COMP1150/MMCC1011 Week 10 Prac

Topics covered:

- Direct and Ambient lighting
- Indirect lighting
- Baked Lightmaps
- Dynamic Indirect Lighting with Light Probes
- Reflections with Reflection Probes
- Emissive materials

Check out your project

Check out your COMP1150-3D-Pracs project from GitHub and open it through Unity Hub. Create a new scene **(File > New Scene)**. You can rename/save the scene as ‘Lighting’.

Lighting

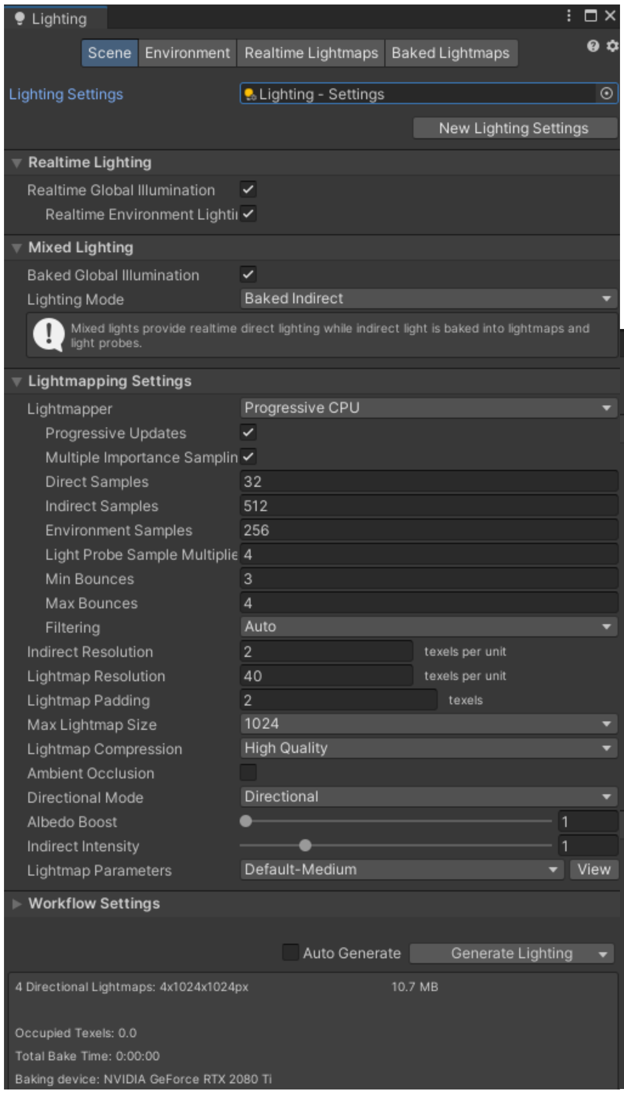
Unity has a variety of options to add different lighting effects to your games. Lighting can make your games feel much more real, but it is expensive to compute and can make performance suffer significantly. Doing lighting well is a matter of balancing quality and performance.

Unity starts with a number of different lighting effects automatically activated. To better appreciate what they all do, we will start by turning everything off, and reactivating them one by one.

Do the following in your newly created scene:

1. Delete the **Directional Light** in the Hierarchy panel.
2. Open the Lighting window **(Window > Rendering > Lighting Settings)**.
   Note: you may want to dock the lighting window next to your inspector, since we’ll return to it frequently in this prac.
3. Under **Environment**:
   a. Set **Skybox Material** to None
   b. Under **Environment Lighting**, set **Source** to **Color**.
   c. Set the **Ambient Color** to black
4. In the **Scene** tab, click the **New Lighting Settings.** This file contains all the lighting data for your scene. Give it an appropriate name like “Lighting Prac Settings”.
5. Under **Realtime Lighting**, check the **Realtime Global Illumination** checkbox
6. In **Lightmapping Settings**:
   a. Set **Indirect Intensity** to **0**
7. Untick **Auto Generate** Lighting (unless you have a powerful computer).

This creates a scene with no lights in it.


Lighting

Scene | Environment | Realtime Lightmaps | Baked Lightmaps

Lighting Settings: Lighting - Settings

New Lighting Settings

Realtime Lighting
- Realtime Global Illumination: ✓
- Realtime Environment Lighti: ✓

Mixed Lighting
- Baked Global Illumination: ✓
- Lighting Mode: Baked Indirect

Mixed lights provide realtime direct lighting while indirect light is baked into lightmaps and light probes.

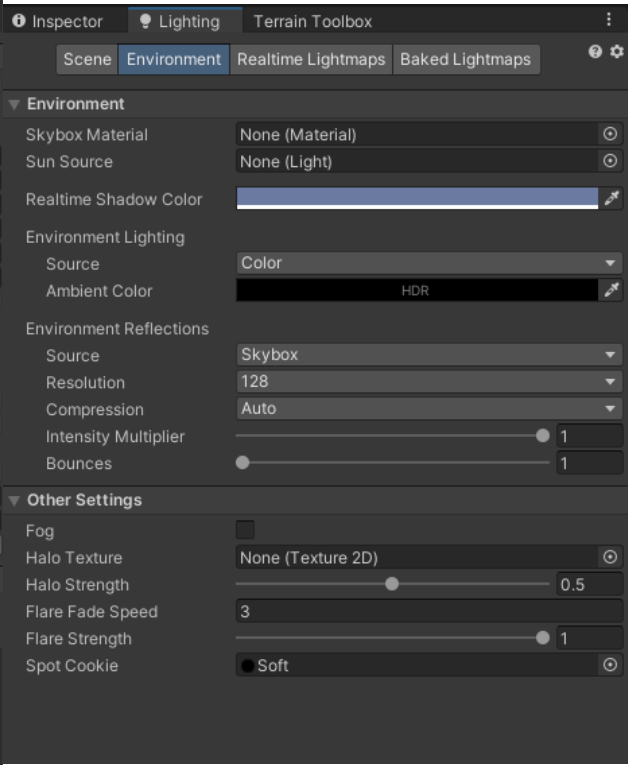
Lightmapping Settings
- Lightmapper: Progressive CPU
- Progressive Updates: ✓
- Multiple Importance Samplin: ✓
- Direct Samples: 32
- Indirect Samples: 512
- Environment Samples: 256
- Light Probe Sample Multiplie: 4
- Min Bounces: 3
- Max Bounces: 4
- Filtering: Auto
- Indirect Resolution: 2 texels per unit
- Lightmap Resolution: 40 texels per unit
- Lightmap Padding: 2 texels
- Max Lightmap Size: 1024
- Lightmap Compression: High Quality
- Ambient Occlusion: ☐
- Directional Mode: Directional
- Albedo Boost: 1
- Indirect Intensity: 1
- Lightmap Parameters: Default-Medium (View)

Workflow Settings

☐ Auto Generate | Generate Lighting

4 Directional Lightmaps: 4x1024x1024px 10.7 MB

Occupied Texels: 0.0
Total Bake Time: 0:00:00
Baking device: NVIDIA GeForce RTX 2080 Ti


Inspector | Lighting | Terrain Toolbox

Scene | Environment | Realtime Lightmaps | Baked Lightmaps

Environment
- Skybox Material: None (Material)
- Sun Source: None (Light)
- Realtime Shadow Color
- Environment Lighting
  - Source: Color
  - Ambient Color: HDR
- Environment Reflections
  - Source: Skybox
  - Resolution: 128
  - Compression: Auto
  - Intensity Multiplier: 1
  - Bounces: 1

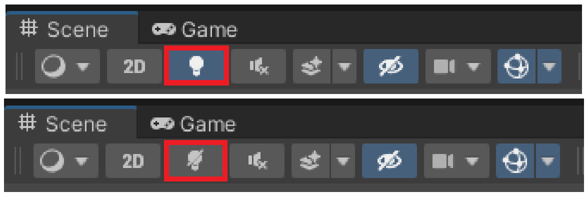
Other Settings
- Fog: ☐
- Halo Texture: None (Texture 2D)
- Halo Strength: 0.5
- Flare Fade Speed: 3
- Flare Strength: 1
- Spot Cookie: Soft

Note: As indicated above, you should only leave the **Auto Generate** checkbox (at the bottom of the example above, next to **Generate Lighting**) ticked if you believe your computer is powerful enough that the wait for Unity to recompute baked lighting will be very short. If you are on a laptop/weak computer, untick this checkbox (to improve performance and prevent automatic light-baking every time you make a change in the scene, as discussed in previous pracs). **Remember**, if the checkbox is unticked then you must press the **Generate Lighting** button every time you want Unity to recompute lighting (e.g. after moving or adding new objects to the scene).

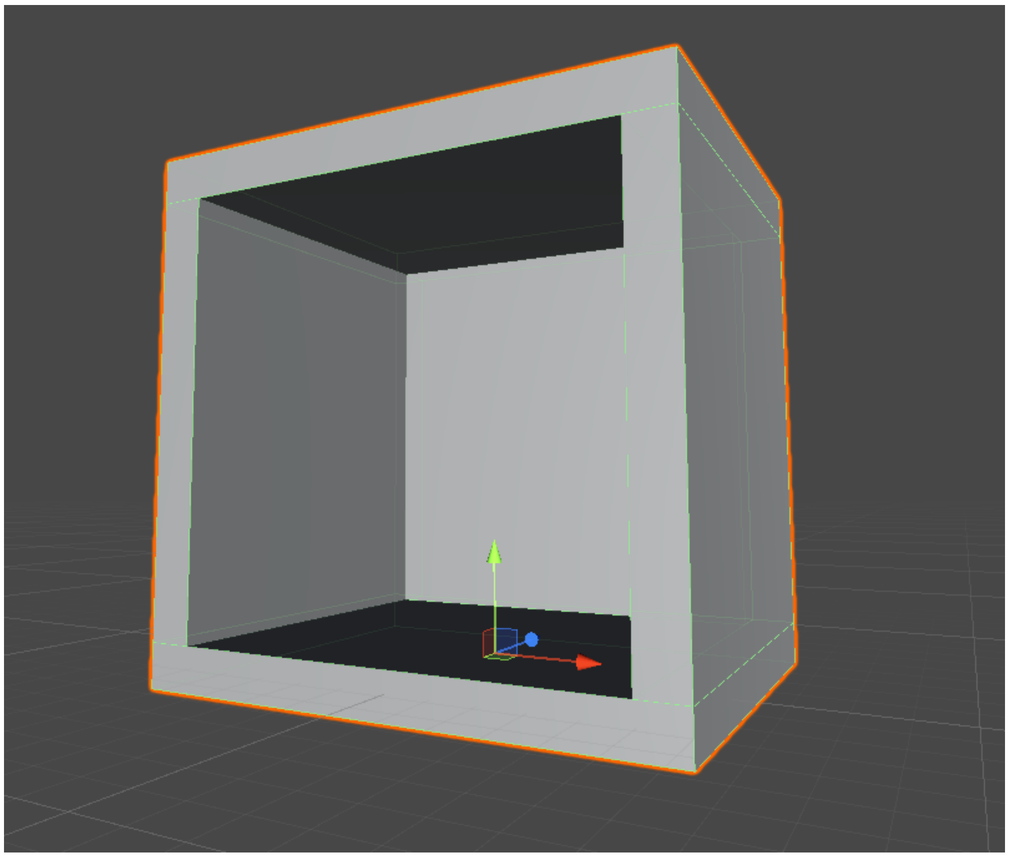
Conversely, if you are in the labs or on a powerful computer, you can switch the **Lightmapper** setting to **Progressive GPU** (since this should help speed up calculations).

By default the Scene view uses the in-game lighting to display the scene. This will make things difficult with all the lighting turned off. Fortunately, there is an option to disable lighting in the

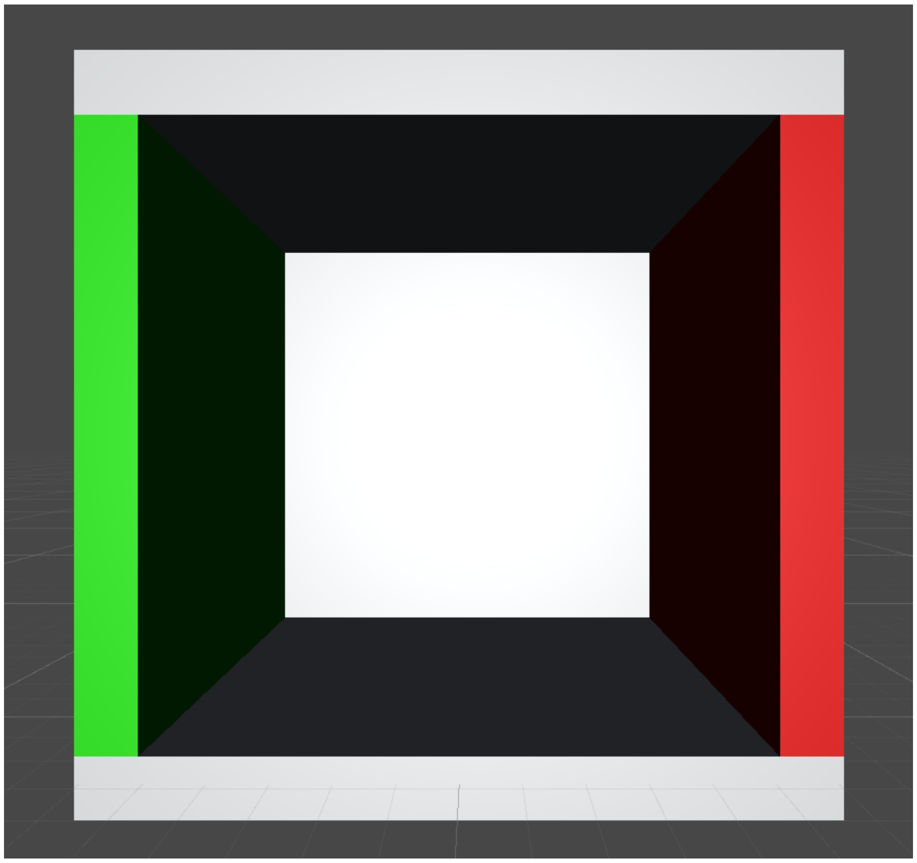
Scene view. **Click the light bulb button** at the top of the Scene window (between the 2D and the mute buttons) to toggle scene lighting on and off.



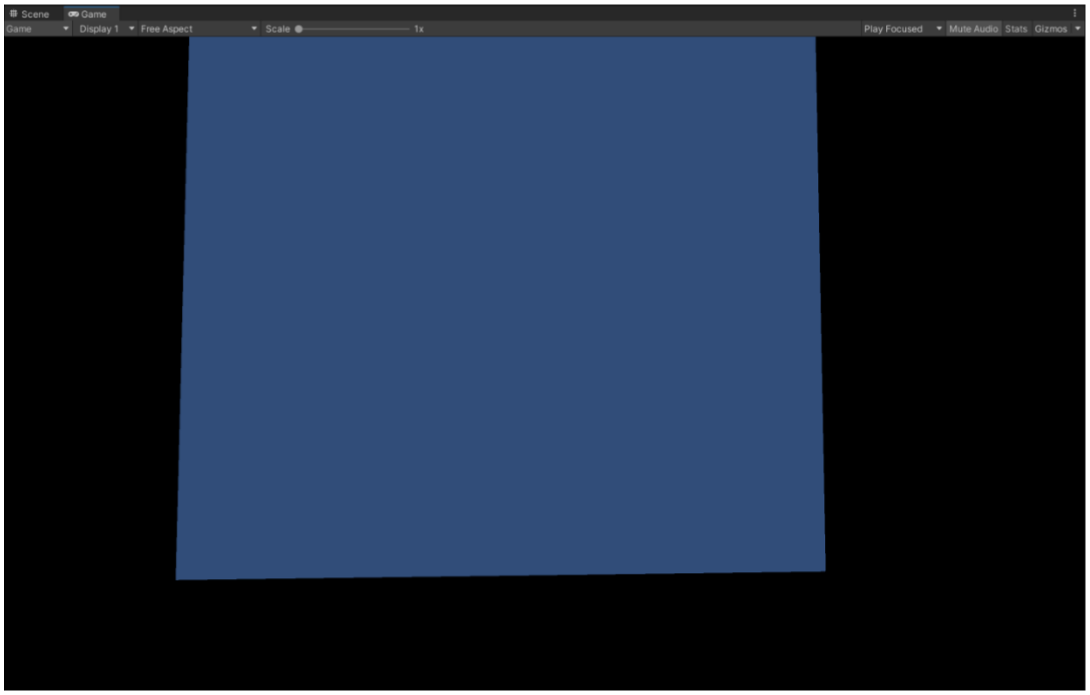
Now, let's create a lightbox. In your new scene, **create a large open box** by creating five (5) 10x10x1 cubes (using Unity Primitives, not ProBuilder) to act as the floor, ceiling and three walls (left, right and back). Leave one side open. It should look something like the below image (note, here the ceiling and floor have one direction scaled to 12 rather 10 to cap off the top and bottom).



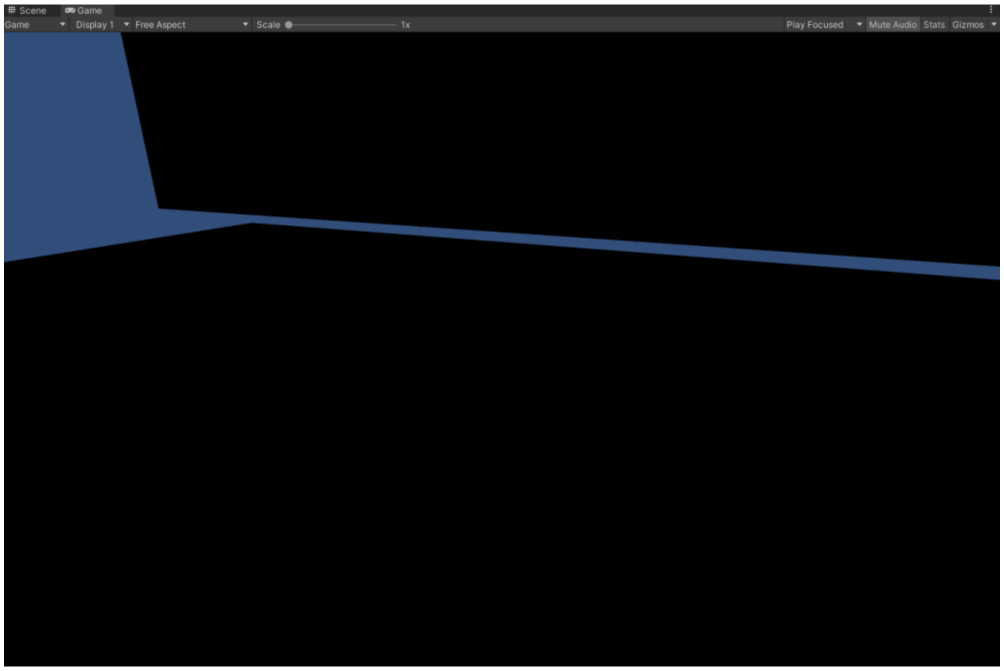
Add a new default (white) material to the back wall, ceiling, and floor. **Add a green material** to the left wall, and a **red material** to the right wall. Make sure you give these materials appropriate names, and remember that even when we leave a surface white, we must still add a material to it.



Delete the **Main Camera** and put the **Player** prefab in the scene, inside the box. **Play** the game. You should find yourself in a dark room with one side open the to the blue sky.

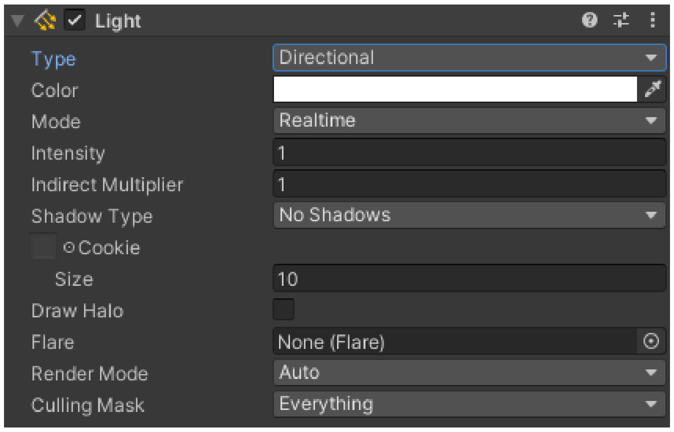


If you notice any cracks of light in the corners, then you probably haven't got your walls lined up perfectly. Seal any gaps before proceeding.



Add a light

Create an empty object called "Light". Place it inside the 'room' and then add a [Light]() component to it **(Add Component > Rendering > Light).** By default, Unity creates a Point light, but we will start with directional lights. In the Light component, change **Type** from 'Point' to **Directional**.



Light

Type	Directional
Color	
Mode	Realtime
Intensity	1
Indirect Multiplier	1
Shadow Type	No Shadows
Cookie	
Size	10
Draw Halo	
Flare	None (Flare)
Render Mode	Auto
Culling Mask	Everything

Turn the scene lighting button back on (the light bulb between the 2D and mute buttons at the top of your scene panel) so you can see the effects of lighting. **Try the following**:

- Move the light (this should have no effect)
- Rotate the light in the scene and see how it changes the apparent colour of the walls

- Adjust the **Color** and **Intensity** of the light and see what effect this has.
- Set **Shadow Type** to **Hard Shadows** and rotate the light some more
- If Auto-Generate lighting is turned off, then re-bake the lighting (by pressing the **Generate Lighting** button in the Lighting window).

Unity, by default, supports four main types of light: 'Directional', 'Point', 'Spot' and 'Area'. Unity also has built in functionality for other types/lighting systems, such as Volumetric Lighting, but we won't be covering those in this unit.

> A 'directional' light represents a distant light source like the Sun, which casts all light in the same direction. Moving the source doesn't affect anything, as it is assumed to be very far away. In particular, the light will still cast shadows even if it is inside the box.

Now change the light's **Type** to **Point** and **repeat the experiment**.

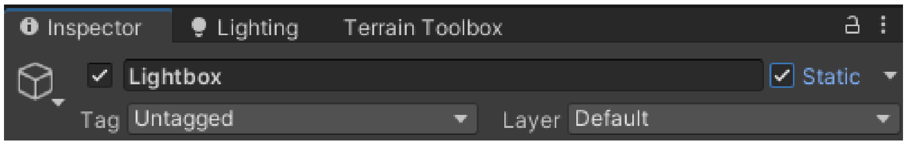
> 'Point' lights shine in all directions from a fixed point in the world, like a bare lightbulb. Rotating a point light has no effect, but moving it does. Point lights also have a 'Range' setting which determines how far the light spreads. Objects beyond this range will not be lit.

Change the **Type** to **Spot** and **experiment some more**.

> 'Spot' lights cast a cone of light in a particular direction, like an electric torch. Both the position and rotation of a spot light matters. Like a point light, they have a range, but they also have a Spot Angle which determines how wide the beam is.

A flashlight is a nice example of a spotlight. To make one, in the Hierarchy, attach the Light to your **Player** prefab. In its **Transform**, set both **Position** and **Rotation** to (0,0,0). **Play** the game and you will have a spotlight that points in the direction of the camera.

The fourth type of lights, 'Area' lights, are very resource intensive, and work by illuminating an object from many different angles by creating lights from its edges. As they are so expensive, they will only render if you rebake the lighting. They will also only work on 'Static' objects, so if you want to see the area light then you'll need to set all the walls of your lightbox to 'Static' in the inspector. Given can take a while to set up and bake, we won't be encouraging you to experiment with them today (or to implement them in your Game Design Task).



Lightbox
✓ Static
Tag Untagged
Layer Default

Reminder: Save, Commit and Push to GitHub

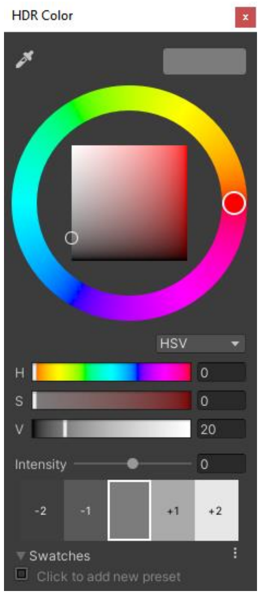
Ambient Light

At the moment, all the shadows in our scene are completely black. If there is no light shining directly on an object then it cannot be seen. This is what we call **direct lighting**. However, in the real world, light bounces in all directions. The light from the Sun or from a bulb bounces from one object to another. So even objects in shadow can still be seen. This is called **indirect lighting**, which we'll cover soon.

First, as a simple and cheap implementation of something between these two types of lighting, ambient lighting fills the space with uniform light in all directions, making everything a little lighter. For some games (e.g. games with a cartoon aesthetic), this might actually be better for the art style.

To set up ambient lighting, in the Lighting panel underneath 'Environment Lighting', change the **brightness** (or **V / Value** if set to HSV) of the **Ambient Color** (see image below). You should notice that the whole scene gets lighter. You can also adjust the **Hue**, **Saturation,** and **Intensity** to give everything a coloured tinge.

Notice that the ambient light does not cast any shadows or create gradients. It is uniform everywhere, not taking into account the shape of your world, and so looks fairly unrealistic. For this reason, it should not be your primary source of light. You generally only want a little ambient light in your scene (assuming you want everything to be at least slightly visible to the player).


HDR Color

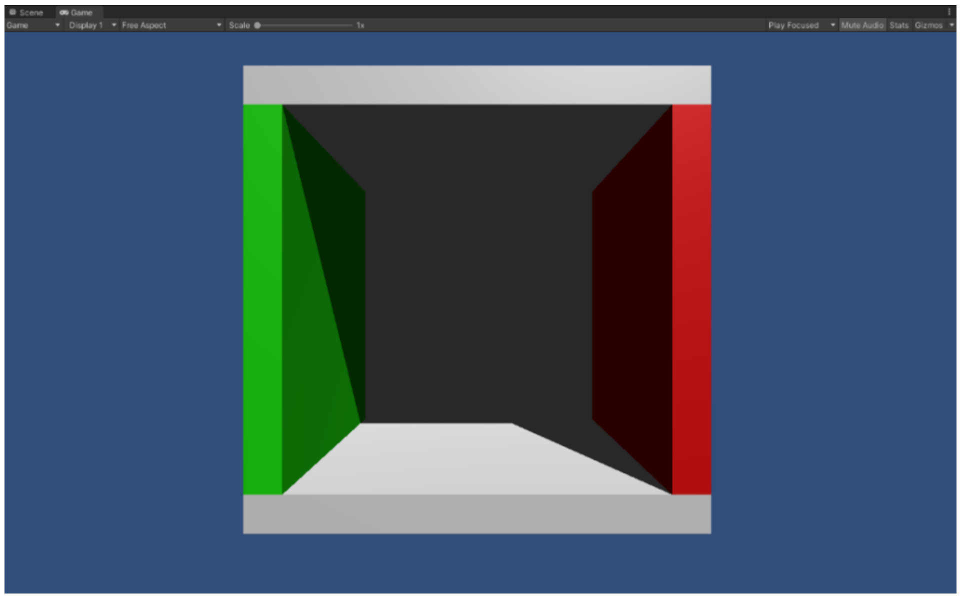
HSV

H 0

S 0

V 20

Intensity 0

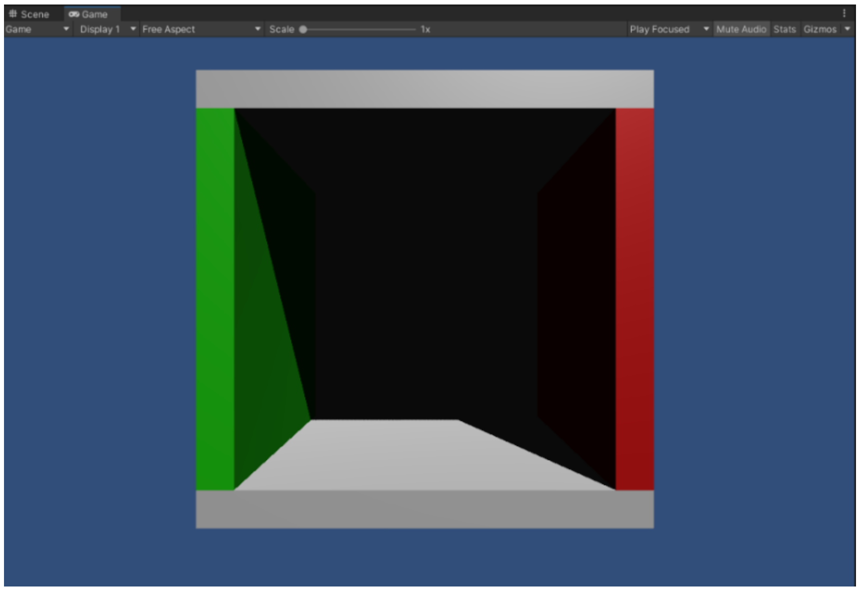


To reset your scene for the next section of the prac (as shown above):

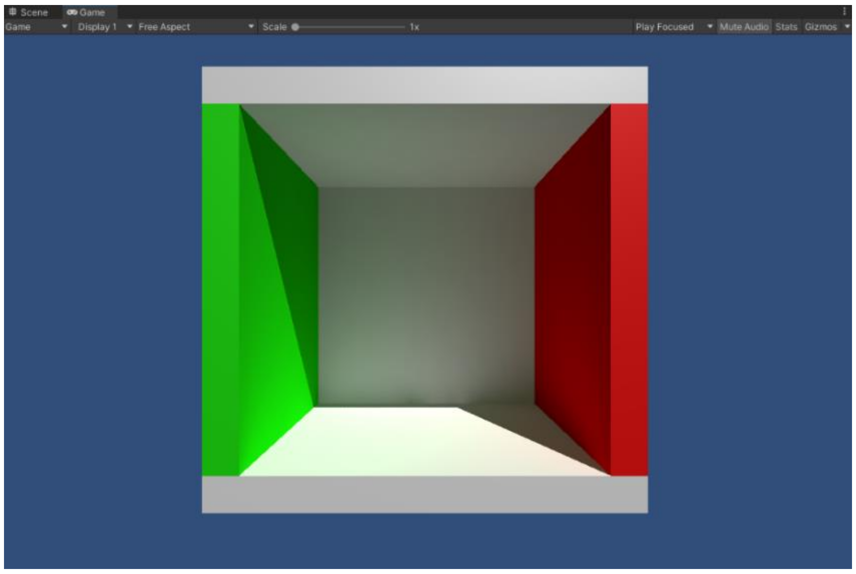
- Leave your ambient light settings on a **dark grey** setting (e.g. V / Value of 20).
- Make sure your light object is set as a directional light, with **Shadow Type** set to either **Hard** or **Soft Shadows**
- **Delete your Player prefab** and **replace it with a regular camera** pointed at your lightbox.

Indirect Lighting

So far we have learned that ‘Direct light’ is light that comes directly from a source. We found that direct lighting on its own left the unlit parts of the world unnaturally black.



We learned to add ambient lighting, but it is uniform. In real life, light bounces off one object onto another. Objects that are not directly facing a light source are still visible because light bounces off other nearby objects onto them. This is called ‘indirect lighting’. Using indirect lighting thus leads to a more natural looking image, as shown below.



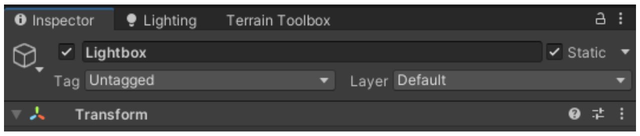
The shadows now vary in darkness depending on how much indirect light can reach each corner of the room. You should also see that some of the colour from the red and green walls is reflected onto the neighbouring white surfaces.

There are other advanced techniques to make lighting and scenes look even more natural, like volumetric lighting and raycast lighting/shadows, which are gaining popularity with the release of more advanced hardware, but these techniques are beyond the scope of this unit (and likely many of your computers!). Some of these techniques simulate indirect lighting mathematically, but since this takes more computational effort than most PC's can currently handle, a variety of different tricks are used to make indirect lighting more accessible.

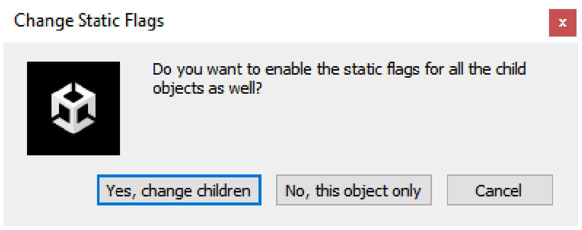
The first trick is to do as much of the computation as possible ahead of time. If we can calculate the lighting while we're making the game (rather than when the game is being played) then we can save it and re-use this later. This is called 'lightmapping'. It is a very useful technique, but it only works if we know the geometry of the world will not change. As such, lightmapping works for stationary objects like walls and fixed furniture but not for objects that move around the scene.

The second trick is to use the static lightmaps to approximate the lighting on a dynamic object by averaging the light values at fixed points in its vicinity. These points are called 'light-probes'. Using light probes does not look as good as doing the lighting calculation properly, but is much faster/cheaper and gives acceptable results at realistic speeds.

To add indirect lighting, we need to tell Unity that the walls are all static, i.e. they do not move. You may have already set your walls/cubes to static to test area lighting; regardless, make a **new empty game object** and call it "Lightbox". Make this object the **parent of all the cubes** that make up your lightbox. Finally, tick the **Static** checkbox at the top-right of the Inspector window.



Unity will prompt you if you want to make the change to all the children as well. Agree by selecting **Yes, change children**.

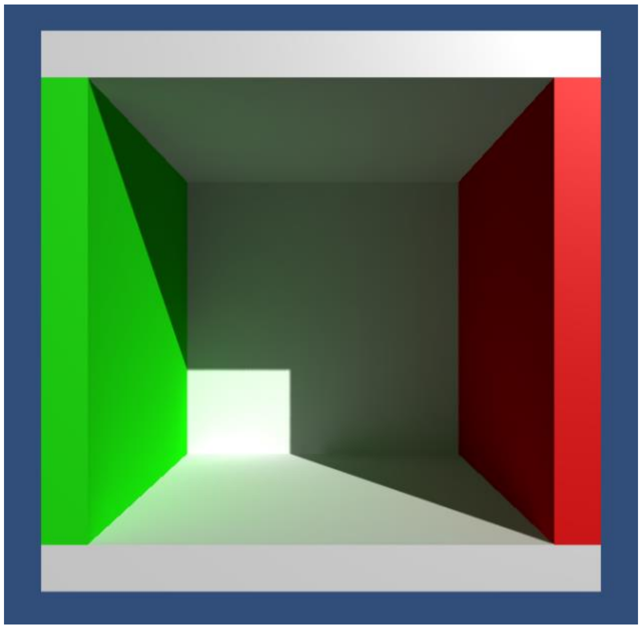

Change Static Flags

Do you want to enable the static flags for all the child objects as well?

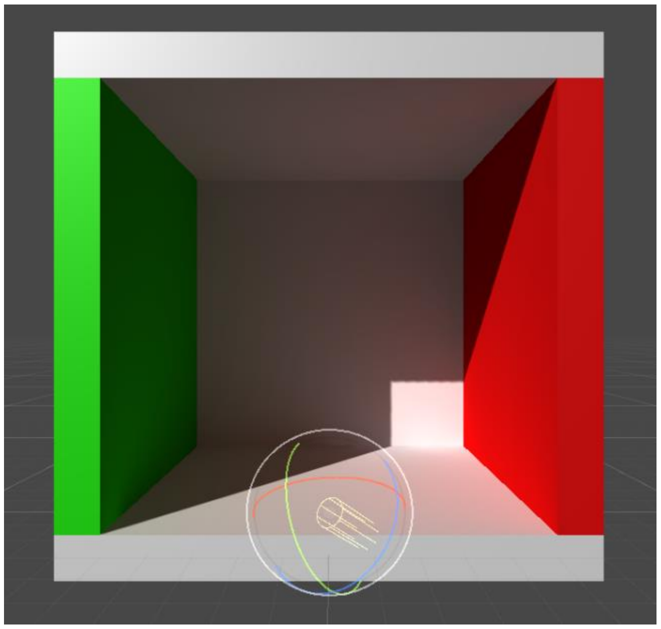
This is another reason why grouping your objects is a good idea. All your static game-architecture should be grouped under a single parent object so you can enable or disable this setting in one go, rather than having to change every object individually.

Once you have made all your walls static, you can turn up the ‘Indirect Intensity’ (**Lighting** panel **Lightmapping Settings > Indirect Intensity**). Try adjusting this value to see how it changes the intensity of the indirect lighting in the scene.

Note: don’t forget to rebake by pressing **Generate Lighting** to see the effect after each change for the rest of the prac (if Auto Generate is unchecked). You may also be prompted to save your scene before lighting generates.

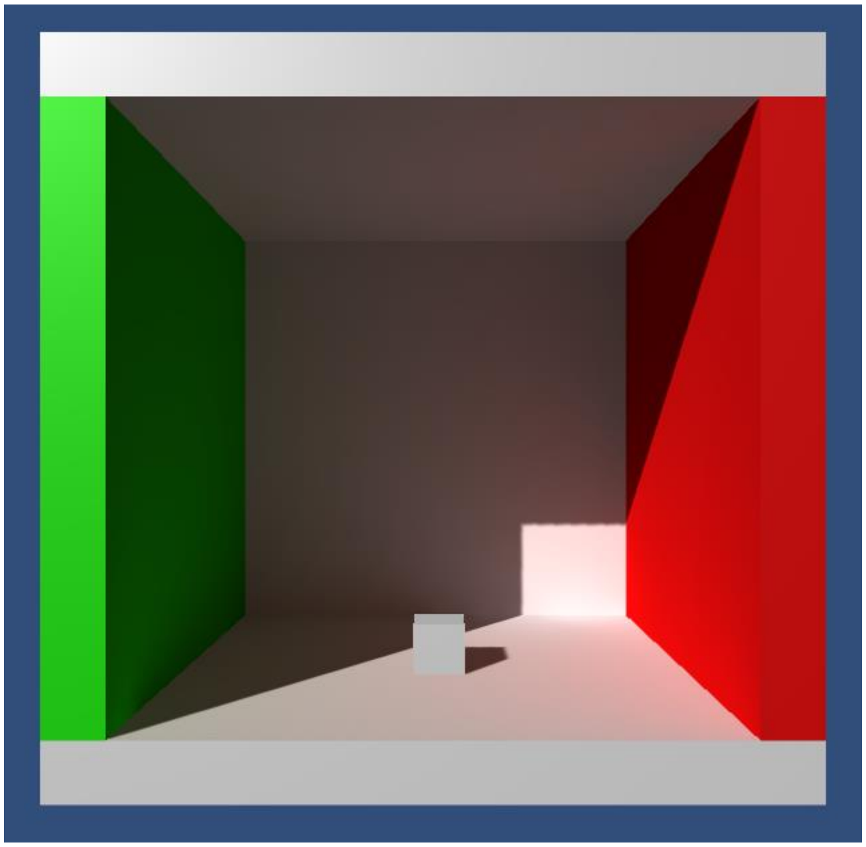


Rotate the directional light to see how this affects the lighting. You should notice that the indirect light changes colour depending on which wall is brightly lit.



Dynamic Indirect Lighting

Add a cube to your scene and **apply the same material** as you have on your white walls.

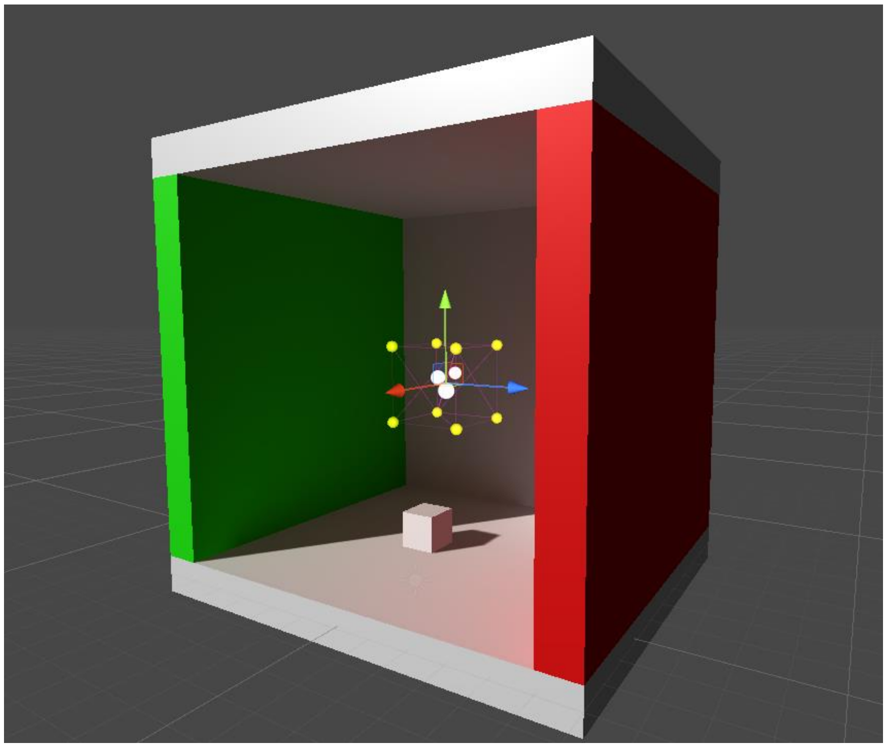


Add an animation to the cube to make it move around the room. You should see that it responds correctly to direct lighting (e.g. moving in and out of shadow) but doesn't respond to indirect lighting. This is because it is not static, unlike the walls of our lightbox, so the lightmapping has not been pre-computed for it.

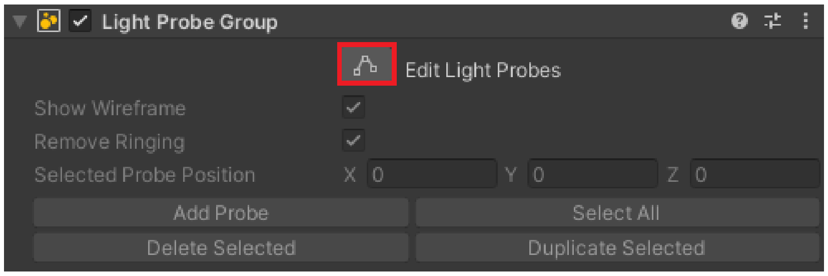
Set the cube to static by ticking its **Static** checkbox. When you play your game scene you will notice that the animation stops working properly because Unity will not allow you to move static objects.

To make dynamic (non-static/movable) objects respond to indirect lighting, we need to use 'light probes'. A light probe is like an invisible sensor that can be placed at a specific position in the world. When lighting calculations are measured during a bake the probe stores the relevant lighting information for its position (including for indirect lighting). During gameplay, the game engine can then approximate the light on a dynamic object by averaging the lighting conditions from all the nearby probes through a 'light probe group'.

To implement this in your scene, select **GameObject > Light > Light Probe Group** from the menu. You should see a network of light probes (yellow circles) appear in the Scene view.



Rearrange the probes by pressing the **Edit Light Probes** button so that there is one in each corner of the room. You can either do this by hand in the scene view or by changing their positions in the Inspector.



Light Probe Group

Edit Light Probes

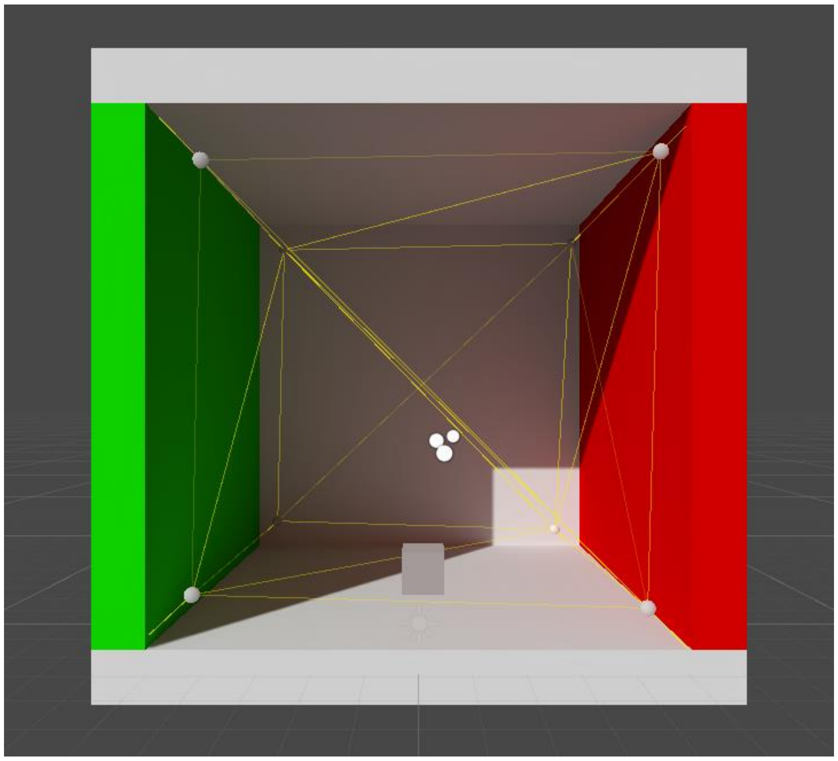
Show Wireframe ✓

Remove Ringing ✓

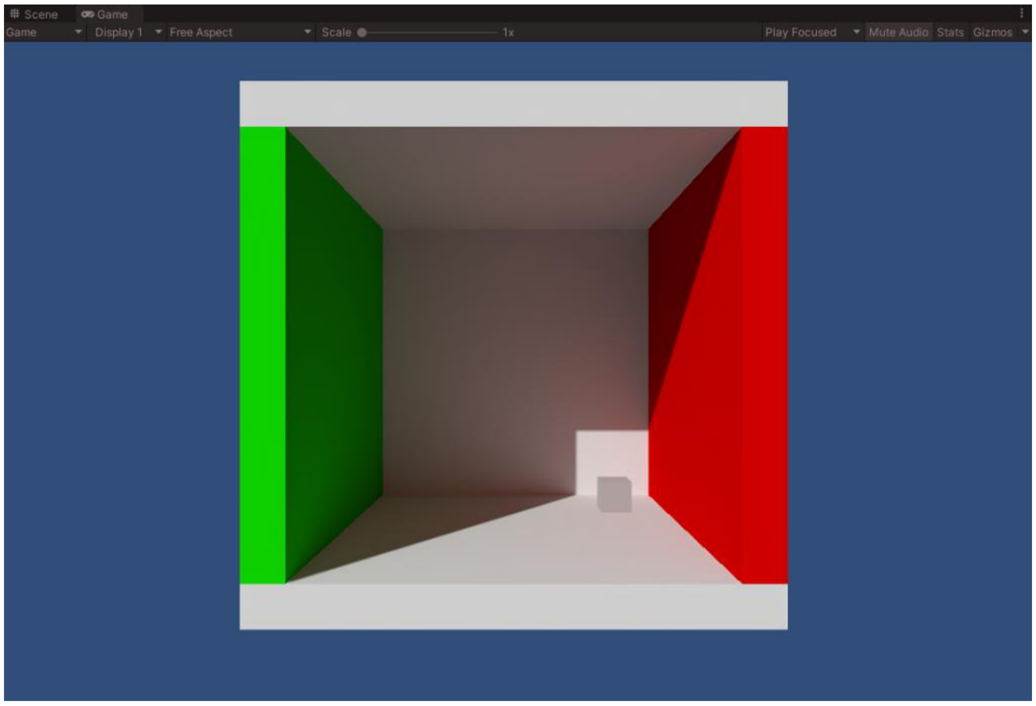
Selected Probe Position X 0 Y 0 Z 0

You generally want to use as few probes as possible, while also making sure all the different lighting conditions are properly sampled. More probes will result in more accurate lighting but will take much longer to compute. For this room, eight probes (one in each corner) will be plenty.

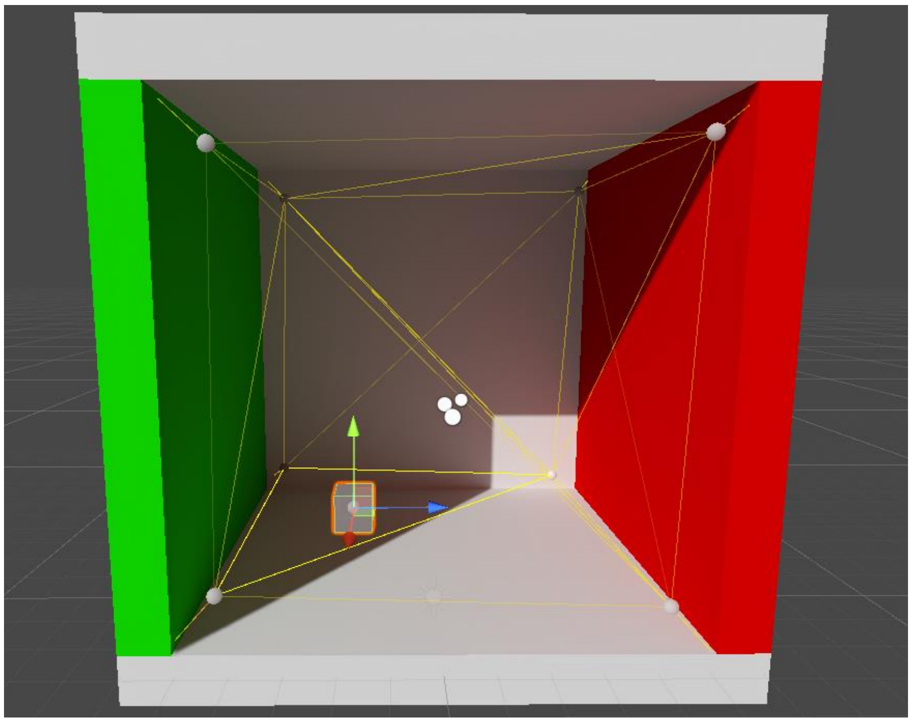
Once this is done, set your Directional Light to **Baked**, and in your Lighting panel, under the Scene tab set **Lighting Mode** to **Baked Indirect**. Your lighting probes should change colour (based on the light around them), and have yellow lines (cells) between each probe (reminder: don’t forget to hit the generate lighting button again if auto-generate is off). By default, if you have red cells it means they are “invalid”, and should go back and tweak the position of the probes. When your probes and settings are correctly changed, your Scene view should look something like this.



Play your game. You should notice immediately that the lighting on the cube is much more realistic, and it should respond appropriately to different lighting conditions as it moves around the room.

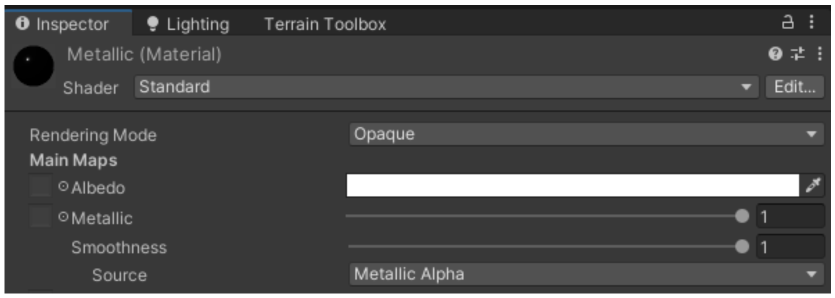


If you select the cube in the Scene view and move it around, Unity will show you which of the light probes are being used to light it at each position by highlighting the adjacent lines used for the current area the object is in.



Reflective Objects

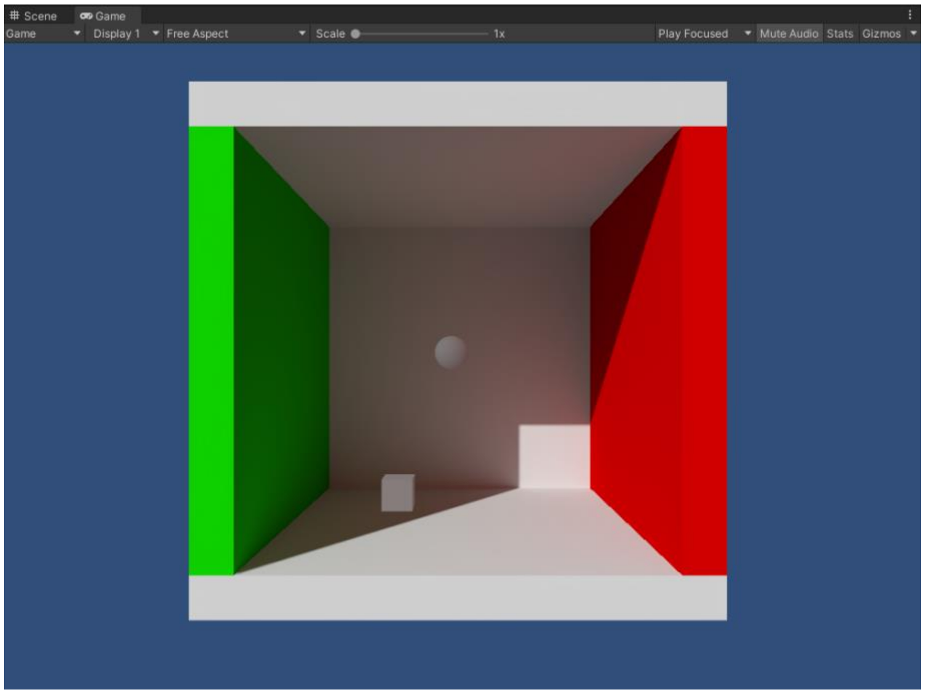
In the standard Material there is an option for how Metallic the material should be:



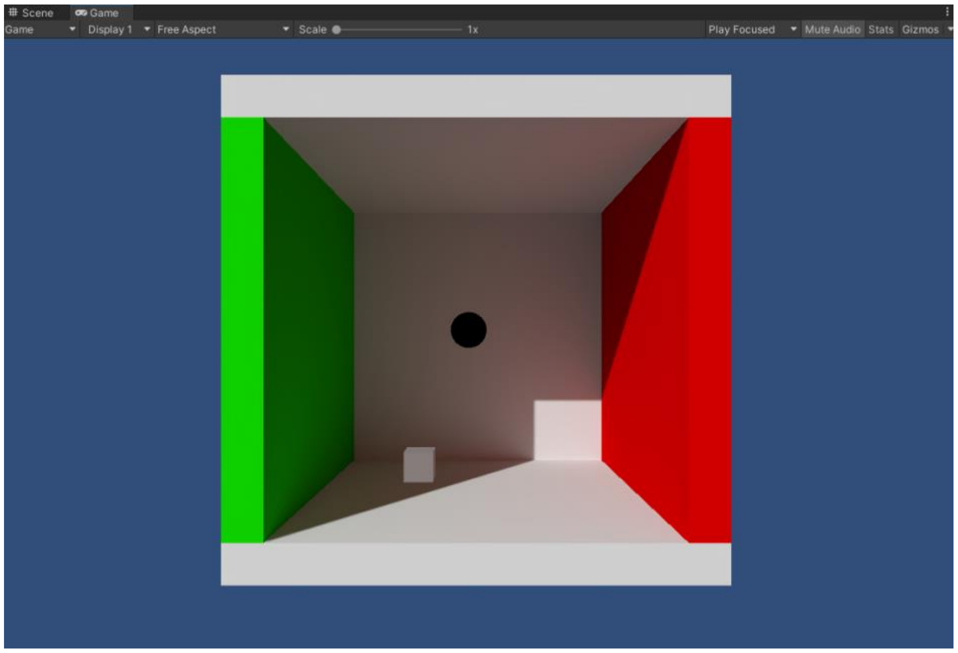
Metallic (Material)
Shader Standard
Rendering Mode Opaque
Main Maps
Albedo
Metallic 1
Smoothness 1
Source Metallic Alpha

Metallic objects are more reflective, but in order to compute reflections we need to do a bit of extra work. Again, we face the problem that calculating physically correct reflections is a computationally intensive process and cannot typically be done in real time. However, there are a number of cheaper approximate solutions which will often look good enough.

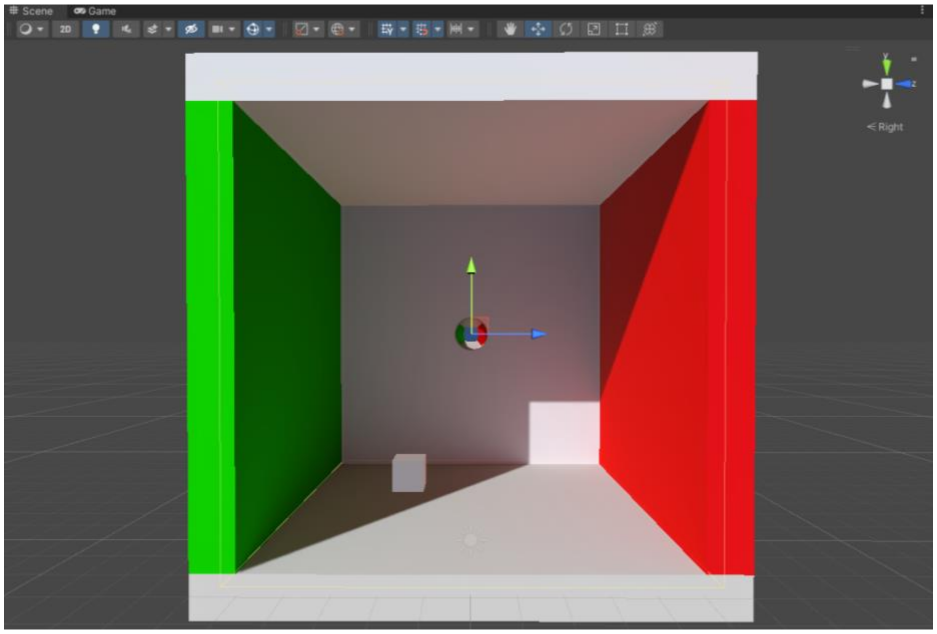
Create a Sphere and place it in the middle of the room. Make it larger than the one shown below (so that you'll easily be able to see reflections in it).



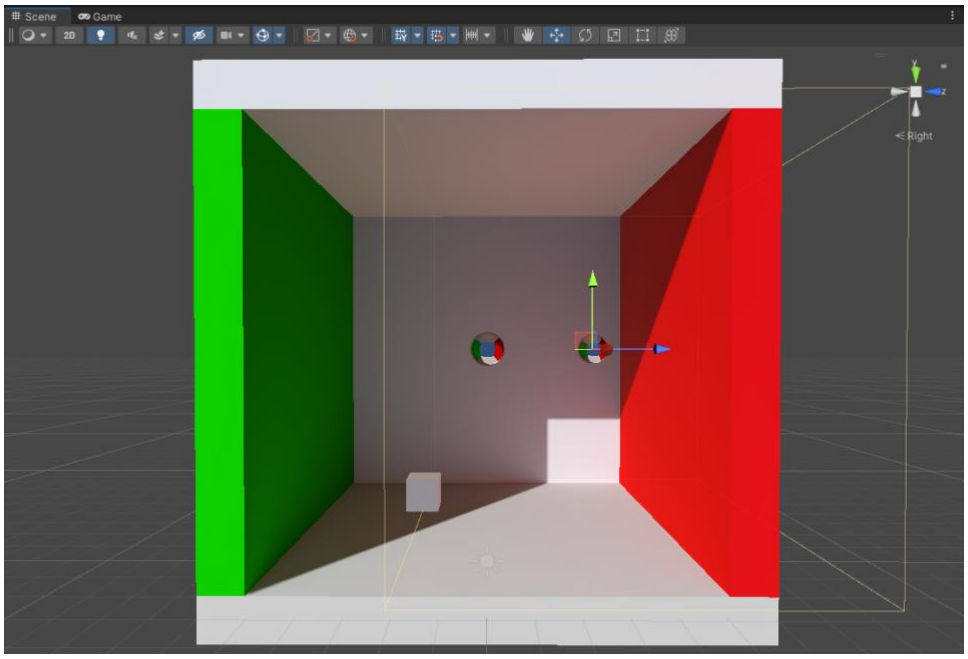
Now create a new material for the sphere and set its **Metallic** and **Smoothness** properties both to **1**. The sphere should turn black.



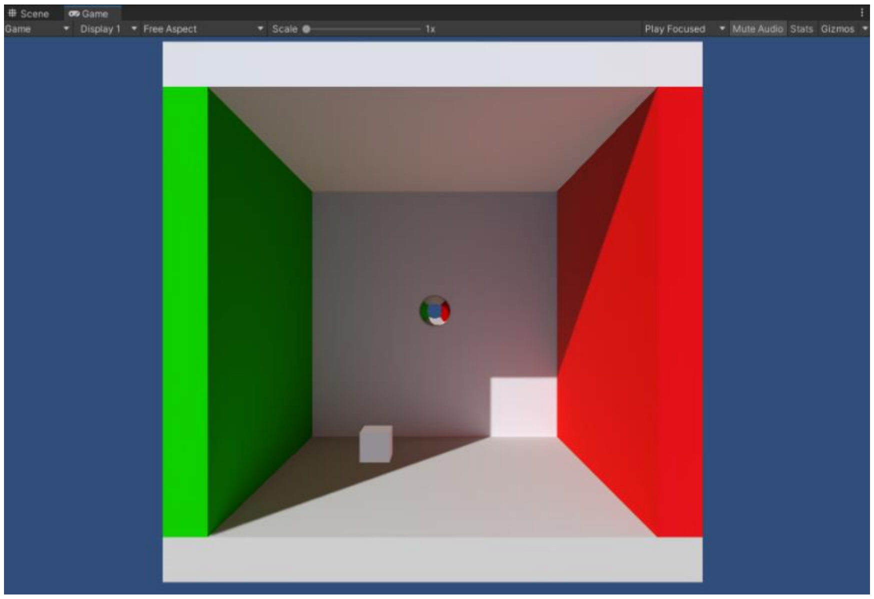
The sphere is trying to show a reflection of the room, but it doesn't know what that should look like. To create a reflection, we need to add a 'reflection probe'. Select **Game Object > Light > Reflection Probe**. Position the new object in the centre of the room, and in the reflection probe component, press the **Bake** button. Your sphere should now reflect the surrounding environment.



If you move the probe around, you will see that it has a box around it. The reflection probe only applies to objects inside this box.



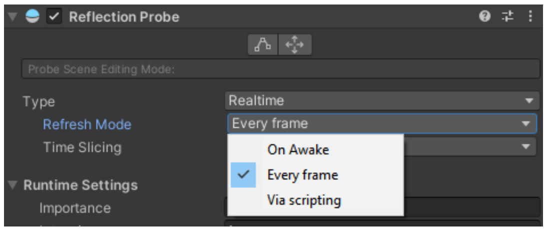
A reflection probe is like an omnidirectional camera sitting at a point in the world. Reflective objects in the probe's space use that camera to work out what reflections to show. Notice how the sphere now shows a reflection of the room on its surface (after you re-generate the lighting).



As you may have noticed before, by default the reflection probe’s Type is set to **Baked**. This means that the reflections are being computed in the editor once, and saved as textures. This is efficient because it means that the reflection calculations only have to be done once, but it also means that only static objects are shown in the reflection. If you play your game now, you should notice that the moving cube is not reflected in the sphere.

If you have enough computing power, you can change the **Type** of your reflection probe to **Realtime**. Realtime probes are recomputed while the game is running, and so they can reflect non-static objects.

By default, realtime probes still only recompute reflections when the scene is reloaded, to have your reflections update dynamically, you need to change the **Refresh Mode** to **Every frame**.



Reflection Probe

Probe Scene Editing Mode:

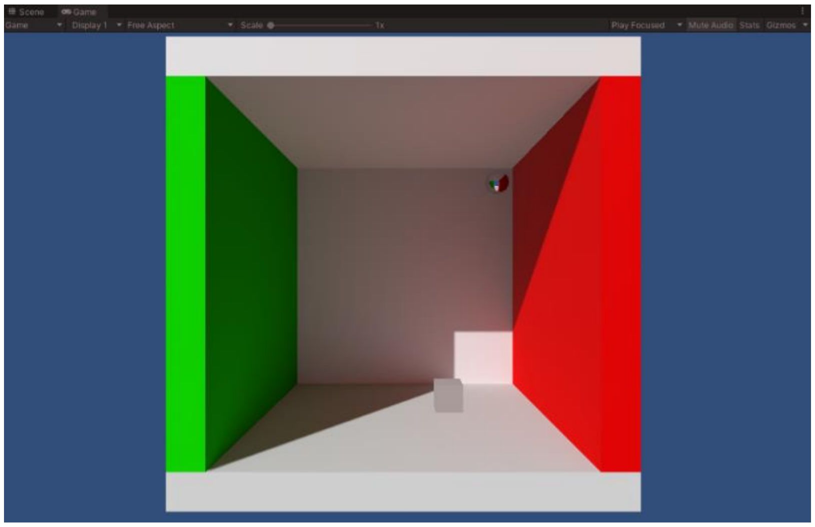
Type Realtime
Refresh Mode Every frame
Time Slicing

On Awake
✓ Every frame
Via scripting

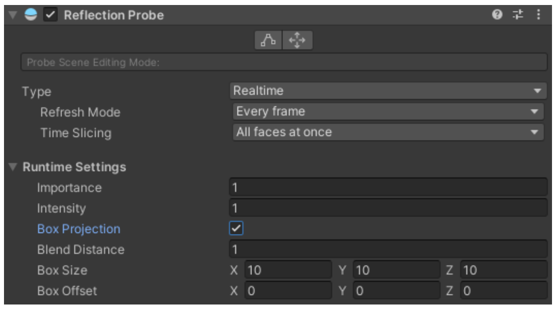
Runtime Settings
Importance

Play the game now and you should see the moving cube in the reflection on the sphere. You may also notice that the game slows down, depending on the speed of your computer.

Now try moving the sphere around the room (leaving the reflection probe where it is). The result is odd: the reflection doesn’t change depending on the sphere’s position.



Again, this is a cost-saving measure. Often it is good enough, but if we are willing to spend extra resources, we can turn on the **Box Projection**, which distorts the reflection appropriately depending on the position of the object relative to its surroundings.



Reflection Probe

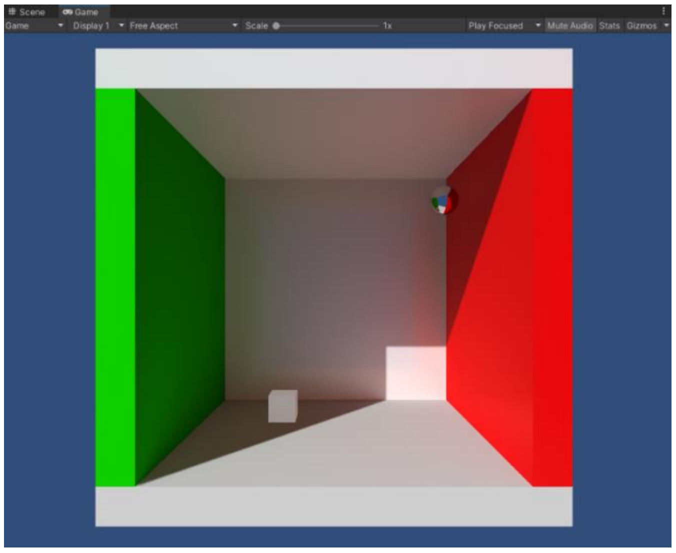
Probe Scene Editing Mode:

Type	Realtime
Refresh Mode	Every frame
Time Slicing	All faces at once

Runtime Settings

Importance	1					
Intensity	1					
Box Projection	✓					
Blend Distance	1					
Box Size	X	10	Y	10	Z	10
Box Offset	X	0	Y	0	Z	0

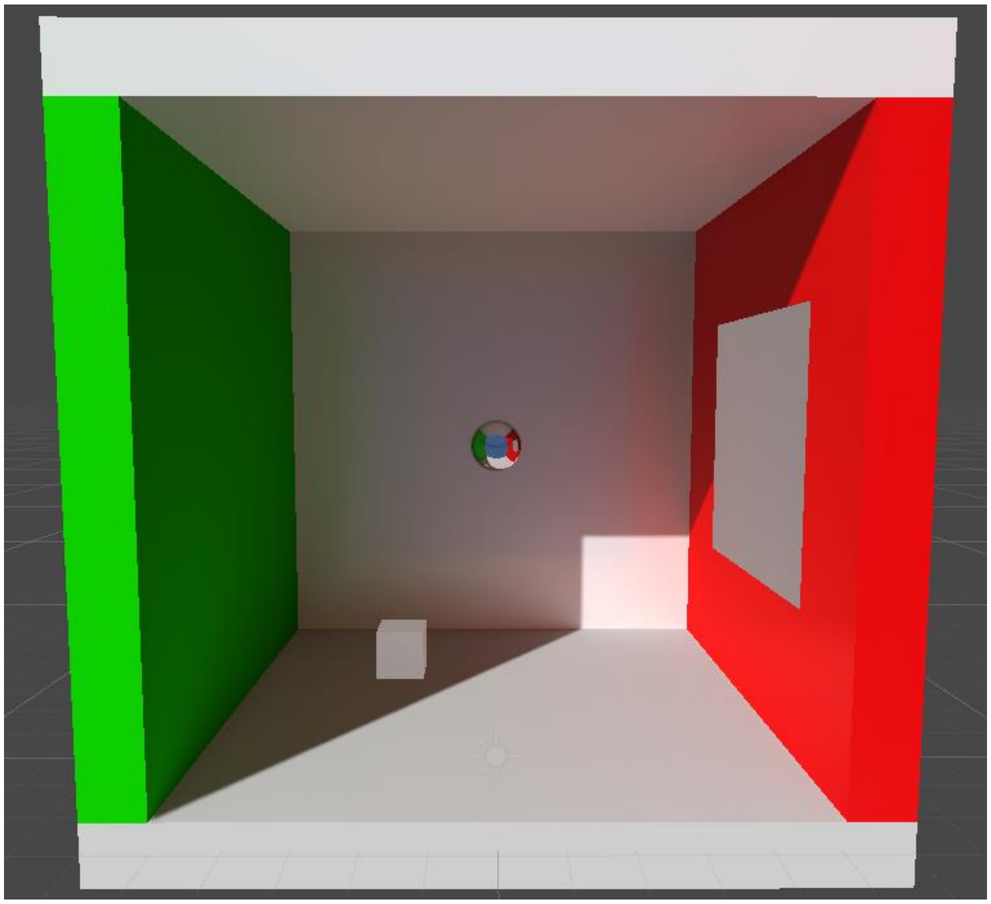
The results are not perfect, but the reflective texture on the sphere should now be broadly accurate as you reposition it around the room.



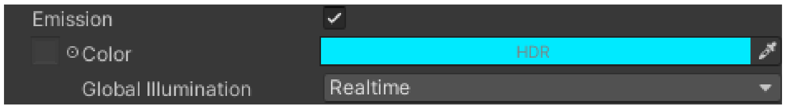
Emissive materials

There is another way to add light to a scene that we haven't discussed yet: emissive materials. These are materials for objects which give out their own light, like computer screens and lighting panels.

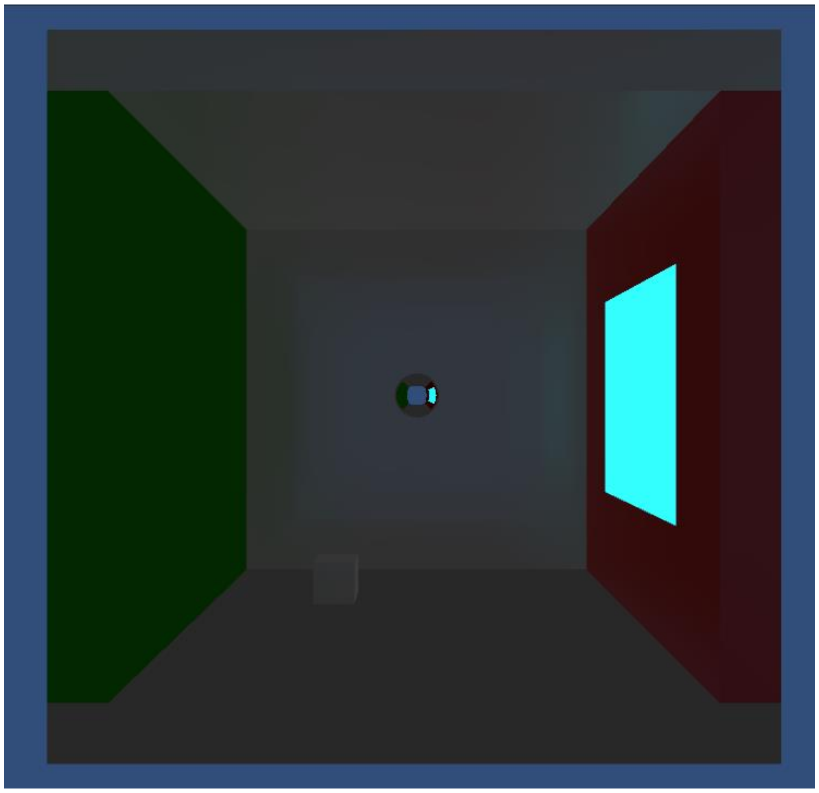
Add a quad to your scene and place it so that it's attached to one of the walls, like a screen.



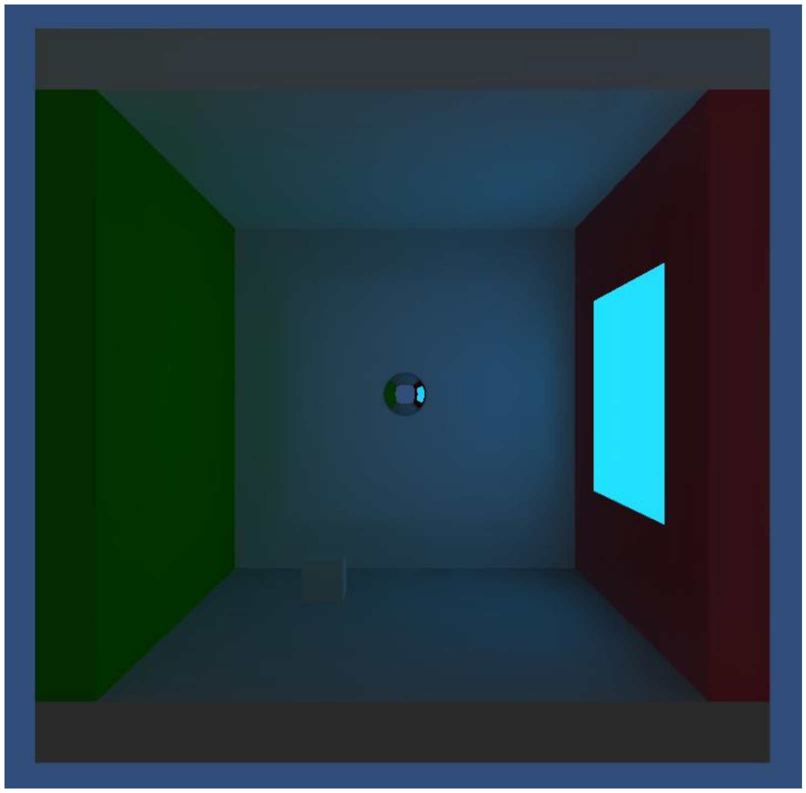
Create a new material for this screen. Set the **Albedo** to **black,** tick the **Emission** checkbox, and set the Emission **Color** to a pale blue colour.


Emission ✓
Color HDR
Global Illumination Realtime

Now the quad should glow with pale blue light in your scene. You can notice this most easily if you disable the 'Directional Light'.

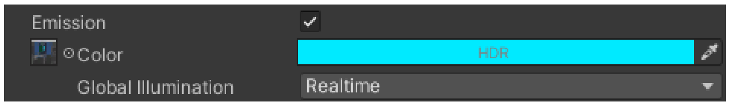


You can make the light from the screen affect the rest of the scene if you make the quad **static**.



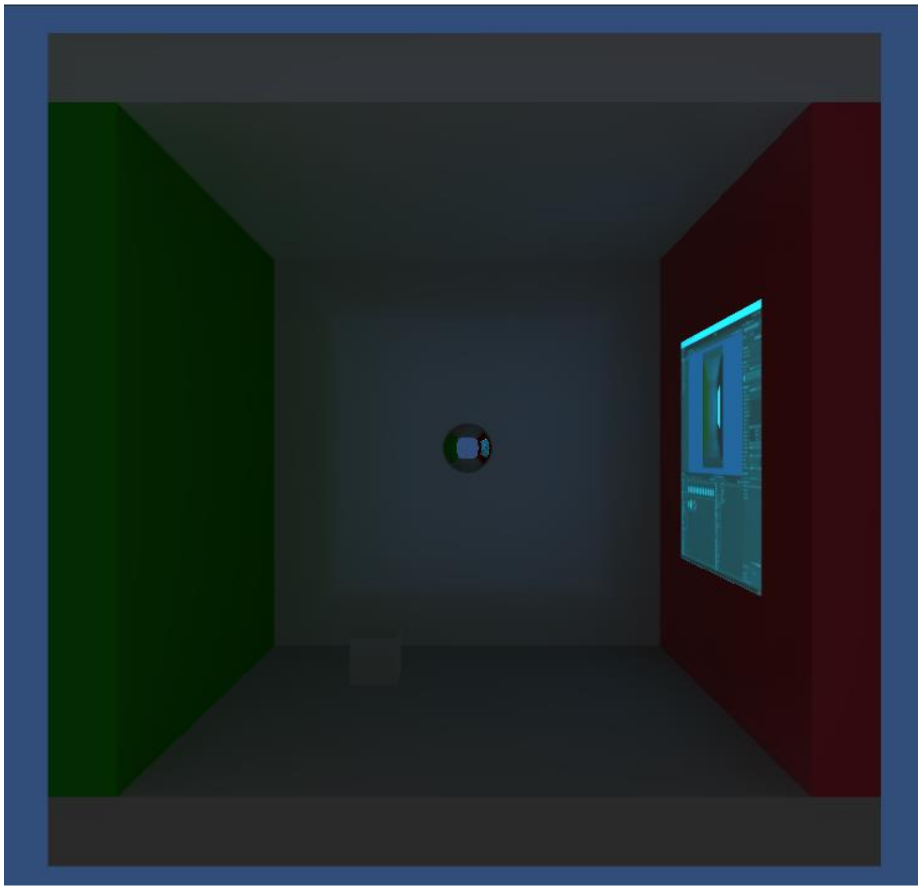
Try the following:

1. Take a screenshot of your computer
2. Import the image into you project
3. Drop the image into the **Emission** texture slot of the material.



Emission ✓
Color HDR
Global Illumination Realtime

You should now have an image of your screen on the screen in the game.



And much more…

Doing realistic lighting well is a complicated trade-off of many different factors. This prac has introduced you to the most basic ideas. There are a lot of technical details which will affect the quality of the result in your game. Refer to the Unity Manual chapter on Lighting for a much more detailed overview.

Reminder: Save, Commit and Push to GitHub

Show your demonstrator

To demonstrate your understanding of this week's content to your prac demonstrator you might show:

- The different types of lights implemented in your scene
- The difference between realtime direct and baked indirect lighting in your scene
- The use of light probes to demonstrate indirect lighting on a dynamic object
- The use of reflection probes to make mirror-like objects
- The use of emissive materials to make glowing objects

If you are on a laptop and baking the lighting is taking a long time, you can ask your marker which specific effects they would like you to demonstrate (so that you can turn the others off before baking).