

LabW06 – Mobile Augmented Reality

Objectives:

1. Understand the motion tracking and spatial understanding capabilities employed by mobile devices to deliver AR services
2. Explore the ARCore platform for Augmented Reality by Google
3. Familiarise with the Sceneform framework used in building AR apps using ARCore

Tasks:

1. Setup a device or emulator to enable ARCore
2. Build an application demonstrating 3D models rendered over the camera view using the Sceneform framework over ARCore

One of the most popular mobile applications now are utilising Augmented Reality to deliver novel experiences to the users. In this tutorial, we will focus on learning how to utilise the ARCore for building an AR application.

This tutorial has been adapted from the ARCore Quickstart tutorial


<https://developers.google.com/ar/develop/java/quickstart>

and the Google Codelabs tutorial

<https://codelabs.developers.google.com/codelabs/sceneform-intro>

Task 1: Enabling ARCore on a device or Emulator

In this preliminary task, we'll make sure that the Android device or virtual device (via emulator) we will be using to run AR application supports ARCore. If you are going to use a physical Android device to run your developed AR app, make sure it is listed [on this ARCore supported devices](#); if it is, then you may proceed straight to Task 2.

1. Open Android Studio. You may open any project in the meantime.
2. To enable ARCore in a virtual device in the Android Emulator of Android Studio, make sure that the following packages are installed:
 - a. Go to **Tools > SDK Manager**. Under **SDK Platforms**, make sure **Android 8.1 (Oreo)** installed.
 - b. Under **SDK Tools**, add **Android Emulator 27.2.9** or later.
3. Configure a virtual device where we'll be running our AR application:
 - a. Select the **Pixel** or **Pixel 2** hardware profile.
 - b. Select the **Oreo: API Level 27: x86: Android 8.1 (Google Play)** system image. Install it if not yet available as a system image.
 - c. In **Verify Configuration**, go to **Show Advanced Settings** and make sure that **Camera Back** is set to **VirtualScene**. Press **Finish** to create your virtual device and run it.
4. Configure the emulator to support ARCore and Sceneform:
 - a. Update Google Play Services for AR on the emulator. Download the latest **Google_Play_Services_for_AR_1.19.0_x86_for_emulator.apk** from the GitHub [releases](#) page. Drag the downloaded APK onto the running emulator. Let it install.
 - b. Click  in the running emulator's toolbar. Select **Settings > Advanced > OpenGL ES API level > Renderer maximum (up to OpenGL ES 3.1)**.
 - c. Restart the emulator. When prompted, do not save the current state.

Note: If after these configurations and the AR application does not run properly, your desktop does not support OpenGL ES 3.0 and you should instead use a physical device to run Sceneform apps. For more info on enabling ARCore on the emulator, refer to <https://developers.google.com/ar/develop/java/emulator>

Task 2: Build AR application

In this task, we will build a Sceneform application that demonstrates a simple AR application that allows you to render a 3D model over a detected surface seen through the camera view.

1. Create a new project "SceneformTutorial" targeting API level 24 (Android 7) or later with a **Basic Activity** layout.
2. Import the Sceneform plugin into your project. Refer to this link for the steps: <https://developers.google.com/sceneform>
3. In the **Gradle Scripts**, we need to enable **Java 8** and add the dependencies for Sceneform.

- a. In the **android {}** section of **app/build.gradle** add:

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}
```

- b. In the **dependency {}** section add:

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

- c. Press "Sync Now" to update the project.

4. Add ARCore **AndroidManifest.xml** entries

- a. In the **<manifest>** section add:

```
<uses-permission android:name="android.permission.CAMERA" />  
<uses-feature android:name="android.hardware.camera.ar" />
```

- b. In the **<application>** section add:

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

5. Add **ArFragment** to the **app/res/layout/content_main.xml** file. Replace the existing **TextView** element with the following fragment:

```
<fragment  
    android:id="@+id/sceneform_fragment"  
    android:name="com.google.ar.sceneform.ux.ArFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```



6. Now, in the **MainActivity** add the **ArFragment** field.

a. Declare a member variable at the top of the class:

```
private ArFragment fragment;
```

b. Initialize fragment at the bottom of the **onCreate()** method using the fragment manager.

```
fragment = (ArFragment)
    getSupportFragmentManager().findFragmentById(R.id.sceneform_fragment);
```

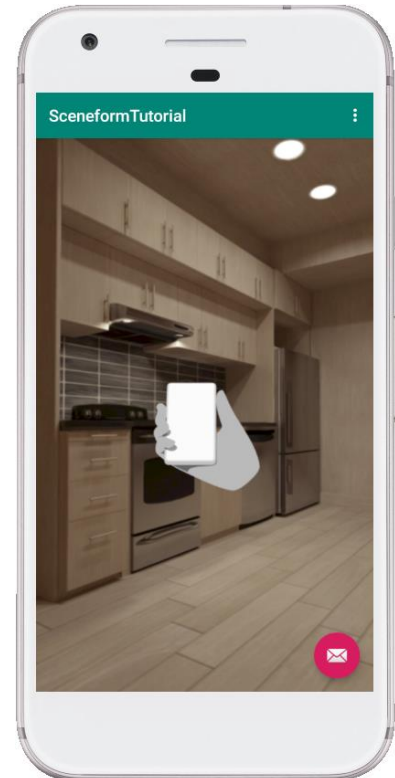
The **ArFragment** handles the *permissions*, ARCore *session creation*, and the *plane discovery* UI.

7. Build and run your app to do a test run.

IMPORTANT: Make sure the target device, whether actual or virtual, is setup with ARCore. If not, refer back to Task 1 to check if you missed a step.

It should first request to use the camera; then, you'll see an indicator that you need to **move the phone around** to detect a plane. See sample screenshot on the right -->

If you have trouble implementing this, you may refer to the reference codes provided from the lab files.



8. Now, let's add a pointer that is centered on the screen and changes based on the tracking state from ARCore, i.e. when it is tracking and has detected a plane.

- Let's create a new *Drawable* Java class for this pointer. Click **File > New > Java Class** and name it **PointerDrawable**.
- Set the Superclass to Drawable, i.e. **android.graphics.drawable.Drawable**.
- At the top of the class, add a Paint object and a Boolean flag "enabled" for indicating whether our pointer is enabled or disabled.
- Move your cursor on the flag. Click on the red light bulb icon in the editor and select "Create getter and setter for 'enabled'".

```
private final Paint paint = new Paint();
private boolean enabled;

public boolean isEnabled() {
    return enabled;
}

public void setEnabled(boolean enabled) {
    this.enabled = enabled;
}
```

e. Implement the **draw** method necessitated by a Drawable class.

```
@Override
public void draw(@NonNull Canvas canvas) {
    float cx = canvas.getWidth()/2;
    float cy = canvas.getHeight()/2;

    if (enabled) {
        paint.setColor(Color.GREEN);
        canvas.drawCircle(cx, cy, 10, paint);
    } else {
        paint.setColor(Color.GRAY);
        canvas.drawText("X", cx, cy, paint);
    }
}
```

9. Now, let's use the pointer in the **MainActivity**.

a. Add the three variables to the **MainActivity** class.

```
private PointerDrawable pointer = new PointerDrawable();
private boolean isTracking;
private boolean isHitting;
```

b. At the bottom of **onCreate()** add a *listener* to the ArSceneView scene which will get called before processing every frame. In this listener we can make ARCore API calls and update the status of the pointer.

We use a *lambda* to call the **onUpdate()** method like so.

```
fragment.getArSceneView().getScene().addOnUpdateListener(frameTime -> {
    fragment.onUpdate(frameTime);
    onUpdate();
});
```

c. Implement the **onUpdate()** method:

```
private void onUpdate() {
    boolean trackingChanged = updateTracking();
    View contentView = findViewById(android.R.id.content);
    if (trackingChanged) {
        if (isTracking) {
            contentView.getOverlay().add(pointer);
        } else {
            contentView.getOverlay().remove(pointer);
        }
        contentView.invalidate();
    }

    if (isTracking) {
        boolean hitTestChanged = updateHitTest();
        if (hitTestChanged) {
            pointer.setEnabled(isHitting);
            contentView.invalidate();
        }
    }
}
```

d. Implement the **updateTracking()** method:

```
private boolean updateTracking() {
    Frame frame = fragment.getArSceneView().getArFrame();
    boolean wasTracking = isTracking;
    isTracking = frame != null &&
        frame.getCamera().getTrackingState() == TrackingState.TRACKING;
    return isTracking != wasTracking;
}
```

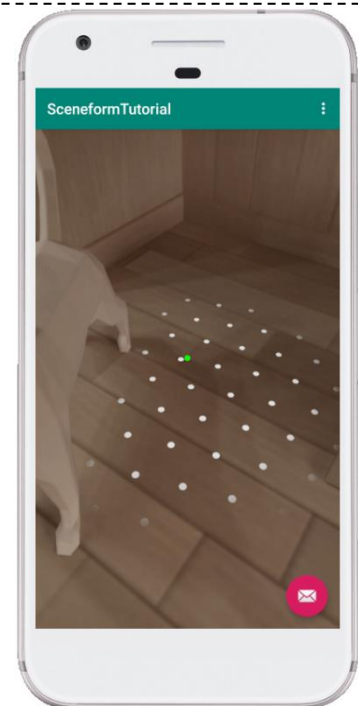
e. Implement the following additional methods:

```
private boolean updateHitTest() {
    Frame frame = fragment.getArSceneView().getArFrame();
    android.graphics.Point pt = getScreenCenter();
    List<HitResult> hits;
    boolean wasHitting = isHitting;
    isHitting = false;

    if (frame != null) {
        hits = frame.hitTest(pt.x, pt.y);
        for (HitResult hit : hits) {
            Trackable trackable = hit.getTrackable();
            if (trackable instanceof Plane &&
                ((Plane) trackable).isPoseInPolygon(hit.getHitPose())) {
                isHitting = true;
                break;
            }
        }
    }
    return wasHitting != isHitting;
}

private android.graphics.Point getScreenCenter() {
    View vw = findViewById(android.R.id.content);
    return new android.graphics.Point(vw.getWidth()/2, vw.getHeight()/2);
}
```

- f. You can test and run the pointer! At the start, the pointer should not be visible until ARCore starts tracking. As you move around the phone, a plane will be detected, and you should see the pointer be enabled (turn green) and disabled as the pointer moves on and off the plane. See the screenshot on the right for an example.



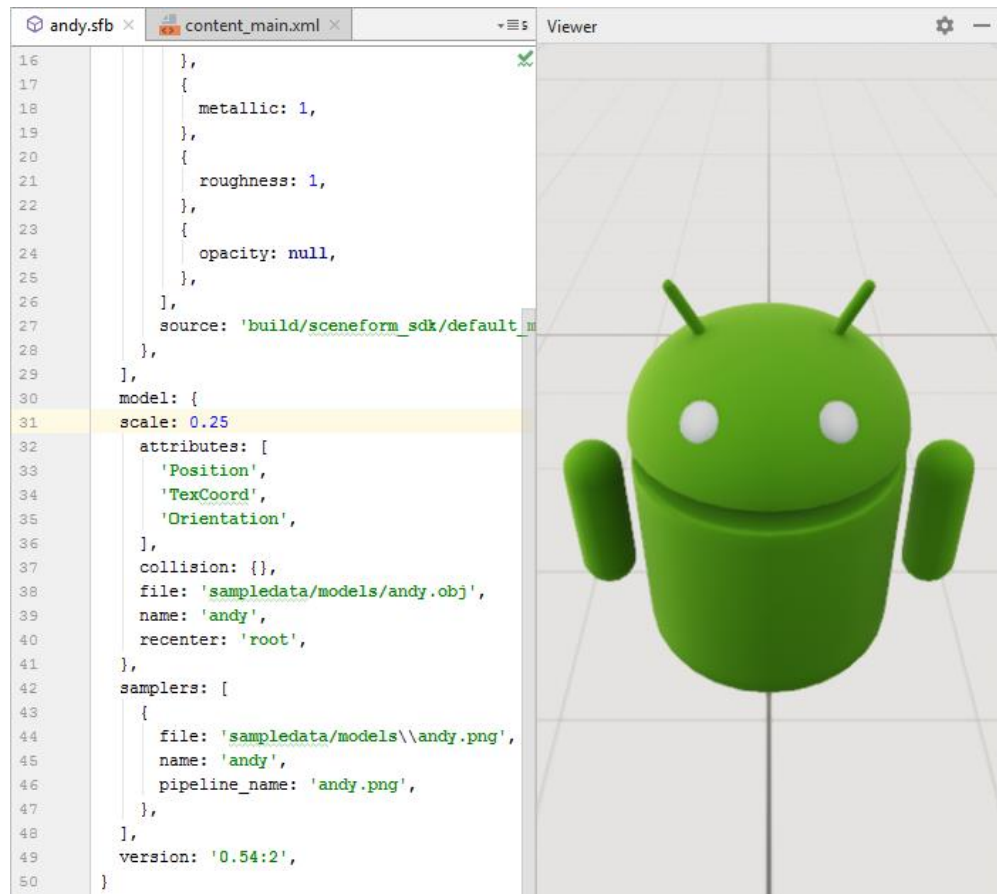
10. Let's import 3D models which we will render over a detected surface

- Create a Sample Data Directory where we'll keep our 3D assets. In the Project window, right-click on **app** and select **New > Sample Data Directory**. If this does not work, manually create the **sampledata** directory through the File Explorer but make sure it is inside the **app** directory.
- Copy the **models** directory from the lab files into **app/sampledata**.
- Copy all the PNG files in the **drawable** directory from the lab files into **app/src/main/res/drawable**.

- d. To import the 3D models, go to **app/sampleddata/models** and right click on **andy.obj** then pick **Import Sceneform asset**. This opens the import wizard. Click Finish to start the import.

(You can check for other 3D models with a CC-BY license at <https://poly.google.com/>)

Once the import has finished, the **andy.sfb** file will be opened in the editor. The Sceneform viewer is also opened showing the imported 3D model. See sample screenshot below.



You may add a **scale** parameter with a value **0.5** within the **model: {}** object in the **sfb** file. An example is shown below. If later your model seems to be too small, you may remove the scale.

```

model: {
  scale: 0.5,
  attributes: [
    'Position',
    'TexCoord',
    'Orientation',
  ],
},

```

11. Let's use the imported model in the **MainActivity**.

- a. Add the renderable model variable to the **MainActivity** class:

```
private ModelRenderer andyRenderable;
```

- b. Then, add the model builder within the **onCreate** method:

```
ModelRenderable.builder()
    .setSource(this, Uri.parse("andy.sfb"))
    .build()
    .thenAccept(renderable -> andyRenderable = renderable)
    .exceptionally(
        throwable -> {
            Toast toast =
                Toast.makeText(this, "Unable to load andy renderable",
                    Toast.LENGTH_LONG);
            toast.setGravity(Gravity.CENTER, 0, 0);
            toast.show();
            return null;
        });
```

- c. Lastly, at the end of the onCreate method, add the screen tap *listener* which waits for a tap on the screen (click if using the emulator) and renders the model after a successful hit test, i.e. the tapped spot is either a plane or point which the rendered model can anchor on to.

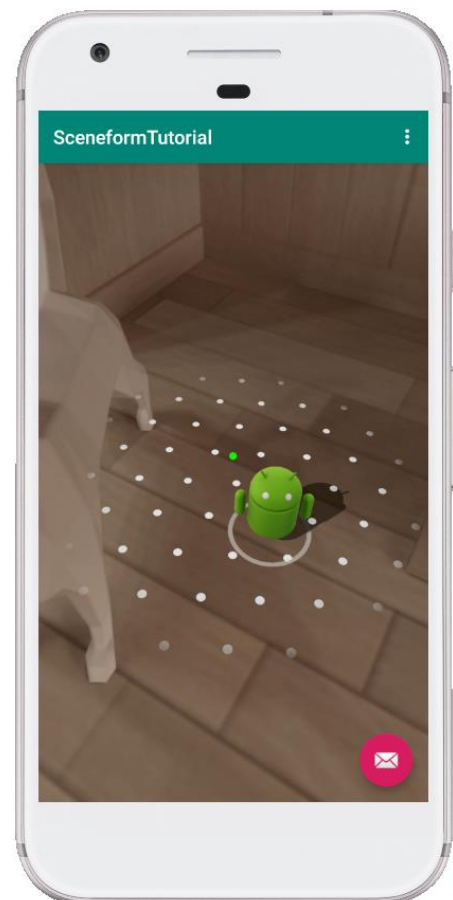
```
fragment.setOnTapArPlaneListener(  
    (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {  
        if (andyRenderable == null) {  
            return;  
        }  
  
        // Create the Anchor.  
        Anchor anchor = hitResult.createAnchor();  
        AnchorNode anchorNode = new AnchorNode(anchor);  
        anchorNode.setParent(fragment.getArSceneView().getScene());  
  
        // Create the transformable andy and add it to the anchor.  
        TransformableNode andy = new  
TransformableNode(fragment.getTransformationSystem());  
        andy.setParent(anchorNode);  
        andy.setRenderable(andyRenderable);  
        andy.select();  
    });
```

The method called by the listener using lambda builds two *nodes* and attaches them to the `ArSceneView`'s scene object.

The first node is of type **AnchorNode**. Anchor nodes are positioned based on the pose of an ARCore Anchor. As a result, they stay positioned in the sample place relative to the real world.

The second Node is a **TransformableNode** provides the interaction functionality to handle *moving*, *scaling* and *rotating* based on user gestures.

12. Finally, run your code! Once a plane is detected, tap on the screen and see that an Android, i.e. **andy**, model is rendered where you tapped. Try moving, rotating, and scaling the model!
13. [Optional] Now, that you have successfully created an AR application using ARCore, you can add further functionalities. Let's add the photo capturing. Activate the floating action button and use it to capture a photo of your screen with the rendered augmentations and save it as an image into the photos directory!



For more information, please visit:

1. ARCore Quickstart for Android:
<https://developers.google.com/ar/develop/java/quickstart>
2. Updating Google Play Services for AR (ARCore service) on the emulator:
<https://developers.google.com/ar/develop/java/emulator#update-arcore>
3. Introduction to Sceneform codelab:
<https://codelabs.developers.google.com/codelabs/sceneform-intro>
4. Polygon library: <https://poly.google.com/>
5. Recognize and augment images with **Augmented Images for Android**:
<https://developers.google.com/ar/develop/java/augmented-images/>
6. Shared AR experiences with **Cloud Anchors**:
<https://developers.google.com/ar/develop/java/cloud-anchors/overview-android>