Ubiquitous and Mobile Computing Introduction to ARCore

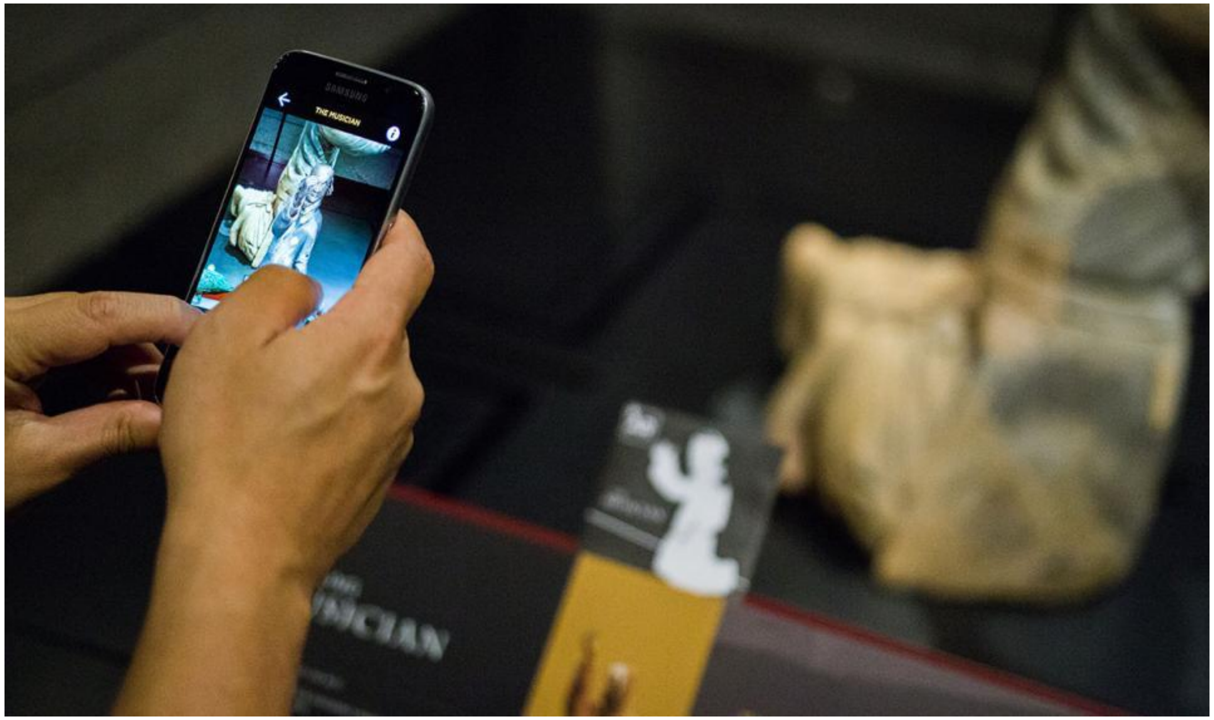
- Manas Mehta
- Theodoros Konstantopoulos
- Skyler Kim
- Khulood Alkhudaidi
- Aritra Kundu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

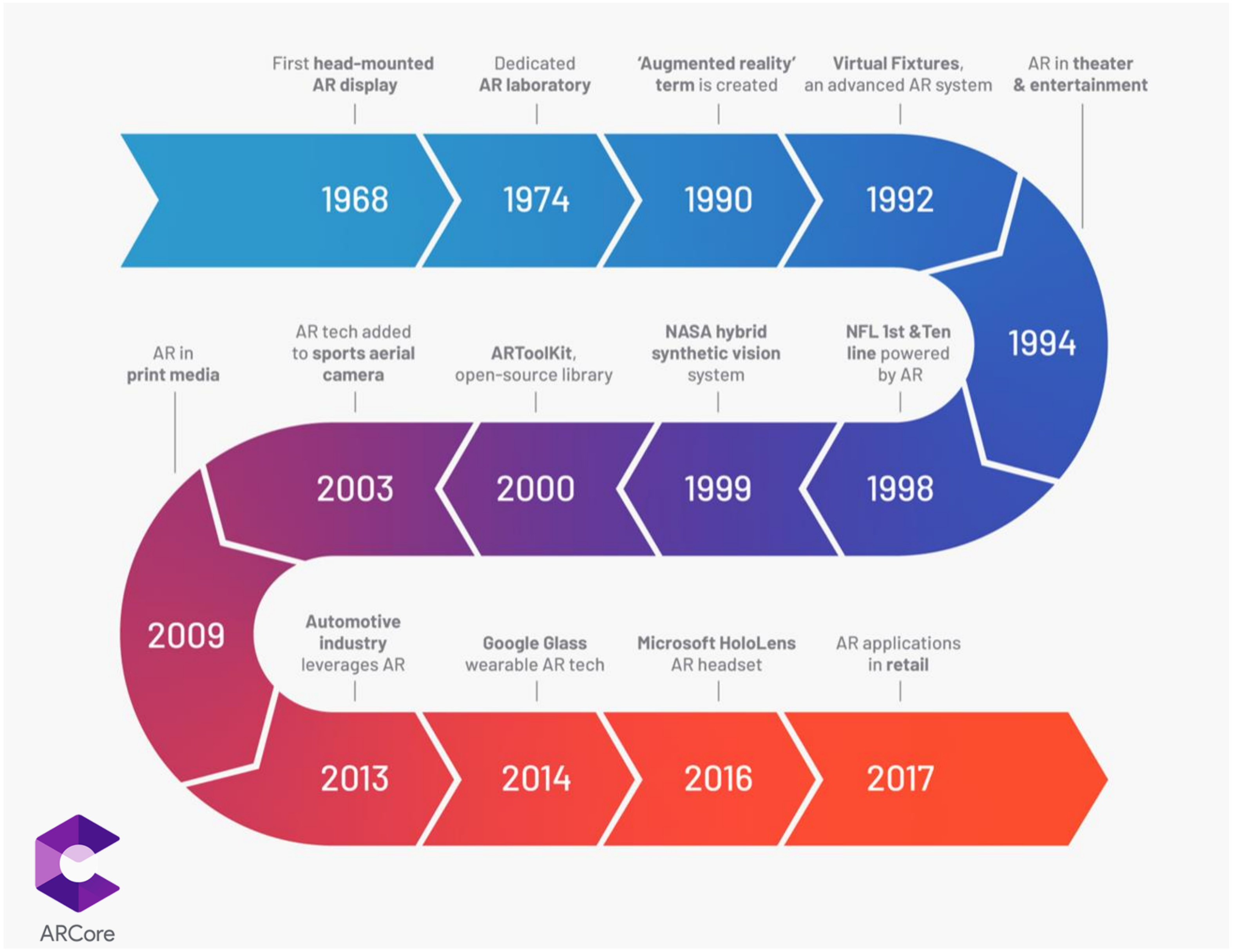
What is Augmented Reality?

- Projection of digital objects onto reality.
- Large application with mobile phones.



ARCore

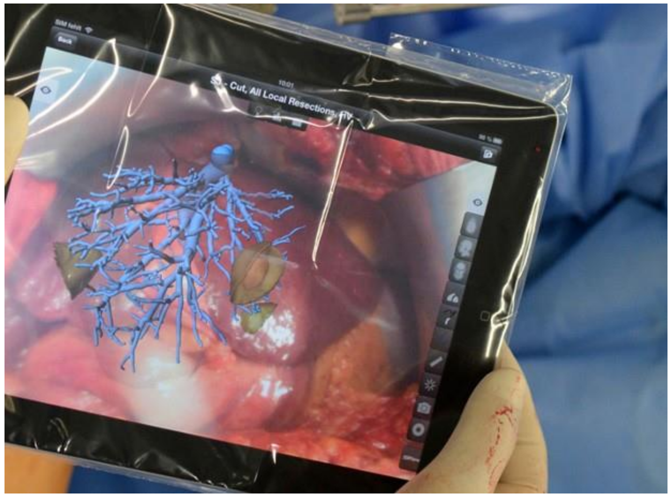
The History of Augmented Reality?

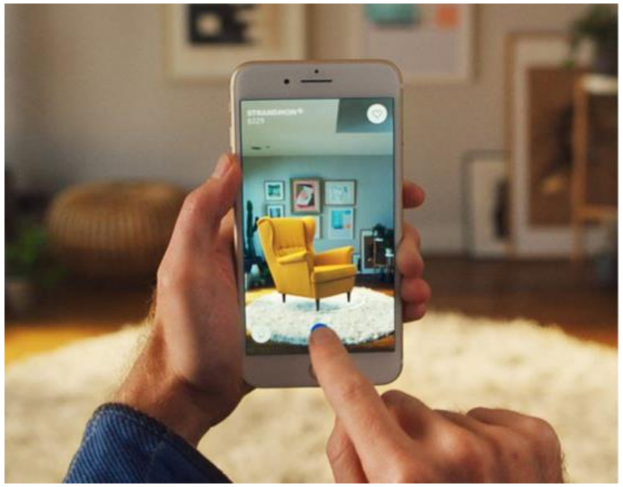


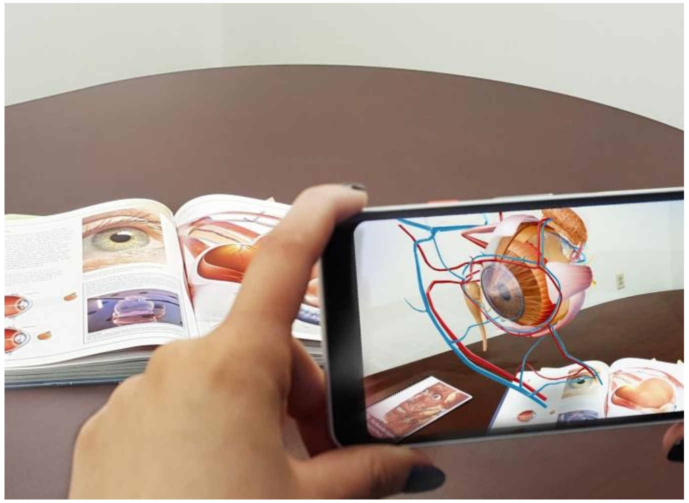
First head-mounted AR display
1968
Dedicated AR laboratory
1974
'Augmented reality' term is created
1990
Virtual Fixtures, an advanced AR system
1992
AR in theater & entertainment
1994
NFL 1st & Ten line powered by AR
1998
NASA hybrid synthetic vision system
1999
ARToolKit, open-source library
2000
AR tech added to sports aerial camera
2003
AR in print media
2009
Automotive industry leverages AR
2013
Google Glass wearable AR tech
2014
Microsoft HoloLens AR headset
2016
AR applications in retail
2017
ARCore

Problems can be solved by AR

- Medical training
- Education
- Online Shopping
- Logistics







ARCore

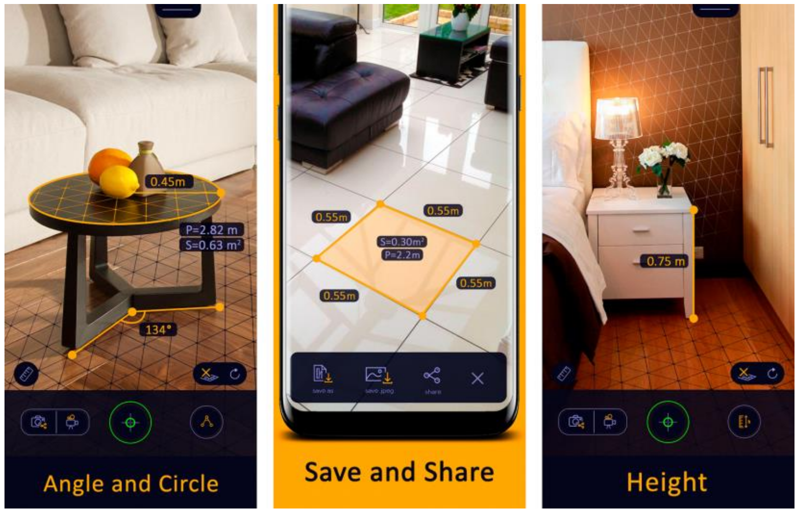
What is ARCore?

- ARCore is Google’s Software development kit for building augmented reality experiences.

- ARCore uses three key technologies:
  - Motion Tracking
  - Environmental Understanding
  - Light estimation

ARCore

Apps Using ARCore



ARCore

Other Uses of ARCore

unity

UNREAL
ENGINE

ARCore

Setting Up Project for AR Core

- Add permissions and data to Android Manifest.

```xml
<uses-sdk android:minSdkVersion="24" />

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


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

<application>
    …

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

- Add dependency to android app’s build.gradle.

```
dependencies {

    implementation 'com.google.ar:core:1.13.0'

}
```

Creating AR Core Session

```java
Session mSession;
boolean mUserRequestedInstall = true;

@Override
protected void onResume() {
    super.onResume();

    // request camera permissions
    requestPermissions(new String[]{Manifest.permission.CAMERA}, CAMERA_REQUEST_CODE);

    try {
        if (mSession == null) {
            switch (ArCoreApk.getInstance().requestInstall(this, mUserRequestedInstall)) {
                case INSTALLED:
                    // Success, create the AR session.
                    mSession = new Session(this);
                    break;
                case INSTALL_REQUESTED:
                    // Ensures next invocation of requestInstall() will either return
                    // INSTALLED or throw an exception.
                    mUserRequestedInstall = false;
                    return;
            }
        }
    } catch (UnavailableUserDeclinedInstallationException e) {
        ...
        return;
    } catch (UnavailableArcoreNotInstalledException | UnavailableApkTooOldException | UnavailableSdkTooOldException
        e.printStackTrace();
        return;
    }
```

ARCore

Tracking Planes in Scene

- Create GL Surface View in Activity.
- Create OpenGL program using Android calls.
- Get trackable planes from AR Core Session.
- Get Pose translation from AR Core Camera.
- Draw Planes with OpenGL calls based off Plane and Camera data.

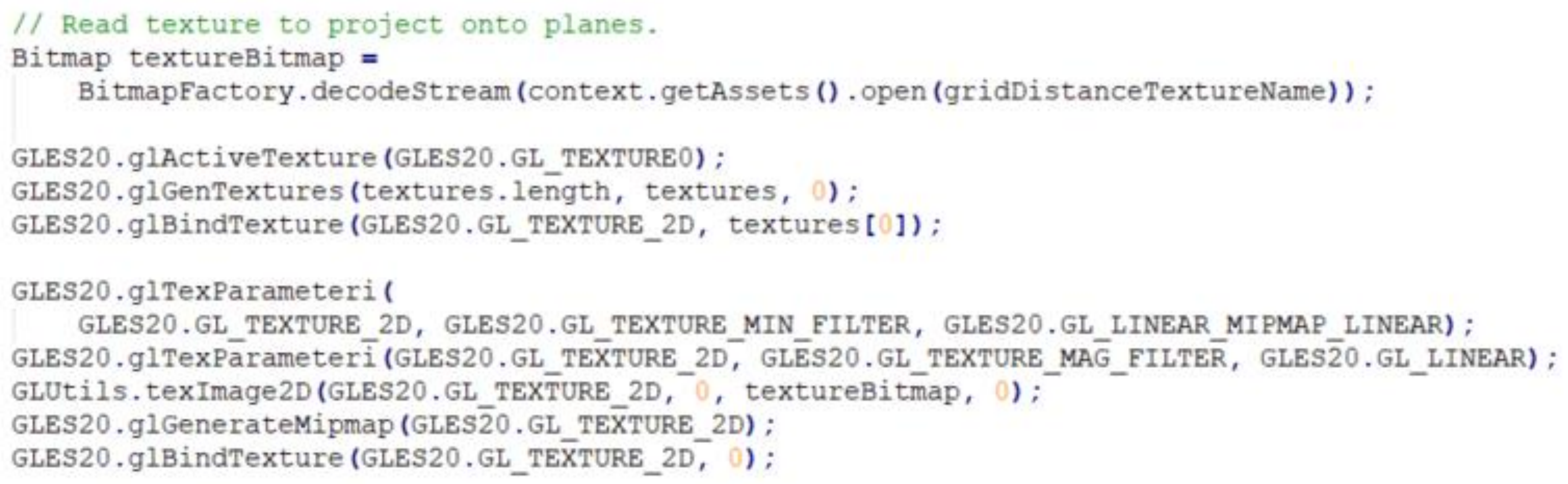
ARCore

Tracking Planes in Scene

```java
// Set up renderer.
surfaceView = findViewById(R.id.surfaceview);
surfaceView.setPreserveEGLContextOnPause(true);
surfaceView.setEGLContextClientVersion(2);
surfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0); // Alpha used for plane blending.
surfaceView.setRenderer(this);
surfaceView.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
surfaceView.setWillNotDraw(false);


// Create shaders to distinguish planes
int vertexShader =
    ShaderUtil.loadGLShader(TAG, context, GLES20.GL_VERTEX_SHADER, VERTEX_SHADER_NAME);
int passthroughShader =
    ShaderUtil.loadGLShader(TAG, context, GLES20.GL_FRAGMENT_SHADER, FRAGMENT_SHADER_NAME);

planeProgram = GLES20.glCreateProgram();
GLES20.glAttachShader(planeProgram, vertexShader);
GLES20.glAttachShader(planeProgram, passthroughShader);
GLES20.glLinkProgram(planeProgram);
GLES20.glUseProgram(planeProgram);
```

ARCore

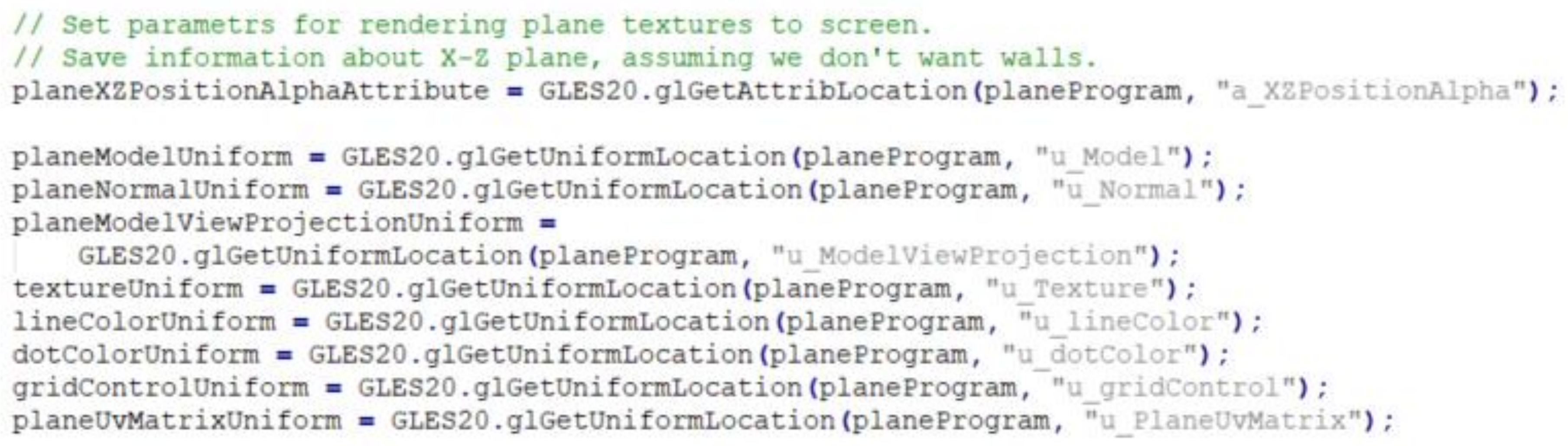
Tracking Planes in Scene



```
// Read texture to project onto planes.
Bitmap textureBitmap =
    BitmapFactory.decodeStream(context.getAssets().open(gridDistanceTextureName));

GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glGenTextures(textures.length, textures, 0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[0]);

GLES20.glTexParameteri(
    GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR_MIPMAP_LINEAR);
GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR);
GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, textureBitmap, 0);
GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, 0);
```

ARCore

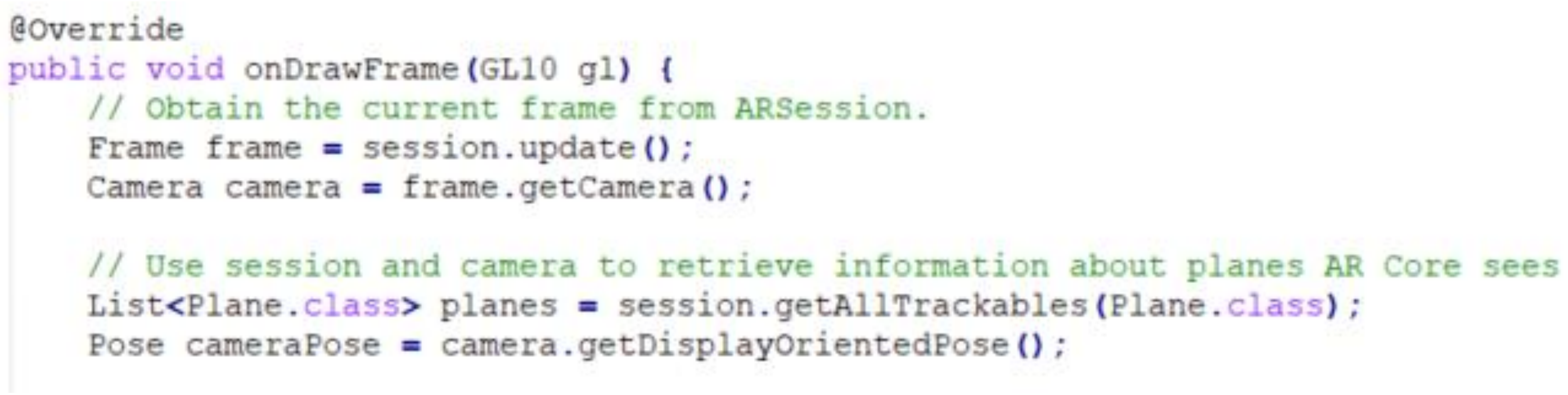
Tracking Planes in Scene



```
// Set parametrs for rendering plane textures to screen.
// Save information about X-Z plane, assuming we don't want walls.
planeXZPositionAlphaAttribute = GLES20.glGetAttribLocation(planeProgram, "a_XZPositionAlpha");

planeModelUniform = GLES20.glGetUniformLocation(planeProgram, "u_Model");
planeNormalUniform = GLES20.glGetUniformLocation(planeProgram, "u_Normal");
planeModelViewProjectionUniform =
    GLES20.glGetUniformLocation(planeProgram, "u_ModelViewProjection");
textureUniform = GLES20.glGetUniformLocation(planeProgram, "u_Texture");
lineColorUniform = GLES20.glGetUniformLocation(planeProgram, "u_lineColor");
dotColorUniform = GLES20.glGetUniformLocation(planeProgram, "u_dotColor");
gridControlUniform = GLES20.glGetUniformLocation(planeProgram, "u_gridControl");
planeUvMatrixUniform = GLES20.glGetUniformLocation(planeProgram, "u_PlaneUvMatrix");
```

ARCore

Tracking Planes in Scene



```
@Override
public void onDrawFrame(GL10 gl) {
    // Obtain the current frame from ARSession.
    Frame frame = session.update();
    Camera camera = frame.getCamera();

    // Use session and camera to retrieve information about planes AR Core sees
    List<Plane.class> planes = session.getAllTrackables(Plane.class);
    Pose cameraPose = camera.getDisplayOrientedPose();
```

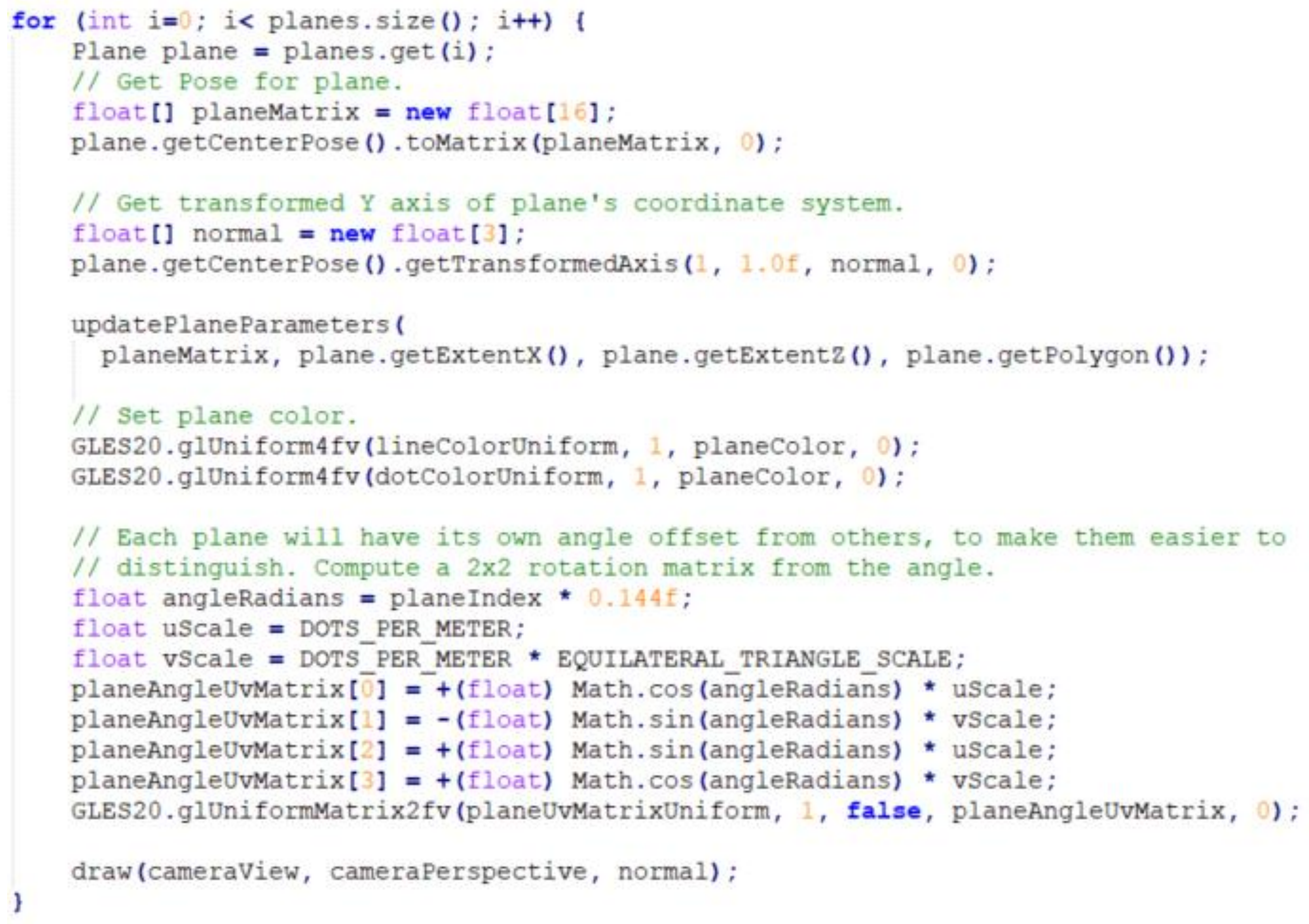
ARCore

Tracking Planes in Scene

```
// Planes are drawn with additive blending, masked by the alpha channel for occlusion.
// Start by clearing the alpha channel of the color buffer to 1.0.
GLES20.glClearColor(1, 1, 1, 1);
GLES20.glColorMask(false, false, false, true);
GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
GLES20.glColorMask(true, true, true, true);

// Disable depth write.
GLES20.glDepthMask(false);

// Additive blending, masked by alpha channel, clearing alpha channel.
GLES20.glEnable(GLES20.GL_BLEND);
GLES20.glBlendFuncSeparate(
    GLES20.GL_DST_ALPHA, GLES20.GL_ONE, // RGB (src, dest)
    GLES20.GL_ZERO, GLES20.GL_ONE_MINUS_SRC_ALPHA); // ALPHA (src, dest)

// Set up the shader.
GLES20.glUseProgram(planeProgram);

// Attach the texture.
GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[0]);
GLES20.glUniform1i(textureUniform, 0);

// Shared fragment uniforms.
GLES20.glUniform4fv(gridControlUniform, 1, GRID_CONTROL, 0);

// Enable vertex arrays
GLES20.glEnableVertexAttribArray(planeXZPositionAlphaAttribute);
```

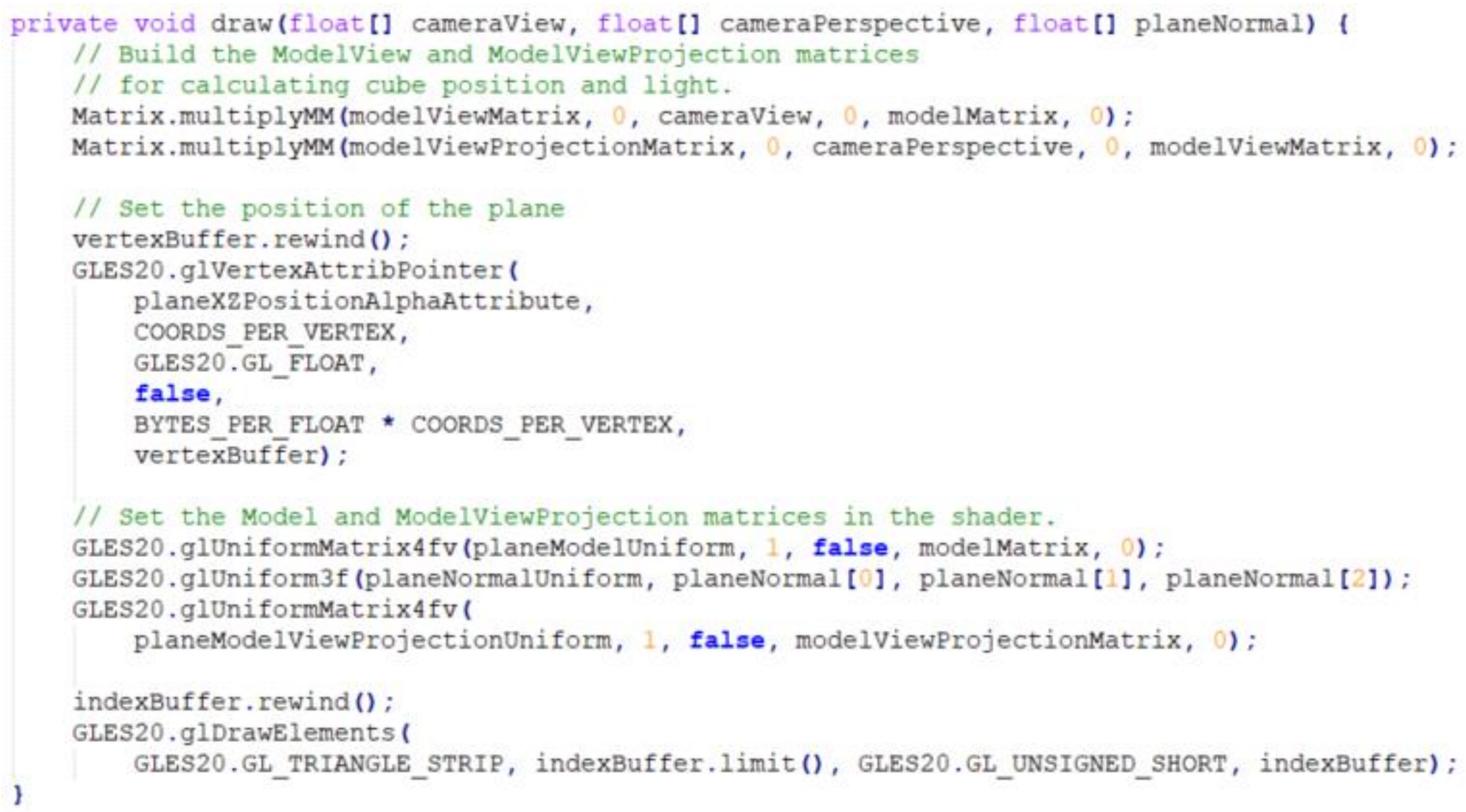
ARCore

Tracking Planes in Scene



```java
for (int i=0; i< planes.size(); i++) {
    Plane plane = planes.get(i);
    // Get Pose for plane.
    float[] planeMatrix = new float[16];
    plane.getCenterPose().toMatrix(planeMatrix, 0);

    // Get transformed Y axis of plane's coordinate system.
    float[] normal = new float[3];
    plane.getCenterPose().getTransformedAxis(1, 1.0f, normal, 0);

    updatePlaneParameters(
      planeMatrix, plane.getExtentX(), plane.getExtentZ(), plane.getPolygon());

    // Set plane color.
    GLES20.glUniform4fv(lineColorUniform, 1, planeColor, 0);
    GLES20.glUniform4fv(dotColorUniform, 1, planeColor, 0);

    // Each plane will have its own angle offset from others, to make them easier to
    // distinguish. Compute a 2x2 rotation matrix from the angle.
    float angleRadians = planeIndex * 0.144f;
    float uScale = DOTS_PER_METER;
    float vScale = DOTS_PER_METER * EQUILATERAL_TRIANGLE_SCALE;
    planeAngleUvMatrix[0] = +(float) Math.cos(angleRadians) * uScale;
    planeAngleUvMatrix[1] = -(float) Math.sin(angleRadians) * vScale;
    planeAngleUvMatrix[2] = +(float) Math.sin(angleRadians) * uScale;
    planeAngleUvMatrix[3] = +(float) Math.cos(angleRadians) * vScale;
    GLES20.glUniformMatrix2fv(planeUvMatrixUniform, 1, false, planeAngleUvMatrix, 0);

    draw(cameraView, cameraPerspective, normal);
}
```

Tracking Planes in Scene



```
private void draw(float[] cameraView, float[] cameraPerspective, float[] planeNormal) {
    // Build the ModelView and ModelViewProjection matrices
    // for calculating cube position and light.
    Matrix.multiplyMM(modelViewMatrix, 0, cameraView, 0, modelMatrix, 0);
    Matrix.multiplyMM(modelViewProjectionMatrix, 0, cameraPerspective, 0, modelViewMatrix, 0);

    // Set the position of the plane
    vertexBuffer.rewind();
    GLES20.glVertexAttribPointer(
        planeXZPositionAlphaAttribute,
        COORDS_PER_VERTEX,
        GLES20.GL_FLOAT,
        false,
        BYTES_PER_FLOAT * COORDS_PER_VERTEX,
        vertexBuffer);

    // Set the Model and ModelViewProjection matrices in the shader.
    GLES20.glUniformMatrix4fv(planeModelUniform, 1, false, modelMatrix, 0);
    GLES20.glUniform3f(planeNormalUniform, planeNormal[0], planeNormal[1], planeNormal[2]);
    GLES20.glUniformMatrix4fv(
        planeModelViewProjectionUniform, 1, false, modelViewProjectionMatrix, 0);

    indexBuffer.rewind();
    GLES20.glDrawElements(
        GLES20.GL_TRIANGLE_STRIP, indexBuffer.limit(), GLES20.GL_UNSIGNED_SHORT, indexBuffer);
}
```

ARCore

Rendering Objects In AR

- Save .obj file in res folder.
- Create OpenGL program using Android calls.
- Save anchor positions of places where objects should be.
- Use OpenGL program to render 3D object at anchor point.
- Update render from OpenGL program on SurfaceView Callback.

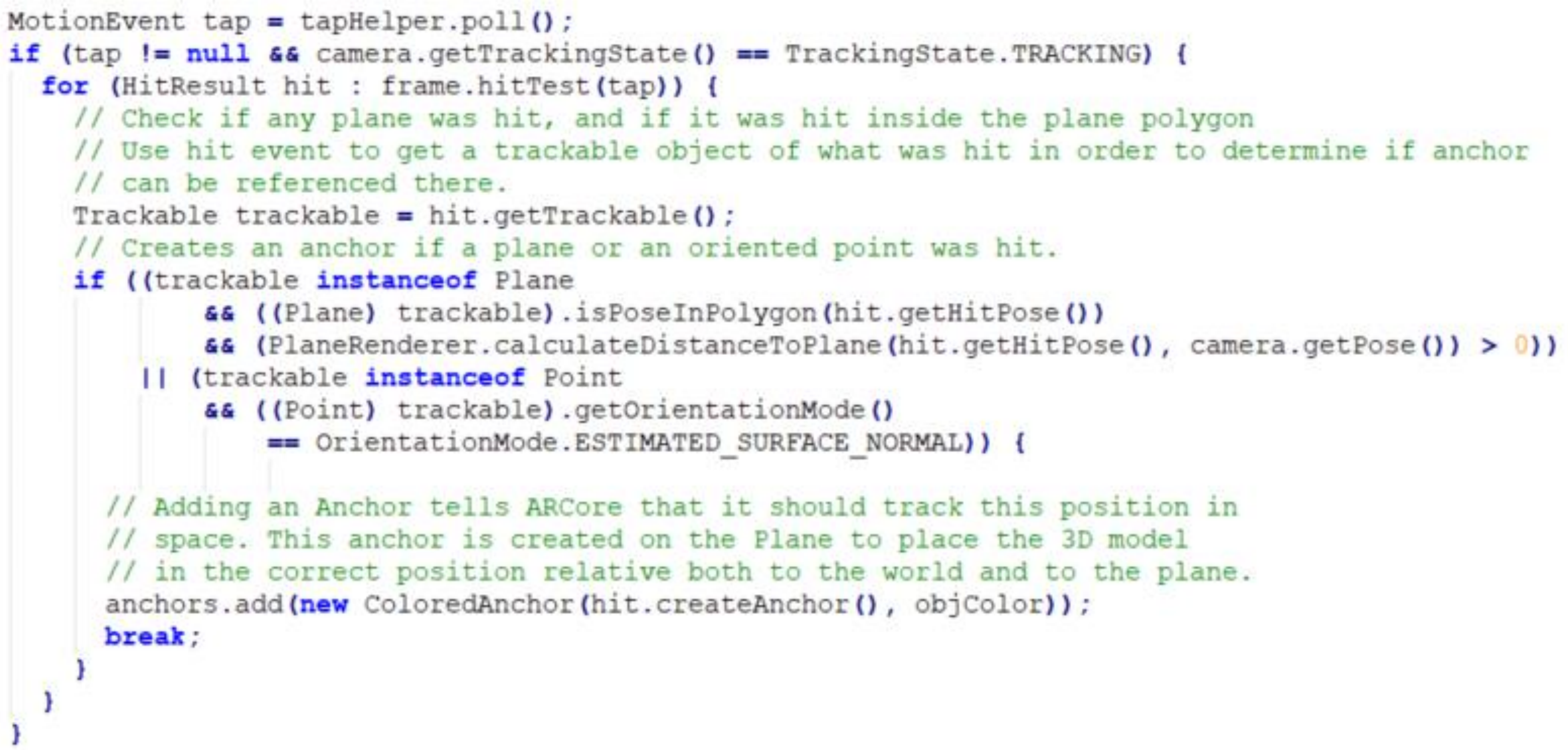
ARCore

Prepare 3D Models for Render

```
// Read the obj file.
InputStream objInputStream = context.getAssets().open(objAssetName);
Obj obj = ObjReader.read(objInputStream);
obj = ObjUtils.convertToRenderable(obj);

// Obtain the data from the OBJ, as direct buffers:
IntBuffer wideIndices = ObjData.getFaceVertexIndices(obj, 3);
FloatBuffer vertices = ObjData.getVertices(obj);
FloatBuffer texCoords = ObjData.getTexCoords(obj, 2);
FloatBuffer normals = ObjData.getNormals(obj);

// Convert int indices to shorts for GL ES 2.0 compatibility
ShortBuffer indices =
    ByteBuffer.allocateDirect(2 * wideIndices.limit())
        .order(ByteOrder.nativeOrder())
        .asShortBuffer();
while (wideIndices.hasRemaining()) {
    indices.put((short) wideIndices.get());
}
indices.rewind();
```

ARCore

Setting Anchors



```
MotionEvent tap = tapHelper.poll();
if (tap != null && camera.getTrackingState() == TrackingState.TRACKING) {
  for (HitResult hit : frame.hitTest(tap)) {
    // Check if any plane was hit, and if it was hit inside the plane polygon
    // Use hit event to get a trackable object of what was hit in order to determine if anchor
    // can be referenced there.
    Trackable trackable = hit.getTrackable();
    // Creates an anchor if a plane or an oriented point was hit.
    if ((trackable instanceof Plane
            && ((Plane) trackable).isPoseInPolygon(hit.getHitPose())
            && (PlaneRenderer.calculateDistanceToPlane(hit.getHitPose(), camera.getPose()) > 0))
        || (trackable instanceof Point
            && ((Point) trackable).getOrientationMode()
                == OrientationMode.ESTIMATED_SURFACE_NORMAL)) {

      // Adding an Anchor tells ARCore that it should track this position in
      // space. This anchor is created on the Plane to place the 3D model
      // in the correct position relative both to the world and to the plane.
      anchors.add(new ColoredAnchor(hit.createAnchor(), objColor));
      break;
    }
  }
}
```

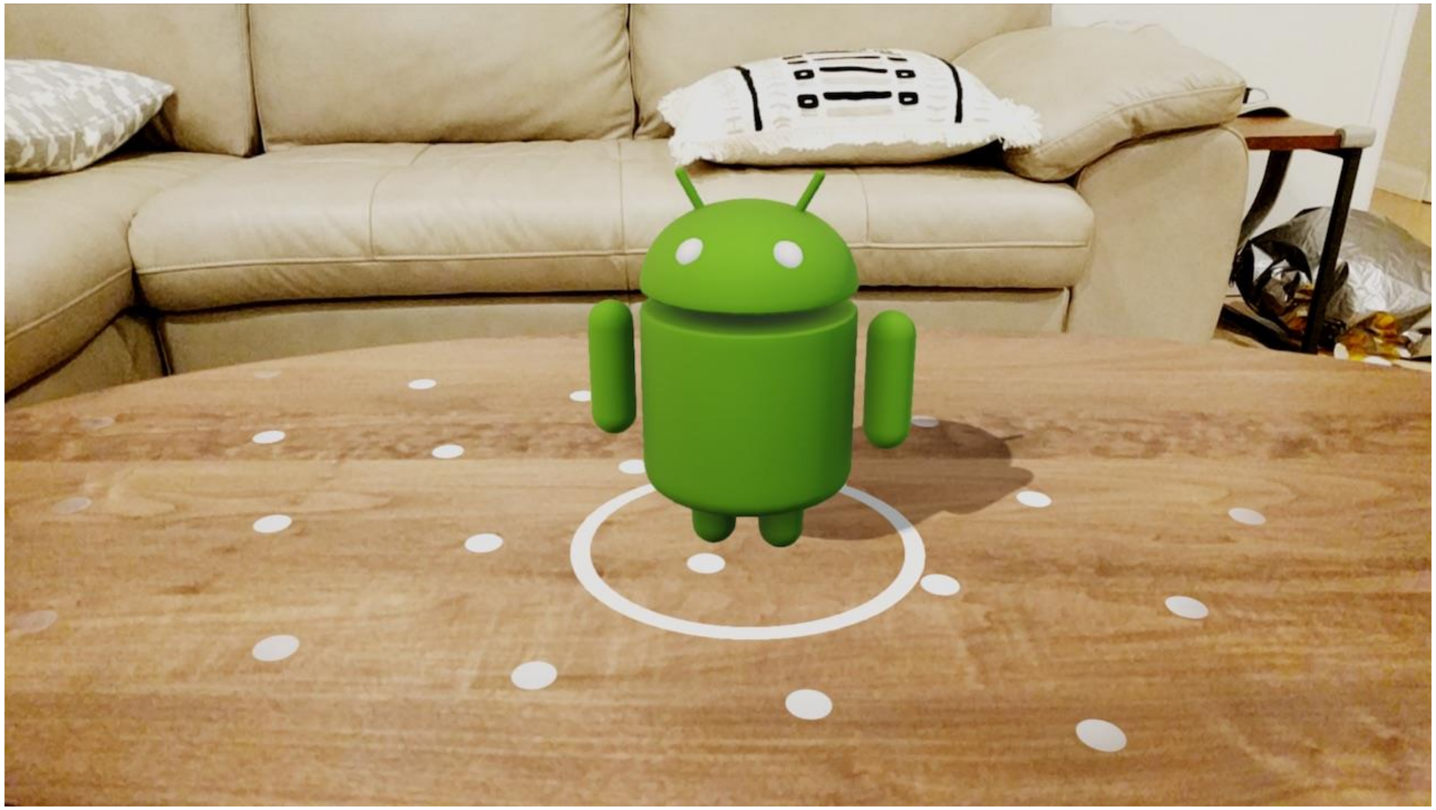
ARCore

Drawing Objects

```
// Get projection matrix.
float[] projmtx = new float[16];
camera.getProjectionMatrix(projmtx, 0, 0.1f, 100.0f);

// Get camera matrix and draw.
float[] viewmtx = new float[16];
camera.getViewMatrix(viewmtx, 0);

float scaleFactor = 1.0f

// Go through each anchor and render a model.
for (ColoredAnchor coloredAnchor : anchors) {
    if (coloredAnchor.anchor.getTrackingState() != TrackingState.TRACKING) {
        continue;
    }
    // Get the current pose of an Anchor in world space. The Anchor pose is updated
    // during calls to session.update() as ARCore refines its estimate of the world.
    coloredAnchor.anchor.getPose().toMatrix(anchorMatrix, 0);


    // Update 4x4 matrix containing translation from virtual world to real world.
    float[] scaleMatrix = new float[16];
    Matrix.setIdentityM(scaleMatrix, 0);
    scaleMatrix[0] = scaleFactor;
    scaleMatrix[5] = scaleFactor;
    scaleMatrix[10] = scaleFactor;
    Matrix.multiplyMM(modelMatrix, 0, modelMatrix, 0, scaleMatrix, 0);

    // Draw the .obj file
    draw(viewmtx, projmtx, colorCorrectionRgba, coloredAnchor.color);
}
```

ARCore

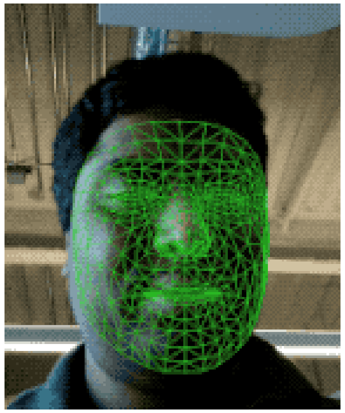
AR Core Example

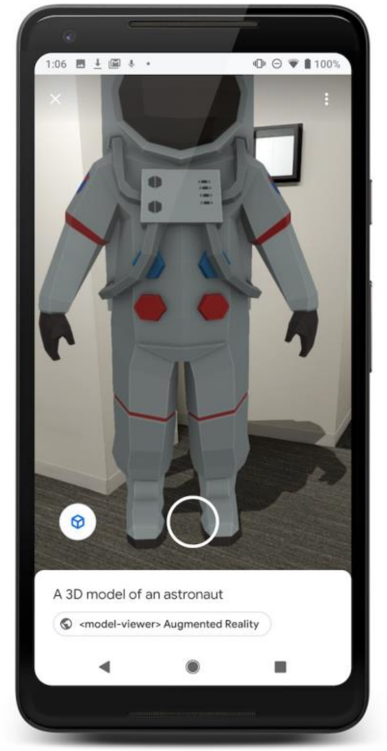


ARCore

Other Features of AR Core

- Viewing 3D models in AR from an Android Browser.
- Augmented Faces + Face detection.
- Cloud Anchor objects via AR Core Cloud Anchor API.





ARCore

Questions?

References

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

https://github.com/google-ar/arcore-android-sdk

https://learn.g2.com/history-of-augmented-reality

https://www.ualberta.ca/science/science-news/2018/january/augmented-reality-tech-see-under-skin-without-scalpel

https://www.wikitude.com/blog-augmented-reality-101-ar-top-use-cases/

https://www.fi.edu/what-is-augmented-reality

ARCore