CE318: High-level Games Development

Lecture 4: Cameras, Audio, Lights and Shadows

Diego Perez

dperez@essex.ac.uk
Office 3A.527

2016/17

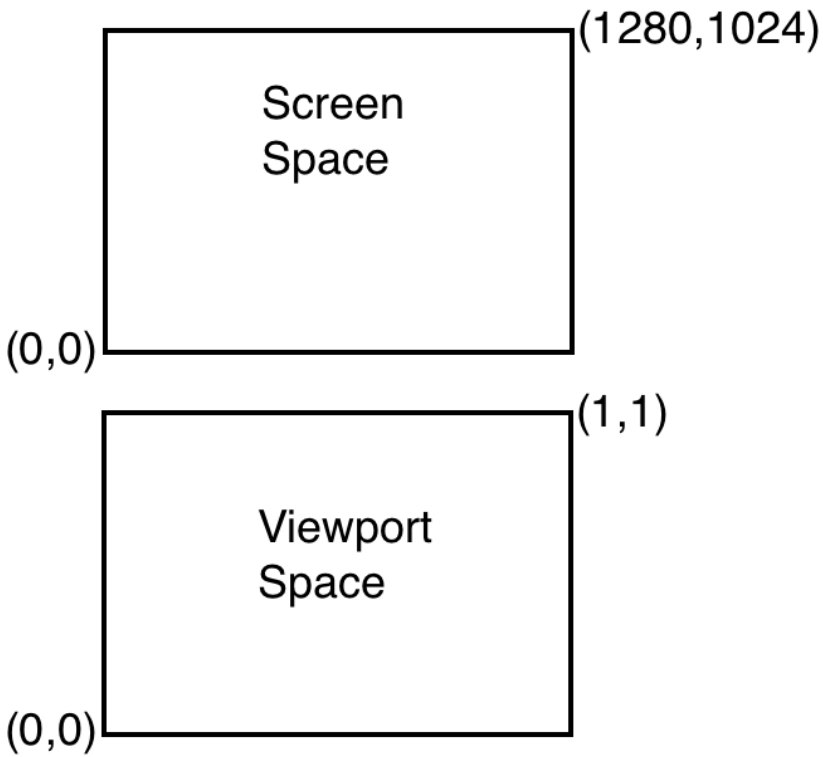
Outline

Outline

Cameras

Cameras are the devices that capture and display the world to the player. They can be customized, scripted, or parented just as any other game object. You can create multiple cameras in the scene.

Every scene in Unity needs to have a camera, or nothing will be shown.

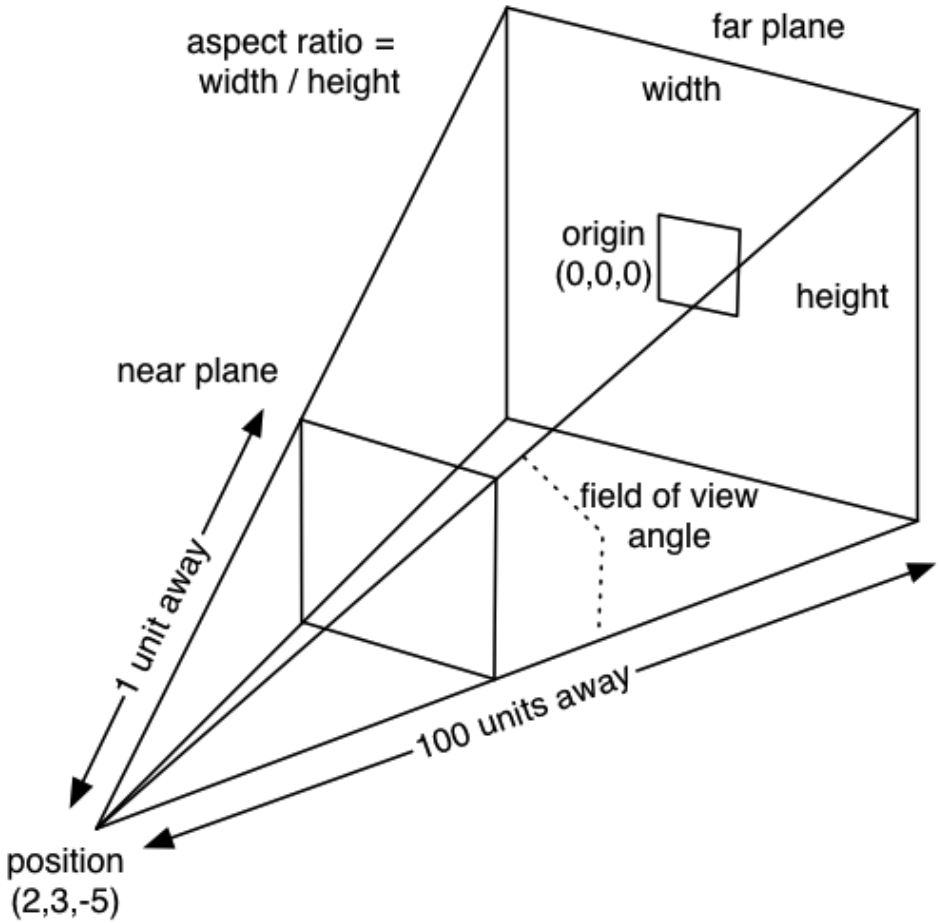


Screen Space is a 2D area with the origin in the lower left corner, and with a size equal to your resolution.

Viewport Space is a normalized area, from $(0, 0)$ to $(1, 1)$.

Camera Projection

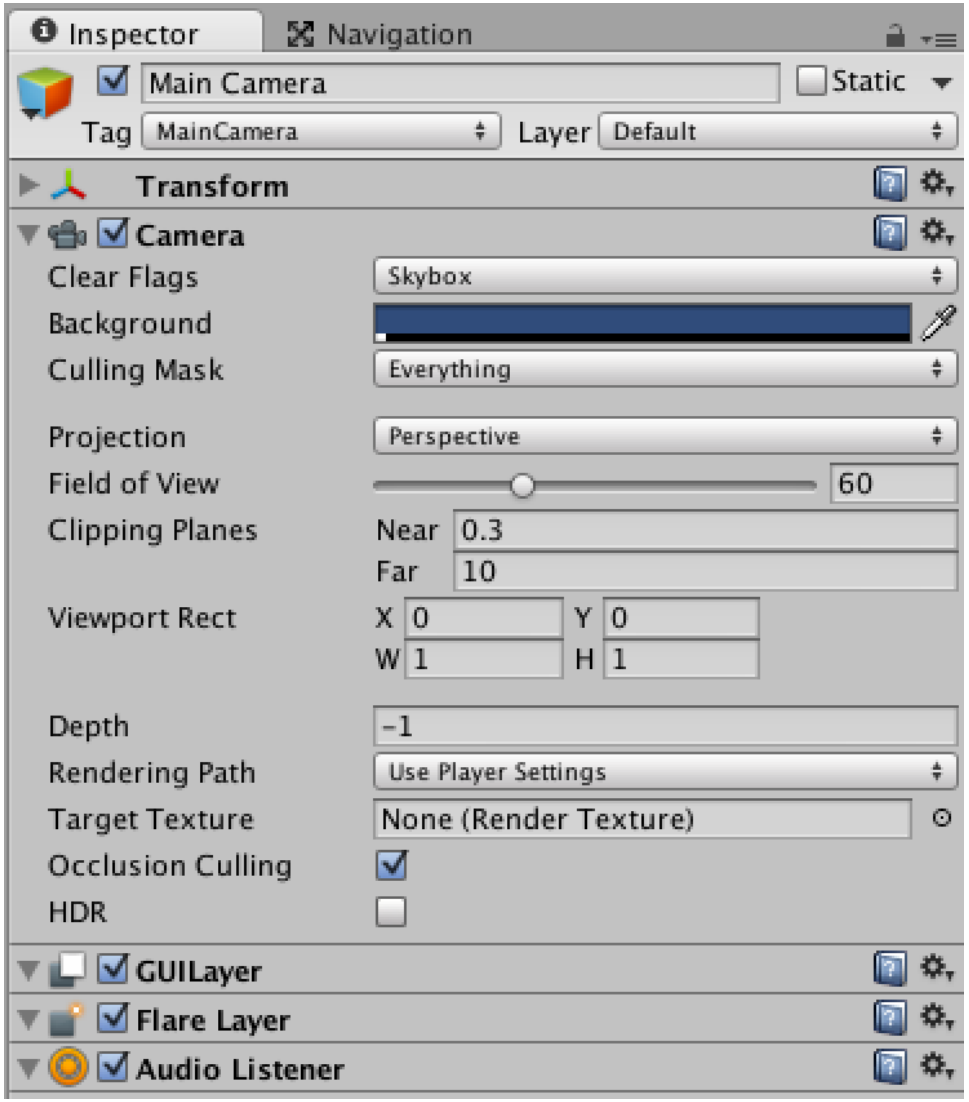
The **Viewing Frustum** is the shape of the region that can be seen and rendered by a camera. The **projection** of the camera allows it to simulate perspective in the rendered scene.



In a **Perspective Projection**, the viewing frustum of the camera takes the shape of a pyramid.

In an **Orthographic Projection** the size of the near plane and the far plane is the same (therefore, the viewing frustum is shaped as a cube). Thus all objects of equal size are rendered the same size independent of distance to the camera (i.e. the perspective is lost).

Camera objects (1/4)



- Transform: transform component of the camera.
- Camera: camera component (can be attached to any game object!).
- GUILayer: to enable/disable rendering of 2D GUIs.
- Flare Layer: to enable/disable Lens Flares (lights refracting inside camera lens) in the image.
- Audio Listener: calculates the output audio. There can only be **one** audio listener in the scene!

Camera objects (2/4)

Settings of the Camera Component:

Clear Flags: Each Camera stores **color** and **depth** information when it renders its view. The portions of the screen that are not drawn in are empty. You can set the clear flags setting to specify what to draw in these empty areas:

- **Skybox:** Any empty portions of the screen will display the current Cameras skybox (chosen in the *Lighting* window).
- **Solid Color:** Any empty portions of the screen will display the current Camera's Background Color.
- **Depth Only:** Nothing is drawn in the empty portions of the screen. This is typically used with more than one layered cameras (bad setting for only one camera).
- **Don't Clear:** Scene rendered in the previous screen does not get deleted.

Camera objects (3/4)

Background Color: Color for the background.

Culling Mask: used for selectively rendering groups of objects using Layers.

Example: All user interface elements are set to a UI Layer, that is rendered by a separate camera. A "scene" camera draws the rest of the scene. In order for the UI to display on top of the "scene" Camera, you need to set the *Clear Flags* to *Depth only* and make sure that the UI Cameras Depth is higher than the "scene" Camera (see below for the explanation of all these parameters).

Projection: Perspective or Orthographic. Orthographic projection lacks perspective, and it is mostly useful for making isometric or 2D games, and they are great for making 3D user interfaces.

Field Of View: Width of the Cameras view angle, measured in degrees along the local Y axis.

Clipping planes: Far and Near planes, specified in Unity units. Near is the closest point relative to the camera that drawing will occur, while Far is the furthest point relative to the camera that drawing will occur.

Camera objects (4/4) *

Viewport Rect: Four values that indicate where on the screen this camera view will be drawn (values between 0 and 1). (X, Y) are the beginning position where the camera view will be drawn, and (W, H) are the width and height of the camera output on the screen.

Depth: The cameras position in the draw order. Cameras with a larger value will be drawn on top of cameras with a smaller value.

Rendering Path: Options for defining what rendering methods will be used by the camera. For more details see: `http://docs.unity3d.com/Manual/RenderingPaths.html`

Target Texture: Reference to a Render Texture that will contain the output of the Camera view. It can be used to create sports arena video monitors, surveillance cameras, transparencies, etc.

Occlusion Culling: Disables rendering of objects when they are not currently seen by the camera because they are obscured by other objects.

HDR: Enables High Dynamic Range rendering for this camera (for more a realistic effect of lights in colours). `http://docs.unity3d.com/Manual/HDR.html`

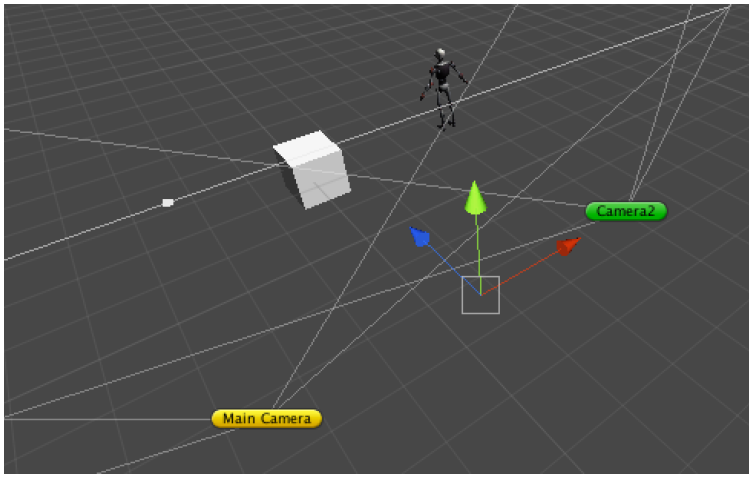
An Example of a Follow Camera

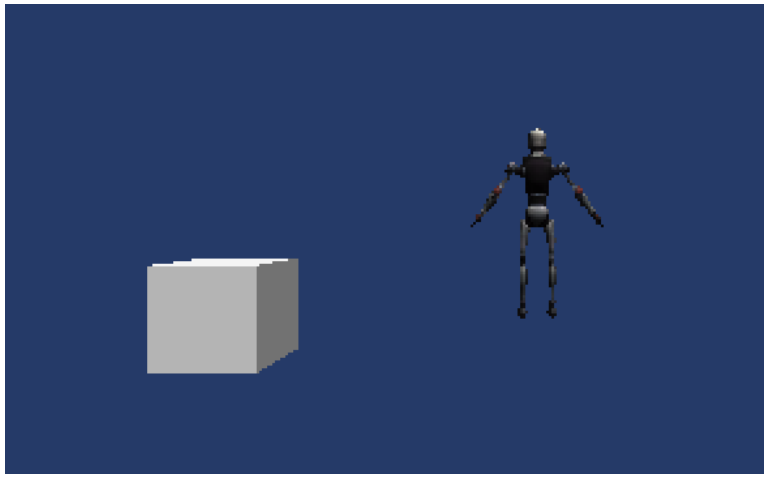
```
1  public class CameraFollow : MonoBehaviour
2  {
3      // The position that the   camera will be following.
4      public Transform target;
5      // The speed with which the camera will be following.
6      public float smoothing = 5f;
7      // The initial offset from the target.
8      Vector3 offset;
9
10     void Start ()
11     {
12         // Calculate the initial offset.
13         offset = new Vector3(0,10,-10);
14     }
15
16
17     void LateUpdate ()
18     {
19         // Create a postion the camera is aiming for
20         //     based on the offset from the target.
21         Vector3 targetCamPos = target.position + offset;
22
23         // Smoothly interpolate between the camera's
24         //     current position and it's target position.
25         transform.position = Vector3.Lerp (transform.position,
               targetCamPos, smoothing * Time.deltaTime);
26     }
27 }
```

Also: Consider using `Transform.LookAt` to set the camera's transform forward vector.

Using Multiple Cameras (1/3) *

When having multiple cameras, the *Depth* parameter controls the order in which the cameras draw the scene. If the values are different, the camera with the lower *Depth* value will be drawn first. If the values are equal, the first camera added to the scene will be drawn first.

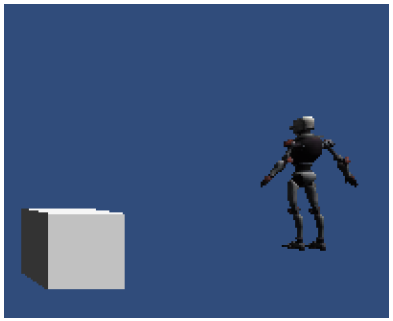




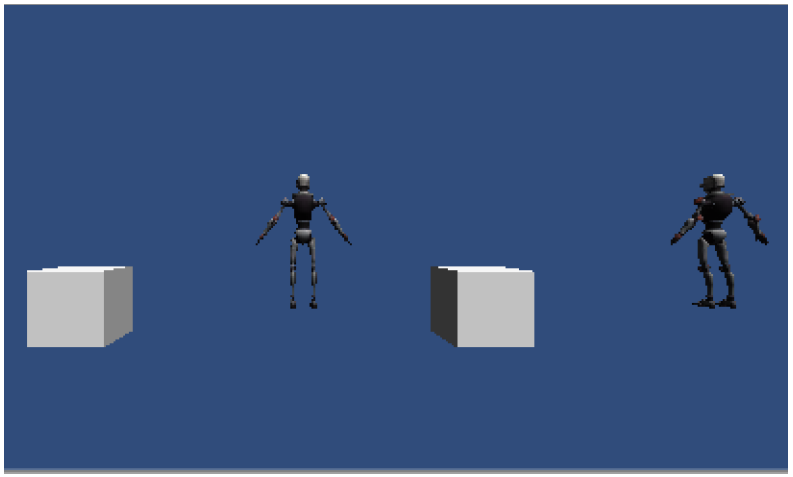
Note that **all** cameras actually draw the scene, but the last one overrides all the others if its *Clear Flags* is not set to *Depth Only*.

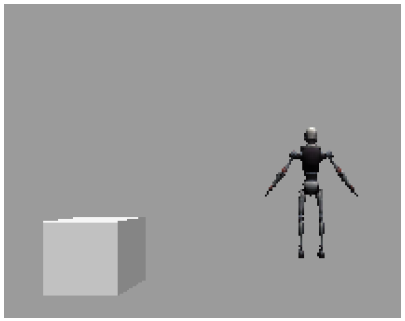
Using Multiple Cameras (2/3) *

It is possible to avoid this overriding by setting the property *Clear Flags* to *Depth Only* in one of the cameras.



Clear Flags = *Skybox*.



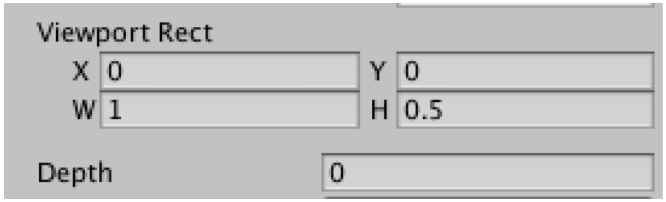


Clear Flags = *Depth Only*.

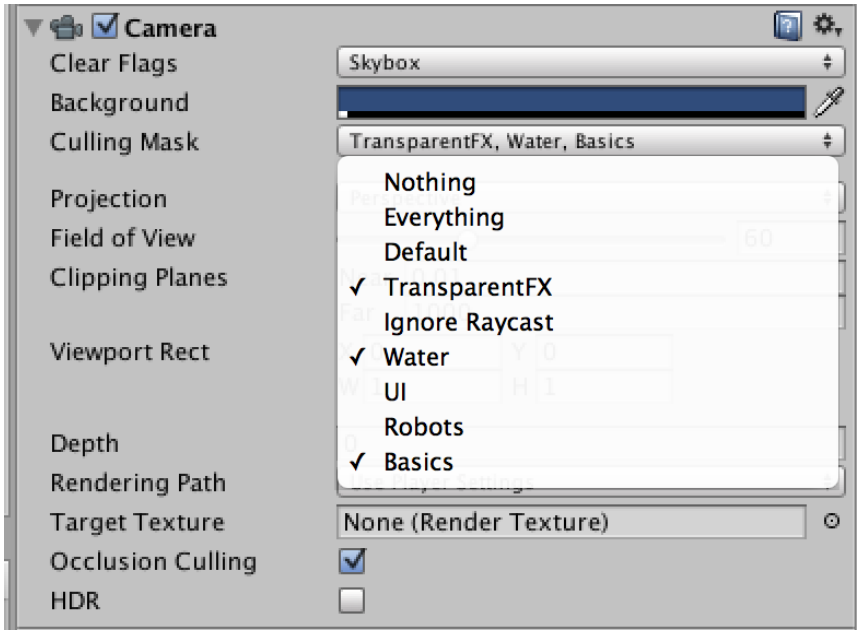
The camera with *Clear Flags* = *Depth Only* only draws those objects hit by the camera. The rest is not drawn.

Using Multiple Cameras (3/3) *

Split Screen: 2 cameras drawing in different parts of the viewport. They can both be assigned to the same Depth, as they are not overriding each other. In order to indicate which parts of the viewport each camera draws, modify the *Viewport Rect* attribute of the cameras.



Culling Masks: By assigning objects to layers, and setting the culling mask property of the cameras, it is possible to make cameras draw objects from specific layers:



Outline

Audio

In Unity, sounds are produced by **Audio Sources** and received by an **Audio Listener**. There can be only one Audio Listener in the scene, while there can be many Audio Sources in the scene. The way a sound is perceived depends on a number of factors, including distance and source position.

To simulate the effects of position, Unity requires sounds to originate from Audio Sources attached to objects. The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera. Unity can then simulate the effects of a source's distance and position from the listener object and play them to the user accordingly.

Unity can import **audio files** in AIFF, WAV, MP3 and Ogg formats in the same way as other assets, simply by dragging the files into the Project panel.

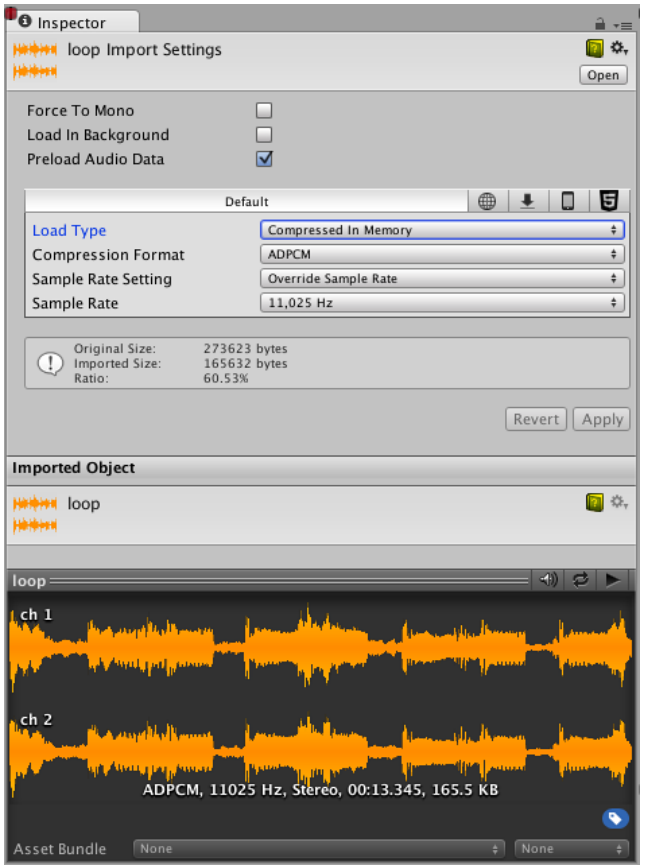
Finally, Unity can access the computers microphones from a script and create Audio Clips by direct recording.

Audio Clips

Audio Clips contain the audio data used by Audio Sources. Unity supports mono, stereo and multichannel audio assets. The audio file formats that Unity can import are .aif, .wav, .mp3, and .ogg. Unity can also import tracker modules in the .xm, .mod, .it, and .s3m formats.

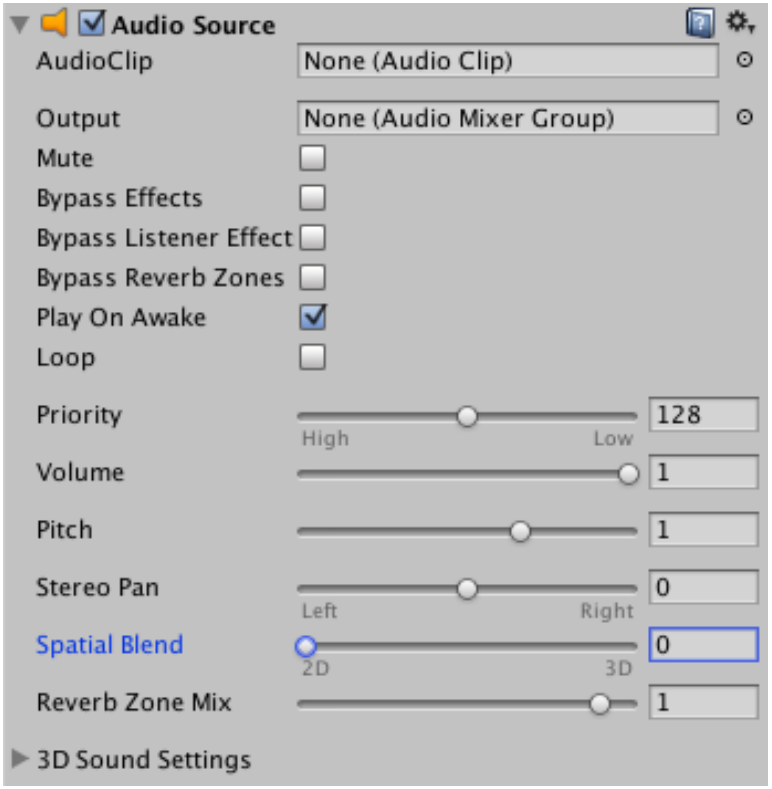
The import settings of an Audio Clip include:

- *Forced to mono:* to force the audio clip to be mixed into a single channel.
- *Load in Background:* loaded not in the main thread (default: *off*).
- *Preload Audio Data:* Audio loaded before scene starts (default: *on*).
- Load type (decompress on load, compressed in memory or streaming), compression format and sample rate.



Audio Source (1/2)

The Audio Source plays back an Audio Clip in the scene.



Audio Clip: The clip to play back.

Output: The sound can be output through an Audio Listener or an Audio Mixer.

Mute: If enabled the sound will be playing but muted.

Play On Awake: If enabled, the sound will start playing the moment the scene launches. If disabled, you need to start it using the Play() command from scripting.

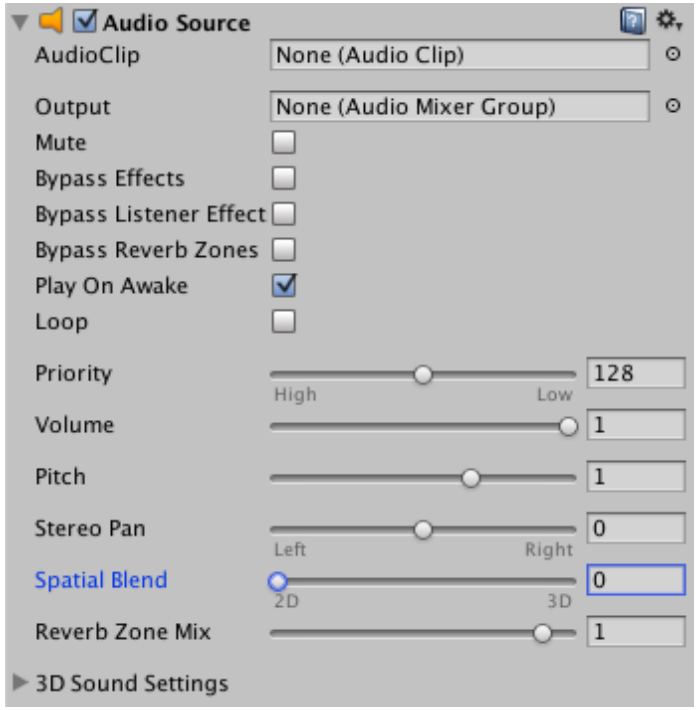
Loop: Enable this to make the Audio Clip loop when it reaches the end.

Audio Source (2/2)

Priority: Determines the priority of this audio source among all the ones that coexist in the scene. (0: most important to 256: least important). Use 0 for music tracks to avoid it getting occasionally swapped out.

Volume: How loud the sound is at a distance of 1 world unit (1 meter) from the A. Listener.

Pitch: Amount of change in pitch due to slowdown/speed up of the Audio Clip (i.e. pace the clip is played). 1 is normal speed.

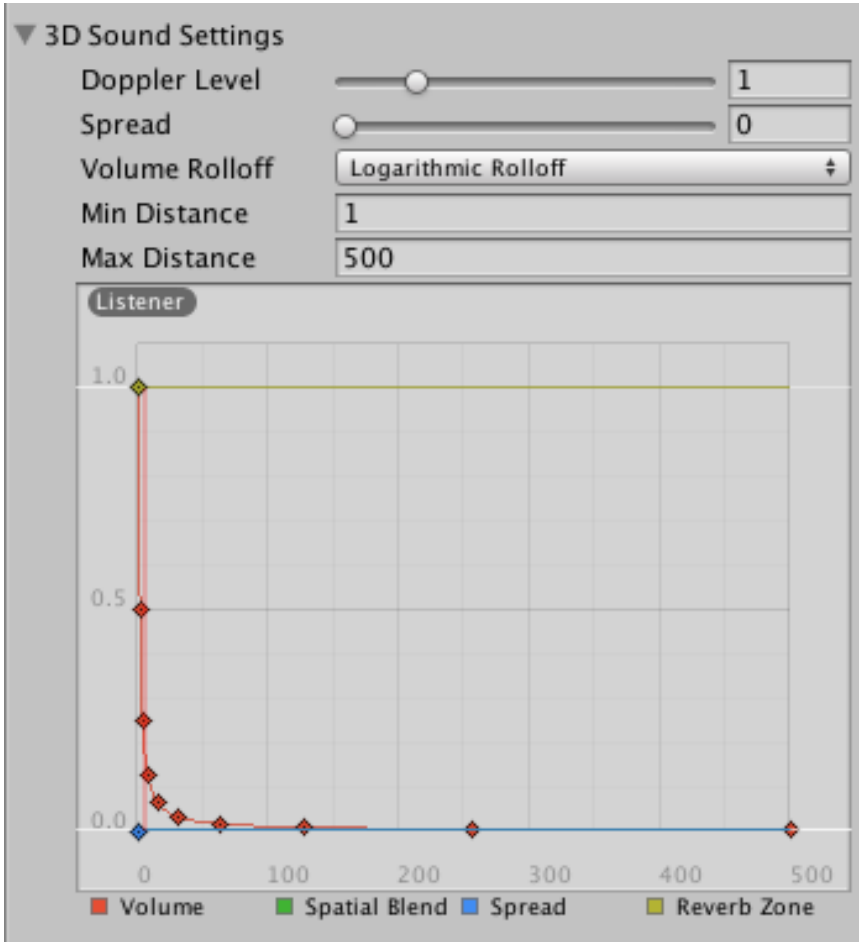


Stereo Pan: Set the position of the stereo field of 2D sounds.

Spatial Blend: Set how much the 3D engine has an effect on the audio source. For total control (complete 3D sound), the source is played back at a given position and will attenuate over distance.

Reverb Zone Mix: Amount of output signal routed to the reverb zones.

Audio Source - 3D Sound Settings (1/2)

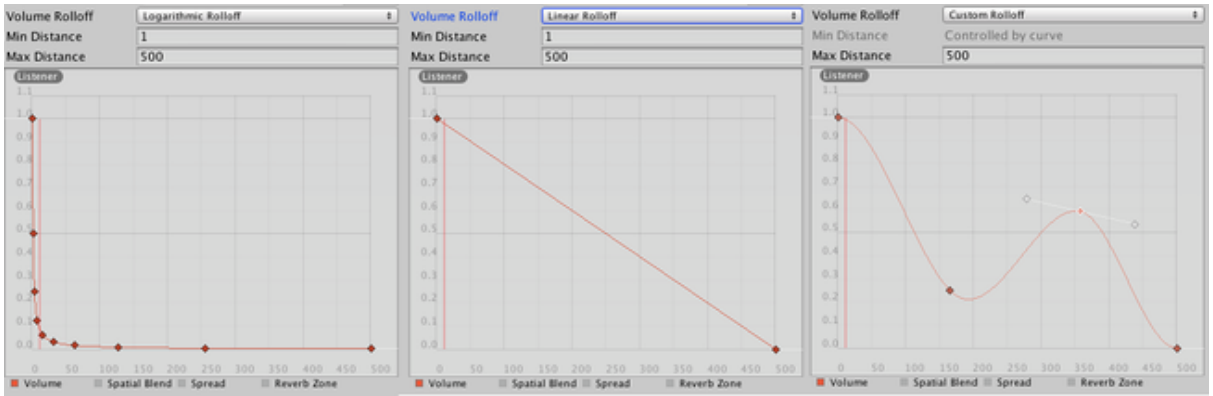


Audio Source - 3D Sound Settings (2/2) *

Doppler Level: Determines how much doppler effect will be applied to this audio source (0: no effect).

Spread: Sets spread angle to 3D stereo or multichannel sound in speaker space. At 0 the sound is playing in mono. At 360 is fully stereo.

Volume Rolloff: How does the sound fades according to its distance to the Audio Listener. Pan level and Spread can also be adjusted like this.



Min Distance: Within the mean distance, the sound will stay at its loudest, attenuating with distance.

Max Distance: The distance where the sound stops attenuating at. Beyond this point it will stay at the volume it would be at MaxDistance units from the listener and will not attenuate any more.

Audio Listener

The Audio Listener receives input from any given Audio Source in the scene and plays it. The Audio Listener has no properties. It simply must be added to work. It is always added to the Main Camera by default and there can only be one in the scene.

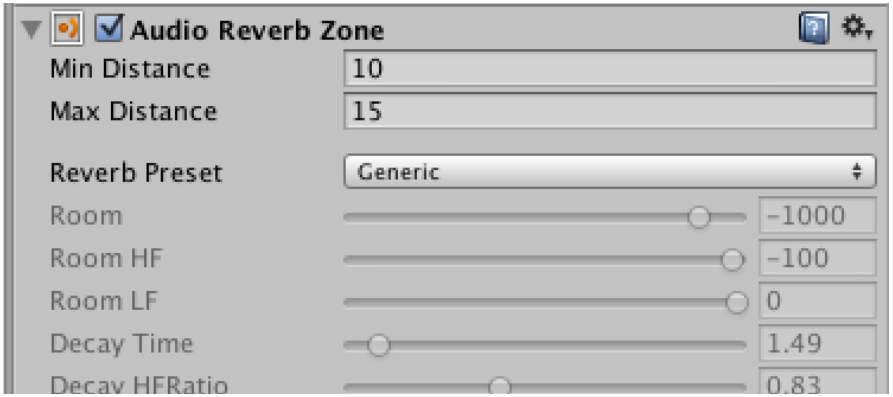
When the Audio Listener is attached to a GameObject in your scene, any Sources that are close enough to the Listener will be picked up and output to the computer speakers.

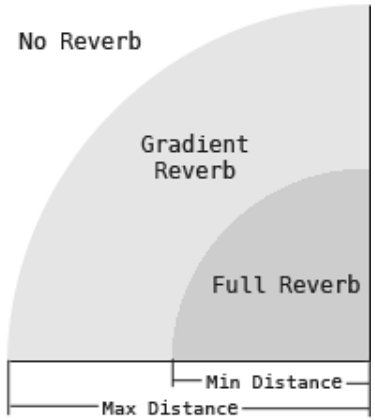
If the Sources are 3D, the Listener will emulate position, velocity and orientation of the sound in the 3D world.

You access the project-wide audio settings using the *Audio Manager*, found in the *Edit* $\rightarrow$ *Project Settings* $\rightarrow$ *Audio* menu.

Reverb Zones

Reverb Zones take an Audio Clip and distorts it depending where the audio listener is located inside the reverb zone. They are used when you want to gradually change from a point where there is no ambient effect to a place where there is one, for example when you are entering a cavern.





- Min Distance: Represents the radius of the inner circle in the gizmo, this determines the zone where there is a gradually reverb effect and a full reverb zone.
- Max Distance: Represents the radius of the outer circle in the gizmo, this determines the zone where there is no effect and where the reverb starts to get applied gradually.
- Reverb Preset: Determines the reverb effect that will be used by the reverb zone (cave, arena, room, bathroom, auditorium, etc.).

Scripting - Audio Source (1/2)

Interesting Variables and Functions:

`bool isPlaying`: Is the clip playing right now (Read Only)?

`bool loop`: Is the audio looping (Read/Write).

`float minDistance, maxDistance`: Distances at which the audio can be heard.

`bool mute`: Mutes/unmutes the AudioSource.

`bool playOnAwake`: If `true`, the source will automatically start playing on awake.

`int priority`: Sets the priority of the AudioSource.

`float volume`: The volume of the audio source (0.0 to 1.0).

`void Play()`: Plays the Audio Clip.

`void PlayDelayed(float delay)`: Plays the clip, with a delay specified in seconds.

`void PlayOneShot(AudioClip clip, float volumeScale = 1.0F)`: Plays a clip scaling the volume by *volumeScale*. This audio will not be stopped by any other.

`void Pause()`: Pauses playing the clip.

`void Stop()`: Stops the clip.

Scripting - Audio Source (2/2)

Examples:

```
1  //Note that 'audio' does not need to be declared!
2  void Update()
3  {
4      if(Input.GetButtonDown("Space")
5      {
6          if(audio.isPlaying){
7              audio.Stop();
8          }else{
9              audio.PlayDelayed(1f);
10         }
11     }
12 }
```

```
1  public AudioClip shootSound;
2  private AudioSource source;
3
4  void Awake()
5  {
6      source = GetComponent<AudioSource>();
7  }
8
9  void Update()
10 {
11     if(Input.GetButtonDown("Fire1")
12     {
13         float vol = Random.Range(0.5f, 1.0f);
14         source.PlayOneShot(shootSound, vol);
15     }
16 }
```

Outline

Lights

Lights are used to illuminate the scene and objects, creating a desired mood and flavour for the game.

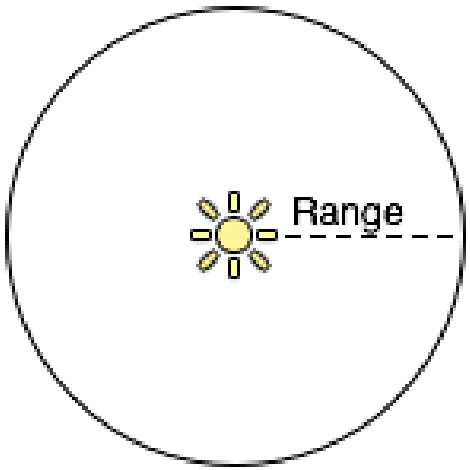
There are four basic light types in Unity:

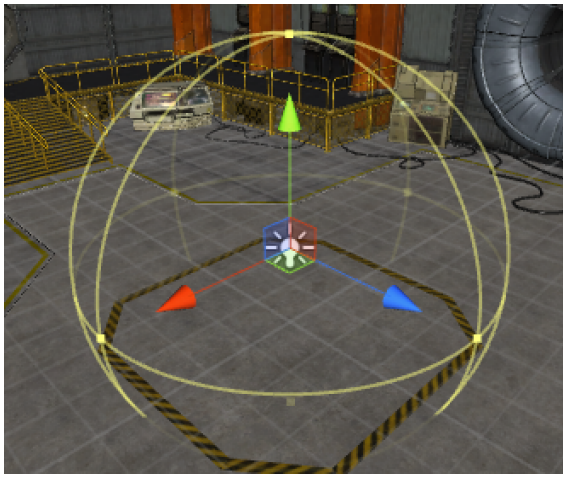
- **Point lights:** shine from a location equally in all directions, like a light bulb.
- **Spot lights:** shine from a point in a direction and only illuminate objects within a cone - like the headlights of a car.
- **Directional lights:** placed infinitely far away and affect everything in the scene, like the sun.
- **Area lights:** (only available for lightmap baking) shine in all directions to one side of a rectangular section of a plane.

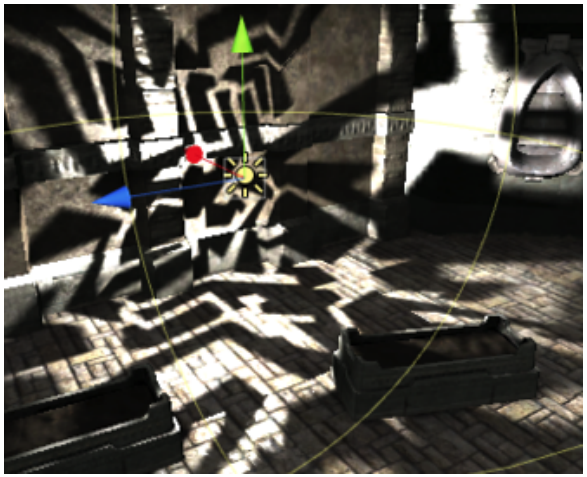
Lights can also cast Shadows. Shadow properties can be adjusted on a per-light basis.

Point Lights

- Point lights shine out from a point in all directions.
- They have an average cost on the graphics processor (though point light shadows are the most expensive).
- They are the most common lights in computer games - typically used for explosions, light bulbs, etc.



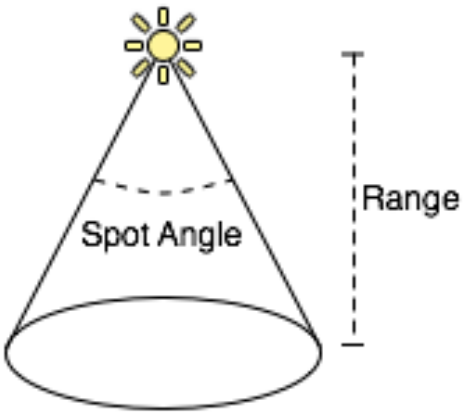


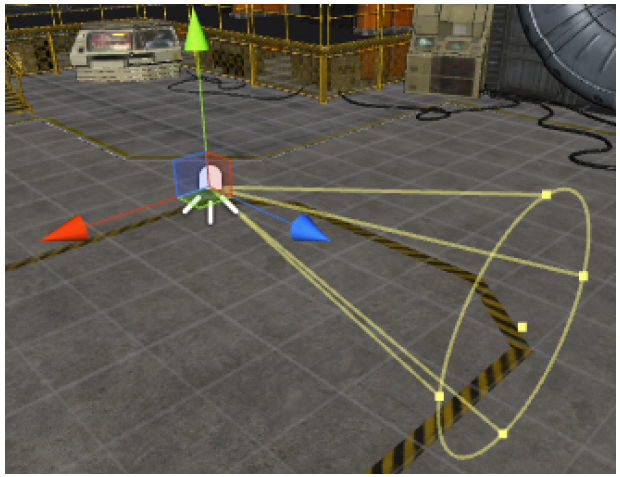


Point lights can have cookies (right image) - Cubemap texture with the alpha channel. This Cubemap gets projected out in all directions.

Spot Lights

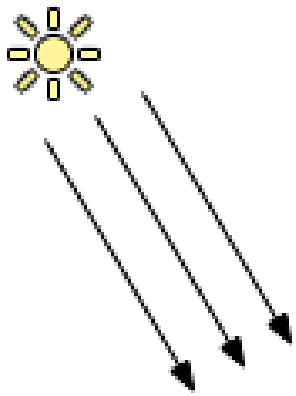
- Spot lights only shine in one direction, in a cone.
- They are the most expensive on the graphics processor.
- They are perfect for flashlights, car headlights or lamp posts.
- Spot lights can also have cookies - a texture projected down the cone of the light. This is good for creating an effect of light shining through the window.

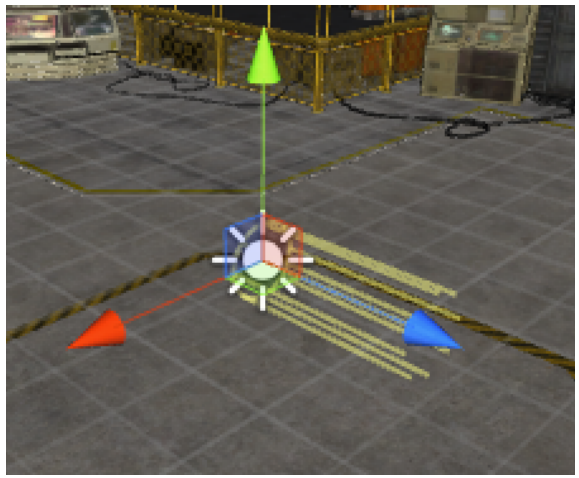




Directional Lights

- Directional lights are placed infinitely far away (its position in the scene is not relevant) and affect all surfaces of objects in your scene.
- Directional lights are used mainly in outdoor scenes for sun and moonlight.
- They are the least expensive on the graphics processor.

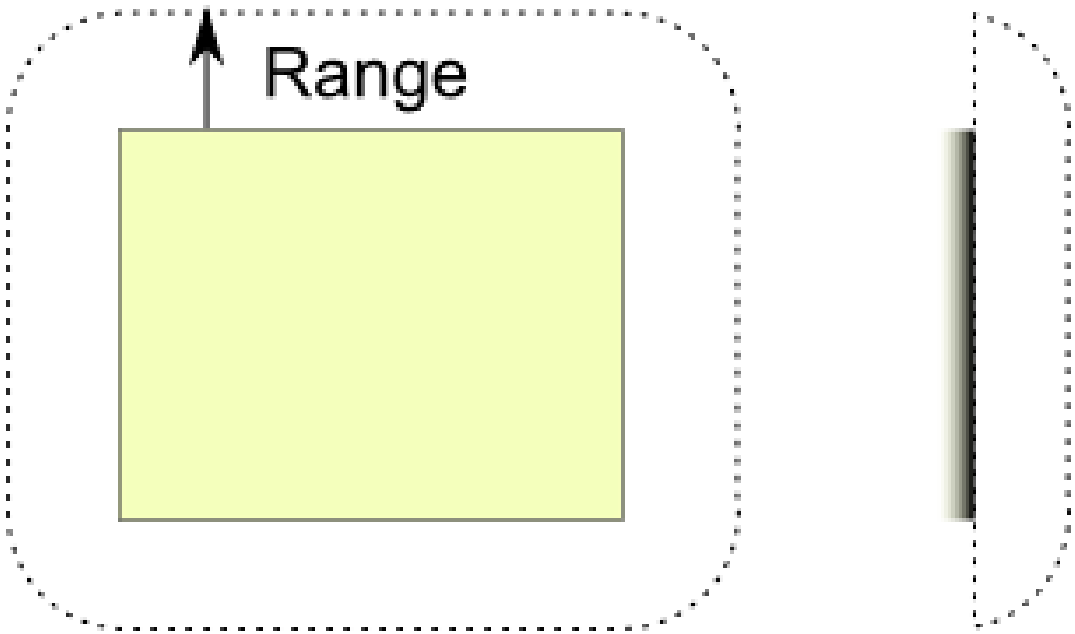




When directional light has a cookie, it is projected down the center of the light's Z axis (Forward vector). The size of the cookie is controlled with the Cookie Size property. Set the cookie textures wrapping mode to Repeat in the Inspector.

Area Lights

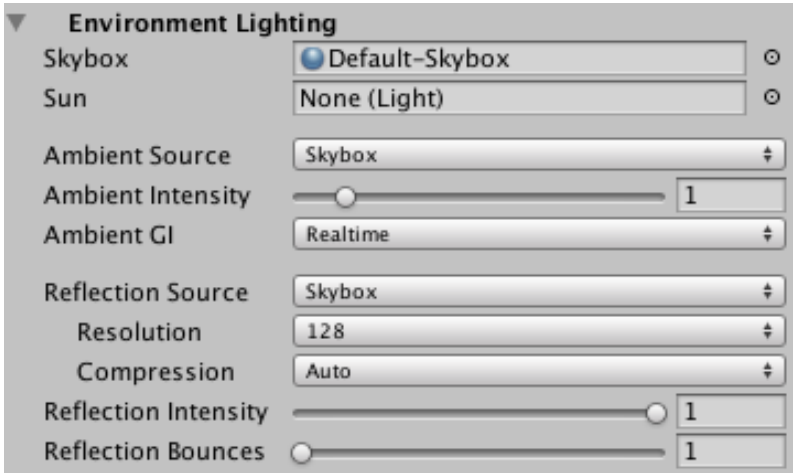
- Light is cast on all objects within the lights range.
- The size of the rectangle is determined by the Width and Height properties and and the planes normal (i.e. the side to which light is cast) is the same as the light's positive Z direction (Forward vector).



Since the lighting calculation is quite processor-intensive, area lights are not available at runtime and can only be baked into lightmaps.

Environment Lighting (1/2)

Access it at *Window* → *Lighting*:



Skybox: Image that appears behind everything else in the scene.

Sun: To specify direction of Directional Light (only procedural skyboxes).

Reflection Source allows to specify whether to use the skybox for reflection effects (the default) or alternatively choose a cubemap to use instead.

Environment Lighting (2/2)

Ambient Light is a light from no particular point that lights all surfaces of the screen. Three possible sources:

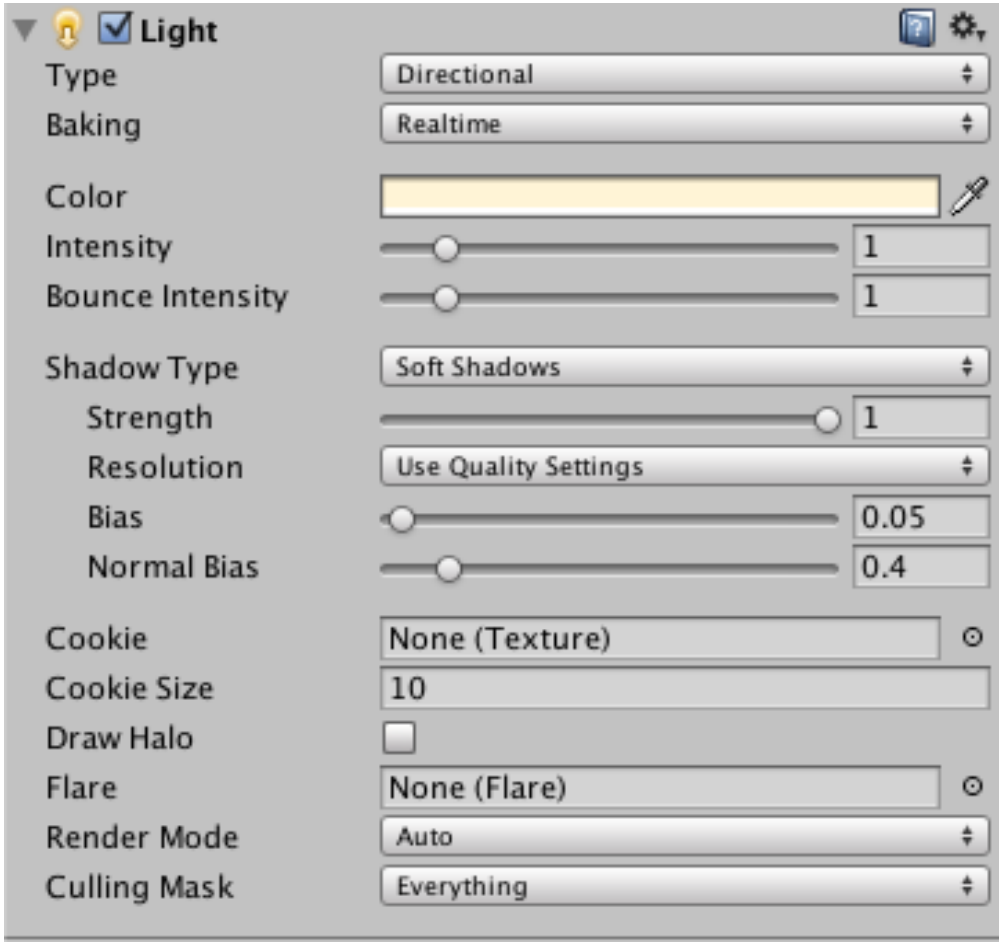
- Color: a flat colour for the light.
- Gradient: colours for sky, horizon, ground.
- Skybox: colours from the skybox.

Basing all your lighting on the ambient light is usually not a good idea, as it is too flat. You should use the previously seen lighting options in your games.

You could even turning it off completely by setting its colour to black $(0, 0, 0)$.

However, it's a good light for illuminating all objects equally to create an effect of day or night: you can change the Ambient Light to be much brighter at daytime, and much darker at night.

The Light Component (1/2)



Type: Type of the light (spot, directional, light, area).

Baking *Realtime*, *Mixed* or *Baked*; see below.

The Light Component (2/2)

Range: How far light is emitted from the object's center. *Point*/*Spot* only.

Spot Angle: Determines the angle of the cone in degrees. Spot light only.

Color: The color of the light emitted.

Intensity: Brightness of the light.

Bounce Intensity: Brightness of the light after bouncing.

Shadow Type: Shadow cast by this light.

Cookie: The alpha channel of this texture is used as a mask that determines how bright the light is at different places. See below.

Cookie Size: Scales the projection of a Cookie. Directional light only.

Draw Halo: To draw a spherical halo of light with a *Range* radius.

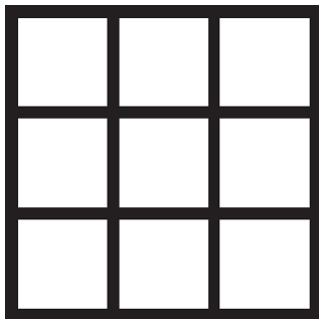
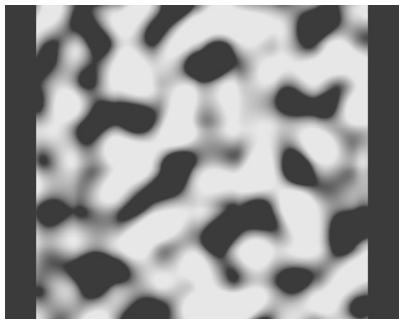
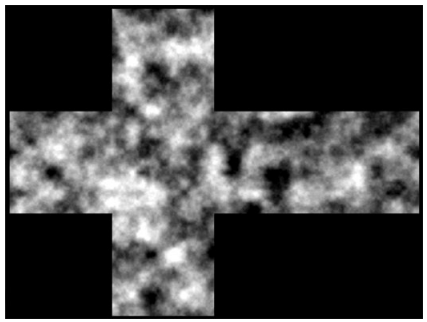
Flare: To add a flare at the light's position.

Render Mode: *Auto*, *Important* or *Not Important*. See below.

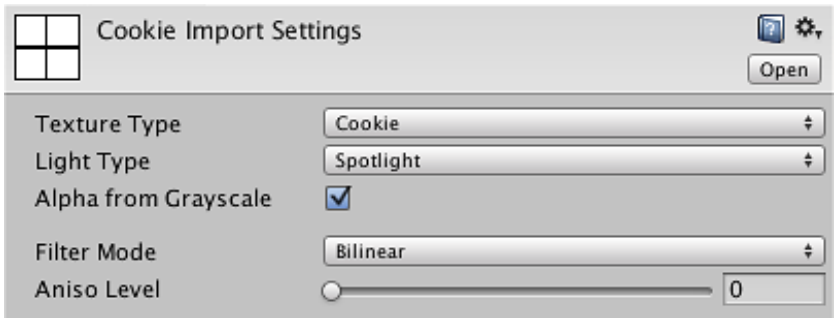
Culling Mask: Use to selectively exclude groups of objects (**layers**) from being affected by the light.

Cookies

A **cookie** is an image that can be used as a mask to determine the brightness of the light in different parts, according to its alpha channel, as it illuminates the scene. If the light is a Spot or a Directional light, this image must be a 2D texture. If the light is a Point light, it must be a Cubemap.

When creating/importing a cookie, a few parameters must be set:

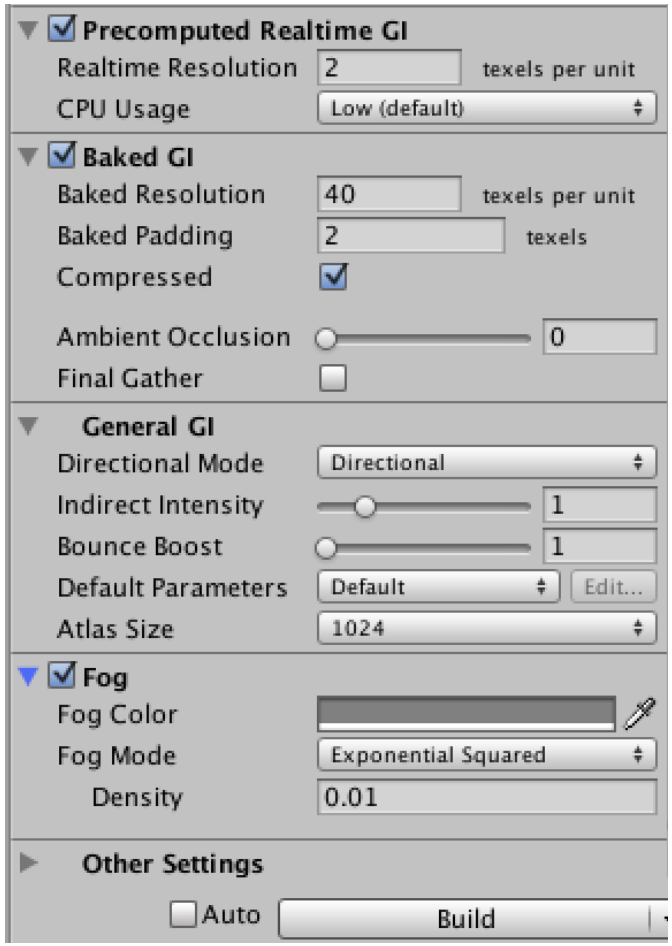


- Texture Type: *Cookie*.
- Light Type.
- Alpha from Grayscale: so Unity converts a grayscale image to an alpha map.

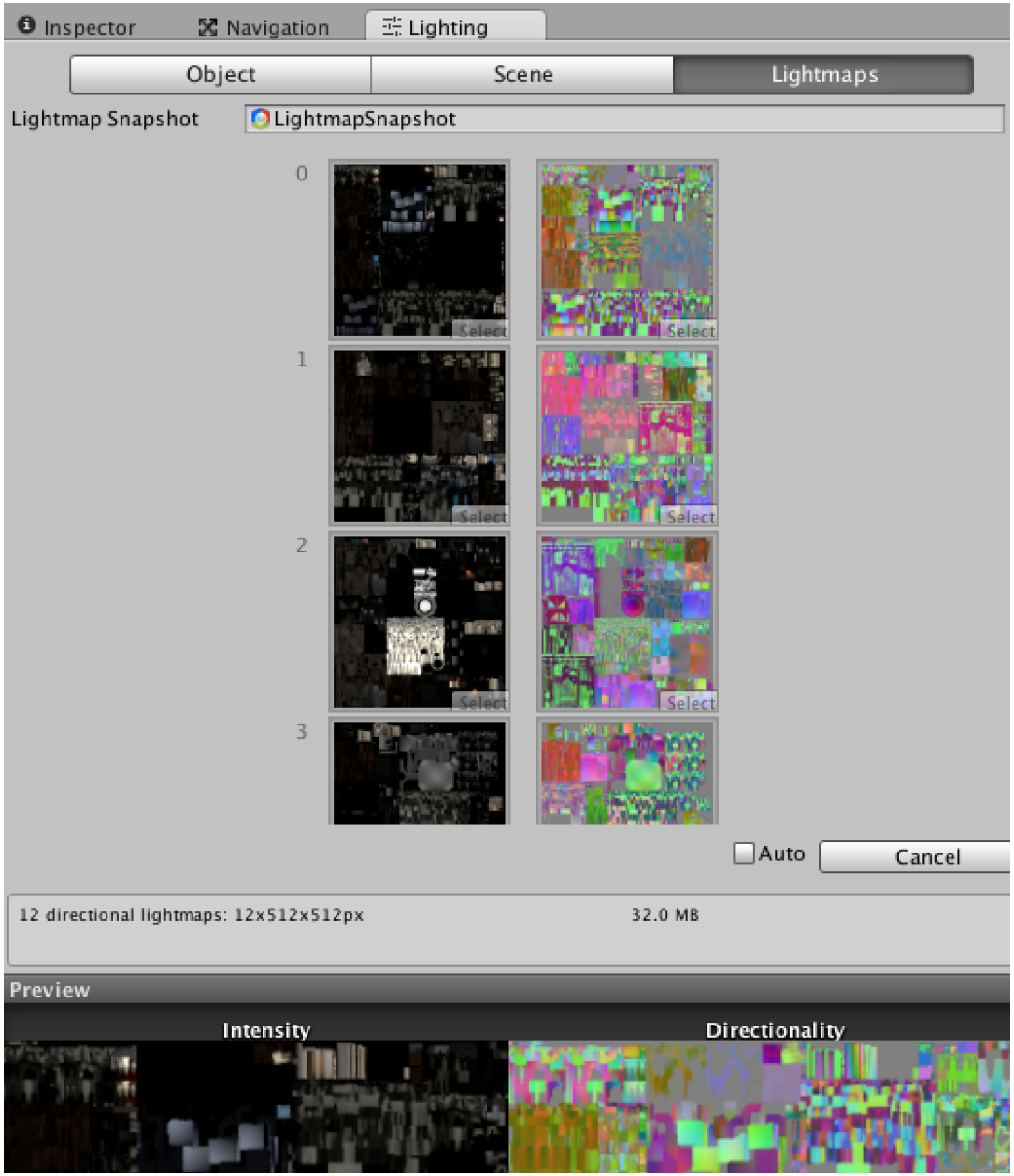
Lightmapping (1/2)

A **lightmap** is a data structure that contains the brightness of the **static** surfaces of the scene. Lightmaps are pre-computed (baked) for a scene, taking into account how meshes, materials, textures and lights are distributed.

- Precomputed Realtime Global Illumination (GI): Stores additional information about how would light bounce to use it in realtime.
- Three types of lightmapping:
  - Non-directional: default case, stores information about how much light does the surface emit.
  - Directional: additional information about the direction the light comes in.
  - Directional with Specular, extra information for indirect lighting.



Lightmapping (2/2)



Shadows

Unity uses **shadow mapping** to cast shadows of objects in the scene. The shadow is created by checking if a given pixel is illuminated by a light source. The light's view is rendered, creating a *shadow map* as a texture, which stores the depth of every surface seen. The scene is then rendered from the camera, adding the information from the shadow map to colour the scene and shadows.

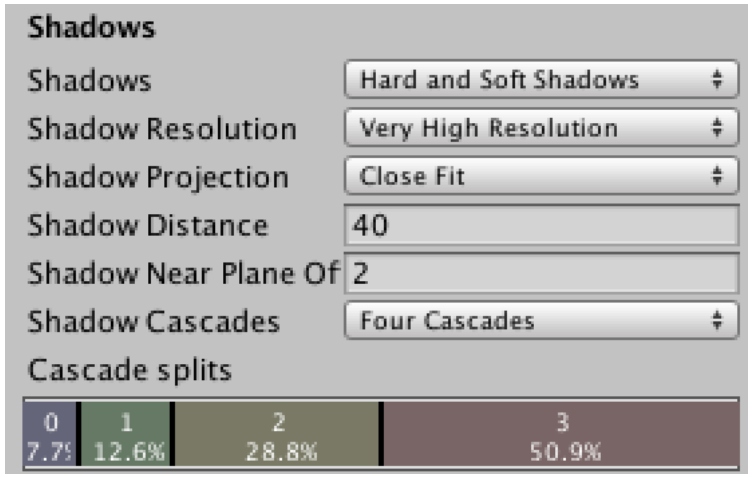
The quality of the shadow map depends on two factors:

- The resolution (**shadow size**) of the shadow maps. The larger the shadow maps, the better the shadow quality.
- The filtering of the shadows. Hard shadows take the nearest shadow map pixel. Soft shadows average several shadow map pixels, resulting in smoother looking shadows. However, soft shadows are more expensive to render.

Real-time Shadows (1/2)

- Real-time shadows are computationally expensive and take a lot of resources, so they should be used with care.
- Soft shadows are more expensive to render than hard shadows.
- The cost is entirely on the graphics card, so shadows do not make any impact on the CPU or memory.

Quality Settings (*Edit* → *Project Settings* → *Quality*) contains a section for Shadows, that affects the whole Unity project.



Real-time Shadows (2/2)

Shadows: Determines which type of shadows should be used: *Hard and Soft Shadows*, *Hard Shadows Only* or *Disable Shadows*.

Shadow resolution: Shadows can be rendered at several different resolutions: Low, Medium, High and Very High. The higher the resolution, the greater the processing overhead.

Shadow Projection: There are two different methods for projecting shadows from a directional light:

- *Close Fit* renders higher resolution shadows but they can sometimes wobble slightly if the camera moves.
- *Stable Fit* renders lower resolution shadows but they will not wobble with camera movements.

Shadow Distance: The maximum distance from camera at which shadows will be visible.

Shadow Cascades: The number of shadow cascades can be set to 0, 2 or 4. A higher number of cascades gives better quality but at the expense of processing overhead (see below).

Real-time Shadows - Directional Lights

In a Directional Light, shadows have the following three properties:

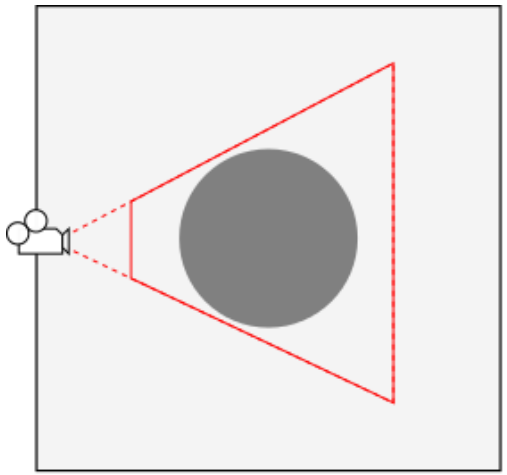
- *Strength:* The darkness of the shadows. Values are between 0 and 1.
- *Resolution:* Detail level of the shadows. It can be set to whatever *Quality* Settings says, or set specifically for this light to *Low*, *Medium*, *High* or *Very High* resolution.
- *Bias:* Determines how close the shadow is to the object that projects it. A value close to 0 may cause shadows on the object's surface (*shadow acne*), but a very high value may cause the shadow to "float" (*peter panning*).

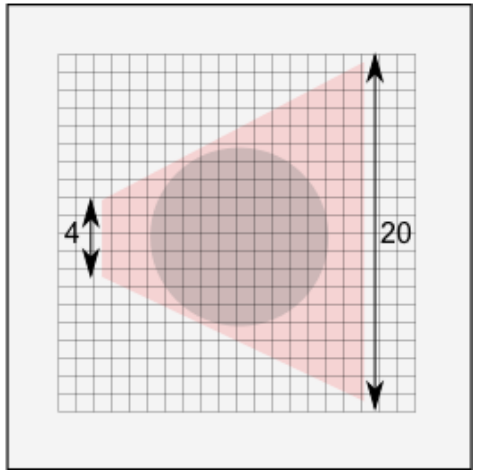
Soft shadows add two more properties, *Softness* and *Softness Fade*, that allow to determine how the transition from lit to shadowed regions is managed.

Shadow Cascades

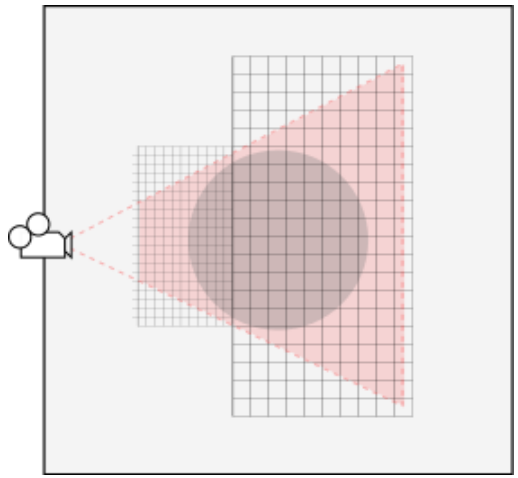
Shadow Cascades is a process that consists of dividing the viewing area in progressively larger areas, using **the same shadow size** on each one of them. The result is that objects close to the viewer get more shadow map pixels than objects far away.

With no shadow cascades (value set to 0), the shadow texture covers the entire viewing area (4 vs 20 pixels, but same size on screen).





With 2 cascades, the entire shadow distance is divided into a smaller chunk near the viewer and a larger chunk far away. In exchange for some performance, we get better shadow resolution up close.



New to Unity 5.4

Cinematic Image Effects (Check the Asset Store). It lets you play with:

- Antialiasing;
- Camera effects;
- Reflectivity and glowing effects;
- Lens aberration and blurriness.

Outline

Progress Test

Next lecture:

- Room **LTB03**, Tuesday 8th November, 1pm - 2pm Progress Test 1
- Room **EBS.2.65**, Tuesday 8th November, 2pm - 3pm Terrains (1h)

The test:

- It contains 20 multiple-choice questions.
- You have 30 minutes.
- The test is to be taken under exam conditions.
- You will receive a reminder 5 minutes to the end.
- Do not turn over your test paper until asked to do so.
- Leave the test and answer book on your table when finished.
- It is worth 15% of the final mark.

Bring **your own pencil** and **rubber**.

UROP & Lab Preview

Lab: You'll finish the Space Shooter assignment.

UROP: Undergraduate Research Opportunities Programme (UROP)

- Useful information:

  `https://www.essex.ac.uk/urop/students/useful-forms.aspx`
- Web Game Engine placement:
  - **General Video Game Artificial Intelligence: Playing and Data Collection on the Web**:
  - `https://www.essex.ac.uk/urop/students/placement-details.aspx?ID=198`