ARCore & Sceneform Workshop

- Welcome! Before we get started...
- Clone the project
- Check that Android Studio is the correct version or higher
- Set up an ARCore supported device...
- ... or emulator

About ARCore

- Google released ARCore in February 2018.
- ARCore helps devs build apps that can understand the environment around a device and place objects and information in it.

About Sceneform

- Google followed up with Sceneform at I/O 2018.
- Sceneform helps devs render 3D scenes on Android without needing to learn OpenGL.

Add the dependency

In `app/build.gradle`

```
dependencies {
    // ...

    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.4.0"
}
```

Configure the manifest

Setup camera in the `<manifest>` section

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

In the `<application>` section, add a Play Store filter for users on devices that are not supported by ARCore.

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

Add the ARFragment

In `content_main.xml`

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

Run it!

Emulator troubleshooting

- Check that your Android Emulator is updated to 27.2.9 or later.
- Follow the instructions linked in the README.

Motion Tracking

ARCore detects visually distinct **feature points** in each captured camera image and uses these points to compute a device's change in location over time.

Motion tracking

Visual feature point information is combined with measurements from the device's Inertial Measurement Unit (IMU).

Motion tracking

This combined data is used to estimate the **pose**, defined by **position** and **orientation**, of the device camera relative to the world over time.

Pose

Everything in an AR scene has a **pose** within a 3D world coordinate space.

Each **pose** is composed of:

- x-axis translation
- y-axis translation
- z-axis translation
- rotation

Pose

Sceneform aligns the pose of the **virtual camera** that renders your 3D content with the pose of the device's camera provided by ARCore.

Pose

Because the rendered virtual content is overlaid and aligned on top of the camera image, it appears as if your virtual content is part of the real world.

Feature points

feature points are visually distinct features that ARCore detects in each captured camera image (e.g. the corner of a table, a mark on a wall)

Plane detection

ARCore looks for clusters of feature points that appear to lie on common horizontal or vertical surfaces and provides this data to Sceneform as **planes**.

Plane detection

A **plane** is composed of:

- A center **pose**
- An **extent** along each axis
- A **polygon**, a collection of 2D vertices approximating the detected plane.

Plane detection

The Sceneform fragment renders a plane-grid to indicate via the UI where these planes exist.

Goal:

When a user taps a point on the screen that intersects with a plane, we want to place an object at that point.

Hit test

We want to determine if a **ray** extending from the **x, y** coordinate of our tap intersects a plane.

If it does, we'll place an **anchor** at the **pose of the intersection**.

Hit test

In `MainActivity.kt`, set up the `ArFragment` with an `OnTapArPlaneListener`.

```
private lateinit var arFragment: ArFragment

override fun onCreate(savedInstanceState: Bundle?) {
    //...
    arFragment = fragment as ArFragment
    arFragment.setOnTapArPlaneListener { hitResult, plane, motionEvent ->
        // We've hit a plane!
    }
}
```

Hit Result

Represents *an intersection between a **ray** and estimated real-world **geometry*** (e.g. a `Node` or a `Plane`).

We can use a `HitResult` to determine the `Pose` of the intersection, the distance from the camera,
or to create a new `Anchor` at the pose of intersection.

Shapes

Let's start out by adding a sphere at that pose.

We can use Sceneform's `ShapeFactory` API to create renderable shapes: cubes, cylinders, and spheres to which we can apply materials such as surface colors or textures.

Shapes

Create a new function in `MainActivity.kt`:

```
private fun addSphere(color: Int, anchor: Anchor, radius: Float, centerX : Float, centerY: Float, centerZ : Float) {
    MaterialFactory.makeOpaqueWithColor(this, com.google.ar.sceneform.rendering.Color(color))
        .thenAccept { material ->
            val shape = ShapeFactory.makeSphere(radius, Vector3(centerX, centerY, centerZ), material)
            addNodeToScene(anchor, shape)
        }
}
```

Nodes

All of the virtual content in a AR experience is organized as a **scene graph**.

A **scene graph** is basically an n-tree, made up of **nodes** which can each have 0...n children.

Nodes

We need a way to add our sphere to the **scene**. We'll do that by creating a `Node`, attached to an `Anchor` at the point of intersection.

Nodes

Create a new function in `MainActivity.kt`:

```
    private fun addNodeToScene(anchor: Anchor, renderable: Renderable) {
        val anchorNode = AnchorNode(anchor)
        val node = TransformableNode(arFragment.transformationSystem)
        node.renderable = renderable
        node.setParent(anchorNode)
        arFragment.arSceneView.scene.addChild(anchorNode)
    }
```

Anchors

Now we want to create an **anchor** based on the **hit result**, so that we can anchor our node to the pose on the plane we tap.

Anchors

Create an anchor based on the `HitResult`, and use it to add a sphere to the scene.

```
arFragment.setOnTapArPlaneListener { hitResult, plane, motionEvent ->
    val anchor = hitResult.createAnchor()
    addSphere(Color.RED, anchor, 0.1f, 0.0f, 0.15f, 0.0f)
}
```

Run it!

Sceneform plugin

Sceneform has an Android Studio plugin for importing, editing, and previewing 3D models.

Sceneform plugin: installation

Android Studio > Preferences > Plugins > browse repositories
Google Sceneform Tools (Beta)

Supported formats

Look in `app/sampledata/models`. We've provided some models.

- `.obj` - encodes the 3D geometry of the model (e.g. vertices, polygon faces)
- `.mtl` - material referenced by the `.obj`, describes the surface of the model (e.g. color, texture, reflection)
- `.png` - optional visual texture referenced by the `.mtl` to be mapped onto the surface of the model

Supported formats

In addition to Wavefront `obj`, Sceneform also supports importing:

- FBX, with or without animations
- glTF (animations not supported)

Sceneform assets: importing

1. Select `app/sampledata/models/coffee.obj` and then right mouse click to get the menu.
2. Pick `New > Sceneform asset`.
3. Click `Finish`.

Sceneform assets: editing

We've now converted into Sceneform's `.sfa` and `.sfb` formats.

- `.sfb` - Sceneform Binary, points to the models, material definitions, and textures in the source asset.
- `.sfa` - Sceneform Asset Definition, a human-readable description of the `.sfb`

Sceneform assets: editing

Loading assets

Renderables

Run it!

Snap a photo

We can take a photo of our AR scene, including the virtual content, by capturing the `SurfaceView` (the class that `ArSceneView` descends from).

Snap a photo

The app already includes a `CameraHelper.kt` class. Let's wire it up so that when the button is clicked, we take a photo.

```
fab.setOnClickListener { view ->
    camera.snap()
}
```

Run it!

Now for the fun stuff...

Cloud anchors

Augmented Faces

Grazie!