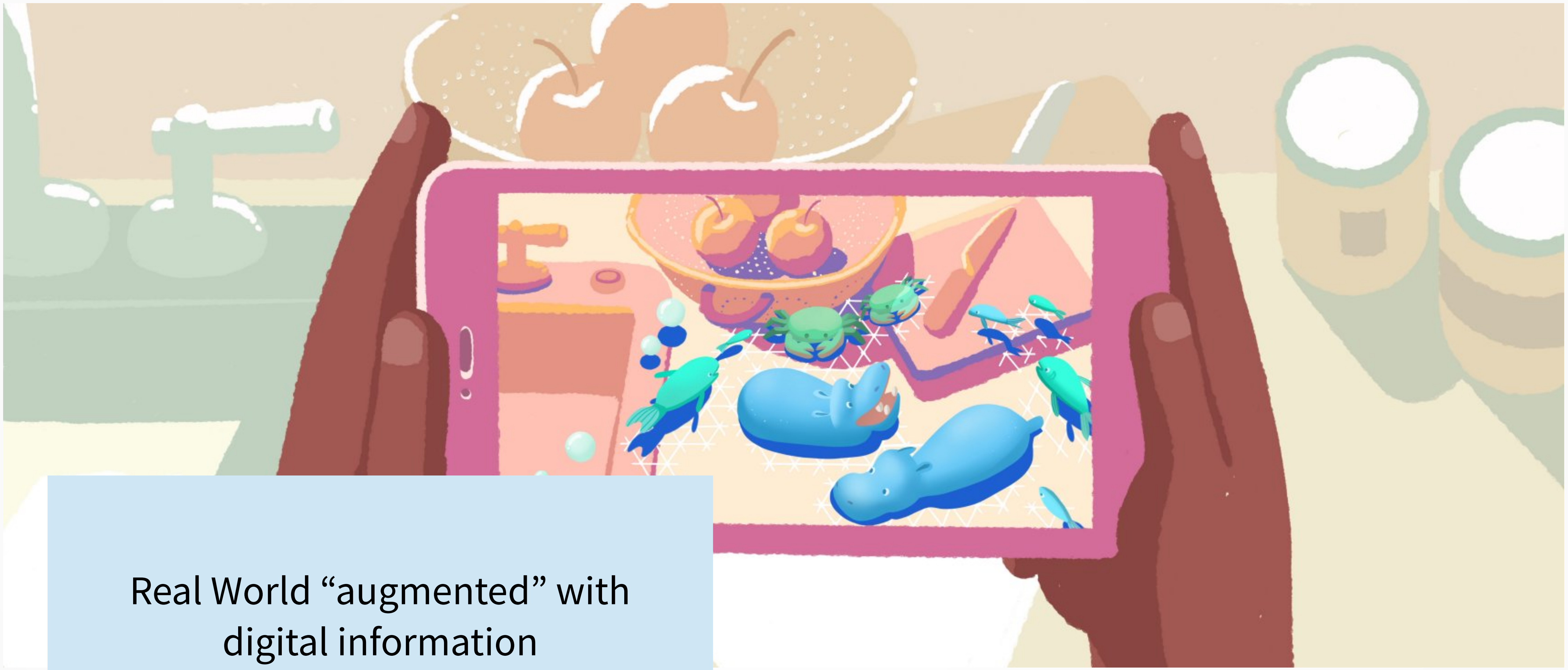
Android Augmented Reality

Programming Models for Emerging Platforms



Real World “augmented” with digital information

Real World “augmented” with digital information

Need some “window” into the digital world

A Couple of Neat Examples

- Microsoft Hololens
  - https://www.youtube.com/watch?v=xga kdcEzVwg

https://www.youtube.com/watch?v=gZhQCV

- Apple AR Kit
  - https://www.youtube.com/watch?v=gZh QCVSvq5E

ARCore

- Modern framework for developing augmented reality apps
- Modern framework means modern API
  - Uses a DSL style which we will examine later
  - Relies on Java 1.8 lambda expressions

Java Lambda Expressions

- Java's answer to the growing desire for more *general* programming abstractions
  - Anonymous functions
- Most modern languages are patching these abstractions in (and they fit in odd ways due to backward compatibility)
- We just need to understand the basics

Java Lambda Expressions

```
(int x)->System.out.println(2*x);
```

A function that takes an int x and prints 2*x

But functions aren’t quite first class values like they are in Go

Java Lambda Expressions

```
interface FuncInterface {
    void abstractFun(int x);
}

class Test {
  public static void main(String args[]) {
    FuncInterface fobj =
      (int x)->System.out.println(2*x);
    fobj.abstractFun(5);
  }
}
```

Create a “unctional interface” (interface with a single abstract method)

Implement the interface via a lambda

Java Lambda Expressions

```
interface FuncInterface {
    void abstractFun(int x);
}

class Test {
  public static void main(String args[]) {
    FuncInterface fobj =
      (int x)->System.out.println(2*x);
    fobj.abstractFun(5);
  }
}
```

Create a “unctional interface” (interface with a single abstract method)

Implement the interface via a lambda

Are lambdas orthogonal to Java ?

Java Lambda Expressions

- When a features are not orthogonal, you will see a “wort” in the language
  - Java lambda / functional interfaces *feel* weird
- Java lambda expressions better suited for Java streams

Java Streams

- Java’s answer for growing desire for less mutability (more “pure”) in a language
  - Immutable code is less error prone, easier to program for concurrency, easier to reason about

Java Streams

```java
List<Integer> numbers = Arrays.asList(2,3,4,5,1)
numbers.stream()
   .map(x -> x * x)
   .filter(x -> x > 9)
   .forEach(x -> System.out.println(x));
```

Chain a bunch of operations together on a “stream” of data

stream method converts java Collection interface into stream

Java Streams

```
double average = roster
    .stream()
    .filter(p -> p.getGender() == Person.Sex.MALE)
    .mapToInt(Person::getAge)
    .average()
    .getAsDouble();
```

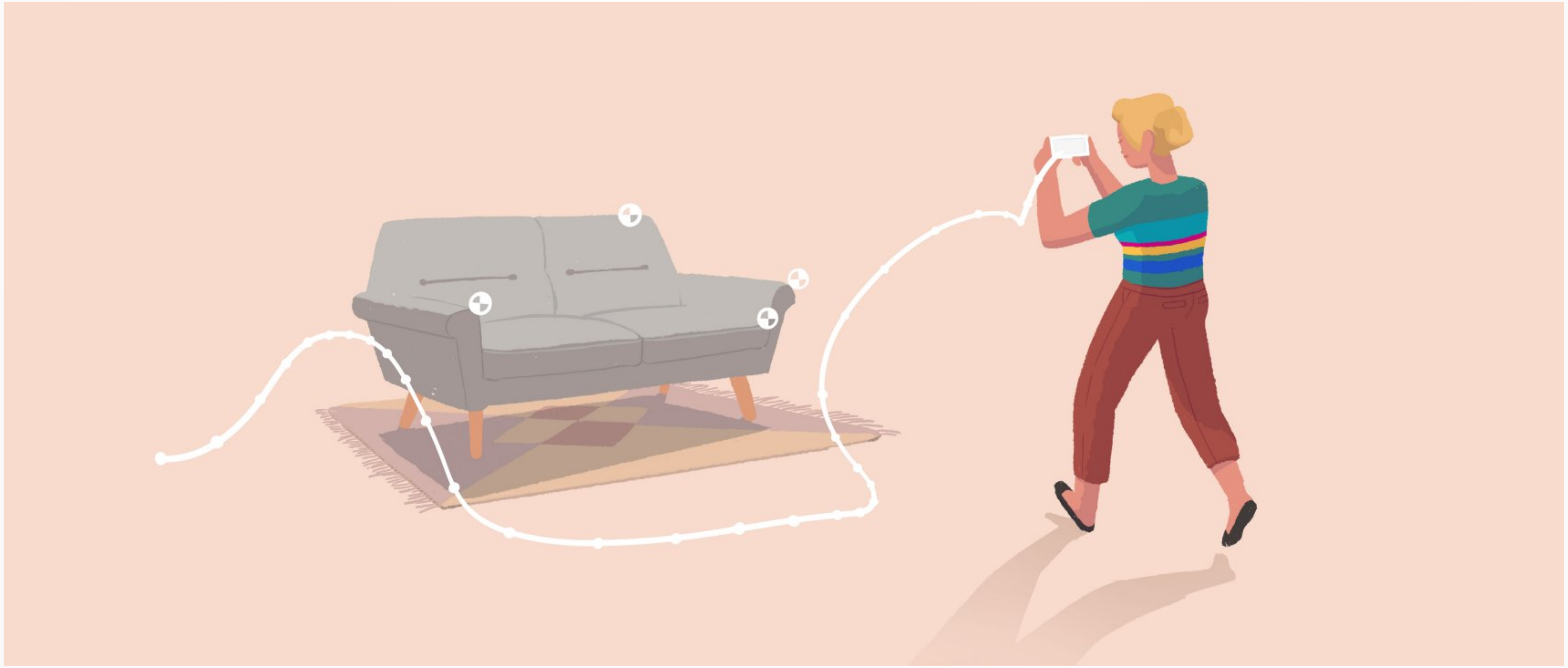
```
double average = roster
    .parallelStream()
    .filter(p -> p.getGender() == Person.Sex.MALE)
    .mapToInt(Person::getAge)
    .average()
    .getAsDouble();
```

Simply change the method to make parallel (although we know its not that simple)

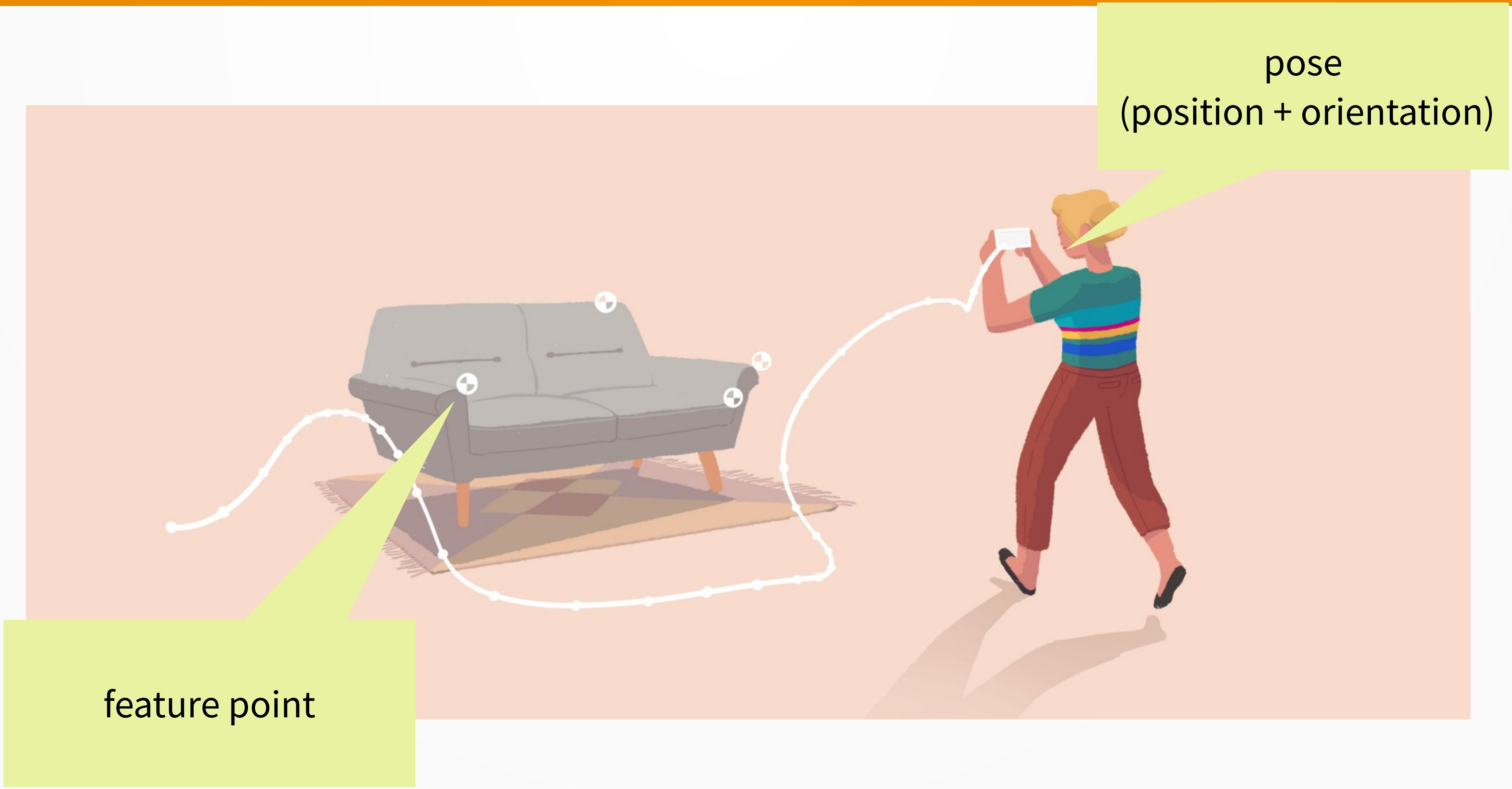
ARCore

- Google framework for building augmented reality applications
  - 1. Motion tracking
  - 2. Environment understanding
  - 3. Light estimation

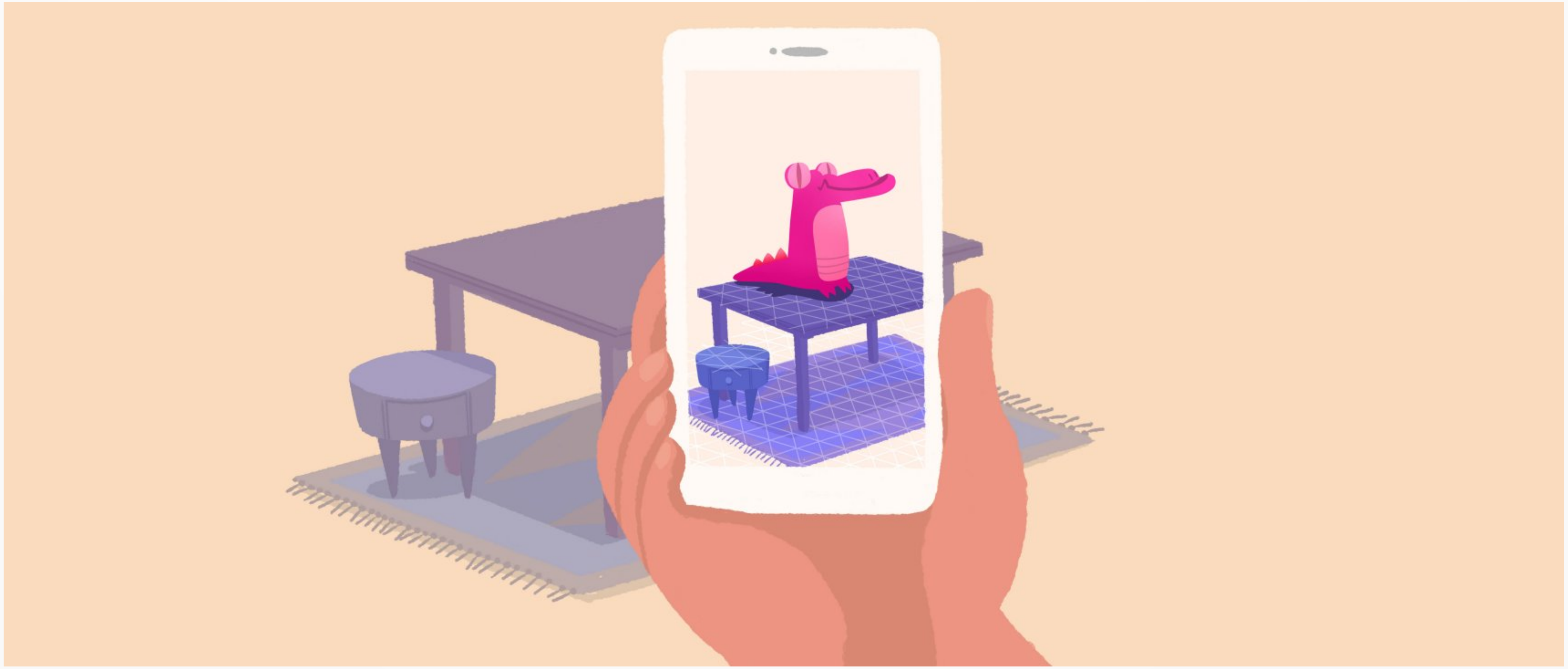
Motion Tracking



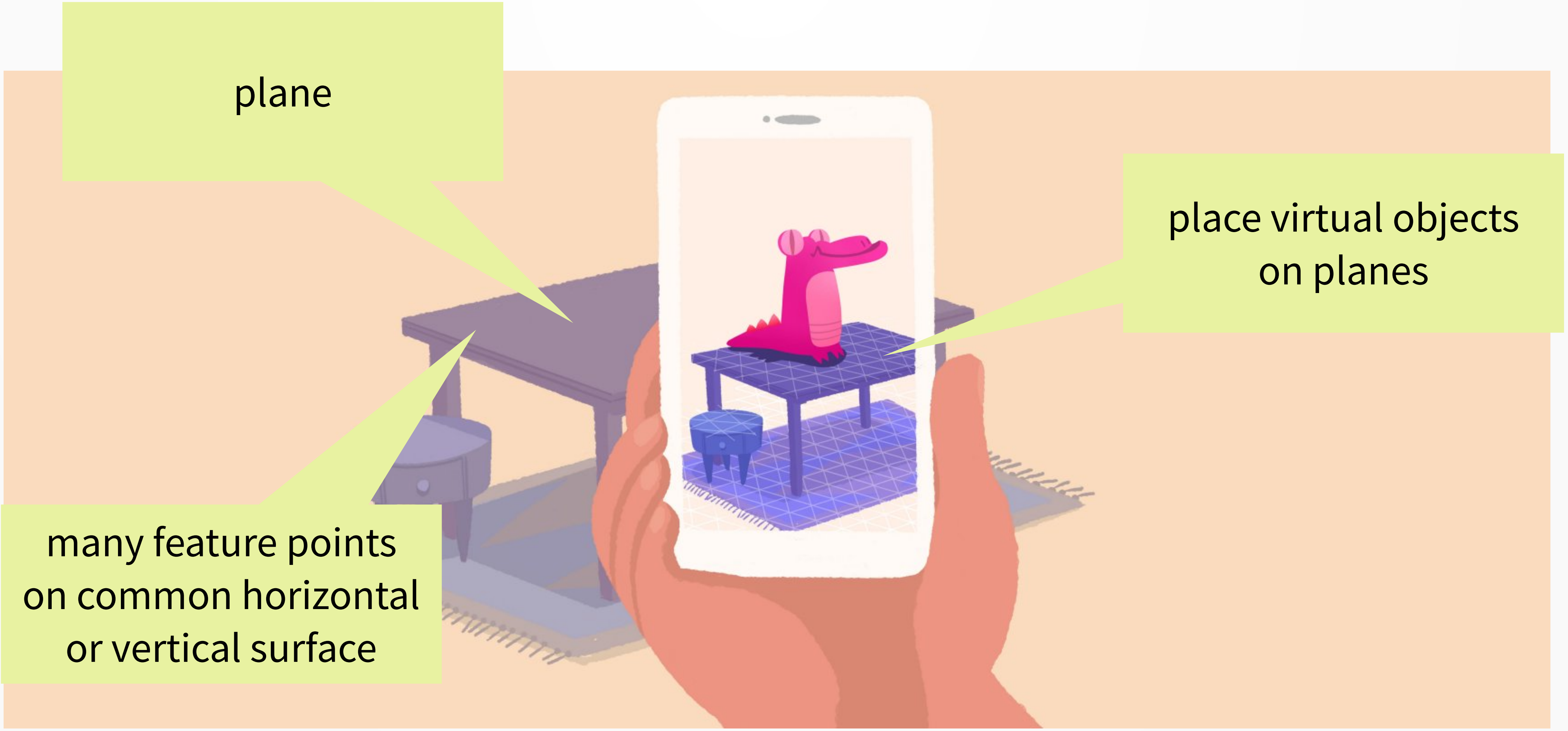
Motion Tracking



Environmental Understanding



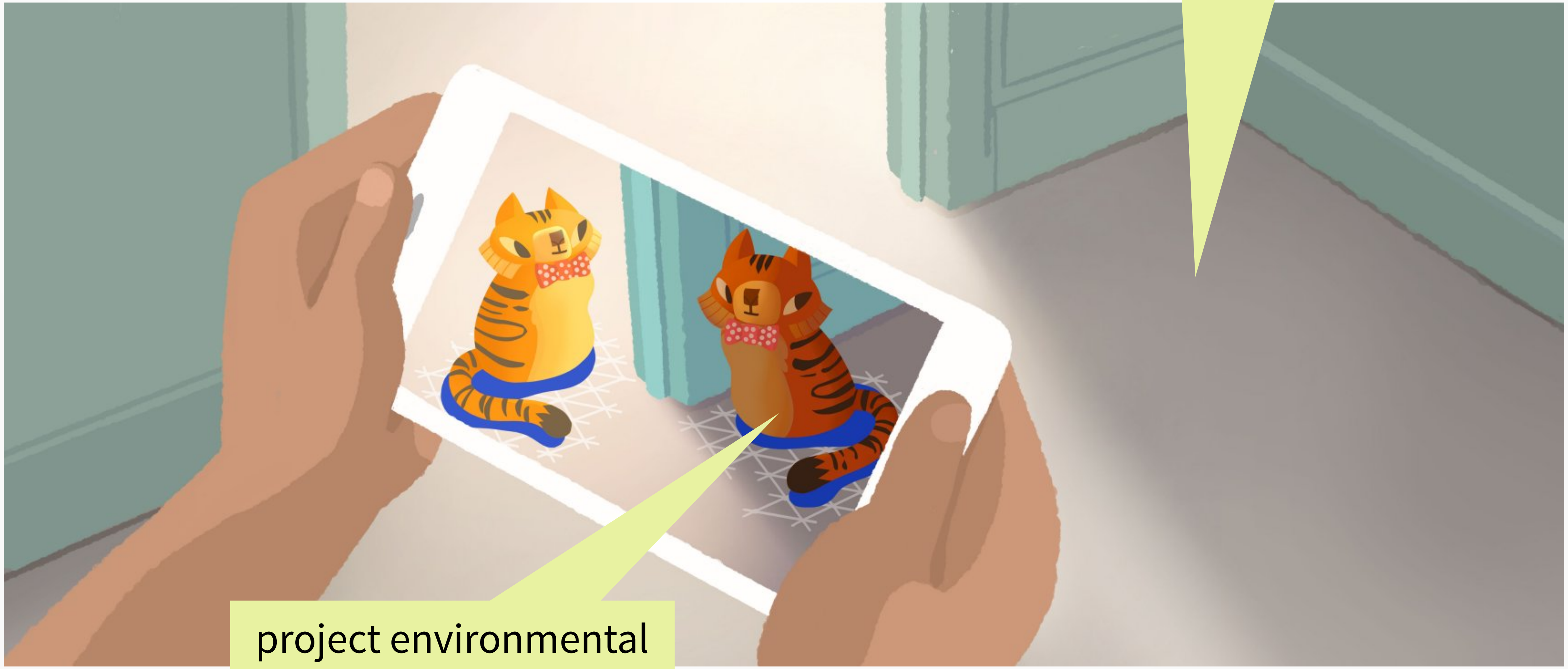
Environmental Understanding


plane
place virtual objects on planes
many feature points on common horizontal or vertical surface

Light Estimation

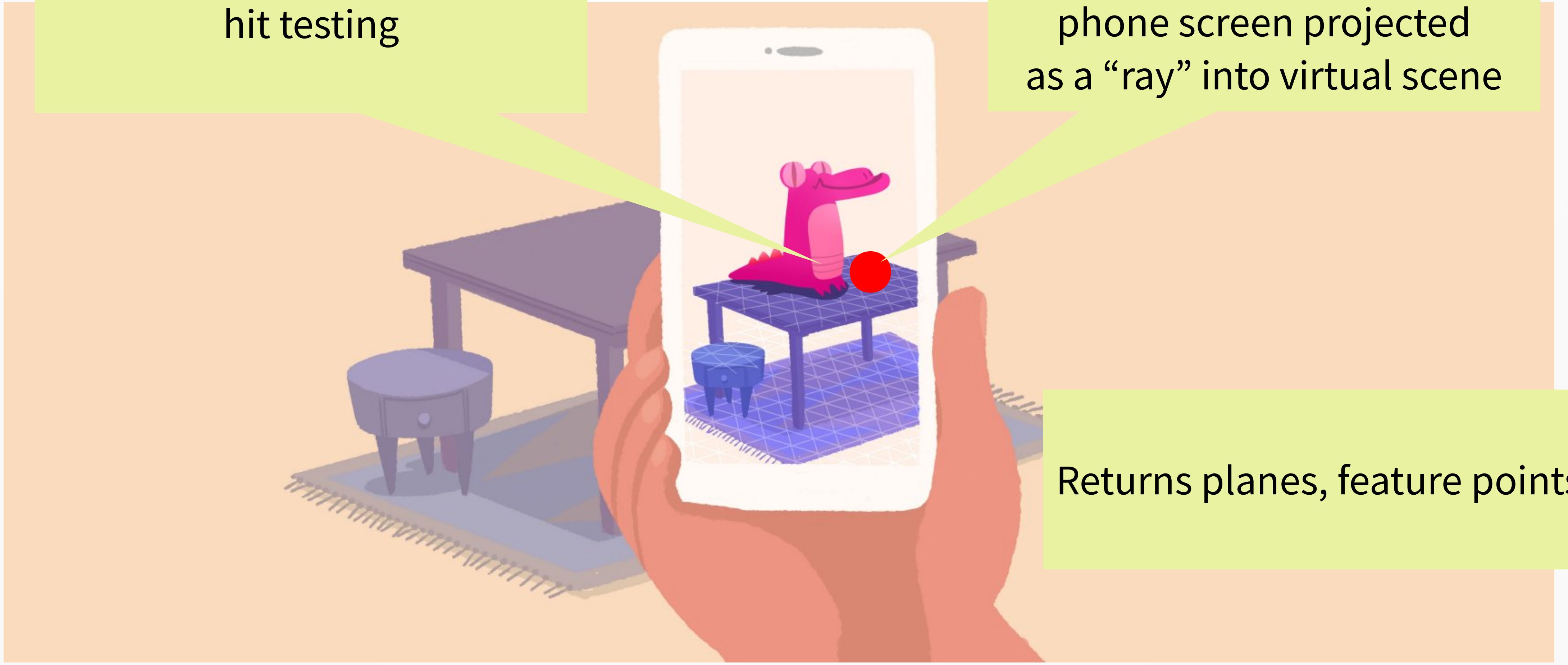
Light Estimation

project environmental
onto virtual objects
for realism



project environmental
onto virtual objects
for realism

User Interaction


hit testing
(x,y) co-ordinate of phone screen projected as a “ray” into virtual scene
Returns planes, feature points

Android ARCore

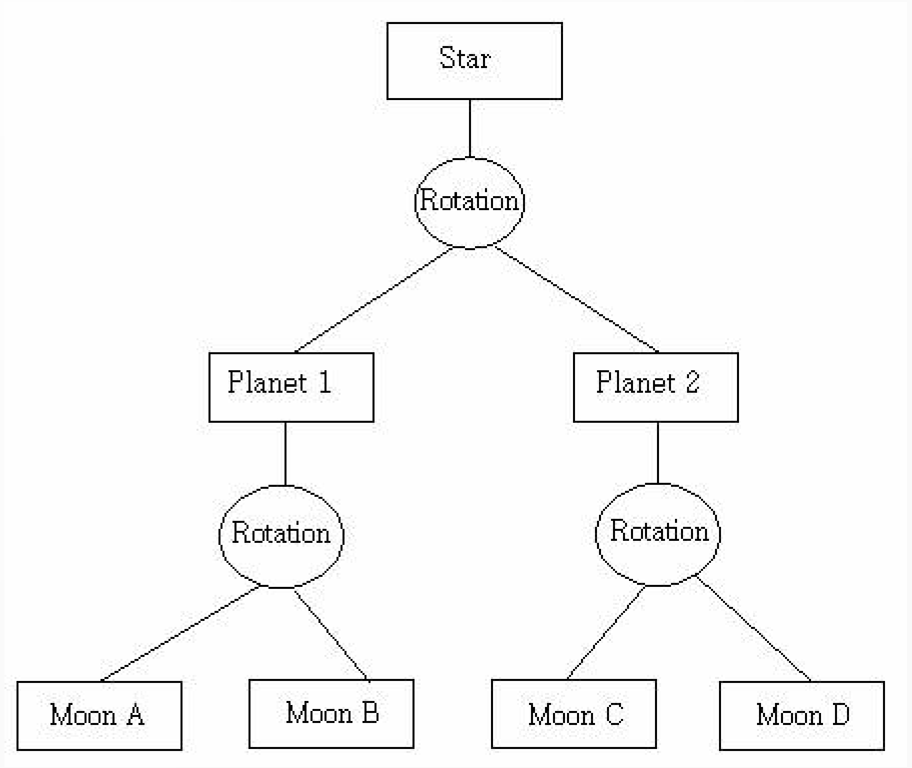
- ARCore provides the sensory detection for augmented reality
- Independent of the *rendering* (although without AR + Rendering, we don’t really get anything interesting)
  - We could use OpenGL, but that requires advanced graphics knowledge
  - Alternative is to use a higher-level API for managing a *Scene*

Sceneform

- Makes it straightforward to render 3D scenes without using OpenGL
  - High-level scene graph API
  - *Physically* based renderer
  - Android studio plugin for working with 3D assets

Scene Graph

- Data structure for managing a “Scene”
  - Parent-child relationship among nodes
    - Position parent object, child objects positioned based of parent
  - Used to cull scene
  - Collision detection etc

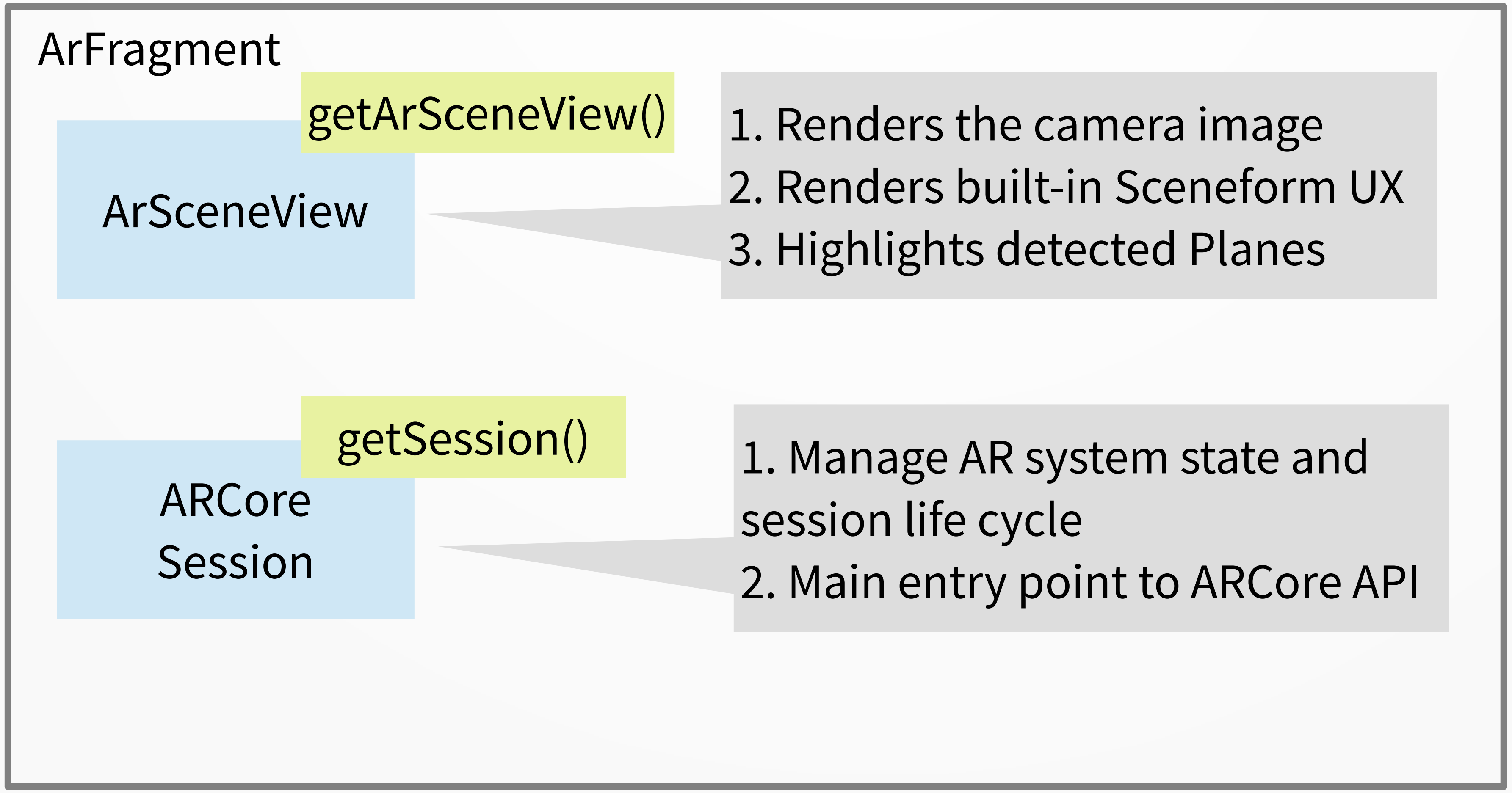


Physically Based Rendering

- Rendering style that strives for accurate modeling of light flow
  - photo-realistic rendering
  - https://labs.sketchfab.com/siggraph2014/

ArFragment

- Bootstrap the app with **`ArFragment`**


ArFragment
getArSceneView()
ArSceneView
1. Renders the camera image
2. Renders built-in Sceneform UX
3. Highlights detected Planes
getSession()
ARCore Session
1. Manage AR system state and session life cycle
2. Main entry point to ARCore API

ArFragment

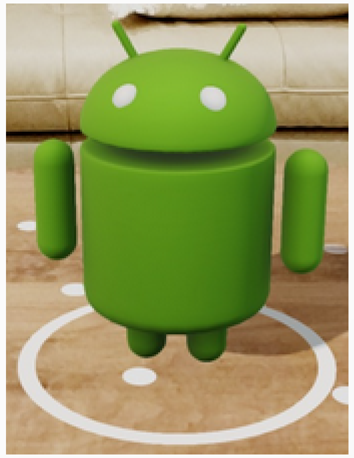
- Bootstrap the app with **`ArFragment`**

In Activity layout

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

ARCore Rendererable

- Represents something that can be rendered in a scene
- Plugin for creating Renderables from 3d models



Add in gradle build file

```
apply plugin: 'com.google.ar.sceneform.plugin'

sceneform.asset('sampledata/models/andy.obj', // 'Source Asset Path' specified during import.
        'default',                 // 'Material Path' specified during import.
        'sampledata/models/andy.sfa', // '.sfa Output Path' specified during import.
        'src/main/res/raw/andy')     // '.sfb Output Path' specified during import.
```

ARCore Rendererable

Somewhere in your ArFragment activity

```
private ModelRenderable andyRenderable;

@Override
protected void onCreate(Bundle savedInstanceState) {
    …

    ModelRenderable.builder()
        .setSource(this, R.raw.andy)
        .build()
        .thenAccept(renderable -> andyRenderable = renderable)
        .exceptionally(
            throwable -> {
            Log.e(TAG, "Unable to load Renderable.", throwable);
            return null;
        });
}
```

May look weird: ARCore uses an embedded DSL syntax for API design

ARCore Rendererable

Somewhere in your ArFragment activity

```
private ModelRenderable andyRenderable;

@Override
protected void onCreate(Bundle savedInstanceState) {
    …

    ModelRenderable.builder()
        .setSource(this, R.raw.andy)
        .build()
        .thenAccept(renderable -> andyRenderable = renderable)
        .exceptionally(
            throwable -> {
            Log.e(TAG, "Unable to load Renderable.", throwable);
            return null;
        });
}
```

Located from the resource directory

Lambda function to set andyRenderable field with result of build

Exception handler

ARCore Rendererable

Somewhere in your ArFragment activity

```
private ModelRenderable andyRenderable;

@Override
protected void onCreate(Bundle savedInstanceState) {
    …

    ModelRenderable.builder()
        .setSource(this, R.raw.andy)
        .build()
        .thenAccept(renderable -> andyRenderable = renderable)
        .exceptionally(
            throwable -> {
            Log.e(TAG, "Unable to load Renderable.", throwable);
            return null;
        });
}
```

Located from the resource directory

Lambda function to set andyRenderable field with result of build

Exception handler

Get used to this API style. It's getting popular.

ARCore Rendererable

- Create Renderables from…
  - Standard Android Views (ViewRenderable)
  - 3D Assets (like previous example)
  - Basic shapes and materials (programmable)

Building a Scene

- Attach Renderables to Nodes in the scene graph
- Nodes have a parent-child relationship

```
Node node = new Node();
node.setParent(arFragment.getArSceneView().getScene());
node.setRenderable(andyRenderable);
```

Add it to the root node

Node will "render" the andy object

Building a Scene

- Scene graph abstracts away some of the rougher details with rendering and collision detection
- Specifically built to work with ARCore
  - hit testing
  - planes
  - more

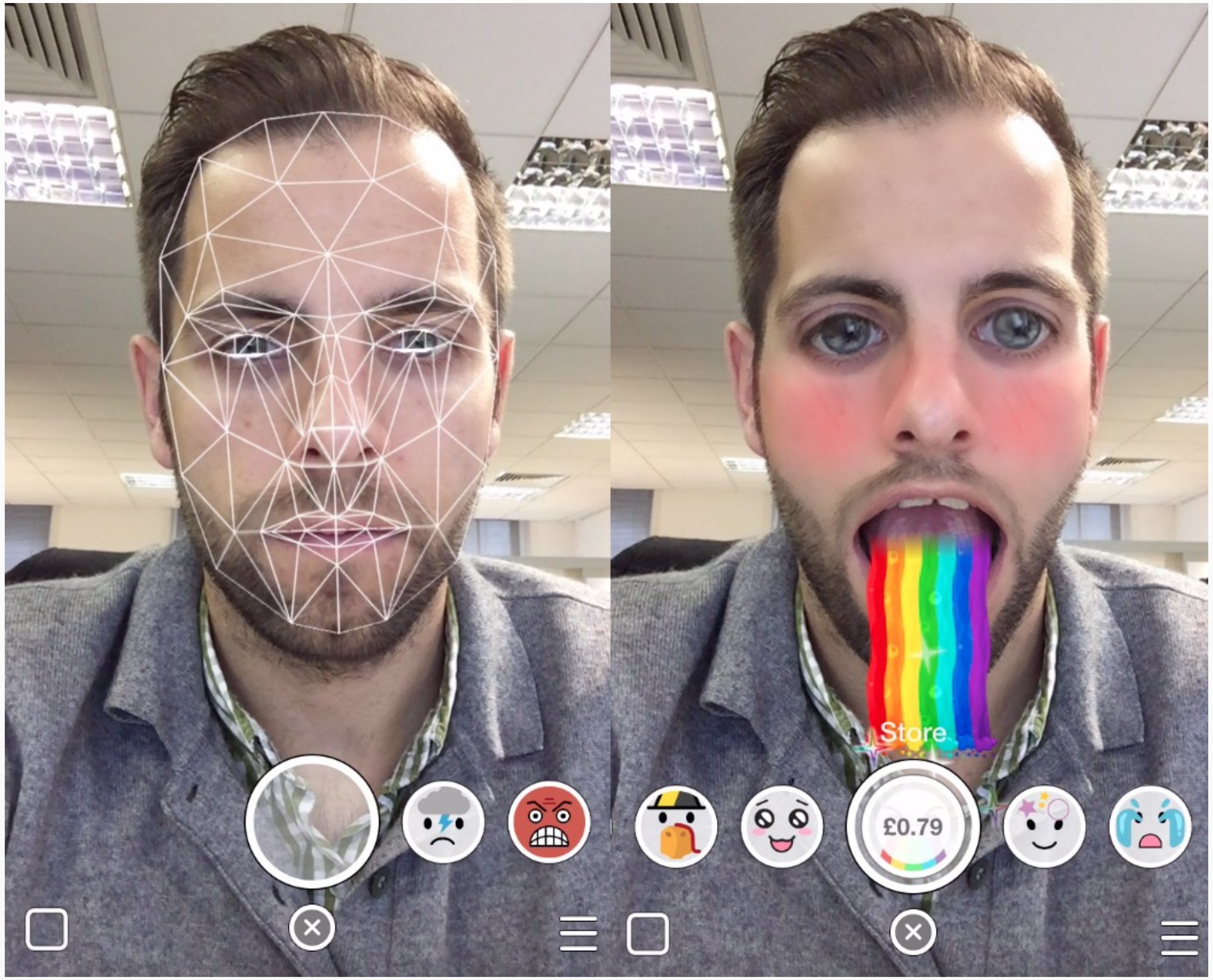
HelloSceneform

- Illustrate these concepts with a demo app, *hellosceneform*

SolarSystem

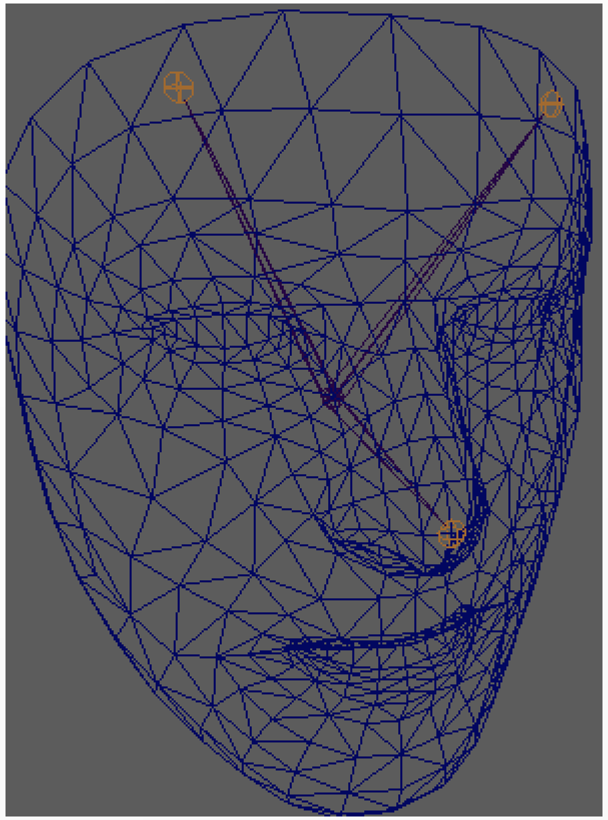
- Illustrate these concepts with a demo app, *hellosceneform*

Instagram / Snapshot AR



AugmentedFaces

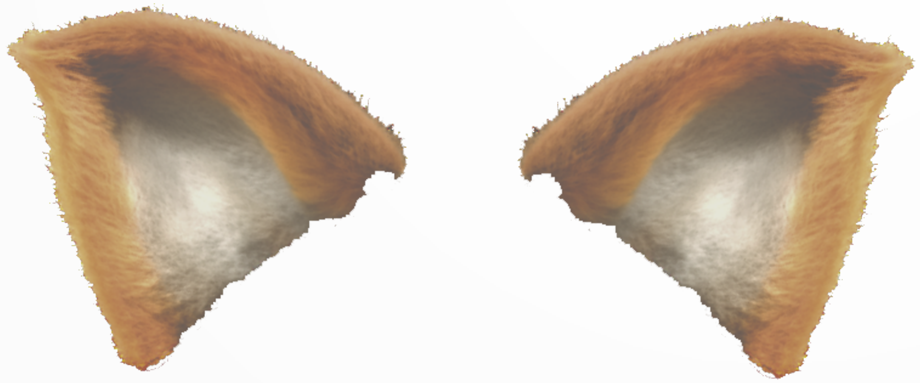
- ARCore has built-in detection for faces
  - Detects regions of the face
    - Left forehead
    - Right forehead
    - Nose
  - Front facing camera only



AugmentedFaces

- ARCore provides a canonical face mesh for building assets that will overlay onto faces
- The details are beyond me. Please see if you are interested in this sort of development:
  - https://developers.google.com/ar/develop/developer-guides/creating-assets-for-augmented-faces

AugmentedFaces



Each is a separate model to attach to a region of the face

Texture overlay completes the effect

Augmented Faces

```
for (AugmentedFace face : session.getAllTrackables(AugmentedFace.class)) {
   if (face.getTrackingState() == TrackingState.TRACKING) {
     // Render face mesh ...
   }
 }
```

AugmentedFace class is a Trackable provided by ARCore

ARCore handles the actual tracking

AugmentedFaces

- Illustrate these concepts with a demo app, *augementedfaces*

Developing these kind of Apps

- You can prototype with the builtin assets, but not very fun
- Naturally, modern platforms exist for sharing, viewing, and purchasing these assets

Sketchfab

- https://sketchfab.com/
- Platform for assets for 3D, VR, and AR
- Android App

Acknowledgments

- https://developers.google.com/ar/
- https://www.geeksforgeeks.org/lambda-expressions-java-8/
- http://archive.gamedev.net/archive/reference/programming/features/scenegraph/index.html
- https://medium.com/@anidaro/how-snapchats-filters-work-86973c3e2e9f