon or special hardware, such as a depth sensor. Rather, it uses the phone's camera and machine learning to provide three snippets of data:

1. **Generates a Face Mesh:** a 468 points dense 3D face mesh, which allows you to pan detailed textures that accurately follow facial moments.
2. **Recognizes the Pose:** points on a person's face, anchored based on the generated Face Mesh, which is useful for placing effects on or close to the temple and nose.
3. Overlays and position textures and 3D models based on the face mesh generated and recognized regions.

How is ARCore Able to Provide a 3D face Mesh from a 2D Image without any Depth Hardware?

It uses machine learning models that are built on top of the TensorFlow Lite platform to achieve this and the crossing pipeline is optimized to run on the device in real-time. It uses a technique called transfer learning wherein we train the neural network for two objectives, one, to predict 3D vertices and to predict 2D contours. To predict 3D vertices, we train it with a synthetic 3D data set and use this neural network as a starting point for the next stage of training.



In this next stage, it uses the annotated data set, annotated real-world data set to train the model for 2D contour prediction. The resulting network not only predicts 3D vertices from a synthetic data set but can also perform well from 2D images. To make sure the solution works for everyone ARCore developers train the network with geographically diverse data sets so that it works for all types of faces, wider faces, taller faces, and all types of skin colors.

And to enable these complex algorithms on mobile devices, we have multiple adaptive algorithms built into the ARCore. These algorithms sense dynamically how much time it has taken to process previous images and adjust accordingly various parameters of the pipeline. It uses multiple ML models, one optimized for higher quality and one optimized for higher performance when computing the resources is really challenging. It also adjusts pipeline parameters such as inference rates so that it skips a few images, and instead replace that with interpolation data. With all these techniques, what you get is a full-frame rate experience for your user. So it provides face mesh and region poses at full camera frame rate while handling all these techniques internal to the ARCore.

Identifying an Augmented Face Mesh

To appropriately overlay textures and 3D models on an identified face, ARCore provides detected regions and an augmented face mesh. This mesh is a virtual depiction of the face and comprises the vertices, facial regions, and the focal point of the user's head. At the point when a user's face is identified by the camera, ARCore performs the following steps to generate the augmented face mesh, as well as center and region poses:

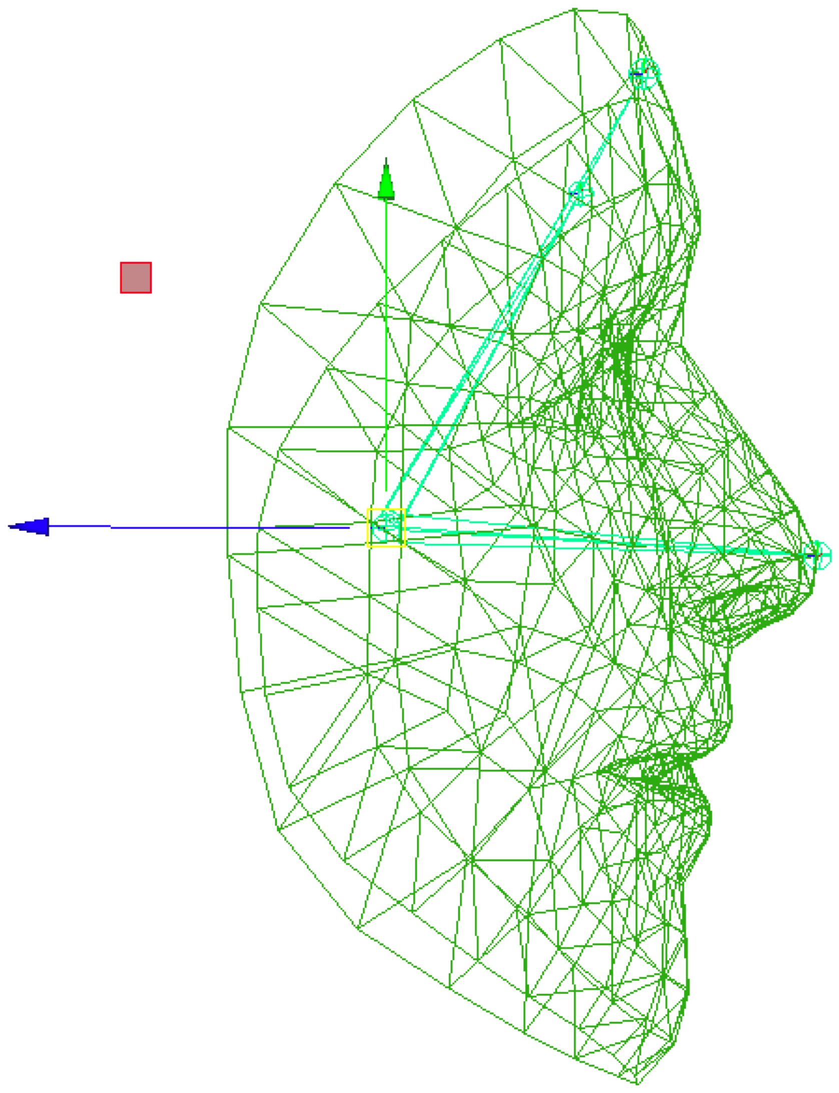
- It distinguishes the center pose and a face mesh.

- The center pose, situated behind the nose, is the actual center point of the user’s head (in other words, inside the skull).
- The face mesh comprises of many vertices that make up the face and is characterized relative to the center pose.

Google Ai

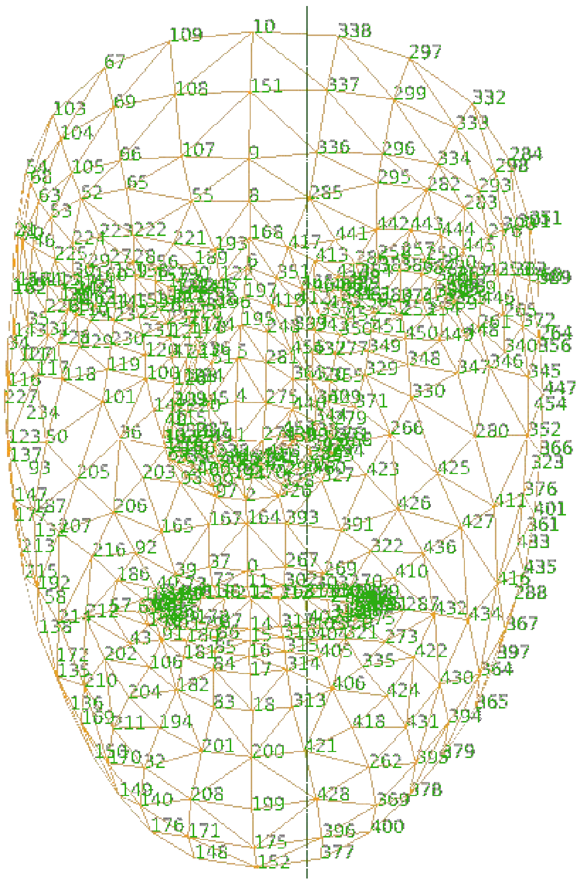
3D

RealTime On Device



- The AugmentedFace class utilizes the face mesh and center pose to distinguish face regions present on the client's face. These regions are:
    - Right brow (RIGHT_FOREHEAD)
    - Left temple (LEFT_FOREHEAD)
    - Tip of the nose (NOSE_TIP)

The Face mesh, center pose, and face region poses are utilized by AugmentedFace APIs as positioning points and regions to place the resources in your app.



468 point face texture mesh

Reference Terminologies

- **Trackable**: A Trackable is an interface which can be followed by ARCore and something from which Anchors can be connected to.
- **Anchor**: It describes a fixed location and orientation in the real world. To stay at a fixed location in physical space, the numerical description of this position will update as ARCore's understanding of the space improves. Anchors are hashable and may for example be used as keys in HashMaps.
- **Pose**: At the point when you need to state wherein the scene you need to put the object and you need to specify the location in terms of the scene's coordinates. The Pose is the means by which you state this.
- **Session:** Deals with the AR framework state and handles the session lifecycle. This class offers the primary passage to the ARCore API. This class permits the user to make a session, configure it, start or stop it and,

above all, receive frames that allow access to the camera image and device pose.
- **Textures:** Textures are especially helpful for Augmented Faces. This permits you to make a light overlay that lines up with the locales of the identified face(s) to add to your experience.
- **ArFragment:** ARCore utilizes an ArFragment that provides a lot of features, for example, plane finding, permission handling, and camera set up. You can utilize the fragment legitimately in your activity, however at whatever point you need custom features, for example, Augmented Faces, you should extend the ArFragment and set the proper settings. This fragment is the layer that conceals all the compound stuff (like OpenGL, rendering models, etc) and gives high-level APIs to load and render 3D models.
- **ModelRenderable:** ModelRenderable renders a 3D Model by attaching it to a Node.
- **Sceneform SDK:** Sceneform SDK is another library for Android that empowers the quick creation and mix of AR experiences in your application. It joins ARCore and an amazing physically-based 3D renderer.

Approach

Step 1: Create a New Project

Step 2: Adding the assets file

- Add any 3D model in sampledata/models folder. We can do this by creating a new folder in the project file directory or directly from the Android Studio. The allowed 3D model extensions are .fbx, .OBJ, .glTF. There are many free models available on the internet. You can visit, here or more.

Step 3: Adding dependencies to the build.gradle(:app) file

Add the following dependencies to the **build.gradle(:app)** file.

```
// Provides ARCore Session and related resources.

implementation ‘com.google.ar:core:1.16.0’

// Provides ArFragment, and other UX resources.
```

```
implementation 'com.google.ar.sceneform.ux:sceneform-ux:1.15.0'

// Alternatively, use ArSceneView without the UX dependency.

implementation 'com.google.ar.sceneform:core:1.8.0'
```

Add the following code snippet to the build.grdale file. This is required(only once) to convert **.fbx asset into .sfb** and save that in the raw folder. Or you can add them by yourself as done in step 2.

```
// required(only once) to convert .fbx asset into .sfb

// and save that in raw folder

sceneform.asset('sampledata/models/fox_face.fbx',

                'default',

                'sampleData/models/fox_face.sfa',

                'src/main/res/raw/fox_face')
```

Step 4: Adding dependencies to the build.gradle(:project) file

Add the following dependencies to the **build.gradle(:project)** file.

```
// Add Sceneform plugin classpath to Project
```

// level build.gradle file

```
classpath ‘com.google.ar.sceneform:plugin:1.15.0’
```

Step 5: Working with the AndroidManifest.xml file

Add the following line to the **AndroidManifest.xml** file.

```
// Both “AR Optional” and “AR Required” apps require CAMERA permission.

<uses-permission android:name=”android.permission.CAMERA” />

// Indicates that app requires ARCore (“AR Required”). Ensures app is only

// visible in the Google Play Store on devices that support ARCore.

// For “AR Optional” apps remove this line. →

<uses-feature android:name=”android.hardware.camera.ar”
android:required=”true”/>

<application>

  …

  // Indicates that app requires ARCore (“AR Required”). Causes Google

  // Play Store to download and install ARCore along with the app.

  // For an “AR Optional” app, specify “optional” instead of “required”.
```

```
        <meta-data android:name=”com.google.ar.core” android:value=”required” />

         …

      </application>
```

Step 6: Modify the activity_main.xml file

We have added a fragment to the activity_main.xml file. Below is the code for the activity_main.xml file.

Note:

Please add your package name to this attribute.

android:name=”com.example.arsnapchat.CustomArFragment”

Step 7: Create a new Java class

Create a new class and name the file as CustomArFragment that extends ArFragment. Below is the code for the CustomArFragment.java file.

Step 8: Modify the MainActivity.java file

Limitations of Ar Core

1. Augmented Faces only works with the Front Camera.
2. Not all devices support ARCore. There’s still a small fraction of devices which doesn’t come with AR Core support. You can check out the list of ARCore supported devices at https://developers.google.com/ar/discover/supported-devices.
3. For AR Optional App minSdkVersion should be 14 and for AR Required App minSdkVersion should be 24.
4. If your app falls in the category of AR Required app then the device using it should have AR Core installed on it.

Notes:

1. Prior to making a Session, it must be verified beforehand that ARCore is installed and up to date. If ARCore isn't installed, then session creation might fail and any later installation or upgrade of ARCore would require restarting of the app, and might cause the app to be killed.
2. The orientation of the face mesh is different for Unreal, Android and Unity.
3. Calling Trackable.createAnchor(Pose) would result in an IllegalStateException because Augmented Faces supports only front-facing (selfie) camera, and does not support attaching anchors.