



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE



# 3D Augmented Reality

A.Y. 2022/2023

**Introduction to Unity 3D & Computer Graphics**

**LAB experience 1**

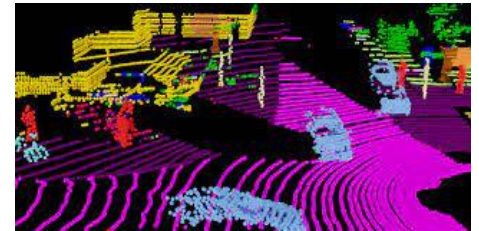
*Elena Camuffo*

[elena.camuffo@phd.unipd.it](mailto:elena.camuffo@phd.unipd.it)

# About Organization

**Elena Camuffo** [PhD student]

- *Previous AR projects:* MR dance (HoloLens + ML1), Ninja Girl Game (Mobile) <https://sites.google.com/view/university-of-padova-3dar/home>
- *Current Research topics:* **Point Cloud Semantic Segmentation, 3D Model Optimization for Immersive Applications**  
[https://stem.elearning.unipd.it/pluginfile.php/285469/mod\\_resource/content/1/THE\\_SIS\\_0622.pdf](https://stem.elearning.unipd.it/pluginfile.php/285469/mod_resource/content/1/THE_SIS_0622.pdf)



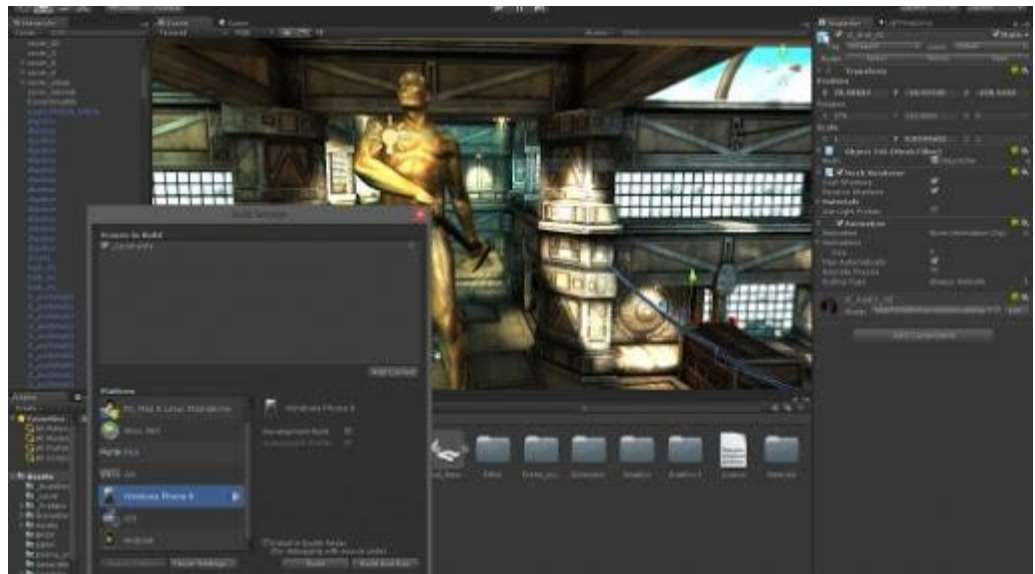
**Lab Schedule** [Wednesday 12:30-14:30]

1. Introduction to Unity 3D and AR
2. Physics and Graphics User Interface
3. Animations and Animator Controllers
4. ***Rendering and Shaders***
5. ***Acquisition and Simulation***
6. ***Deep Learning with ML Agents***



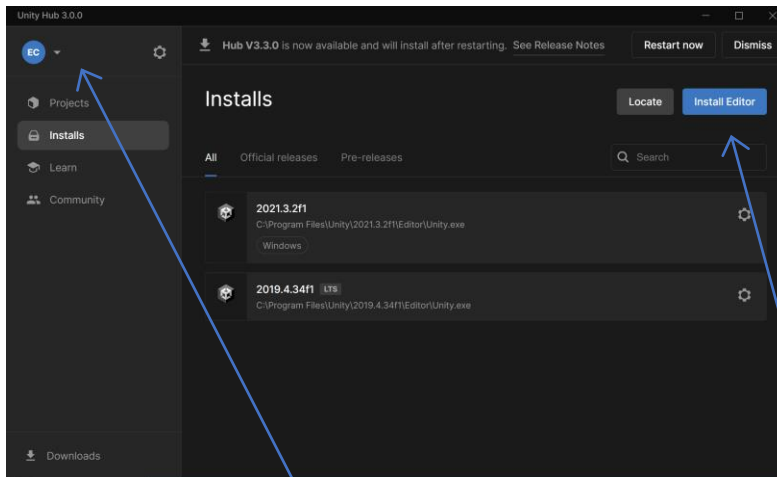
# Introduction to Unity 3D

- Unity 3D is a **cross-platform game engine** and a multi-platform IDE, developed by Unity Technologies.
- It is commonly used to create **3D, 2D, VR and AR** interactive visual content and video games.



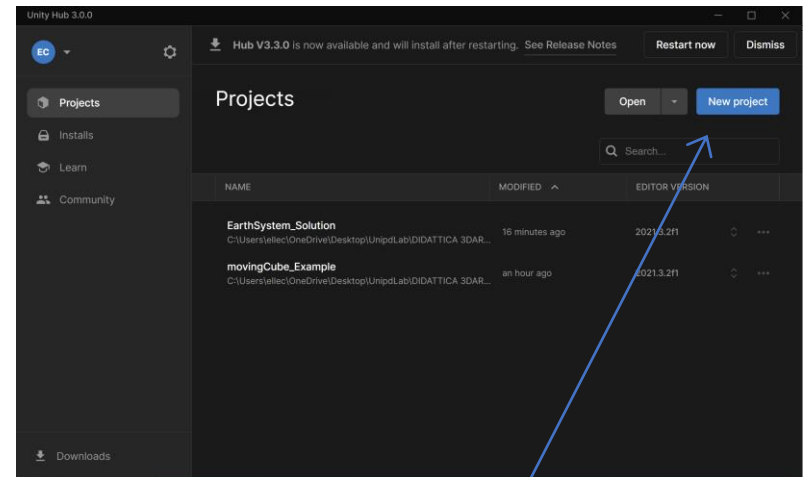
# Unity Installation

1. Download the Unity Hub (manager for multiple Unity installs):  
<https://unity3d.com/get-unity/download>.
2. Create an **account**, open Unity Hub and log-in.
3. Add a Unity release (we are using **Unity 2021.3.2f1**) from the *Installs* tab.
4. Further components can be added to each release to develop for various platforms.



Login

Add release



Create a New Project

# Interface

## Hierarchy window

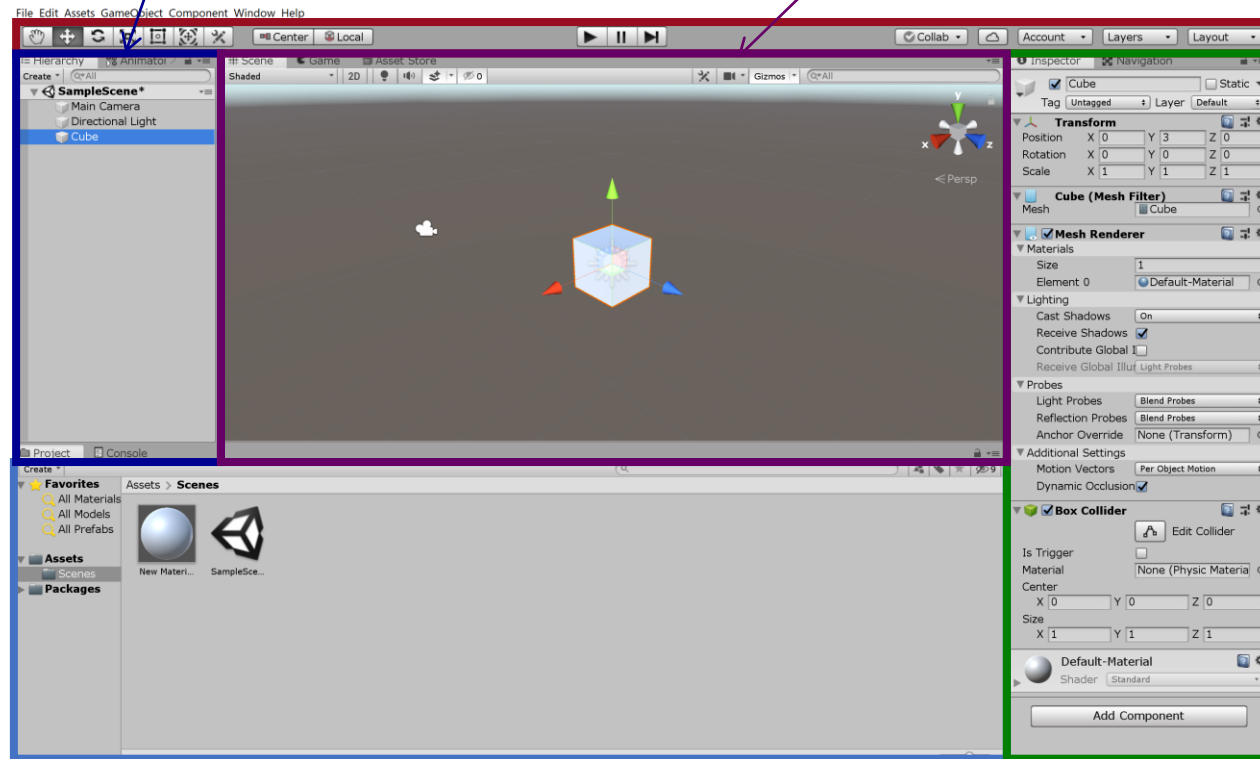
Lists game objects and their relations.

## Scene/Game view

Shows the scene editor/game preview.

## Toolbar

Basic controls for game objects, view change and simulation control (play/pause).



## Project/Console window

Reports the structure of the project (left), the assets contained in the selected folder and outputs messages for debug (right).

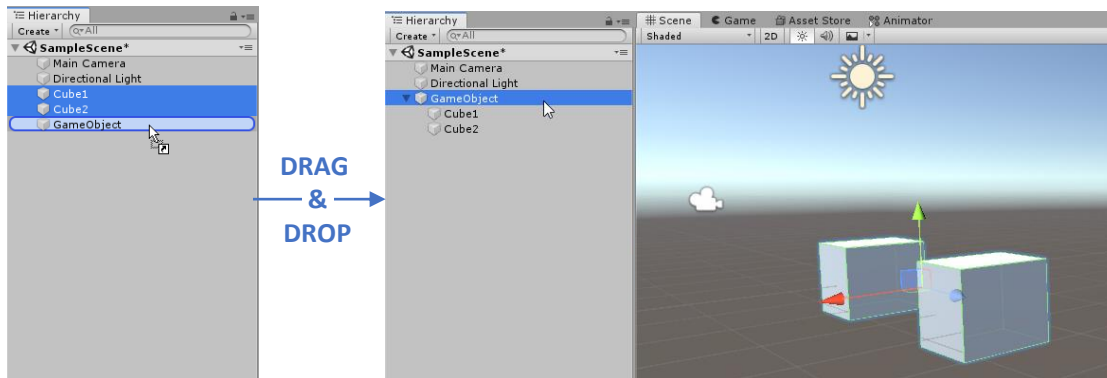
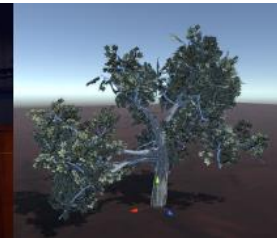
## Inspector window

Shows detailed information about the currently selected Game Object, including all the attached components and their properties.

# Game Objects

- An **Asset** is a generic file that can be used into a project (e.g., a font, an audio clip or a 3D object). It can be built inside Unity or with another program and then imported into Unity.
- When an asset is carried into a game scene it becomes a particular **Game Object**.
- Game Objects can be for example:

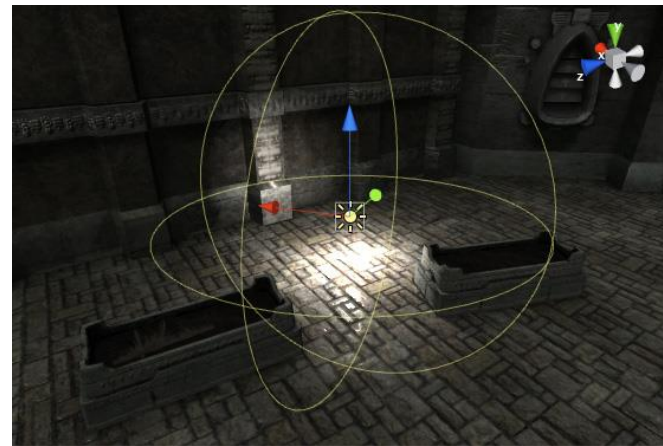
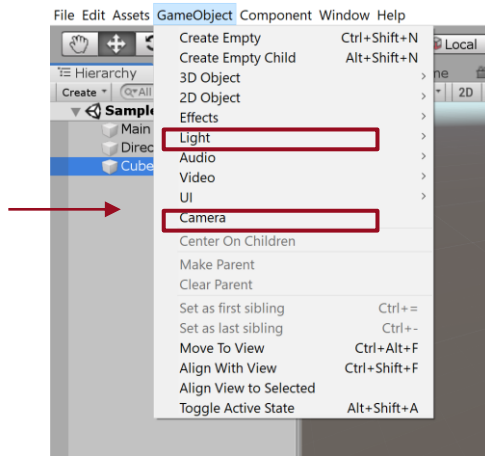
- Characters
- Lights
- Cameras
- Environment objects
- Audios



- They can be organized **Hierarchically**: the parent object acts as a container to the children.

# Cameras & Lights

- The fundamental Game Objects of your scene are the *Camera* and the *Light Source*. When you create a new project they are already in the scene by default.



- A **Camera** is a Game Object that defines a view in the scene space.
- A **Light** is a Game Object that that allows Unity engine to compute the illumination of the scene and the shading of the Game Objects inside.



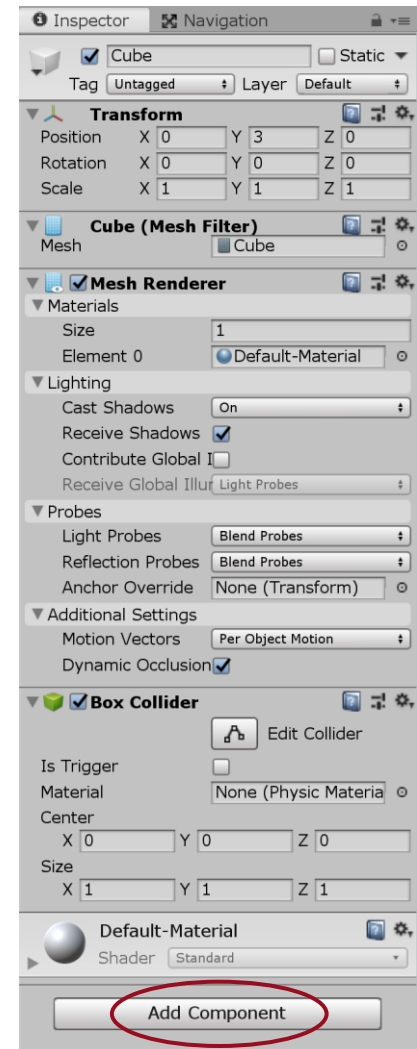


# Components of Game Objects

- Component is a **property** or **behavior** that can be assigned to a specific Game Object and can perform different functions.

*“You can think of a Game Object as an empty cooking pot, and components as different ingredients that make up the recipe of your game”*

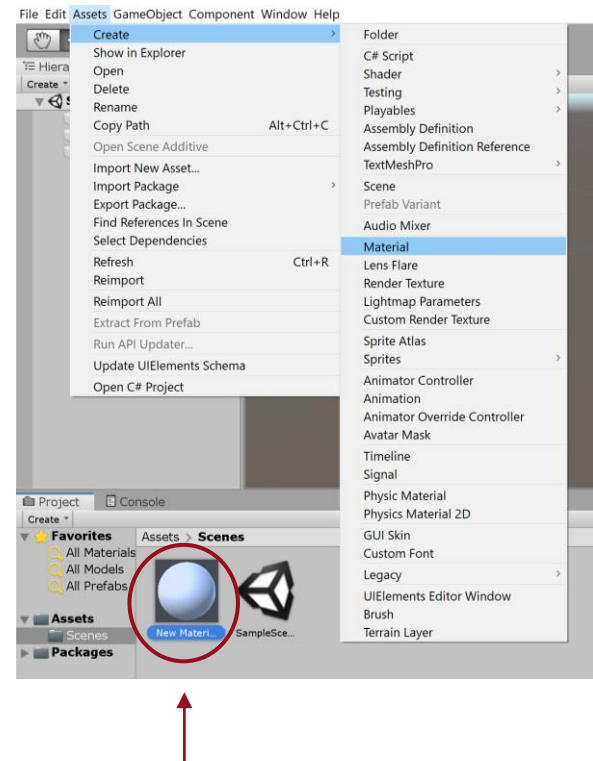
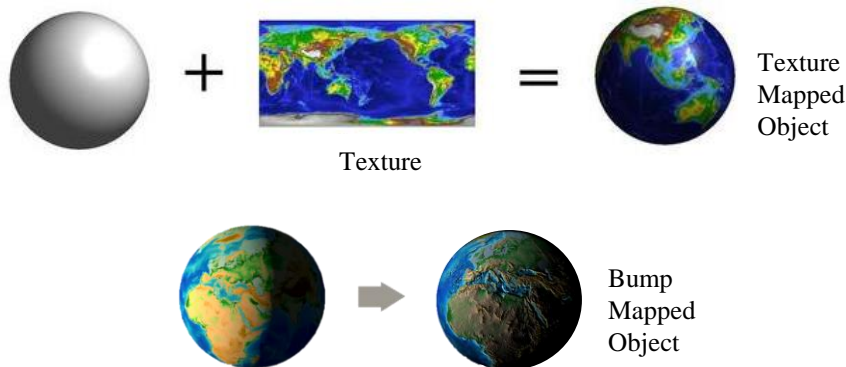
- The most common Components are built-in and offered directly by Unity.
- To add a new component to a Game Object you can click the *Add Component* button, or simply drag and drop the Asset on your Object in the Scene View.





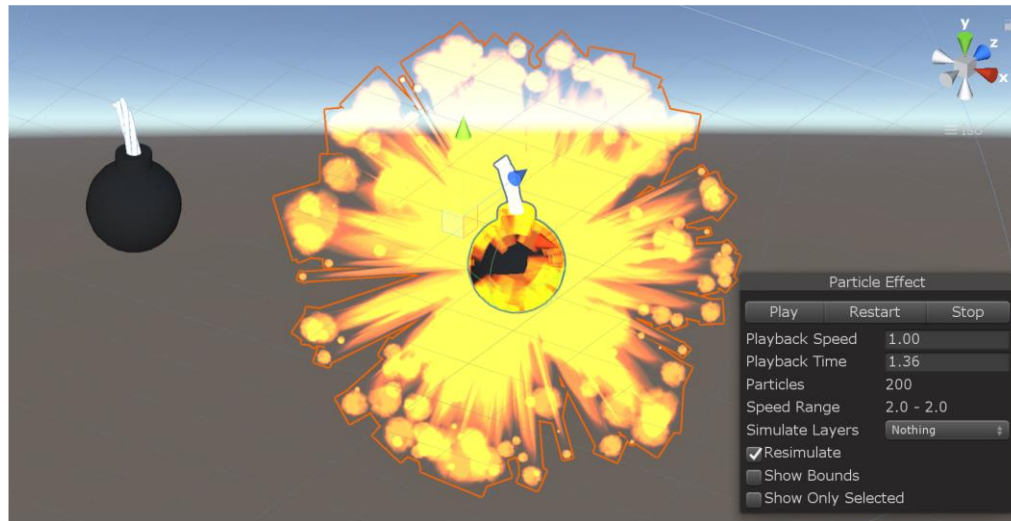
# Materials & Textures

- A **Material** is an Asset that can be attached to a Game Object (drag & drop onto it) defining how a surface of the object should be rendered.
- A **Texture** is a bitmap image that can be added as property (*Albedo*) of a Material, to define the color of the Game Object to which it is attached.
- In addition, Textures can represent many other aspects of a Material's surface such as its reflectivity or roughness.



# Particle System

- A **Particle System** is a Game Object (but it can be also added to a Game Object Component).
- It simulates and renders many small images called particles, to **produce a visual effect**. Each particle represents an individual graphical element, but the system simulates every particle collectively to create the impression of the complete effect.
- An Example of a Particle System is an explosion.



# Scripting

- A script is a **component** attached to a Game Object.
- It that allows to invoke some events, modify some properties and generally perform operations that are not included into the generic Components' functions.
- The programming languages supported are **Javascript** and **C#** (similar to Java).
- Unity relies on external programs to write Scripts' code, i.e., **Visual Studio** (or **Mono Develop** for older versions of Unity 3D).

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class NewBehaviourScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```



## Main Method: Start()

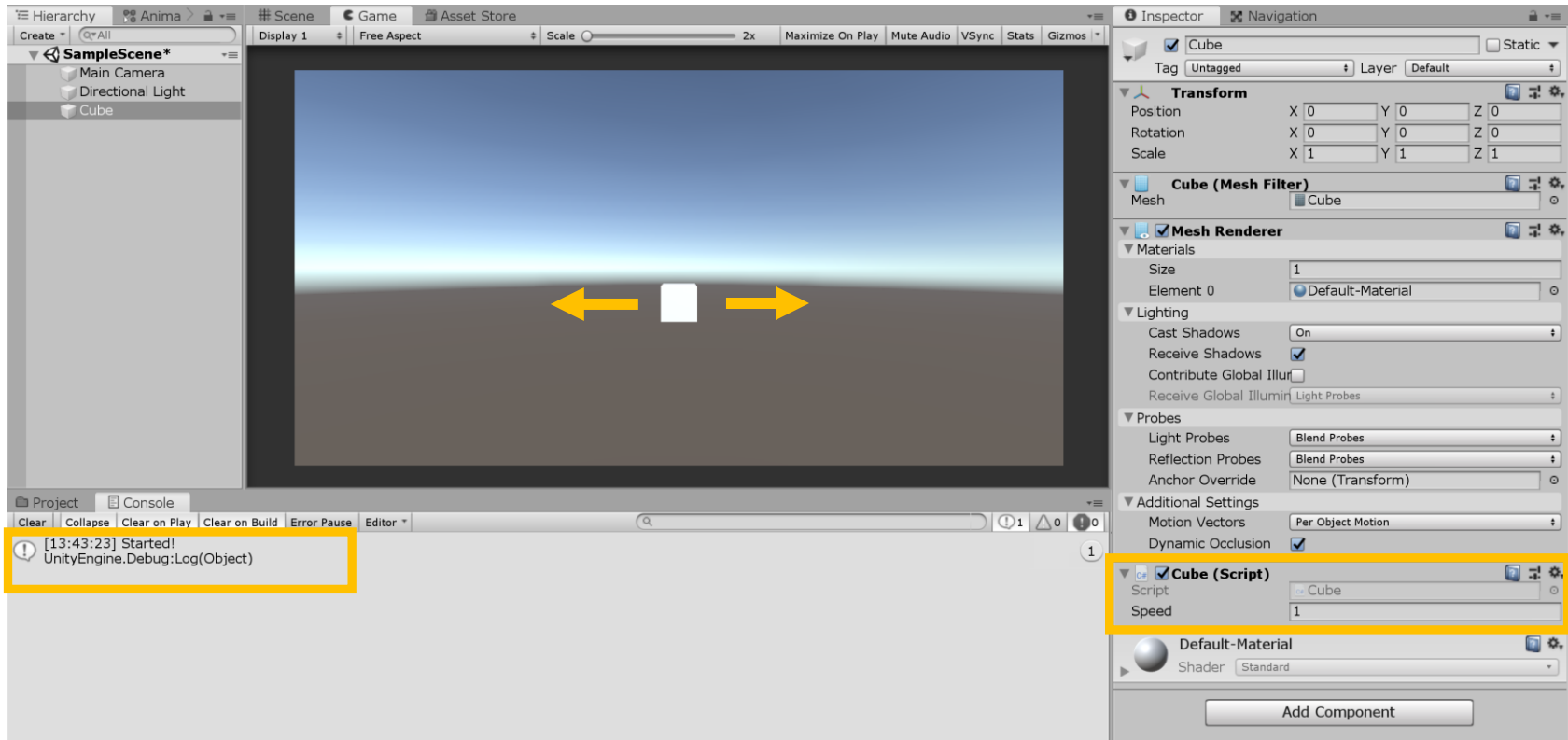
It includes the actions performed when the application is booted.

## Main Method: Update()

It is called every time the reference frame refreshes.

# Example

- Make a cube moving left or right along the x-axis.



# Script Example

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Cube : MonoBehaviour
6  {
7      // Serialized fields are private but visible from Unity
8      [SerializeField]
9      private float speed = 1;
10
11     // Start is called before the first frame update
12     void Start()
13     {
14         //initialize position
15         Debug.Log("Started!");
16     }
17
18     // Update is called once per frame
19     void Update()
20     {
21
22         // move by 'speed' units per second on the x-axis
23         float deltaX = speed * Time.deltaTime;
24
25         // compute the new position vector
26         transform.position = new Vector3(deltaX, 0, 0) + transform.position;
27
28         //limit the distance to 10 to fit the viewport size
29         if (Mathf.Abs(transform.position.x) > 10)
30             transform.position = transform.position - new Vector3(Mathf.Sign(transform.position.x) * 10, 0, 0);
31     }
32 }
```

## Class Variables

It is set as the speed of movement (the sign decides the direction).

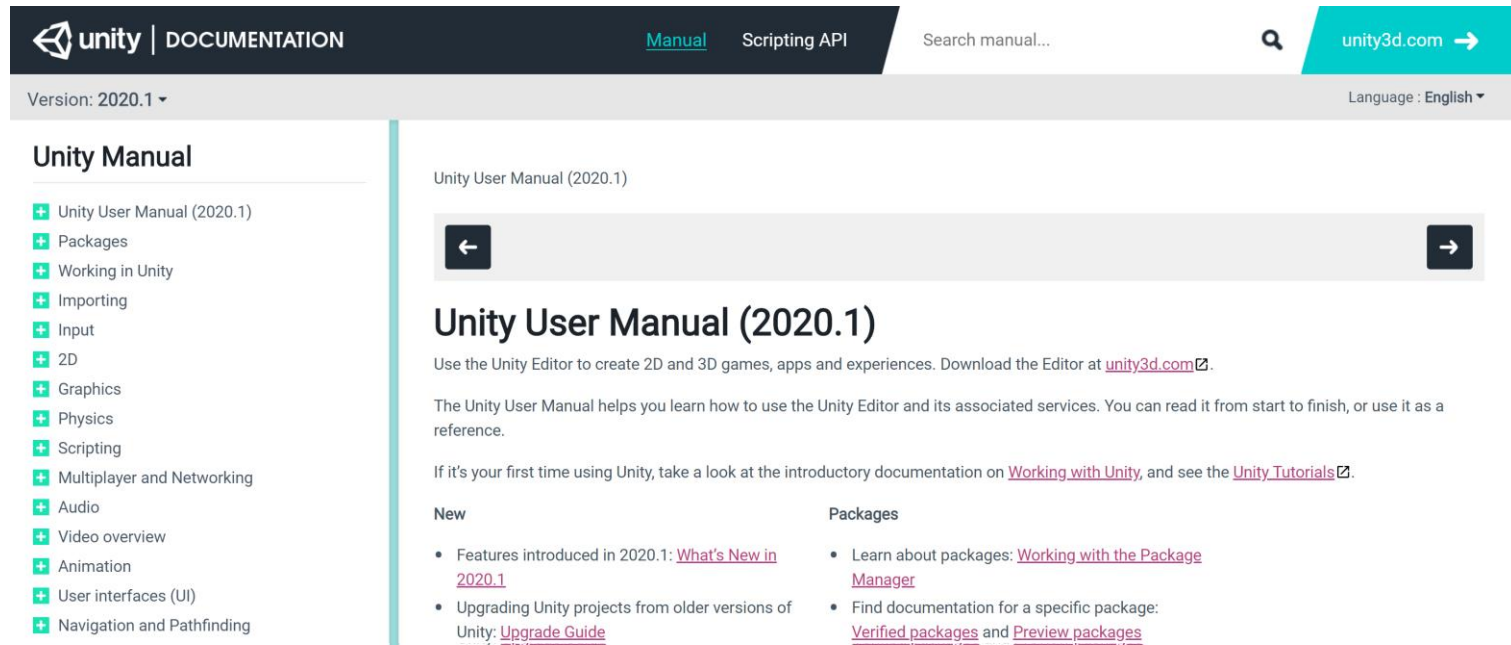
## Start()

Write a debug message when the Cube Game Object initializes.

## Update()

Compute new position along the x-axis and move the Game Object accordingly.

# Documentation



- Unity Documentation is useful to solve problems. Remember to check your **Unity version** and make sure you are looking at the appropriate reference.
- **Manual**  
<https://docs.unity3d.com/Manual>
- **Scripting API**  
<https://docs.unity3d.com/ScriptReference>

# Vuforia

- Vuforia is a **Software Development Kit (SDK)** that allows marker-based tracking for AR.
- For older releases of Unity, download it from <https://developer.vuforia.com/downloads/sdk>.
- To start building AR applications you have to **create an account**.



Check out funny projects @:  
<https://www.theknightsofunity.com/project/ar-projects>





# Add a Vuforia license

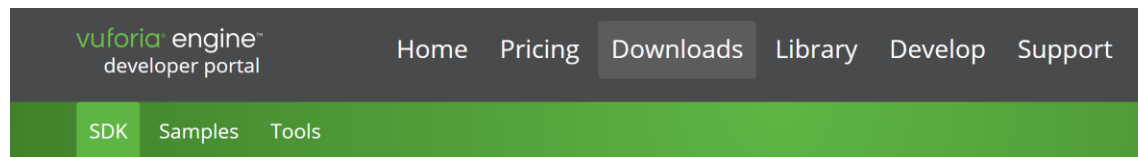
- In the *Develop* section you can create your own license.
- The license is a **code**, which has to be copied and pasted into the *Vuforia Manifest* of your Unity project in order to allow you to export and build your application.

The screenshot shows the Vuforia engine developer portal. The top navigation bar includes links for Home, Pricing, Downloads, Library, Develop, and Support. The 'Develop' section is active, showing sub-links for License Manager, Target Manager, and Credentials Manager. The 'License Manager' page displays a 'Get Basic' button circled in red, with an arrow pointing to it from the text 'Create New License'. Below the buttons is a search bar and a table of licenses. The table has columns for Name, Primary UUID, Type, Status, and Date Modified. The first row, 'CuboProva', is circled in red, with an arrow pointing to it from the text 'Open Existing License'.

Name	Primary UUID ⓘ	Type	Status ▾	Date Modified
CuboProva	N/A	Basic	Active	Jul 25, 2019
AR_Penguin	N/A	Basic	Active	May 01, 2019


# Add Vuforia to Unity

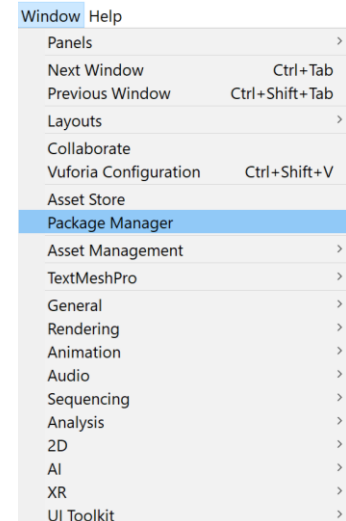
- To add Vuforia to your project, go to the **Downloads** tab and select *Add Vuforia to a Unity Project*. This downloads an executable file that imports automatically Vuforia in your open Unity project.
- Alternatively you can open the **Package Manager** window in Unity and search for Vuforia.



## Vuforia Engine 10.10

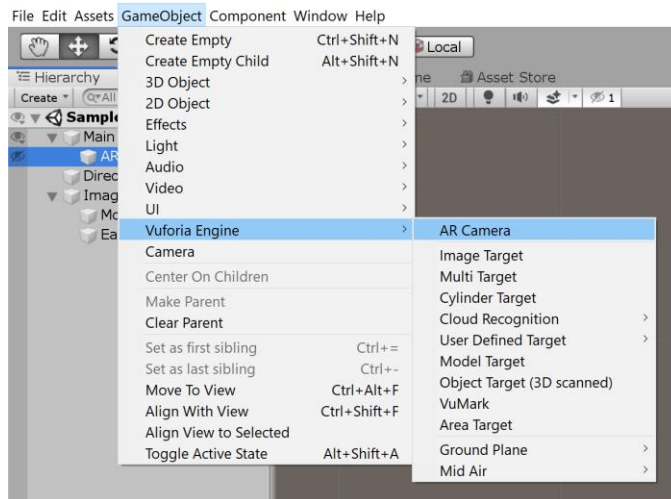
Use Vuforia Engine to build Augmented Reality Android, iOS, and UWP applications for mobile devices and AR glasses. Apps can be built with Unity, Android Studio, Xcode, and Visual Studio. Vuforia Engine can be easily imported into Unity by downloading and double-clicking the .unitypackage below.

 [Add Vuforia Engine to a Unity Project or upgrade to the latest version](#)  
add-vuforia-package-10-10-2.unitypackage (244.31 MB)  
MD5: ce66d6469c349e67acc2defb5173483e



# Set up your Unity project

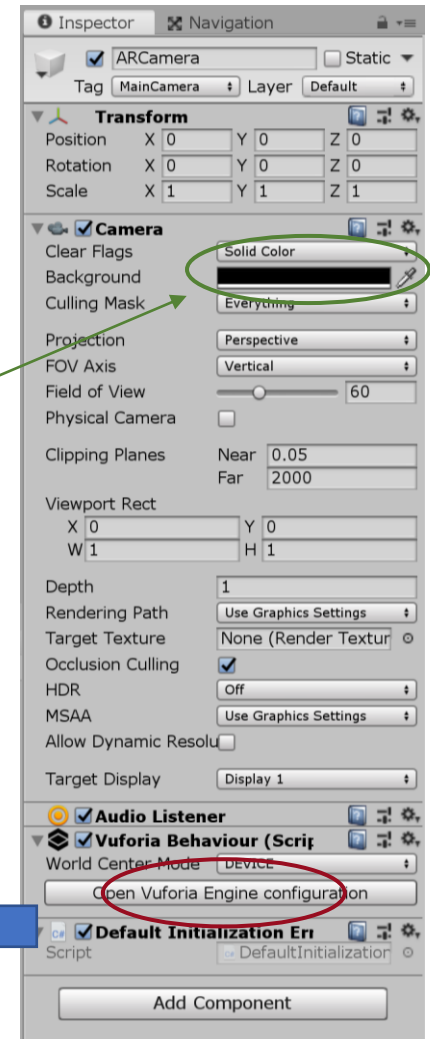
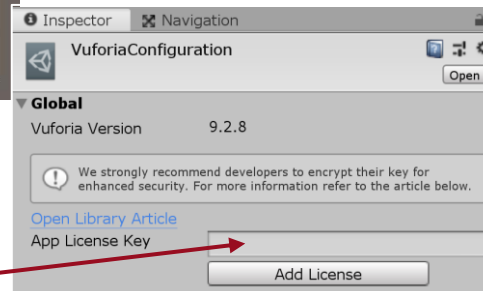
- The fundamental element that is needed to simulate Augmented Reality is the **AR Camera** Game Object.
- From the *Vuforia Behaviour script* component you can reach the *Vuforia configuration*, where to add your license key.



## NOTE:

The solid color (black) of the camera background allows AR independently on the device chosen.

Paste here your license



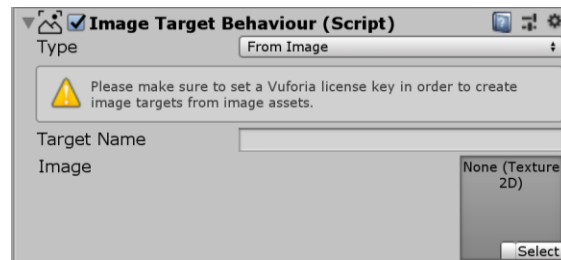
# AR markers

- An **image-based marker** can be added as a Game Object and the target image needs to be added into its *Image Target Behaviour Script* component.

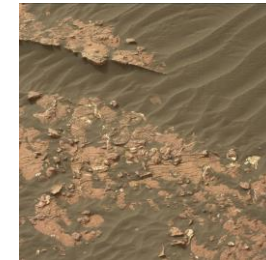
Virtual Object  
Visualized



Real World Marker



DRAG  
&  
DROP



- Starting the simulation, you will see the camera view in place of the solid background.
- If the target image is detected, all the objects that are set as **its children** are visualized and move together with the target.

# EXERCISE: Earth System



## 1. Build the Scene

Create a sphere game object, assign it a new material and add a texture (look at the provided files). Add a light and move/rotate the camera to frame the scene.



## 2. Add an Animation via Scripting

Write a C# script code and attach it to the sphere, to make it rotate around its y-axis.

Suggestion: you can do it by modifying the *Transform.rotation* property of the object, keeping into account that it works with *Quaternions* (Look at the documentation for more information).



## 3. Make the Scene in AR

Add a Vuforia *AR Camera* and a marker (choose it wisely between the ones provided) to visualize your sphere and allow it to move in the real world.



## 4. (ADVANCED) Enrich the scene

Add the Moon and write a script to make it rotate around the Earth. If you want, you can add also the Sun and the other planets to complete the Solar System.



## 5. (OPTIONAL) Particle System

Write a script to make a planet explode whenever *spacebar* is pressed. The explosion can be built as a particle system.