Techniques d'Interaction Avancées

Ferran Argelaguet // Jeanne Hecquard - M2 - 2023/2024

Contents

Introduction

This document introduces the basic concepts for an augmented reality (AR) application and develop simple applications to depict them. One of the main usages of AR is to enhance the real world with computer generated images. In order to achieve this integration, the AR application must gather information from the real environment (tracking) and integrate the computer-generated content (see Figure 1). In order to focus on the user interface instead of the tracking and rendering algorithms we will use a commercial AR toolkit supplied by Qualcomm (https://developer.vuforia.com/), the toolkit can be used with Unity 3D (http://unity3d.com). The SDK provides the basic building blocks for creating an AR application.

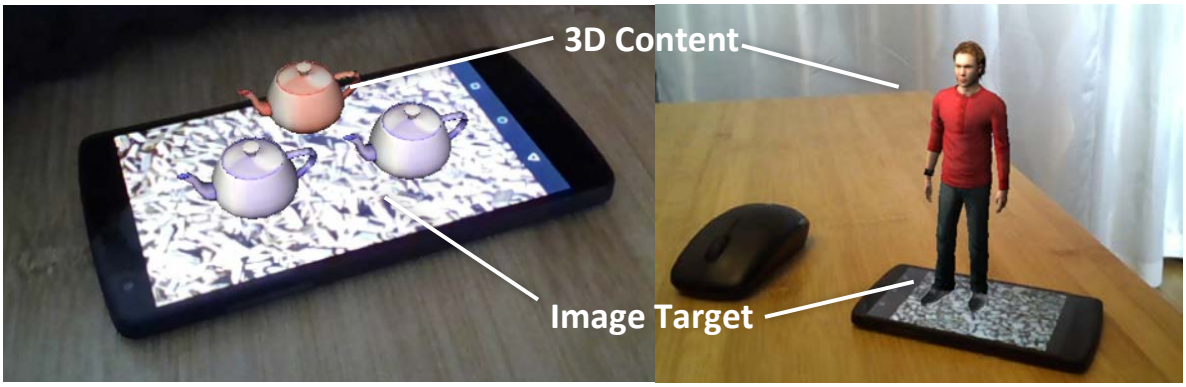


Figure 1 *Examples of image (target) tracking. Once the target is recognized its position and orientation can be computed. This information is used to place the 3D generated scene.*

We will focus on the initialization steps and how to efficiently track the real world (see Figure 1). Most AR applications use known images (known as markers or targets) to improve the tracking process, although more and more high-end systems are able to capture in real-time the layout of the real environment. Regarding image targets, these images can be recognized in the live video feed and its spatial position can be inferred. The computed positional information is then used to display the 3D content. Vuforia (which call these images ImageTargets) provides several tools to make this process as transparent as possible for the developer.

Figure 2 *Unity scene view of the 3D content displayed in Figure 1. Vuforia links the virtual scene with the recognized target and integrates the real and the virtual content.*

Vuforia SDK Main Components

In current Unity releases, the Vuforia SDK is downloaded from the Vuforia Developer Portal (https://developer.vuforia.com). You have to register in the developer portal and then downloaded the Unity Vuforia Engine Package (https://developer.vuforia.com/downloads/sdk). Once downloaded drag and drop the unity package to your asset folder. As we will see later, the Vuforia portal is also used to create and manage image targets and the developer license.

(Optional) In addition, a Vuforia Sample package can be downloaded through the Unity Asset Store (**Vuforia Core Samples**). To install the samples, go to the Unity Asset Store, search and install the aforementioned package.

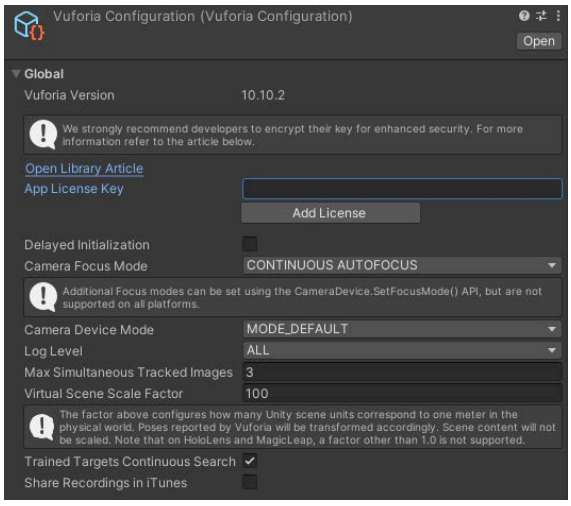
In the following sections, we will detail three of the main Vuforia components that we will use during the different exercises: `ARCamera`, `ImageTargets` and `VirtualButtons`.

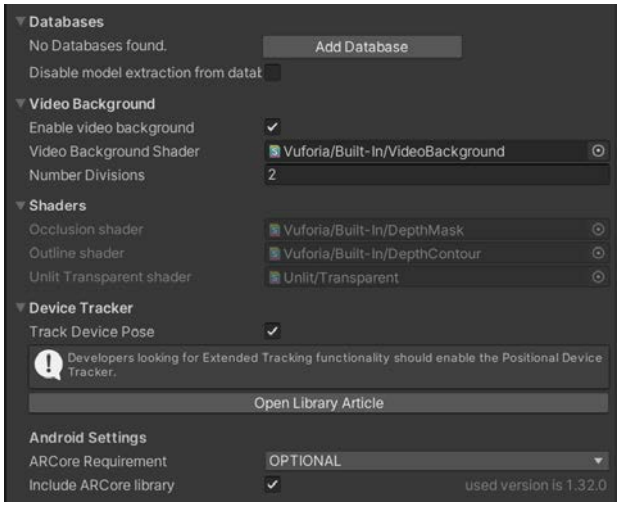
ARCamera

The `ARCamera` is the main component of any Vuforia application as it represents the virtual and the real camera and thus, it is responsible to track the environment and to render virtual objects.

> **Tip**: In order to add the ARCamera game object into our scene, add the ARCamera GameObject (GameObject->Vuforia Engine->AR Camera). This Unity will also import the Vuforia package.

The `ARCamera` game object includes a "Vuforia Behaviour" component, which provides a button to open the Vuforia configuration "*Open Vuforia Engine Configuration*". The "*Vuforia Configuration*" panel allows configuring a number of parameters that will determine the behavior of the image target detection. It handles the hardware (webcam) and the image targets that can be recognized.





The main parameters for the configuration dialog are:

- **App License Key** –Vuforia SDK requires the use of a custom license key. You can create your own App Key from the Develop section of the Vuforia website. Login with your Vuforia account and in the License Manager tab click on "Get Development Key". The "Add Licence" button will open the webpage.
- **Camera Device mode** – Determines the optimization mode (default, fast, and high-quality camera modes).
- **Max Simultaneous Tracked Images**– the number of image targets the tracker should try to track simultaneously. You might have to update this value during the different TPs.

- **Databases:** The list of the image target databases currently available. New databases can be added by pressing the **Add Database** button, which will open the Target Manager in the Vuforia Developer portal.
- **Video Background:** Determines whether the video feed is displayed or not.
- **Webcam:** Allows selecting the active webcam, the default parameter should work.

Tip: Ensure that only one unity "Camera" (the one in ARCamera) is enabled. Remove the Main Camera gameobject if still in the Hierarchy pannel

Tip: To improve the responsiveness of the application set the mode setting to MODE_OPTIMIZE_SPEED.

Exercise 1: Create a new Unity project, add the ARCamera GameObject, configure it and run the application from the Unity editor. Test that the video feed is correctly displayed in the Game window also, you can Test the behavior of the options in the Vuforia configuration dialog.

Image Targets

In order to merge virtual and real content, augmented reality applications have to determine where the camera is placed with respect to the real environment. This allows the system to render the virtual objects from the good perspective, providing a seamless integration. One solution used by Vuforia is to search for known visible and salient features in the camera feed. These features (sharp edges, corners) are called "*Natural Features*" as they are computed from image targets (see Figure 3). The detected features in the image create a unique fingerprint that can be efficiently tracked and recognized in real time. Once the image target is detected, the camera position can be computed.

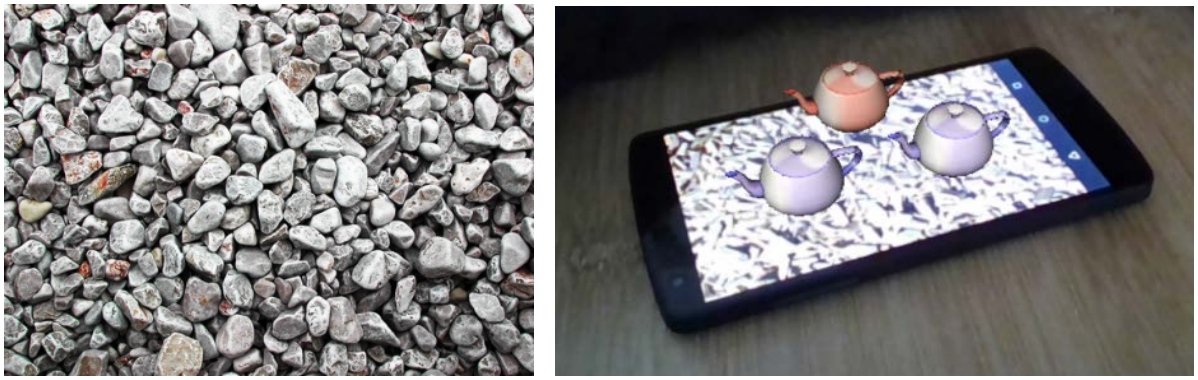


Figure 3. Once the image target (left) is detected in the camera feed, the application can correctly place the 3D content (right).

Image targets can be defined through the `ImageTarget` game object. The `ImageTarget` object provides several handlers which can be reimplemented in order to define the application behavior. In order to add an `ImageTarget` select the GameObject->Vuforia->Image. By default it will select an available Image from the image target database (see Figure 4). Once added to the scene, the default behavior of the `ImageTarget` can be adapted to the application needs.

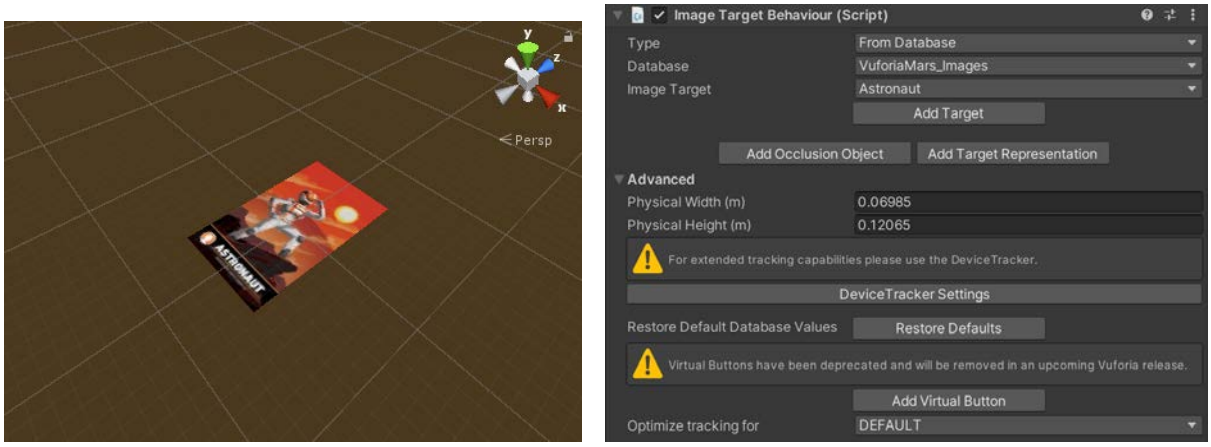


Figure 4. Image target setup in Unity. Left, default image target. Right, Image target properties panel.

The `ImageTarget` gameobject has three scripts: `ImageTargetBehaviour`, `ImageTargetPreview` and `DefaultTrackableEventHandler`.

First, the script `ImageTargetBehaviour` configures the properties and the used image. By default, the type of the target is "From Image", click on the combobox and select "From Database". Vuforia provides the `VuforiaMars_Images`, a set of 4 images which can be selected using the **Database** and **Image Target** attributes. The Vuforia platform allows the definition of customized images targets through a web-based interface. If you want, you can explore the Vuforia developer portal in order to create your own images, just create a new database in the tab.

Tip: You can create new Image target databases from the developer portal. The "Add Database" in the Vuforia Configuration dialog will redirect you to the Target Manager at the developer portal. The process is simple, you have just to create a new database and add the images you want to use. Once the images are validated you will be able to download a Unity package that has to be imported to the unity project. You can obtain additional information regarding how to choose a good image here:

https://library.vuforia.com/articles/Solution/Optimizing-Target-Detection-and-Tracking-Stability

Once the set is defined, you can choose the desired image target. In addition, the `ImageTargetBehaviour` script enables to:

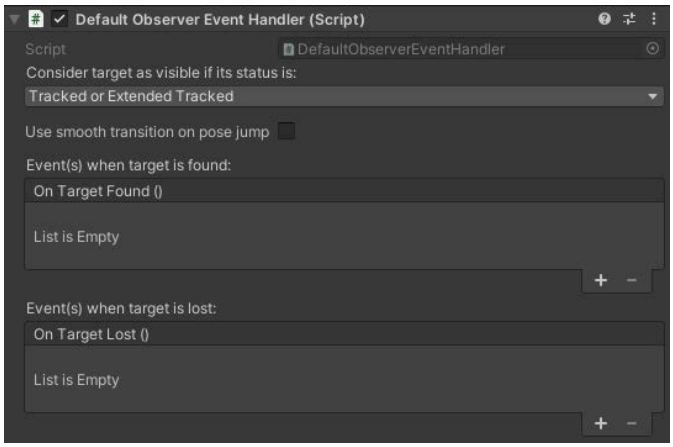
- Set the **width** and **height** of the image target. The image targets have a predefined width and height, which match the physical size of the real image.
- Provide a visual representation of the target either "Add Occlusion Object" or "Add Target Representation". The occlusion object will enable the virtual target to occlude virtual objects, while the target representation will display a 3D representation of the actual image target.
- The advanced features will be discussed later on.

Tip: The width and height of the image target should match the size of the real image target so the virtual content appears at the desired size. Do not change the scale of the image target gameobject!

The second script is `DefaultObserverEventHandler` which handles the behavior of the image target when it is visible (and when it disappears) from the webcam. By default, the script enables all the game objects in the hierarchy when the target is visible, and disables them when the target is not visible.

The third script `ImageTargetPreview` manages the 3D preview of the image target in the editor. If you launch the project, the preview will disappear. For debug purposes, you can enable the visualization of the target during play by pressing the button "Add target representation" in the `ImageTargetBehaviour` component.

The `DefaultObserverEventHandler` also provides two callbacks to manage the "Target Found" and "Target Lost" events. The first event is triggered when the target is first detected on the camera feed, while the second is triggered when the already visible target is no longer visible. The callbacks triggering can be modified using the "Consider target as visible if its status is:" drop down menu. In order to customize this behavior, two alternatives are possible.



First, the `DefaultObserverEventHandler` can be reimplemented and his default behaviors modified. Alternatively, the callbacks can be used to alter its default behavior.

The third script is `ImageTargetPreview`, which hides the representation of the `ImageTarget` (textured plane) at runtime. The textured plane is only displayed for reference purposes.

Exercise 2: Add and configure an Image target in your Unity project. Then, create several simple 3D shapes and add them to the image target (the shapes have to be child of the image target). Run the application and ensure that the 3D shapes are well displayed when the real image is in the real camera viewport. Stress the recognition system in order to determine its limits. What happens when you add the image target representation? What happens if some of the game objects are affected by the unity physics?

Exercise 3: Add another Image Target to Exercise 2 and explore the behavior of the application when more than one target is enabled. Explore and explain what the "World Center Mode" option in the `VuforiaBehaviour` component of the ARCamera game object. Does it impact the behavior you defined in the second exercise?

One key feature while using AR applications, is the ability to recognize the surroundings of the user. Vuforia can track image targets when not visible in the viewport when the "Extended Tracking" functionality is enabled. It can be enabled in the Vuforia Configuration by enabling or disabling the "Track Device Pose".

Exercise 4: Enable and disable the extended tracking functionality and describe the differences between the two modes.

Customizing the behavior of an ImageTarget

In order to change the default behavior, one alternative is to replace the `DefaultObserverEventHandler` script with a new script, which inherits from it.

The `DefaultObserverEventHandler` class provides several methods to monitor the status of the image target. In particular, it contains the `HandleTargetStatusChanged` method, which is called when a change in the tracking state occurs, for example, when the target is found or when it disappears. In addition, two other methods can be reimplemented: `OnTrackingFound` and `OnTrackingLost`. Check the `DefaultObserverEventHandler` for a reference implementation.

Exercise 5: Check the contents of the `DefaultObserverEventHandler` and briefly describe is behavior.

```
protected virtual void HandleTargetStatusChanged(Status previousStatus, Status newStatus)
    {
        var shouldBeRendererBefore = ShouldBeRendered(previousStatus);
        var shouldBeRendererNow = ShouldBeRendered(newStatus);
        if (shouldBeRendererBefore != shouldBeRendererNow)
        {
            if (shouldBeRendererNow)
            {
                OnTrackingFound();
            }
            else
            {
                OnTrackingLost();
            }
        }
        else
        {
            if (!mCallbackReceivedOnce && !shouldBeRendererNow)
            {
                OnTrackingLost();
            }
        }

        mCallbackReceivedOnce = true;
    }
```

Code Listing 1. Handler called each time the tracking state changes.

The simplest solution is to hook to the callbacks to a new script which defines the behavior of the image target when the visibility of the target changes. Alternatively, you can create a new script that inherits from `DefaultObserverEventHandler` and reimplement the required methods. If you reimplement the `DefaultObserverEventHandler` you will have to replace the old component by the new component for each of the targets in your scene. You can create different scripts to define different behaviors.

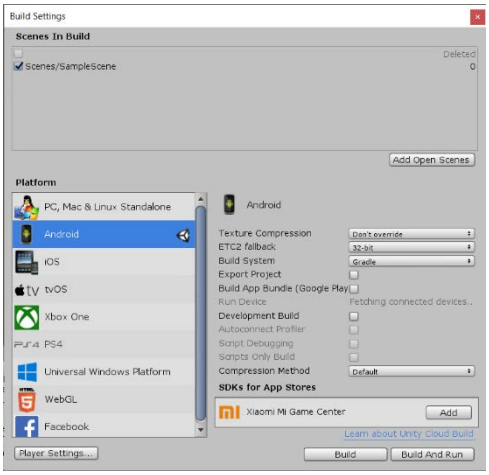
Exercise 6: Create a new scene, which contains one Image Target. Modify the default behavior of the `DefaultObserverEventHandler` script to display a different object every time that the image target becomes visible. For example, you can have three different objects attached to the Image Target and cycle them every time that the target becomes visible.

Tip: The Vuforia Unity API can be found here:

https://library.vuforia.com/sites/default/files/references/unity/index.html

Android Deployment

For testing purposes, we recommend you to execute your application directly from the Unity editor. If you want to test your application in your tablet you can build and launch your application using the File -> Build & Settings option.

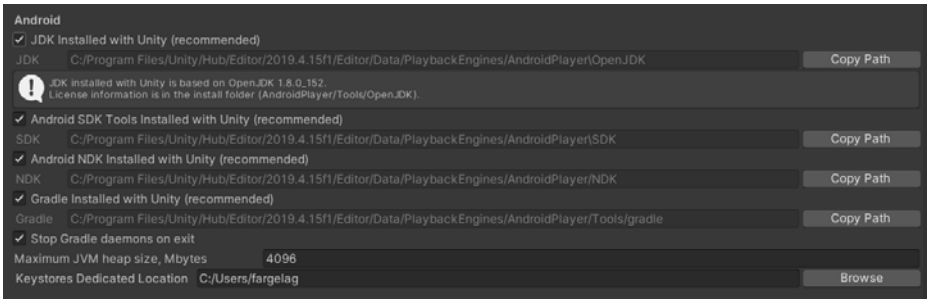


First, you have to ensure that the Unity scene that you want to test (you can have multiple scenes) is in the "*Scenes In Build"* list and that it is checked. By default the list is empty. You can add the active scene by pressing the "*Add Current"* button.

Second, you have to determine the target platform for your application. In our case, we will use the Android Platform. There is no need to change the default options.

If your tablet is properly configured, the build process will copy and launch the compiled application in your tabled. Please ensure that:

1. There are no issues with the Android SDK. By default, unity provides a version of it. If you encounter some issues, you might need to install the Android SDK. It can be downloaded from: http://developer.android.com/sdk/index.html#Other
   - Install the Android SDK in a user folder.
   - Open the Android SDK manager and install API (just click on install).
   - Update the Unity configuration to use the custom Android SDK. (**Edit** > **Preferences**)



2. Enable USB debugging from the tablet.
   - Preference Panel -> About Tablet. Press on Build number several times to enable the developer panel.
   - Developer options -> Enable the USB debugging option.
3. Connect the tablet using the USB cable and allow debugging once asked.

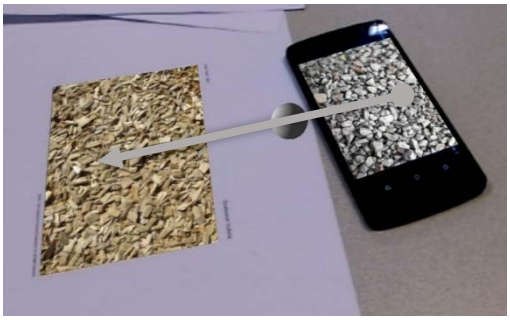
If some of this requirements are not meet, the build process will be interrupted and Unity will show an error. In addition, in some cases, there might be some errors during the build, the more common ones are

- Conflict with the build versions for ARCore. Solution, disable the use of ARCore. Go to the ARCamera gameobject, open the Vuforia configuration, and uncheck "Include ARCore Library".
- The minimum Android SDK is too low. Solution, go to Edit->Preferences, select "Player". In the options panel, go to the "Other settings" tab, search for the "Minum Level API" option, and set it to the minimum level indicated in the error while building.If some of this requirements are not meet, the build process will be interrupted and Unity will show an error.

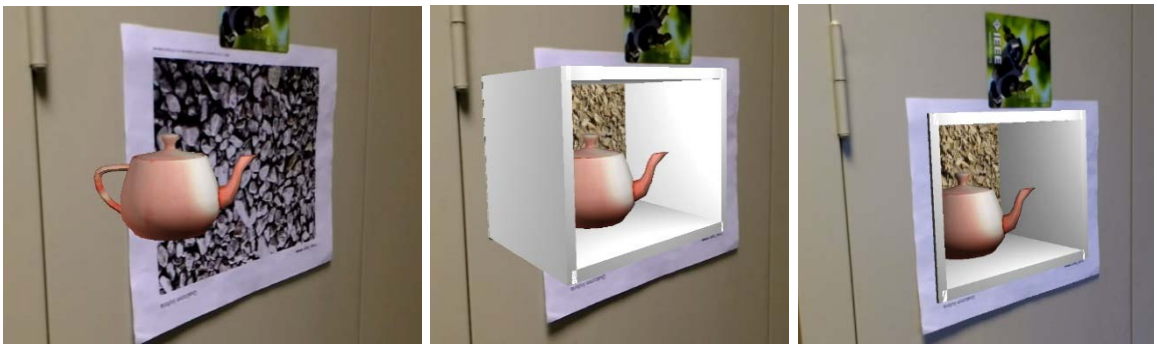
Exercise 7: Build and deploy exercise 5 and 6 in an Android device (phone or tablet). Describe the differences encountered with respect to the tests in the Unity editor.

Exercises

1. The goal of the second exercise is to change the default behavior of the `ImageTarget` prefab. In this exercise, two image targets will be required. The application behavior will be the following: If both targets are visible, a virtual ball has to be displayed on top of one of them, and afterwards an animation will be triggered to translate the sphere towards the other target.



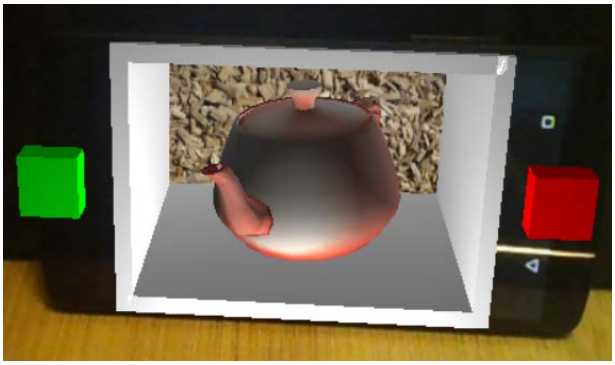
2. Sometimes, the integration between the real and the virtual content creates artificial image compositions due to inconsistent image occlusions. In the leftmost figure, the teapot should be displayed beneath the image target, however, the application does not know how to do it. The goal of this exercise is to correctly display an object behind an image target without creating any inconsistent composition.



In order to properly display the content, first, we have to "hide" the real content that should not be visible. Additional 3D objects should be added to the scene in order to hide it. However, as depicted in the middle figure, this is not enough, because the real content is not able to correctly hide the virtual content. The final step is to create virtual occlusions for the 3D content which can be achieved by reproducing the real geometry (e.g. door). This geometry will not be rendered, but it will occlude the other 3D objects.

This can be achieved using the "VR/SpatialMapping/Occlusion" shader. Create a plane, which will represent the virtual surface, and a new material. Attach the material to the plane object and in the Shader dropbox select the VR/SpatialMapping/Occlusion item. Explore which is the best way to achieve the desired effect.

3. Add two buttons besides the virtual object to enable its rotation. Each button, once pressed (on the tabled) should trigger a rotation of the virtual object. In order to create such behavior, you can attach a script for each object which implements the methods from MonoBehavior: (OnMouseUp, OnMouseDown, OnMouseOver). In addition, the color of each button must show its status, whether it's pressed or not. For the buttons you can use the basic Cube game object. For this exercise you do not need the Vuforia virtual buttons, just create standard unity triggers.



```
using UnityEngine;
using System.Collections;

public class RotateScript : MonoBehaviour {

    ...
    void Start()    {... }

    void OnMouseDown()    {... }

    void OnMouseOver()    {... }

    void OnMouseUp()    {... }
}
```