

LAB ONE

© 2019–2022 Wiley InterScience. Reprinted by permission.

Nguyen D. & Burdea G. Virtual Reality Laboratory Manual, 3rd Ed.

The First Scene



✖ Requirements

1. Unity 3D (using version 2018.4+ and after)

Unity will be the primary tool you will use to develop 3D games. In the Prelab, you learned the basics of getting around the Unity editor. In this Lab, you will be able to create your very first project.

The structure of all the labs will have two primary sections – Basic and Advanced. What this really means is for graduate students, there will be extra work and/or things to learn in the Advanced section, while undergraduate students are only expected to complete the basic section. For this first lab, only the post lab will have more work.

Contents

Start of a Scene	01
Basic Components	02
Materials & Shaders	03
Main Maps	
Other Maps & More	
Lighting a Scene	07
Skyboxes	10
Cameras	11
First-Person Character Controller	
Audio Sources & Listeners	13
Post Lab Homework Assignment	14
References & Additional Sources	15

Previously, you learned about the modules and how to navigate the Unity interface. We will apply that knowledge here and focus on the usage of *assets*, *GameObjects*, *components*, *lighting*, and more. We will not go into detail over low-level things such as navigation, and you should be able to find and navigate the editor on your own. As before, relevant links will be provided throughout the lab if you want to learn more (click on the ‘?’ icons).

Start of a Scene

When you start a new project, you will be presented with a blank template scene. In the hierarchy window you will be able to see that there are two GameObjects already in this template scene: a camera and a directional light. We will be using them soon, so keep them there for now.

When you start a new scene, you need to create GameObjects to fill up the scene and create a world of your own! Unity has a small collection of basic primitive shapes (e.g. cube, sphere, etc.) and others such as light sources, audio sources, and cameras.

In this example, we will be looking at a simple cube. You may remember seeing the inspector window of this cube from the Prelab. We will now go over some of the basic components of this cube and manipulate them to change how the cube looks and behaves.

If you remember from the Prelab, there are two ways to create a new GameObject. You can create it from the hierarchy window, or go to the GameObject dropdown in the top menu bar.

Once you find and create a cube, click on the cube to bring up its inspector window.

Basic Components ?

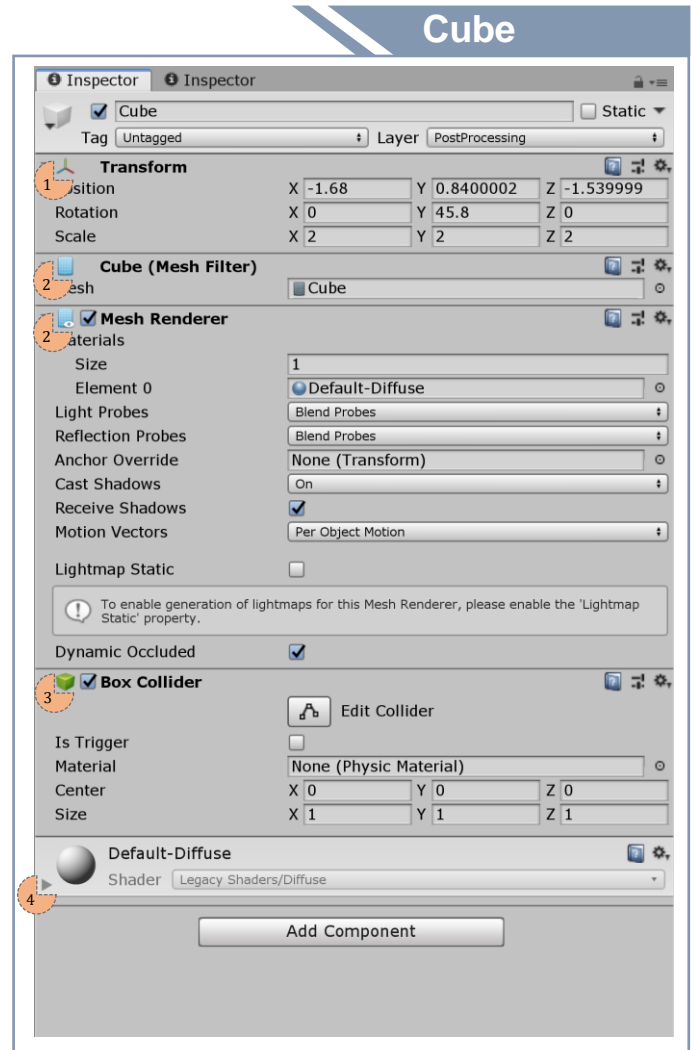
This window should be very familiar to you as we covered some of it in the Prelab. However, we had not covered the components yet. There are a wide range of components, but for the purpose of this course and this lab, we cannot cover them all. If you want to try something that is not covered, refer to the official Unity Manual and learn more.

Here is a list of the most common components that you will see often:

1. Transform;
2. Mesh Filler & Mesh Renderer;
3. Collider (Box, Sphere, Capsule, Mesh);
4. Rigidbody;
5. Audio Source;
6. Light Source;
7. Material.

You can see that the cube we have in **Figure 1** has a total of four components: “Transform”, “Mesh Filler”, “Mesh Renderer”, and “Box Collider”.

The Transform component [**Figure 1, #1**] is on every GameObject and cannot be removed. This is because it defines where the GameObject is located and positioned in virtual space in addition to the scale and rotation.



The inspector window of a cube GameObject.

Figure 1

This can either be relative to the world coordinate system or to a local coordinate system.

The Mesh Filler and Mesh Renderer components [**Figure 1, #2**] are dependent to each other, so you should normally see both or neither. These components define the mesh – the geometry of the object – and how the mesh is rendered or shown. There are more settings for changing the material, enabling or disabling probes (covered later), cast or receive shadows, and allowing static lightmaps to be used on it.

If the GameObject has a material attached, it will be shown at the very bottom of the inspector window [**Figure 1, #4**]. You may notice that it is grayed out. This is because the cube is using a default material that cannot be edited. It is applied automatically to new GameObjects. When you change the material to a custom one, it can be changed and edited within the inspector window. It is important to note that this will change the material itself and any GameObject that is also using that material will be changed too.

Next is the Box Collider [**Figure 1, #3**]. There are a few different kinds of primitive colliders: box (aka cube, also referred to as bounding box in the class), sphere, and capsule. There is another type of collider, called *mesh collider*, where the collision is calculated using the mesh of an object.

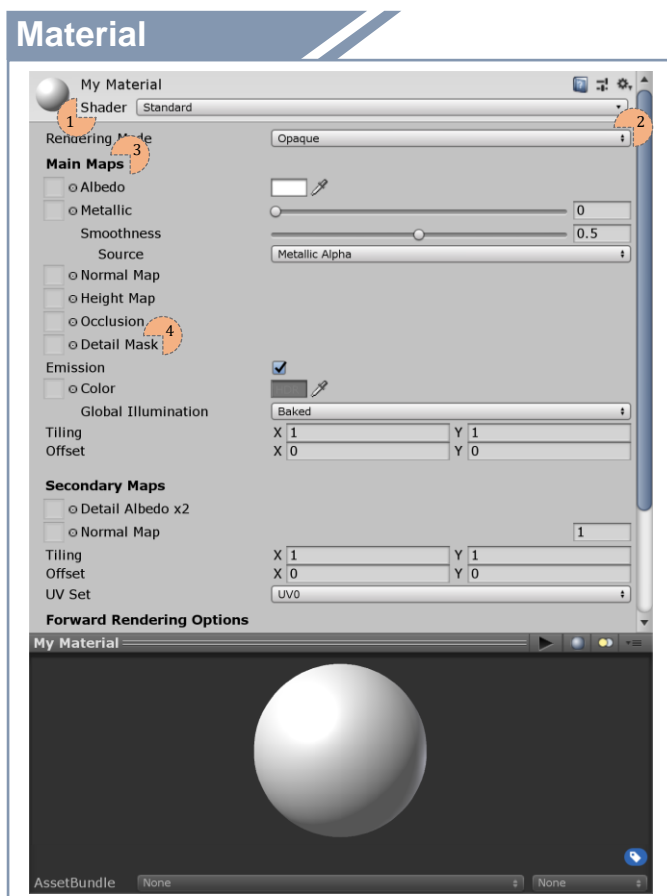
Colliders are used with objects that have physics. These objects are rigidbodies, which are a component that applies physical properties to them such as collision, velocity, gravity, and more. Colliders define where objects are solid and should collide. If the GameObject does not have a collider, other things will be able to pass through it. Sometimes, there is a performance cost when using mesh colliders with complex meshes. This can be reduced by using primitive colliders, which are less accurate, but faster to compute. You can even apply multiple colliders on a single GameObject to create the rough shape of the object's mesh!

The first-person controlled character (FPC for short), has a capsule collider component with a rigidbody component. This applies gravity and collision detection so colliders on the ground and surrounding environment can block the FPC from moving through them. More on this in a later section³. You will be doing more things with colliders and rigidbodies in future labs.

Materials & Shaders

The appearance of a GameObject depends on the material it is using. Materials are defined by textures, colors and shaders. Let's start by creating a material in the project window within the asset folder. You can create a folder dedicated to materials to keep them organized.

Using the right-click menu or “create” drop-down, find “Material” and click on it. This will create a new material. By default, this material uses Unity's “Standard” shader [?]. First, apply it to the cube and then click on the material itself to open its inspector window, as seen in **Figure 2**.



There are many types of shaders that may have varying settings. You can even write your own shader with your own custom settings. For simplicity in this lab, we will stick with the standard shader.

The shader can be changed at the very top of the inspector window **[Figure 2, #1]**. It provides a drop-down menu containing all available shaders.

The rendering mode setting **[Figure 2, #2]** determines if the material is transparent or not and which kind of blending is used.

In the next section we will discuss the main texture maps that define the material **[Figure 2, #3]**. The following maps controls color, texture, shading, and more.

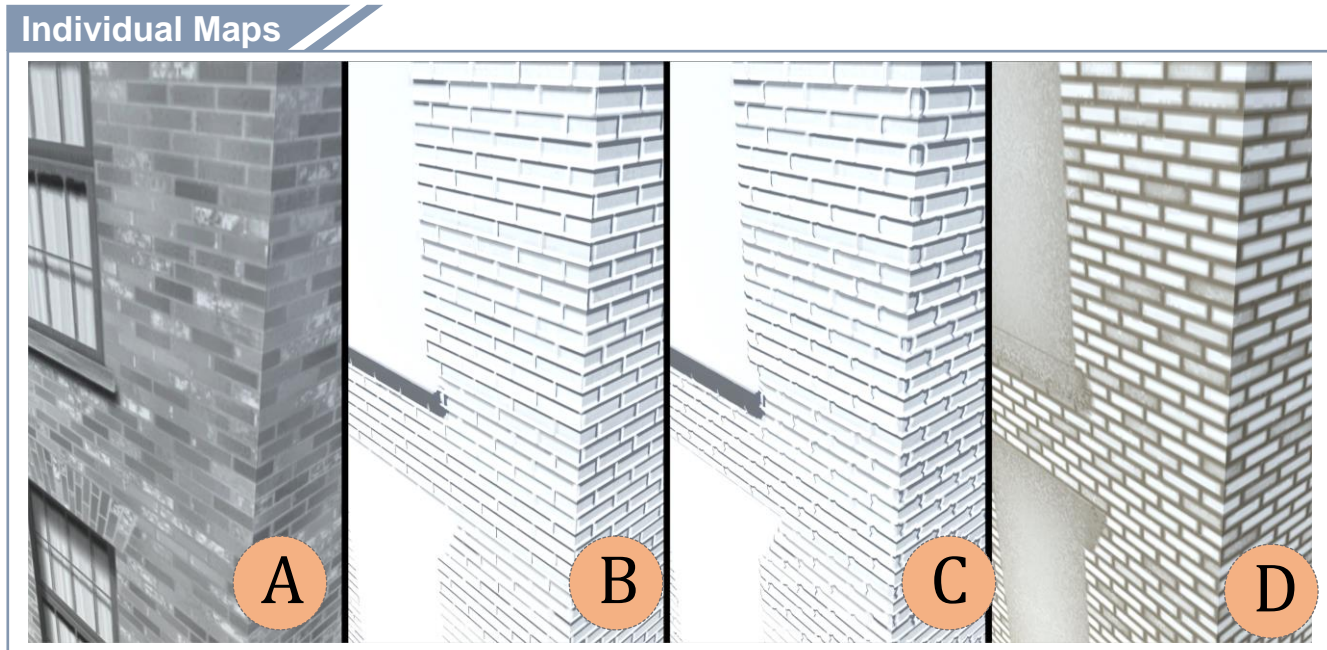
You can add texture assets to the following maps by clicking on the circle to the left of their respective names. Alternatively, you can drag and drop the asset into the leftmost square. 3

The inspector window of a new material.

Figure 2

Main Maps: Albedo, Normal, Height, & Occlusion

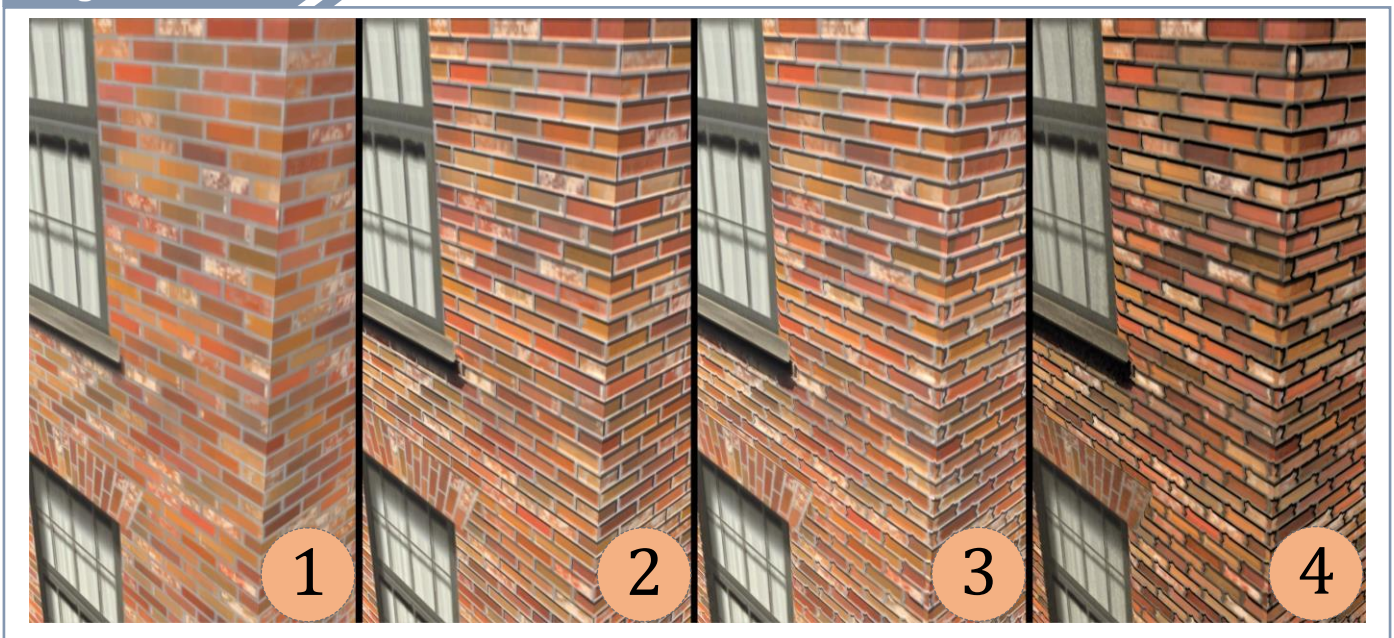
In what follows, we will apply different texture mappings to build up a texture of a brick wall with some windows. Figure 3 shows the different maps individually while Figure 4 shows the progression of layering the different texture maps on top of each other to create a more realistic appearance.



(A) Albedo texture (black & white version), (B) normal map, (C) height map, (D) occlusion map.
Credit to Craig Nisbet for the texture maps.

Figure 3

Progression



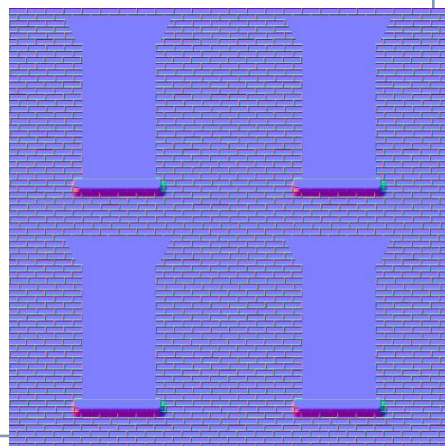
Same order as in Figure 3, except the multi-texture layers build up from the previous one.

Figure 4

? Albedo specifies both the color and texture of the material. The color can be edited by clicking the colored rectangle to the right of it. As an example, we will apply a brick texture. The color of the texture can be shifted using the color selection box to the right. You can leave it at white if you want to use the color of the base texture.

Below that are sliders for metallic and smoothness. These two control the reflectivity and reflective properties of the material. The smoother the material is, the more reflective, just as you would expect. These properties can be used in conjunction with a GameObject called a Reflection Probe, which allows you to create accurate reflections on surfaces. Reflection probes will be covered in a future lab. The smoothness in this example will be reduced to zero, since we want the bricks to not have reflective properties.

Normal Map



The normal texture map of a wall. Figure 5.B

? The height mapping is similar to normal mapping but is more complex and thus has a performance impact. Height maps are normally used in conjunction with normal maps to create even more detail. These maps can create the illusion of protruding bumps on the surface of the object by shifting the visible surface texture around. It is important to note that this does not modify the mesh or geometry of the object even when it looks like it would.

A height map is a grayscale image as opposed to the blue normal map. It represents the high and low areas of the texture (light areas are higher than dark areas). The strength of the mapping can be changed by the slider to the right. It should be noted that setting the strength too high will cause major distortions.

Albedo Map

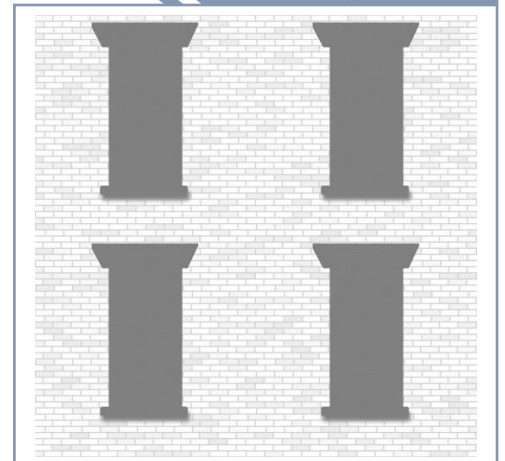


The Albedo texture map of a wall. Figure 5.A

? A normal map is a special texture that allows you to add surface detail without changing the mesh or geometry of the object. This is primarily used for small details so that the geometry can remain simpler, but still yield a realistic effect. The strength of the mapping can be changed in the number field to the right.

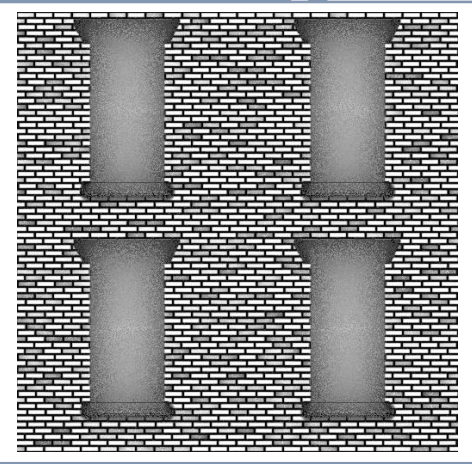
If you don't know what a normal vector is, it is a vector which is perpendicular to the surface at a given point. The normal map data looks blue, but the colors are not intended to be shown. The coloration are just instructions to modify surface normals. By modifying the normal vectors, it is able create shadows or highlights on a flat surface depending on lighting.

Height Map



The height texture map of a wall. Figure 5.C

Occlusion Map



The occlusion texture map of a wall. Figure 5.D

❓ The occlusion map defines what areas should receive high or low ambient lighting. Similar to the height map, this is also a grayscale texture. The lighter areas will receive more ambient lighting while the darker areas will receive less.

This is usually used to darken grooves and bring more lighting to peaks. Realistically, less light will reach inset creases or grooves making them appear darker. In this occlusion map however, the author darkened the individual bricks as well. If you look closely, the image is grainy in some areas which create some additional texture.

In some simple cases, height maps can be used as occlusion maps. In this example however, the texture author created a separate occlusion map that makes the bricks darker as well.

When searching for textures on the internet for your own projects and scenes, it may be hard to find a set of texture maps. Normally, you would only find a single image as the primary texture and no normal, height, or occlusion maps to go with it.

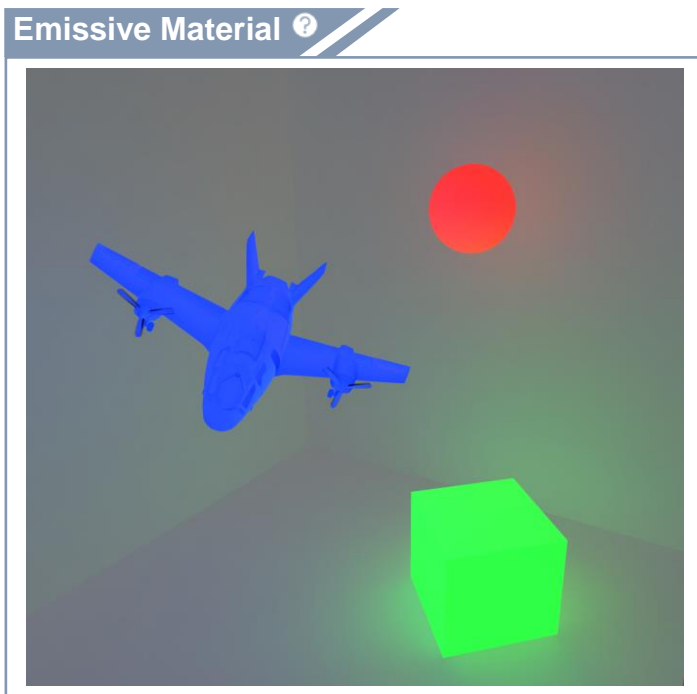
One way to get these maps is by searching for them. There are many normal maps available for download, but they may or may not fit your needs. Another way is to create these maps yourself using the original image and an image editor.

The easiest way, but not the best way to create a normal map, is to duplicate the texture in Unity and select it so it shows up in the inspector window. Then, change the “Texture Type” at the very top to “Normal Map”. This will convert the non-normal map image into an image with a blueish hue that can be used as a makeshift normal map. When importing real normal maps into Unity, this will need to be done as well, and Unity will issue a warning for you to fix it.

Underneath emission, you can find two options called “Tiling” and “Offset”. The tiling option allows you to control the number of times a texture repeats itself. For example, you can repeat the texture in one direction for a very long road in order to prevent the texture from being stretched. Another example would be tiling in both directions to cover a large area. With the offset option, you can adjust the positioning of the texture in case the texture does not line up correctly.

Other Maps & More: Detail Mask, Detail Maps, & Emission ❓

Continuing from [Figure 4, #4], there is an option called detail *mask* (not to be confused with detail *map*). Detail *maps* are textures that overlay the main texture. These are used for fine details when viewing the model up close. Usually they consist of a small texture that is tiled many times. Similar to the main texture maps, the detail map can also be tiled and offset. The detail *mask* allows you to mask off areas where you do not want the detail map to be applied to.



Scene with emissive materials.

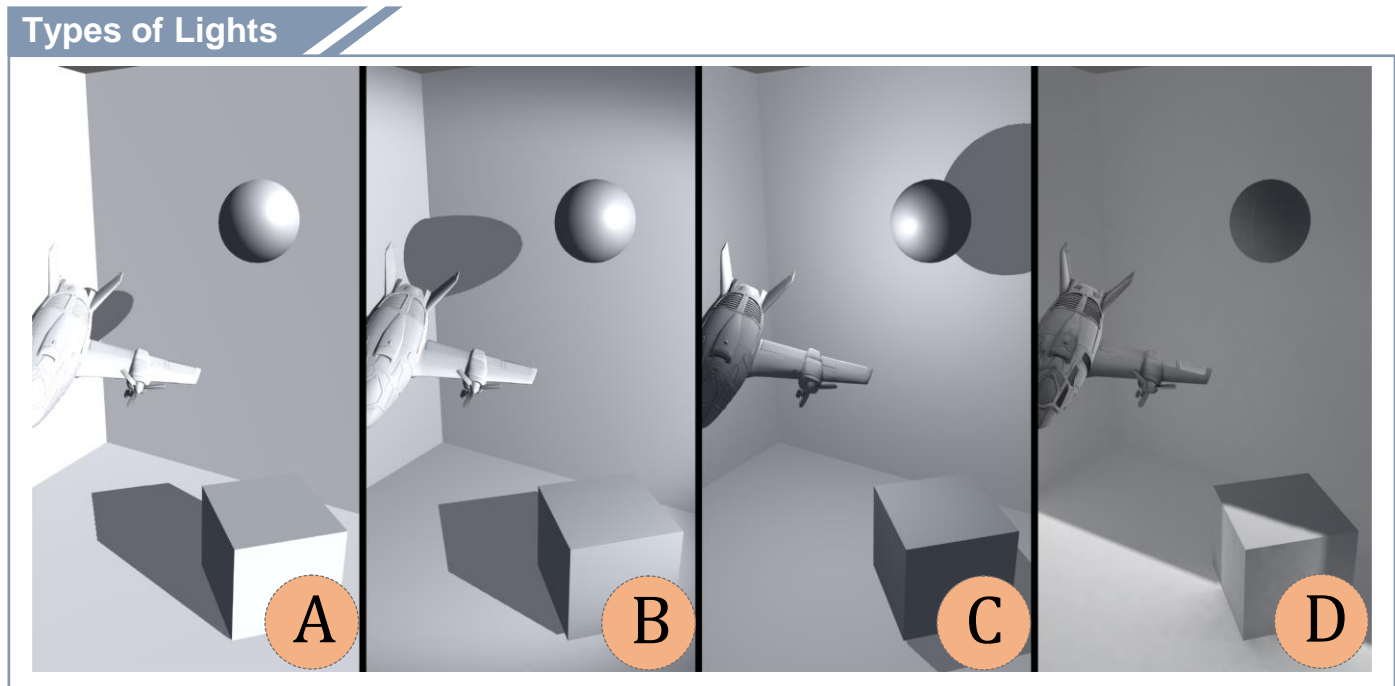
Figure 6

The emission option is what its name suggests; it makes the surface appear lit and can produce a glow to surrounding objects with the right settings. When checked, you will be able to choose a color and use a texture map for the emission. The texture map also allows you to mask with black coloring parts of the object where emission is not wanted.. In order to produce light to nearby surrounding objects, the objects must be set to “Static” which means they cannot move. The lighting will be “baked” which means it is precalculated before you play it (off line during authoring of the scene) and will not change at run time. Runtime illumination is possible with the use of light probe groups, which will be discussed in the next section of the Lab, discussing lighting.

Lastly, there are a few more options at the bottom that allow you to enable and disable reflections and highlights.

Lighting a Scene

Light is a component that can be found on lighting GameObjects. You can create these GameObjects or add the light component to an existing GameObject. There are 4 different types of light sources, each with different uses and properties. A preview is shown in **Figure 7**.



Examples of (A) directional light, (B) spot light, (C) point light, (D) area light (or ambient light).

7
Figure 7

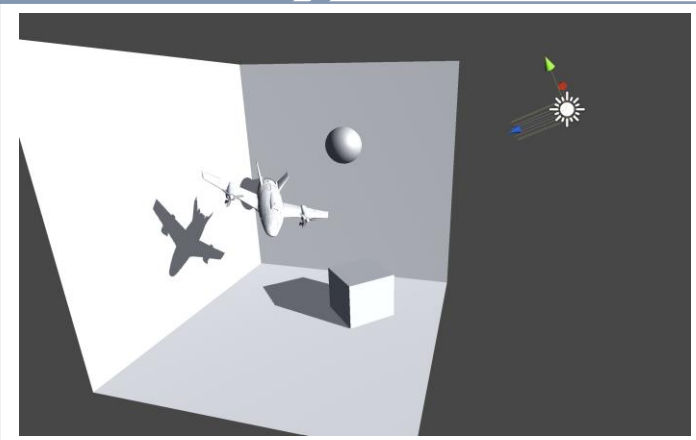
There are many different settings that can be changed to effect how the lighting behaves. Each light source type has varying settings, and some can be seen in **Table 1**. More on documentation[?].

Light Settings

Color	Defines the color of the light that is emitted
Range	The range of the lighting. (Only on spot and point lights)
Spot Angle	The angle in an outwards cone from the light source. (Only spot lights)
Mode	Changes the baking mode of the light. Options: Realtime, Baked, or Mixed.
Intensity	The amount or brightness of light emitted.
Shadows	There are multiple controls for shadows. These determine if the light source can cast shadows and how those shadows behave.
Cookies	Not actual cookies! Cookies specify a texture mask that is used to create shadows for silhouettes or patterns.

Table 1

Directional



Scene with directional lighting.

Figure 8.A

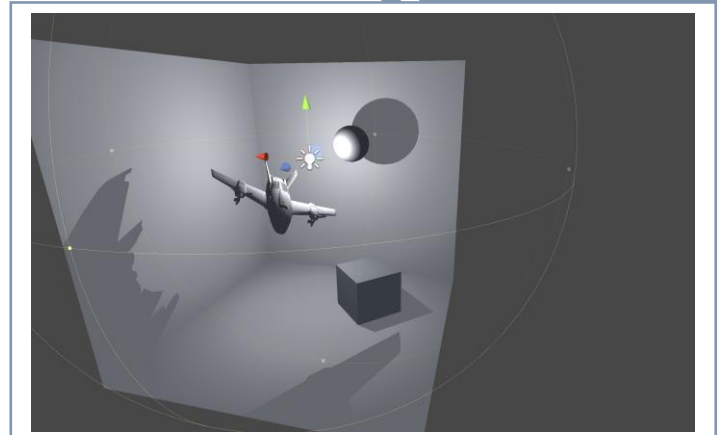
[?] When creating a new scene, Unity will automatically create a “Directional Light”. This type of light is used for creating the effect of sunlight. It does not have an identifiable source position, since all objects are illuminated from the same direction. Some skyboxes, and specifically the standard skybox Unity uses, have a sun that follows the angle of the directional light. We will discuss skyboxes in the next section. It can also be noted that the position of the directional light GameObject does not change anything; only the angle influences the direction of the light.

A point light starts at the central point and emits light in a spherical radius. The radius or range of the light can be changed in the settings.

This type can be used for many things including explosions, sparks, fire, lamps, etc.

For an explosion, the light source can be spawned in momentarily while the explosion is happening with scripts. For a lamp, you can place the light source inside a light bulb which will emit light in a convincing way.

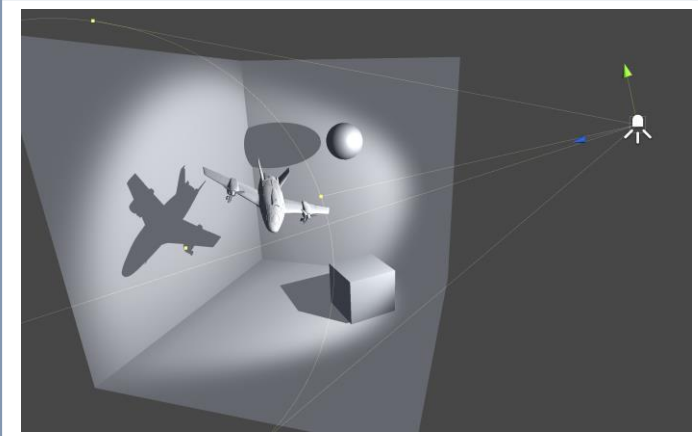
Point



Scene with point lighting.

8 Figure 8.B

Spot




Scene with spot lighting.

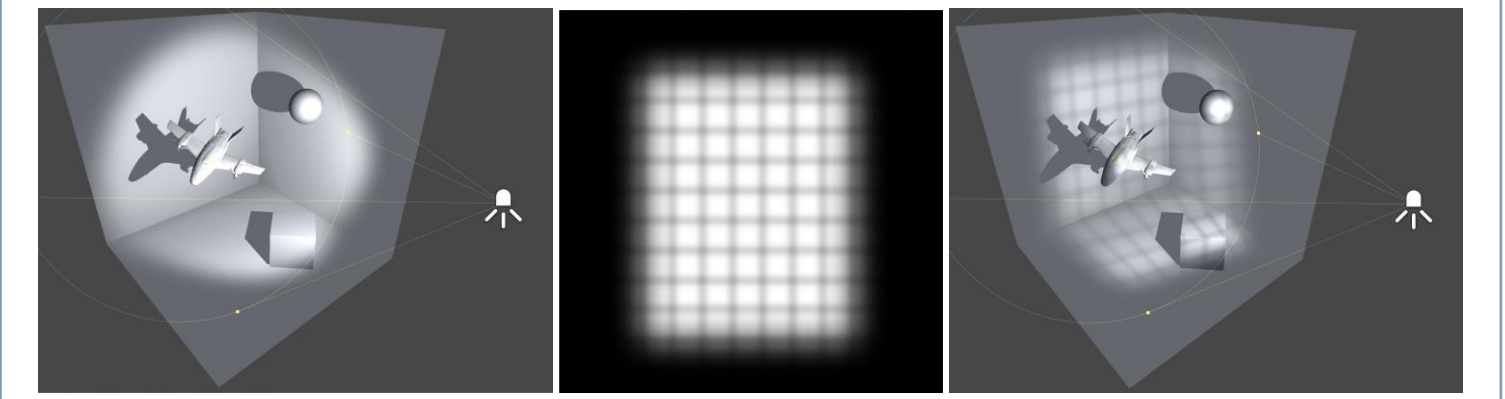
Figure 8.C

A spot light is what you would expect. Light comes from a point and creates a cone shape outwards. The range and size of this light cone can be changed in the settings.

Like a point light, a spot light can also be used to create lighting for lamps.

TIP: Spot lights can be used with  light cookies to create light shadows and patterns such as a flashlight, bars, window, and more. See **Figure 8.C.i** for an example.

Light Cookies



(Left) normal spot light, (Center) light cookie from standard assets, (Right) cookie applied to spot light .

Figure 8.C.i

An area light is a baked type of lighting. Again, this means that it must be precomputed on static GameObjects as a lightmap.

Area lights emit light in one direction from a plane that can be changed in size.

The light from area light is soft, like the way emission materials emit light due to the light coming from multiple different locations.

This can also be used to create more realistic lighting than point or spot lights in some cases.

Area




Scene with area lighting.

Figure 8.D

Another setting that is on all light sources is the *flare asset*. This creates a lens flare when the camera is looking at the light source. Unity's standard assets contain a few lens flares for you to use, or you can create or find your own.

Besides the light sources, there are other forms of light, such as emissive materials – as described before – and ambient light that is present all around the scene. Ambient light does not come from any specific source or GameObject.

Light sources are also capable of creating shadows if allowed to do so. In the mesh renderer component of GameObjects, there are options to enable and disable the casting of shadows and receiving shadows. The quality of the shadows can be changed within the light source component and also in the project settings. Navigating the menu bar going to **Edit > Project Settings > Quality**, you can edit the resolution of shadows (among other things). Reducing the shadow distance can help increase the quality of shadows' edges, with the drawback that shadows will be rendered within a smaller range from the camera.

There are also advanced lighting settings in the dedicated lighting window . You can access it from the menu bar by navigating to **Window > Rendering > Lighting Settings**. This window has settings for ambient lighting, baked lighting quality, fog effects, and more.



(Left) HDR image inspector, (right) cubemap skybox material inspector.

Figure 9

Skyboxes

A *skybox* is composed of multiple textures that wrap around the entire scene used to create a sky or distant background. Unity has a default skybox that has a dynamic sky that changes depending on the angle of the directional light in the scene.

There are several ways to create a skybox. A skybox can consist of 6 textures that are stitched together into a cube or a spherical image – normally a high-dynamic-range (HDR) image. The easiest way is to use a spherical HDR image. You can find these images online (refer to Prelab). When initially imported into Unity, the texture shape will default to 2D.

Open the Inspector window of the image and change the texture shape to cube **[Figure 9, #1]**. Apply the settings at the bottom of the Inspector. This will change the image into a cubemap that can be used for a skybox material.

Next, create a new material and change the shader to Skybox/Cubemap **[Figure 9, #2]**. This will create a skybox material where you can change certain settings, but most importantly use the HDR image as a skybox. Select or drag and drop the image into the Cubemap (HDR) field **[Figure 9, #3]**. You can drag around the preview screen inside the Inspector window to look at the skybox you have just created.

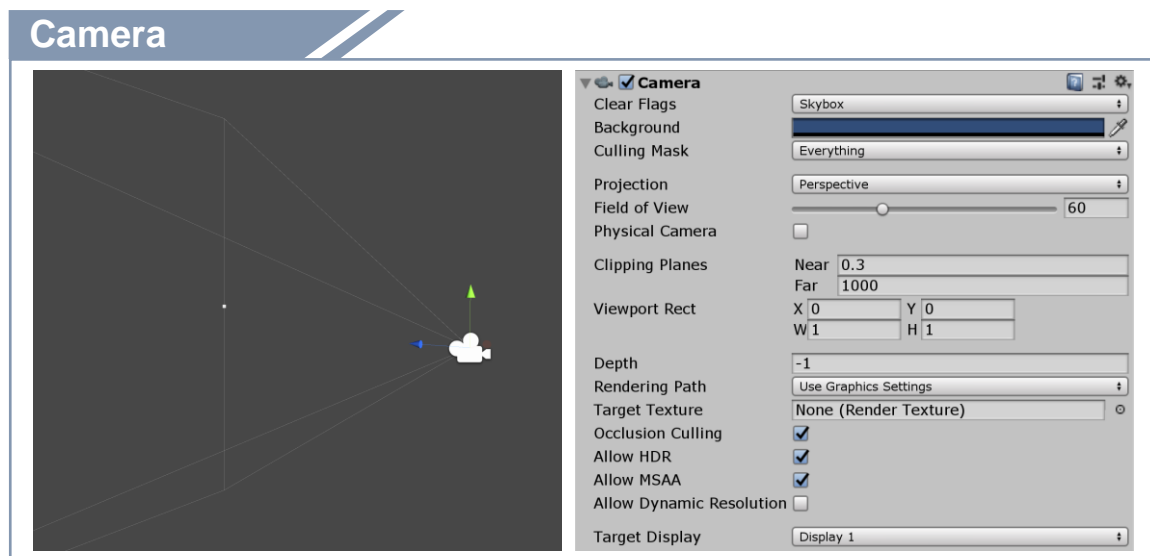
After that, you will be able to drag and drop the skybox material directly into your scene. Another way to change the skybox is to go into the advanced lighting settings window as briefly discussed before.

Another skybox that you can use is similar to the procedurally generated dynamic skybox Unity uses by default. Try creating another material and setting the shader to Skybox/Procedural. There, you can change the sky and ground color along with other parameters like sun strength and exposure.

Cameras

The only way for the player to view your scene when built into an actual game is by adding a camera. A camera captures and displays the world or scene to the player. There are many ways you can use and script cameras to show the player certain things. The most basic way to use a camera is to position it in a static location. This means that it will only show what is in its viewing cone.

When you created a new scene, there is a static camera placed in the scene by default. By clicking on the camera, you can see that it contains the camera component **[Figure 10, Right]** and in the scene window you can see the viewing cone **[Figure 10, Left]**.



(Left) camera viewing cone, (Right) camera component.

Figure 10

There are a few important settings we will discuss, briefly described in Table 2. More may be found in the Unity documentation.

Camera Settings

Culling Mask	Changes what layers are included or omitted in rendering.
Projection	Toggles perspective projection.
Field of View	Changes the angle of the viewing cone.
Clipping Planes	Changes the size of the viewing cone.

Table 2

It is possible you will never need to use these settings in your labs, but they are good to know.

Culling mask setting changes what layers will be rendered or not. Layers, as discussed before in the Prelab, are flags that you can put on individual GameObjects. For the culling mask, any layers that are selected will be rendered while any layers that are not selected, will not. This is typically used with multiple cameras.

Projection settings has two options: perspective and orthographic. By default, the camera renders the perspective projection which should be self-explanatory. Orthographic projection projects the image without depth or 3D perspective. It draws everything flat which is used for 2D or isometric games.

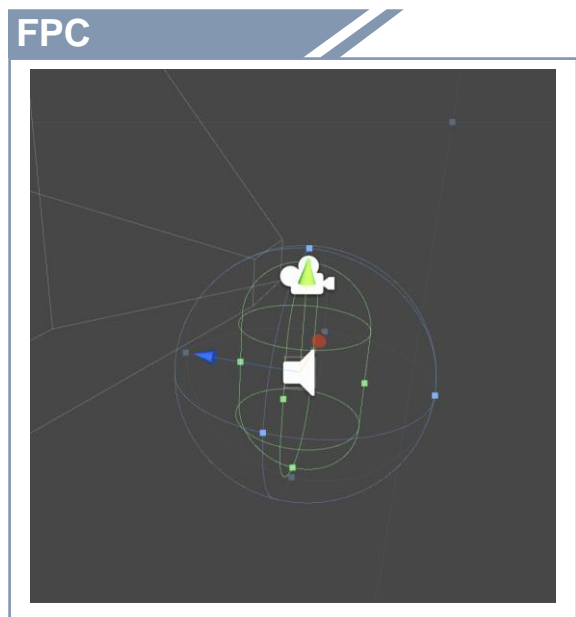
Field of view settings change the viewing angle. By default, the viewing angle is set to 60°. You can change the angle to a greater or lower angle depending on how you want to use the camera. This relates to the clipping planes settings which change the range of the view. The near setting changes where the camera starts viewing, while the far setting determines the where the camera ends viewing.

There are many more settings that you can experiment with, or read about, in the Unity Manual.

First-Person Character Controller (FPC)

The *first-person character* found in Unity's standard assets is very useful for the labs. It has a camera which is attached to a controllable rigidbody character controller. There are certain aspects that are not suitable for everything, so you can create and script your own FPC in the future if you so desire. You can find the FPC in the standard assets by navigating inside the asset folder or searching for FPSController in the search bar. This prefab is shown in Figure 11.

As you can see, there were two FPCs. It is recommended to use the one highlighted in Figure 11 rather than the other one. Click and drag the FPC into the scene. In order to walk around, you need a surface that has a collider. The rigidbody component of the FPC gives it physical properties so that it can detect collision and simulate gravity.



FPC as seen in scene view.

Figure 11

Try creating a plane GameObject which will provide a surface for you to walk on. Scale the plane 5 to 10 times in the X and Z directions (the Y direction will not be affected). Note that planes are one sided and are 2D. They only allow collision from the visible face, while the invisible face has no collision. You can also try adding obstacles such as with cubes, spheres, and walls.

If you click on the FPC, you can see what it consists of and you can explore how it works. It has a capsule collider for collisions, audio source for footstep sounds, and a child camera GameObject for where the “head” of the player should be. There are also scripts that are attached that allow input from the player which controls how the FPC moves. You can try changing the settings in the inspector window to customize the FPC to your needs.

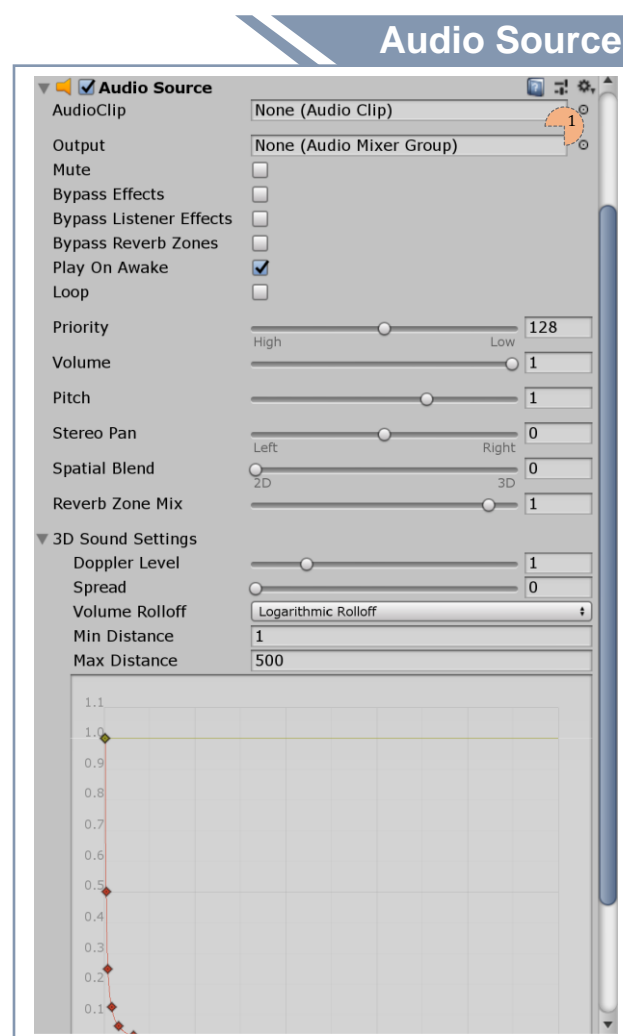
Audio Sources & Listeners ?

Audio sources produce sounds that increase the realism of a scene. Adding ambient noises such as crickets, wind, or water flowing are great ways to immerse the player. You can add an audio source component to any GameObject or create a new audio source GameObject from the create menu. To specify an audio file to play, go to the AudioClip input field [Figure 12, #1] and select an audio clip or drag and drop it in.

By default, “Play on Awake” is true, so when the scene is played, the clip will also play. This can be turned on or off and audio can be played with a script when something happens, such as an explosion. Other settings include loop, volume, pitch, and spatial blend.

Spatial blend specifies 2D or 3D sound. 2D sound means that the volume and direction of the sound is constant everywhere, while 3D sound allows the sound to have a “source” starting at where the audio source GameObject is. This means that audio will sound like it is coming from a certain direction and the volume will vary depending on how close the audio listener is.

? The audio listener is a component which allows audio in the scene to be “heard” (played to the player).



Audio source component.

Figure 12

Lab 1 Homework Assignment

Now that you have learned how to use different objects and components, you will create your very first scene. The scene will encompass a central theme. The theme you will need to design around is an “ominous” or “scary” street. You are free to use whatever assets you like and are not restricted. Here is an example scene:

Spooky Scene



A scene depicting an ominous and windy city street. A lone dark figure stands in the shadows... watching you.

Figure 13

Your scene does not need to be quite as complex as the one in **Figure 13**, but it gives you an idea of what the theme should be. Do not try to copy this scene, rather create something of your own inspired from it. An urban or city scene is the easiest, since the buildings are just cubes with textures on them. However, you can set the location of the scene anywhere – even in space! But you do need to meet the following deliverables:

1. Textured with use of normal maps (and optionally other maps as well to add detail);
2. A long street;
3. Buildings;
4. Light posts;
5. Skybox;
6. Scene should look dark and spooky.

For undergraduate students, a fixed camera is sufficient and is only expected to create whatever is in view.

For graduate students, a first-person controlled character must be used and because of this, the scene needs to have a traversable environment and 360° of scenery.

References & Additional Resources

1. Intro to Components
 - a. <https://docs.unity3d.com/Manual/Components.html>
2. Materials, Shaders, & Textures
 - a. <https://docs.unity3d.com/Manual/Shaders.html>
 - b. <https://docs.unity3d.com/Manual/shader-StandardShader.html>
 - c. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameters.html>
 - d. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameterAlbedoColor.html>
 - e. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html>
 - f. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html>
 - g. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameterOcclusionMap.html>
 - h. <https://docs.unity3d.com/Manual/StandardShaderMaterialParameterDetail.html>
3. Static GameObjects
 - a. <https://docs.unity3d.com/Manual/StaticObjects.html>
4. Lighting
 - a. <https://docs.unity3d.com/Manual/LightingOverview.html>
 - b. <https://docs.unity3d.com/Manual/class-Light.html>
 - c. <https://docs.unity3d.com/Manual/Lighting.html>
 - d. <https://docs.unity3d.com/Manual/Cookies.html>
 - e. <https://docs.unity3d.com/Manual/GlobalIllumination.html>
5. Skyboxes
 - a. <https://docs.unity3d.com/Manual/class-Skybox.html>
6. Cameras
 - a. <https://docs.unity3d.com/Manual/class-Camera.html>
7. Audio Source & Audio Listener
 - a. <https://docs.unity3d.com/Manual/class-AudioSource.html>
 - b. <https://docs.unity3d.com/Manual/class-AudioListener.html>