Visualization of furniture with MindAR

Authors: Yichao Liu, Sanchuan Ma, Ramon Butzer
Lecturer: Dr.-Ing. Sven Wursthorn

1 The Idea

The central concept of our project is to use an Augmented Reality (AR) API to visualise different furniture arrangements in a scaled-down room with a smartphone or similar handheld device. We use the *MindAR* toolkit, which is an open source web augmented reality library similar to the well-known *JSARToolkit*. MindAR supports not only image tracking but also face tracking and integrates AFRAME and three.js as a 3D framework for visualising spatial objects within the virtual environment (Yuen 2024a). We use AFRAME, because it is tailored for AR applications and therefore utilizes easier to use syntax, while three.js is the more general 3D graphics framework.

2 Some theory about the display system

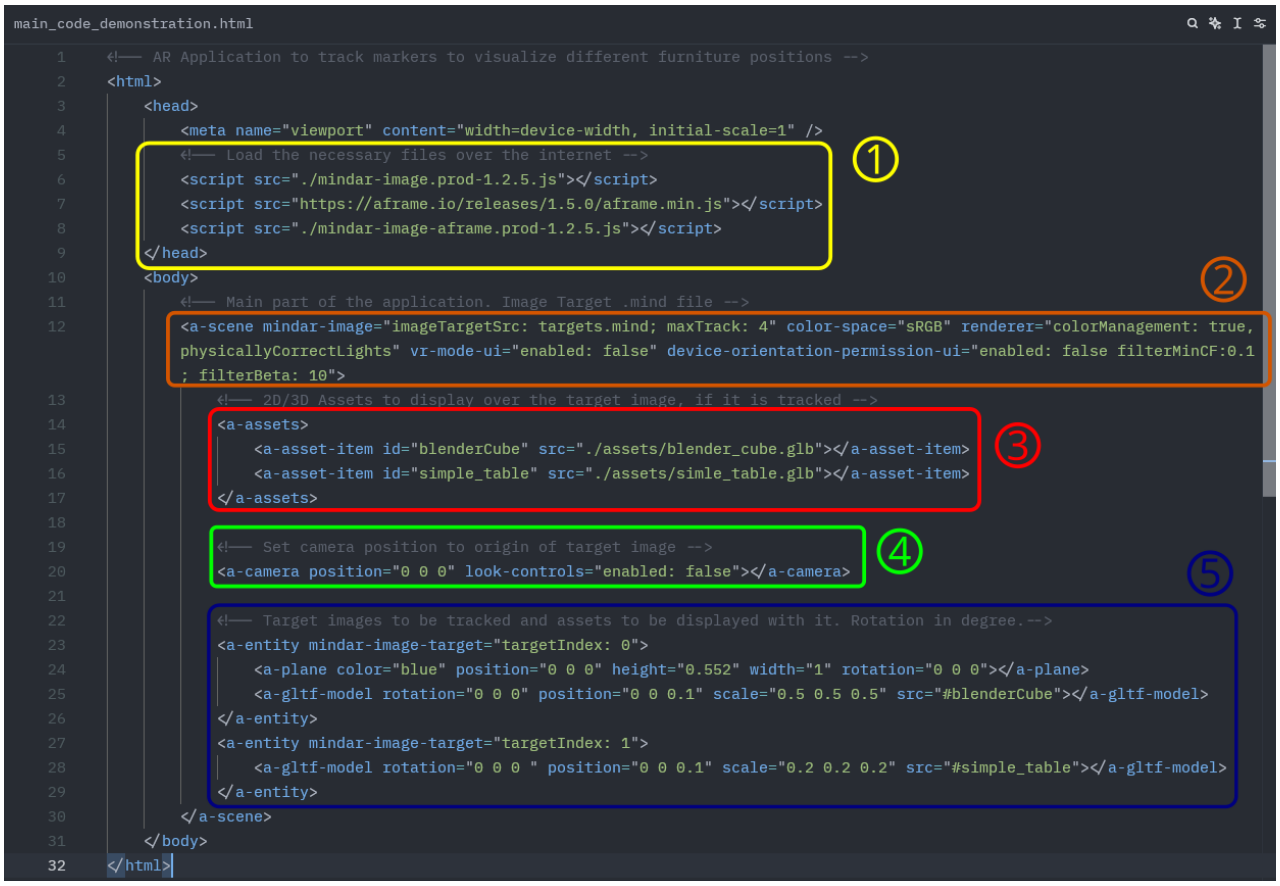
We use a smartphone or similar handheld device as AR platform due to its availability. These devices, together with head-mounted displays, are classified as video-based augmentation display systems (Wursthorn 2025, p. 37). The general disadvantage of video-based augmentation is the need to look at a display that shows the scene, rather than the actual scene itself, as with head-mounted see-through displays or projector-based systems. This additional layer between reality and virtualisation makes the use of the AR system less natural. The outdoor capability of smartphone-based systems is generally acceptable, but not really necessary for the use case of this project, as the presentation of the furniture arrangement is usually done indoors. An important fact about smartphone-based display systems is the type of tracking technology, which is *inside-out*, because of the moving sensor (smartphone) and the static object to be tracked (Wursthorn 2025, p. 95).

3 The Methodology

To achieve the goal of this project, which is a marker-based visualization of 3D furniture models on the smartphone, the mentioned software MindAR is used. In this section, the development process of our application is described in detail. In general, we followed the tutorials in Yuen (2024a) and extended the available code where it was necessary for our application.

3.1 Overview

The application basically runs inside a `html`-file implementing both the `mindar` and `aframe` APIs. Such an example is shown in Figure 1. Firstly, inside the file `<head>` the MindAR and AFRAME dependencies are declared and loaded either from the internet or from local storage (1). In the example of Figure 1, the `mindar-image.prod-1.2.5.js` and `mindar-image-aframe.prod-1.2.5.js` are loaded locally, which is indicated by the syntax `./` meaning the file path to the current directory. The file `aframe.min.js` is loaded via `https`-protocol from the internet. Inside the `<body>` of the `html`-file, a certain `.mind` file containing the information about the used markers is loaded (2). Afterwards the 3D assets are imported as `.glb` or `.gltf` files (3). Here, only `.glb` files are used. A camera object is defined, so the device camera can be accessed for image acquisition (4). Lastly the imported 3D assets are assigned to their marker counterpart while also defining position and orientation of the model relative to the marker origin (5).



```html
<!-- AR Application to track markers to visualize different furniture positions -->
<html>
    <head>
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <!-- Load the necessary files over the internet -->
        <script src="./mindar-image.prod-1.2.5.js"></script>
        <script src="https://aframe.io/releases/1.5.0/aframe.min.js"></script>
        <script src="./mindar-image-aframe.prod-1.2.5.js"></script>
    </head>
    <body>
        <!-- Main part of the application. Image Target .mind file -->
        <a-scene mindar-image="imageTargetSrc: targets.mind; maxTrack: 4" color-space="sRGB" renderer="colorManagement: true,
physicallyCorrectLights" vr-mode-ui="enabled: false" device-orientation-permission-ui="enabled: false filterMinCF:0.1
; filterBeta: 10">
            <!-- 2D/3D Assets to display over the target image, if it is tracked -->
            <a-assets>
                <a-asset-item id="blenderCube" src="./assets/blender_cube.glb"></a-asset-item>
                <a-asset-item id="simple_table" src="./assets/simle_table.glb"></a-asset-item>
            </a-assets>

            <!-- Set camera position to origin of target image -->
            <a-camera position="0 0 0" look-controls="enabled: false"></a-camera>

            <!-- Target images to be tracked and assets to be displayed with it. Rotation in degree.-->
            <a-entity mindar-image-target="targetIndex: 0">
                <a-plane color="blue" position="0 0 0" height="0.552" width="1" rotation="0 0 0"></a-plane>
                <a-gltf-model rotation="0 0 0" position="0 0 0.1" scale="0.5 0.5 0.5" src="#blenderCube"></a-gltf-model>
            </a-entity>
            <a-entity mindar-image-target="targetIndex: 1">
                <a-gltf-model rotation="0 0 0 " position="0 0 0.1" scale="0.2 0.2 0.2" src="#simple_table"></a-gltf-model>
            </a-entity>
        </a-scene>
    </body>
</html>
```

Figure 1: Complete example code of a `html`-file using the mindAR API to visualize two 3D models on corresponding markers, which are defined in the file `targets.mind`.

3.2 Marker creation

MinAR supports both image tracking and face tracking. For image tracking, a binary `.mind` file compiled from all the used reference images has to be loaded in the `html`-file, as was stated in the previous chapter. The author of MindAR provides an online compiler for creating this `.mind` file (Yuen 2024c). The interface of this compiler can be seen in Figure 2. After importing the desired image files via drag-and-drop and starting the process, the compiled file can be conveniently saved with a click of a button.

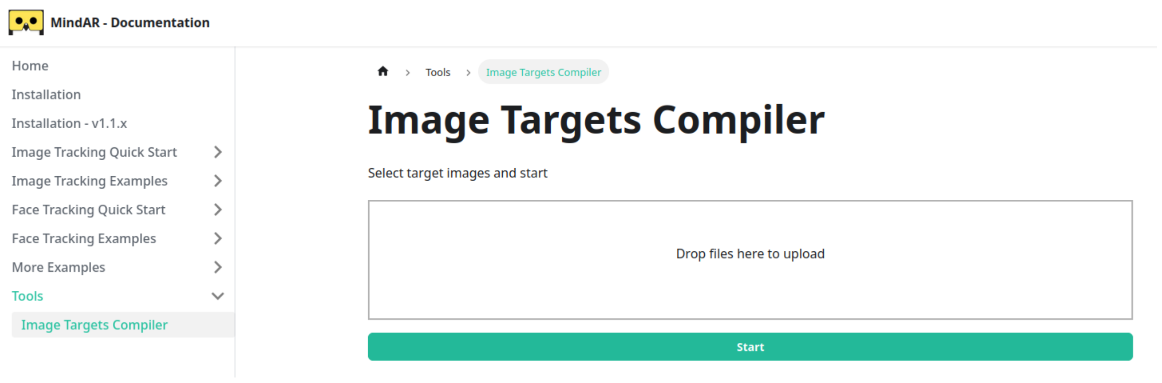
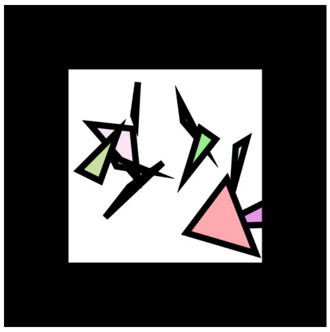


Figure 2: Interface of the online marker compiler.

In general, arbitrary image files can be used as reference images. We chose to create a number of markers with a marker generation software created by Gasques, Scheurer, and Lopes (2021). Some of the markers are shown in Figure 3. It has to be noted, that these images are not optimal representations for markers, because of the big white space and black border, which is identical for all markers. The number of significant features, which can be resolved by the camera and recognised by the algorithm, could be improved upon by choosing more distinct features and better colour discrepancy. In practise, the used markers still showed acceptable results, which is why we did not invest more time in creating more optimal markers.

The marker `.mind` file is referenced in the block `<a-scene mindar-image="imageTargetSrc: targets.mind ... >`, after which all markers included in the `.mind` file can be queried (cf. Figure 1).


(a) Marker image 1.

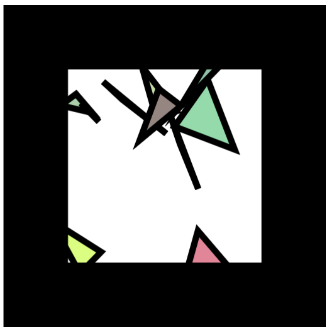
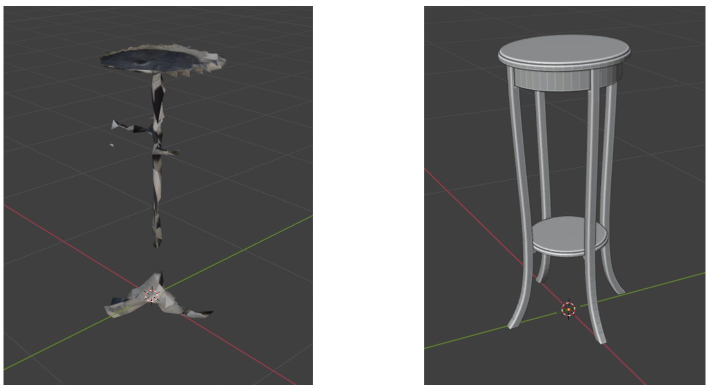

(b) Marker image 2.

Figure 3: Two of the used marker images.

3.3 3D Assets

The main goal of the project is to visualise furniture on a small scale, like a Din A4 piece of paper, in order to find a suitable arrangement in a room. For this purpose, we use two types of 3D assets. First, we use a smartphone app to perform a photogrammetric scan of the desired object and create a textured model of the object. This is done to achieve the most realistic match between the model and the real object. Secondly, we use freely available, sculpted 3D assets from the internet. This is done for performance reasons, as the scanned models have a relatively large file size of several tens of megabytes, whereas the size of the untextured, sculpted models are only a few kilobytes in size. The two types of models are visualised in Figure 4. As can be seen, the scanned model has many artefacts and is only a moderately good representation



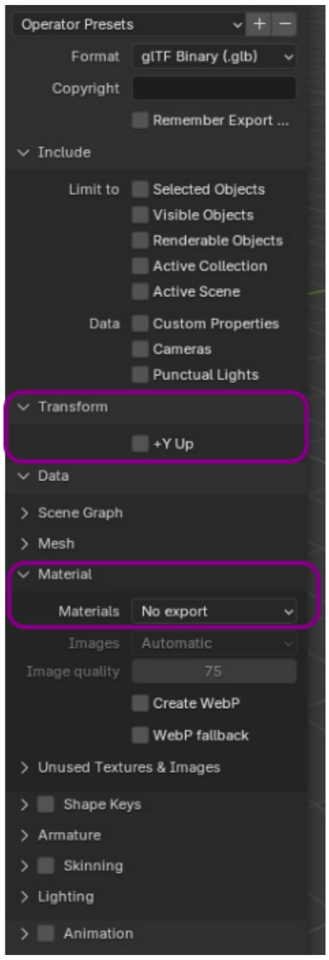
(a) Scanned table from camera images including texture.

(b) Sculpted 3D model of a table without texture.

Figure 4: Examples for the two types of used 3D assets.

of the reality. Therefore, we focus our work on the sculpted models, which due to their low file size also load faster in the finished AR application.

We use *Blender* for minor modifications and file size conversion of the 3D assets. The two main processing options performed with Blender are setting a reasonable transformation for the orientation of the exported model and, in the case of the sculpted models, turning off *materials* such as texture and normal maps. The former option is useful, because the default option for the `.glb` export from Blender is to set the $+Y$-axis as the $+Z$-axis. While the orientation of the model can also be adjusted in our main `html`-file, we found it helpful to simply disable this unnecessary transformation option in the export. The latter option is used to massively reduce the file size of the asset, which helps to create a smoother performance in action. The export options can be seen in Figure 5.



Operator Presets

Format: glTF Binary (.glb)

Copyright:

Remember Export ...

Include

Limit to: Selected Objects, Visible Objects, Renderable Objects, Active Collection, Active Scene

Data: Custom Properties, Cameras, Punctual Lights

Transform

+Y Up

Data

Scene Graph

Mesh

Material

Materials: No export

Images: Automatic

Image quality: 75

Create WebP

WebP fallback

Unused Textures & Images

Shape Keys

Armature

Skinning

Lighting

Animation

Figure 5: Asset export options in Blender. Marked are the possible transformation of the y-axis and the materials.

The 3D assets are loaded into the `html`-file as AFRAME `<a-assets>` and given an `ID`, so they can be accessed by name later (cf. Figure 1).

3.4 Adding it all together and code adjustments

Looking again at Figure 1, we have discussed the main points of the application, which is the import of the necessary libraries in (1), the marker creation and import in (2), the asset definition in (3), the camera setup in (4) and the assignment of 3D assets per marker in (5).

Some further adjustments to the code *have* to be made in order to produce meaningful results. Other code adjustments *can* be made to try to improve the performance of the application. The first important point is the adjustment of the model orientation and scale. For the orientation of the sculpted models, we generally made few adjustments in Blender and exported without the $+Y/+Z$ axis swap option enabled. Because of this, another change in the orientation via the `html`-file was generally unnecessary. The scale of the different models was changed empirically to create true-to-life results in the AR application. These parameters are changed in the `a-entity mindar-image-target` block.

Moreover, we experimented with different values for the filter-parameters `filterMinCF` and `filterBeta`. These two parameters may help with decreased jittering and better render stability. After different considerations, we chose to remain with the default values of `filterMinCF`= 0.001 and `filterBeta`= 1000.

4 The Results

In this section some results of the developed application are presented. Examples for the visualisation of scanned and sculpted models are shown in Figure 6. In practise, the pose of the assets follows the corresponding marker. The size of the models has been adjusted to a visually reasonable level by setting the `scale` parameter of the respective `<a-entity mindar-image-target>`.

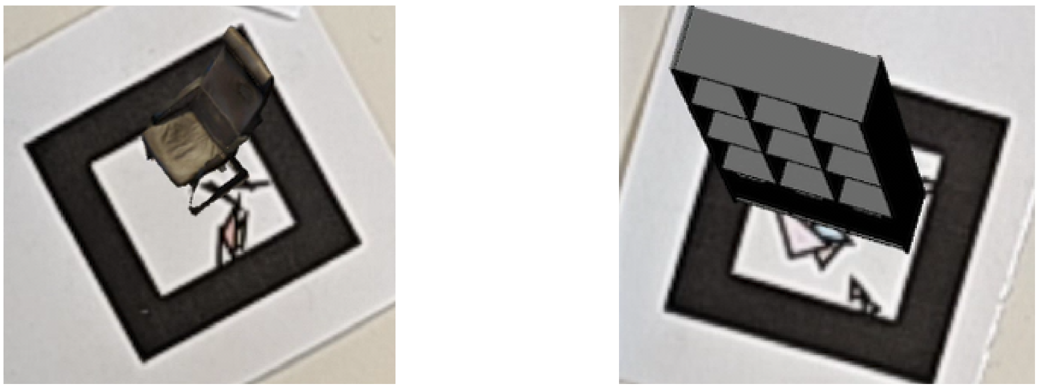

(a) Scanned chair asset. **(b)** Simple shelf asset.

Figure 6: Examples for the AR representation of the two types of used 3D assets.

The result of a complete room scene is shown in Figure 7. In addition to some furniture models we chose to display a wireframe model of a simple rectangular room with a door for better visual representation. Unlike the furniture models, the location of the wireframe room model is situated off-centre of the corresponding marker to minimize occlusion by the furniture markers. As can be seen in Figure 7, a proof of concept of our application has been reached. The 3D-assets are rendered in the AR-application and the markers can be moved around to visualize different furniture constellations.

Still, there are some problems with the application and many things, that could be improved upon. Some of these are discussed in the following.

- **Stability**: Some medium amount of Jittering still exists despite experimenting with the filter parameters described in subsection 3.4. Moreover, the big file size of the textured models derived from scans sometimes lead to crashes on the older of the two smartphones, that were used in testing. The application was most stable, when using the low file size models without texture.
- **General usability**: There exist two fundamental problems in using image markers for the presented task: Scale factor problems and occlusion. The former is responsible for the

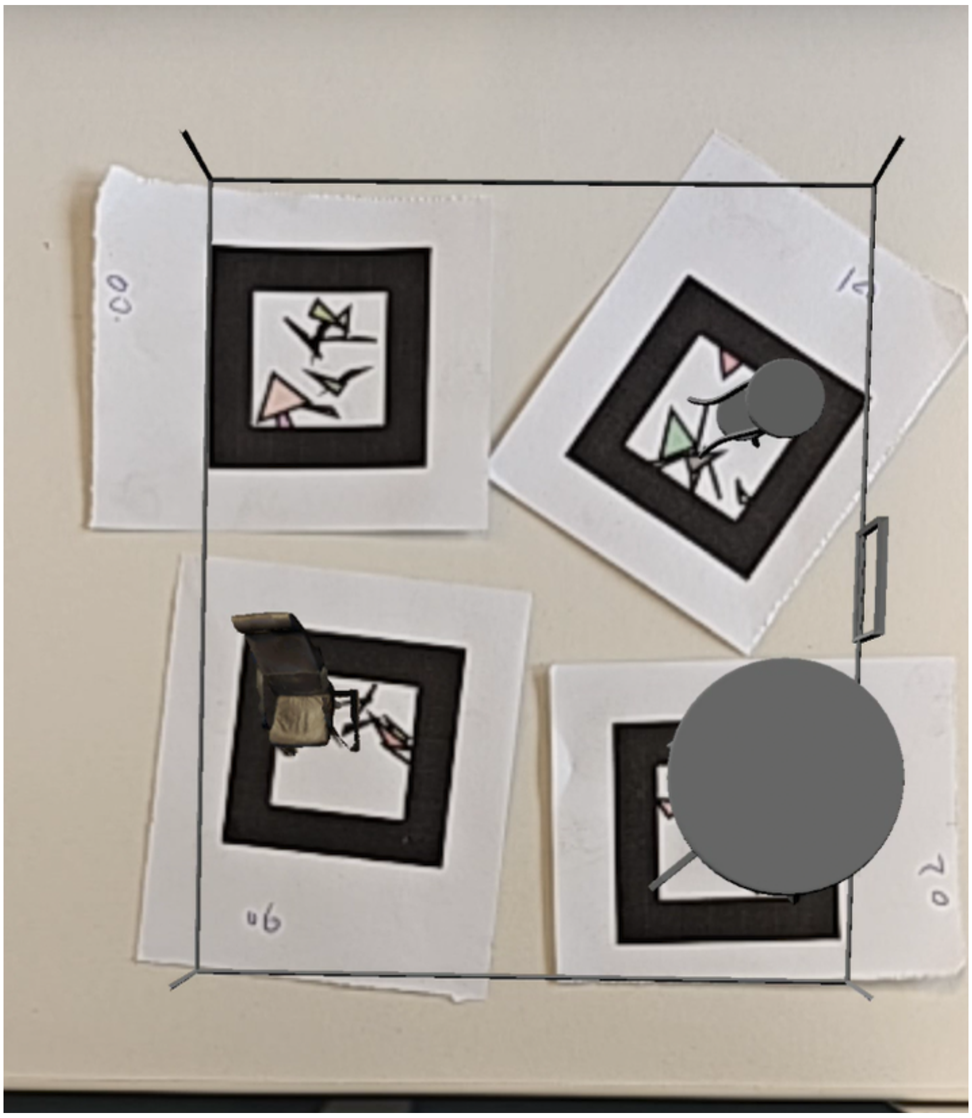


Figure 7: AR representation of the room scene. Top left marker is used for the wireframe model of the room.

decision to use a miniature room model. Instead, a more intuitive visualization for the furniture distribution in the room would be to project lifesize furniture models into the image where they would actually stand. Despite using different marker diameters (ca. 5 cm to ca. 20 cm), this approach was not feasible, because the used cameras weren't able to reliably recognize the markers. The latter is more of an inconvenience, but correlates with the scale factor problem. Because a certain marker size is needed for accurate image recognition, only a small amount of models can actually be displayed in the current form of the implementation of our project.

- **Markers**: Correlated with the earlier points, the chosen markers themselves are part of the problem. They were created in a quick-and-dirty way to minimize time expenditure. Improvements could be made by creating markers with bigger and more easily recognizable features distributed over the entire marker surface. This could lead to better recognition of smaller markers, which could mean the opportunity to show more furniture objects without occlusion.

5 Summary

In this report we use MindAR with AFRAME to create a small AR project for marker-based real-time furniture visualisation for the smartphone platform. By loading 3D assets and a binary file containing marker information, the application, running within a `html`-file, displays 3D models over the corresponding markers. The minimal use case, the correct visualisation of multiple objects, was achieved. Depending on the camera and smartphone used, the application runs relatively smoothly. Possible further improvements could be made by using more distinct markers that are easier for the camera to detect from a distance. Adjusting the scale of the model would in theory allow the application to be used on a 1:1 scale, but only if the markers can be accurately recognized. This is presumably problematic for marker based applications, because of limited camera resolution. Instead, the possibility to place 3D-models into the room via some kind of drag-and-drop feature could be pursued for even more realistic user-experience.

References

Gasques, Danilo, Alexander Scheurer, and Pedro Lopes (2021). *danilogr/AR-Marker-Generator v1.0*. Version v1.0. URL: `https://doi.org/10.5281/zenodo.4466705`.

Marcos, Diego, Don McCurdy, and Kevin Ngo (2024). *A-Frame – A web framework for building 3D/AR/VR experiences*. URL: `https://aframe.io/`. [Accessed 07-02-2025].

Wursthorn, Sven (2025). *Lecture Slides - Augmented Reality, Winter semester 24/25*. URL: `https://ilias.studium.kit.edu/ilias.php?baseClass=ilrepositorygui&cmd=view&ref_id=2488247`. [Accessed 11-02-2025].

Yuen, HiuKim (2024a). *MindAr Documentation*. URL: `https://hiukim.github.io/mind-ar-js-doc/quick-start/overview/`. [Accessed 07-02-2025].

– (2024b). *MindAR Github*. URL: `https://github.com/hiukim/mind-ar-js`. [Accessed 07-02-2025].

– (2024c). *MindAR Image Targets Compiler*. [Accessed 08-02-2025]. URL: `https://hiukim.github.io/mind-ar-js-doc/tools/compile`.