# Coach Buster – A digital fitness coach

Development of an augmented reality fitness app

Martin **Laechele**[1], Thomas **Kalbe**[2] and Zeynep **Tuncer**[3]

This paper describes the concepts and implementation of a native android fitness app in context of augmented reality. Google's frameworks ARCore and Sceneform provide access to 3D features inside the camera view through OpenGL without the need of actually writing plain OpenGL code. The app brings a digital fitness coach to everyone's home through augmented reality to show and explain different fitness exercises.

**Additional Keywords and Phrases:** Native Android App – Augmented Reality – ARCore –Sceneform – Digital Fitness Coach – Kotlin

## 1 INTRODUCTION

Fitness and the awareness of a healthy lifestyle with regular workouts are taking an important role in the life of more and more people. Often, jobs, family and other activities take up nearly the whole daily schedule and only a minimal amount of time is left for sports. Home workouts are one possibility to implement sports into the tight and stressful daily schedule. To bring up motivation and provide ideas for a workout session at home, there are plenty of apps, videos and online courses available on the internet. According to the Statista Digital Market Outlook, around 9. 7 million users in Germany used fitness apps in 2017. According to the forecast, the number of users of fitness apps in Germany could increase to around 18. 3 million users by 2024 [1]. Nevertheless, as a beginner, it can be challenging to identify the essentials and the correct form of an exercise. Correct form prevents injuries but words are not enough to explain the correct form. Videos are more clearly but often lack when it comes to multiple perspectives. A private coach that shows and explains everything can help, but is not a possible option for everyone. Coaches cannot be everywhere every time and often go hand in hand with high costs.

Augmented Reality (AR) provides the possibility to bring digital objects, even humans, into the own rooms. Using AR, everyone can have a coach that is able to show and explain various exercises everywhere. While filming the environment with a camera, e.g. the smartphone camera, the live video is extended by digital objects. Users can interact with those objects, what creates the illusion of an extended reality. Because nearly everyone uses a smartphone in daily life, hardware access is necessary for implementing AR, and performance is important for real-time processing, a native app is the best option to implement the digital fitness coach. A result of a survey in Germany on the frequency of use of sports and fitness apps shows that in 2018, 23 percent of respondents said they used them every day [2].

## 2 THEORETICAL BACKGROUND

Es Implementing AR apps requires working in 3D space. Modelling of 3D objects, manipulating objects in real-time and analysing the camera scene to detect its 3D space are some of the main features used to enable the AR illusion. The Open Graphics Library (OpenGL) [3] – a cross-platform 2D and 3D graphics API – specifies a set of commands to work in the 3D space.

[1]Student, martinlaechele@gmx.de, Wilhelm Büchner Hochschule Darmstadt

[2]Tutor Development of mobile Applications, thomas.kalbe@wb-fernstudium.de, Wilhelm Büchner Hochschule Darmstadt

[3]Professor and head of Media Computer Science and Human-Computer-Interaction, Zeynep.tuncer@wb-fernstudium.de, Wilhelm Büchner Hochschule Darmstadt

OpenGL is available on a wide variety of devices and can be used to display, move, scale and modify 3D graphics. On devices like smartphones or tablets and other embedded systems, OpenGL is available as a subset version called OpenGL ES. Google is developing multiple frameworks to push along the development of AR applications. Other developers can use those frameworks free and some of them are already open-source. Two of those frameworks are ARCore and Sceneform. Both frameworks build the foundation for the developed fitness AR app. ARCore provides APIs for essential AR features like analysing the camera view to detect the 3D space or handling motion controls and light estimations.
The framework is available for multiple development environments like Android and iOS and builds the foundation for developing AR apps. On Android, ARCore can be enabled in two ways – as optional or required. An app that includes only AR features needs to enable ARCore as required. If the app also includes non-AR features that are usable on devices without AR support, ARCore can be enabled as optional. The ARCore state has impact on the minimum required Android SDK version. Apps configured with ARCore as optional require at least SDK version 14. SDK version 24 is required as minimum if ARCore is configured as required [4]. Sceneform is open-source since version 1.16.0 and builds up on ARCore and another of Google's open-source frameworks called Filament. Filament is a physically based rendering engine that provides real-time rendering. This framework is cross-platform but also contains optimizations for running on Android devices. Sceneform provides an abstraction layer to OpenGL that allows developing AR apps without the need of learning the OpenGL commands. To run Sceneform at least OpenGL ES 3.0 needs to be available on the device [5].

## 3 CONCEPT

Due to the integration of the frameworks ARCore and Sceneform, the focus of development is on 3D modelling, model animation [8,9] and the implementation of a user interface. The user interface needs to provide a user experience that does not disturb the illusion of the extended reality, which is the main part of the app.
The app runs mainly in AR mode where the camera is active and allows the creation of an extended reality. The user interface consists of minimal overlay components and 3D graphics that are part of the AR scene. Additionally, the user can control the elements using gestures and can move around to view different perspectives of the objects. Two types of 3D objects are relevant in the app. The first one is a human model that represents the fitness coach and demonstrates the exercises to the user. The second type of objects are user interface components that consist of planes to display information in the AR scene.
As the human 3D model demonstrates various exercises to the user, there needs to be the possibility to move it like a real human body. Using a 3D creation tool to create the model allows attaching bones, animating and much more. Blender is a free 3D creation tool and is used to create and animate the human 3D model. Figure 1 shows screenshots of the first model version with attached bones in Blender.

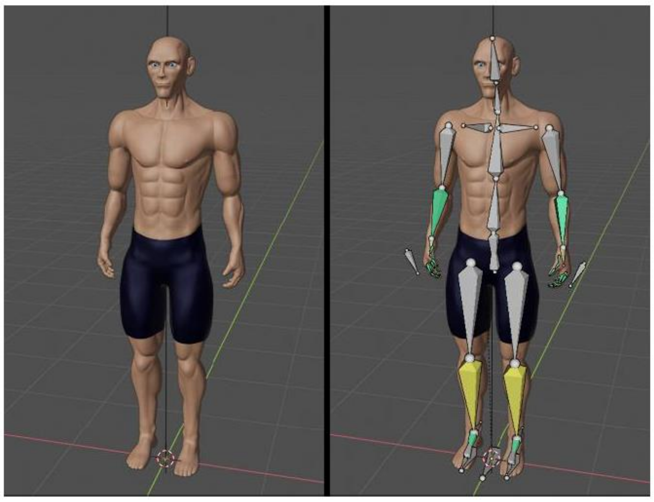
**Figure 1:** Screenshots from human 3D model in Blender that contains bones to create animations.

If a model has attached bones, it can be animated through bone movement. Movements of bone chains are calculated through forward or inverse kinematics. With forward kinematics, every bone has to be moved separately. With inverse kinematics, all parent bones in the chain move when a child bone moves, e.g. the hand bone can move the complete arm [6]. The process of attaching bones and implementing bone dependencies for kinematics is called rigging. Blender provides the possibility to export the model into various formats with included features like animations or light behaviour. The exported human 3D model contains animations for the different exercises and other animations for transitions between animations.
The app uses overlay icons as hints to teach the user the control mechanisms. Figure 2 shows a mock-up of the initial screen after starting the app. On the first start of the app, the users can choose to run a walkthrough for all functionalities. The walkthrough explains how the app works step by step.

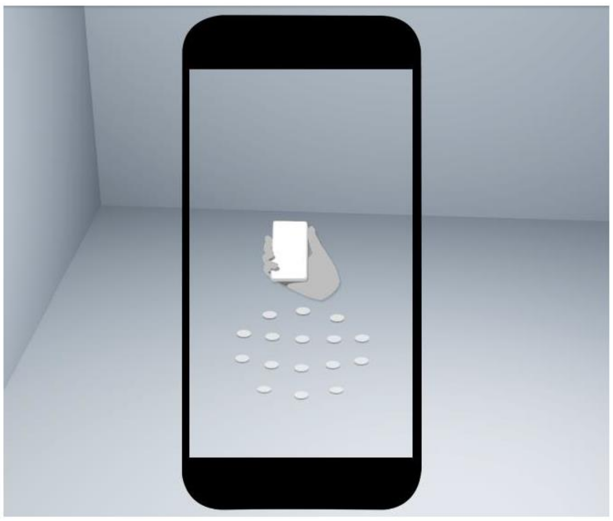
**Figure 2:** Mockup of the initial screen after starting the app.

For the detection of surfaces in the camera scene, the user needs to move the phone to scan the environment. An icon of a hand holding a smartphone appears on the screen and slightly moves around in order to indicate that the user needs to move his phone. This helps the app to identify surfaces in the camera scene. Additionally, a dotted area appears when the app identifies surfaces that allow the placement of 3D objects. A tap on the dotted area places the human 3D model as shown in Figure 3 (Right). The user can interact with the model using basic motions for moving, scaling and rotating.
Overlay components provide the possibility to select and control exercise animations. Figure 4 shows the overlay menu that contains a list of exercises. A floating action button on the bottom of the screen allows showing and hiding the overlay menu.

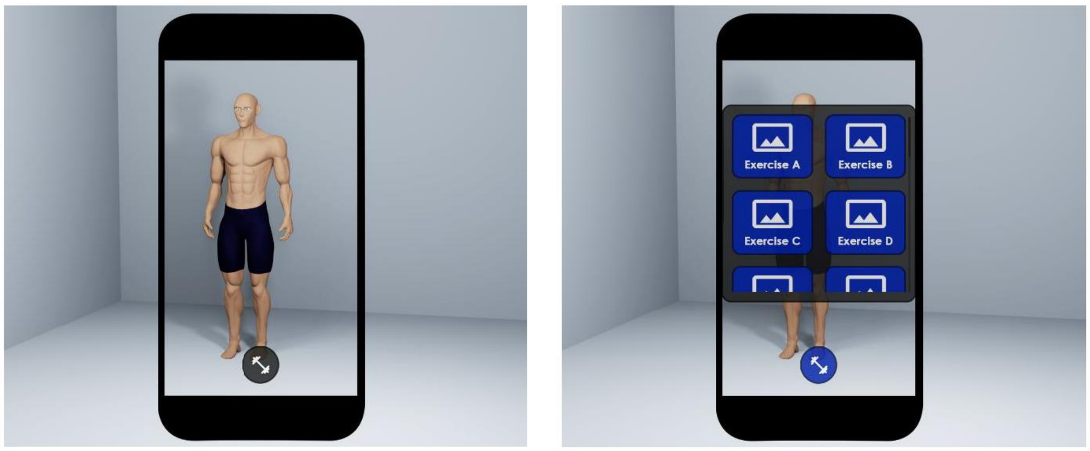

**Figure 3: Right:** Mockup of the screen after the human 3D model is placed on a detected surface.
**Left:** Mockup of the screen with expanded overlay menu for exercise selection.

The menu contains a scrollable list of all available exercises. After selecting an exercise from the overlay menu, the human model moves into the start position of that exercise and information regarding the selected exercise appear on the floor. The user can change the exercise using the menu or start the animation using a play button. Figure 3 (Left) shows this behaviour.

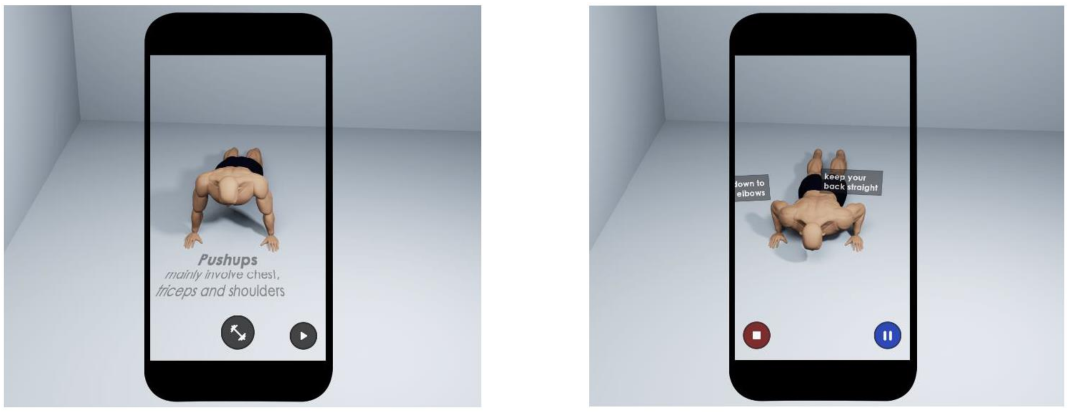

**Figure 5: Right:** Mockup of the screen after selecting an exercise from the menu.
**Left:** Mockup of the screen with selected and running animation.

If the animation of an exercise is running, the user can control the animation using known media keys like play, pause, stop, and repeat. Additionally, while an exercise animation is playing, information panels provide relevant information for a proper form of the exercise like shown in Figure 6. The information panels are part of the AR scene and rotate depending on the user position to provide readability from all perspectives.

## 4 REALIZATION

Kotlin is used as programming language to develop the app. Because the app runs mainly in AR mode and all non-AR features are some overlay user interface components, minimum requirements to run the app are Android SDK 24 and OpenGL ES 3.0. SDK version 24 comes with Android 7.0 (Nougat). Based on Google's statistic of Android OS usages, nearly 75% of all Android users are using at least Android Nougat and 85% of all android devices support OpenGL ES 3.0 or higher [7]. Sceneform is open-source since version 1.16.0 which is also the latest version while developing the app.
Because Sceneform 1.16.0 uses the no longer maintained Android support library for backwards compatibility, the Sceneform project is manually migrated to use the AndroidX library instead. The AR app integrates the modified Sceneform project using Gradle. All other AR related frameworks are already included in the Sceneform project, so no further dependencies are necessary to provide the AR functionality. Sceneform contains an AR fragment that provides all necessary functionalities to create the AR scene. The app includes the AR fragment in its main layout using the XML definition from Listing 1.

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

```kotlin
class MainActivity : AppCompatActivity() {

  private lateinit var arFragment: ArFragment

  override fun onCreate(
      savedInstanceState: Bundle?) {

    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    arFragment = supportFragmentManager
        .findFragmentById(
            R.id.sceneform_fragment
        ) as ArFragment

    arFragment.setOnTapArPlaneListener {...}

    arFragment.arSceneView.scene
        .addOnUpdateListener {...}
  }

  //...
}
```

**Listing 1:** Sceneform AR fragment XML definition.

**Listing 2:** Sceneform AR fragment initialization in the main activity class.

The onCreate-method of the main activity class allows initializing the ArFragment object and adding listeners to the fragment. Listing 2 shows the initialization and methods to add listeners for on-tap and on-update events. The on-tap event is triggered when the user taps on the dotted area that highlights detected surfaces in the AR scene. This allows placing the human 3D model on the tapped surface. Methods passed to the on-update listener are triggered on every frame update of the AR scene. This allows running animations and executing operations that require continuous updates. This basic setup provides all necessary functionalities to activate the smartphone camera, scan the environment, and detect surfaces to place 3D models. To add the human 3D model to the AR scene and animate it, it is necessary to load the model first.

## 4.1 ODEL INTEGRATION

Blender provides the possibility to export 3D models in various formats. Sceneform supports the formats OBJ, glTF / GLB and FBX. Until Sceneform version 1.15.0, an additional plugin is provided that allows to convert various file formats to SFA and SFB files. SFA files contain human-readable data of the 3D model, whereas SFB files are binary representations that are used to load models on runtime. Since version 1.16.0, it is recommended to load models using the glTF / GLB formats. Loaded models – called Renderables in context of Sceneform – can be asynchronously loaded using the method in Listing 3.

```
//...

private lateinit var weakActivityRef
    : WeakReference<MainActivity>
private lateinit var modelRenderable
    : ModelRenderable

override fun onCreate(
    savedInstanceState: Bundle?) {

  //...

  weakActivityRef
      = WeakReference(this@MainActivity)
```

```
  //...

  ModelRenderable.builder()
      .setSource(this, R.raw.model)
      .setIsFilamentGltf(true)
      .build()
      .thenApply { renderable ->
        weakActivityRef.get().let {
          it?.modelRenderable = renderable
        }
      }
      .exceptionally { throwable ->
        // Handle exception
      }
}
```

```
//...

private lateinit var characterNode
    : TransformableNode

private fun onPlaneTapped(hitResult: HitResult)
{
  val anchor = hitResult.createAnchor()
  val anchorNode = AnchorNode(anchor)
  anchorNode.setParent(
      arFragment.arSceneView.scene
  )

  characterNode = TransformableNode(
      arFragment.transformationSystem
  )
  characterNode.renderable = modelRenderable
  characterNode.setParent(anchorNode)
}
```

**Listing 3: Left:** Method for asynchronous loading of a model using the GLB format.
**Right**: Method to add a loaded model to the AR scene at a tapped position.

So-called Nodes are used to attach Renderables and add them to the AR scene. One variant is the TransformableNode that already contains handlers for translating, scaling and rotating a model. As shown in Listing 2 the AR fragment provides an on-tap listener that allows placing models. In addition to the TransformableNode with attached Renderable, an Anchor is necessary to place the model on the tapped position in the AR scene. The tap position is provided as HitResult to the listener and allows to create an Anchor at this position. The Anchor serves as the parent of the TransformableNode. Listing 3 shows the implementation of the process of adding a model to the AR scene at the tapped position.

Wrapping the Renderable into a TransformableNode allows the user to use motions to move, rotate and scale the added model.  Besides 3D models, whole views defined through XML layout files, can be added to the AR scene, too. Sceneform provides the functionality to load the layouts as Renderables and attach them to a plane in the AR scene. Using this functionality, user interface components can be integrated into the scene to optimize the AR illusion.

## 4.2 MODEL ANIMATION

Two ways of creating and running animations are available. One option is the use of basic operations for translating, rotating and scaling objects using a Node to create and run animations programmatically. Another option is to create animations in Blender, export them with the model and just load and run them programmatically. For basic movements of objects, like rotating the information panels based on user position, the first option is enough. To create animations for complex movements of the human body, the second option is preferred. Besides the creation of complex models, animations are one big part of 3D creation tools like Blender. Animations created in Blender can be baked into the model when exporting it and can be accessed using Sceneform after loading the model. Using the second option can reduce the development time of animations a lot. An animation created using Blender consists of keyframes on a timeline and movements stored as bone matrices. Every keyframe stores matrices that represent positions of selected bones at a specific timing. Using Sceneform those animation information are accessible programmatically. Listing 5 shows the code for accessing animation data and running an animation for unlimited time.

```
// ...

private var animator: Animator? = null
private var animationIndex = -1
private var animationStartTime = 0L
private var animationDuration = 0.0F
private var oneSecondInNanosAsDouble
    = TimeUnit.SECONDS.toNanos(1).toDouble()

private fun initAnimator() {
  animator = characterNode
      .renderableInstance!!
      .filamentAsset!!
      .animator

  if (animator!!.animationCount > 0) {
    animationIndex = 0    // first animation
    animationDuration
        = animator!!.getAnimationDuration(0)
    animationStartTime = System.nanoTime()
  }
}

private fun onFrameUpdate () {
  if (animationIndex >= 0) {
    val currentTime = System.nanoTime()
    val time = (
        (currentTime - animationStartTime) /
        oneSecondInNanosAsDouble
      ).toFloat()

    animator?.applyAnimation(
        animationIndex,
        time % animationDuration
    )
  }
}
```

**Listing 5:** Methods to access animation data and running an animation for unlimited time.

The onFrameUpdate-method can be used with the on-update listener of the AR scene to update bone matrices continuously which results in an animated model. **Listing 5** just showed how to animate a model that includes animations. This logic is part of a separate class to provide functionalities for handling all animations of a model with basic start, stop, and resume functionalities.

## 5 CONCLUSION

The combination of ARCore and Sceneform provide a functionality stack that contains all AR relevant functionalities. Processes like analysing the camera view, detecting surfaces and estimate light are handled by

ARCore. Sceneform provides an additional layer on top that removes the need of learning the OpenGL language. 3D models can be loaded, rendered and controlled by using Java or Kotlin only. It is possible to clearly separate the part of modelling and animating 3D graphics from the part of implementing the business logic of the app. While developing, the focus can be set on the own business logic, user interface and 3D models. Besides that, all frameworks are free to use and Sceneform and its rendering engine Filament are open-source and available through Github. When it comes to Sceneform, right now only version 1.15.0 has additional documentation, even though 1.16.0 is the latest version.
Version 1.16.0 contains some breaking changes and some features that worked in 1.15.0 are no longer working, e.g. the usage of Android views as Renderables. However, other features, like loading and rendering 3D models, are optimized and provide better results in 1.16.0. When using Sceneform 1.15.0 and its 3D model conversion to SFA and SFB formats, unexpected behaviours occur in case of models with baked in animation. The model does not look like it looks when modelled in Blender. Surfaces of the model are deformed and animations sometimes are broken while in Blender everything looks fine. With version 1.16.0, the SFA / SFB method is deprecated and the usage of the glTF / GLB format is recommended. Using the GLB format with Sceneform 1.16.0 fixes the issues of the previous version. Although the usage of Android views as Renderables is not working in version 1.16.0, the advantages in loading and rendering of own 3D models prevail. Additionally, Google developers [7] are working actively on the development of Filament and Sceneform. Following the Github activities, it seems that the next release will contain many improvements and cannot be far away.

## 6 FUTURE RESEARCH

This project is still in development. The current state provides the basic functionalities and user interface components to add the human 3D model to the AR scene and control some exercise animations. Some features that are mentioned in chapter "CONCEPT" need to be implemented. The user interface elements that are part of the AR scene, like information panels and text on the floor, are one of those features. One option to implement them are Renderables based on views. Because this does not work in Sceneform 1.16.0, an alternative is to use custom 3D objects with attached textures. This requires creating a texture for each panel and does not provide the flexibility to change texts programmatically based on Strings. Depending on the release date of the next Sceneform release, waiting can be another option. Views as Renderables is one feature that will be available again in the next release. As mentioned, a walkthrough on the first start of the app is planned but not implemented yet. The user will have the possibility to walk through all functionalities of the app. Texts and hints will explain everything the user needs to know to use the app. As the user interface is designed to use known mechanisms to control the components, many users will understand the basics without a walkthrough. A walkthrough is a useful option anyway, especially because AR apps are not in daily use yet.

In addition to the implementation of already planned features, the user interface and control mechanisms can be adapted to remove as much overlay components as possible. On the one hand, if everything is part of the AR scene, the illusion is more consistent. On the other hand, including all components to the AR scene may lead to a more difficult user experience. Feedback of users testing the app is required to identify the optimal user interface. Providing the app to a first group of users will provide more information for possible or necessary adaptions and optimizations.

**REFERENCES**

**[1]** Statista. 2021. URL: https://de.statista.com/statistik/daten/studie/1046996/umfrage/marktentwicklung-von-wearables-und-fitness-apps-in-deutschland/#:~:text=Nutzerentwicklung%20bei%20Wearables%20und%20Fitness%2DApps%20bis%202024,-Ver%C3%B6ffentlicht%20von%20J&text=Laut%20dem%20Statista%20Digital%20Market,18%2C3%20Millionen%20Nutzer%20ansteigen

**[2]** Statista. 2021. URL: https://de.statista.com/statistik/daten/studie/597394/umfrage/nutzung-von-sport-und-fitness-apps-in-deutschland-haeufigkeit/

**[3]** The Khronos Group Inc. OpenGL Overview. 2020. URL: https://www.khronos.org/opengl/, (visited on 20/04/2020)

**[4]** Google Developers. ARCore Overview. 2020. URL: https://developers.google.com/ar/discover, (visited on 05/04/2020)

**[5]** Google Developers. Sceneform Overview. 2020, URL: https://developers.google.com/sceneform/develop, (visited on 05/04/2020)

**[6]** Z. Bhatti, A. Shah, F. Shahidi and M. Karbas. Forward and Inverse Kinematics Seamless Matching Using Jacobian. Sindh Univ. Res. Jour. (Sci. Ser.) Vol. 45, pp. 387 – 392, 2013.

**[7]** Google Developers. Distribution dashboard. 13/04/2020, URL: https://developer.android.com/about/dashboards, (visited on 20/04/2020)

**[8]** A. Chang (Mediafreaks PTE LTD). The Process of 3D Animation. 2019. URL: https://www.media-freaks.com/the-process-of-3d-animation/, (visited on 03/05/2020)

**[9]** J. Petty. Blender Animation Tutorials That'll Take You From Newbie To Expert. 2020. URL: https://conceptartempire.com/blender-animation-tutorials/, (visited on 05/04/2020)