Link to Lab Drive

https://drive.google.com/drive/folders/1jgFTWCJ5JD-JnIY7n_L7s2n2vA1F9nNX?usp=sharing

OR

https://tinyurl.com/ta-comp476-daniel

COMP 376 Lab Slides

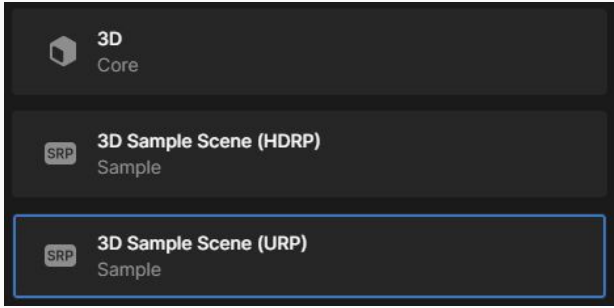
https://drive.google.com/drive/u/0/folders/1iNcgyyXKKZblgykxDGCPJEasnw0Wdlxi

Unity Advanced Features

Render Pipelines

A render pipeline performs a series of operations that take the Scene as input, and outputs a final image to be displayed on a screen.
At a high level, these operations are:

- Culling
- Rendering
- Post-processing



Note: For this course, the built-in render pipeline is satisfactory.

In Unity, you can choose between different render pipelines. Unity provides three prebuilt render pipelines. Different render pipelines have different capabilities and performance characteristics, and are suitable for different games, applications, and platforms.
It can be difficult to switch an ongoing project from one render pipeline to another, because different render pipelines work very differently with regards to shaders (different outputs, features, etc). It is therefore important to understand the different render pipelines that Unity provides, so that you can make the right decision for your project early in development.

- The Built-in Render Pipeline(BIRP) is Unity's default render pipeline. It is a general purpose render pipeline that has limited options for customization.
- The Universal Render Pipeline (URP) is a Scriptable Render Pipeline that is quick and easy to customize, and lets you create optimized graphics across a wide range of platforms.
- The High Definition Render Pipeline (HDRP) is a Scriptable Render Pipeline that lets you create cutting-edge, high-fidelity graphics on high-end platforms.
- You can create your own custom render pipeline using Unity's Scriptable Render Pipeline API.

Universal Render Pipeline

To use the Universal Render Pipeline (URP), you can start a new Project or upgrade an existing Project. There are 2 ways to do this:

- Create a new URP Project from a Template (using Unity Hub). If you are starting a new Project from scratch, this is the best choice. When you do this, Unity automatically installs and configures URP for you.
- Install URP into an existing Unity Project. If you have started a Project using the Built-in Render Pipeline, you can install URP and configure your Project to use URP. When you do this, you must configure URP yourself. You will need to manually convert or recreate parts of your Project (such as shaders or post-processing effects) to be compatible with URP.

As previously mentioned, projects made using URP are not compatible with other render pipelines. Before you start development, you must decide which render pipeline to use in your Project. For a full list of features supported in URP, compared to BIRP see : https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.8/manual/universalrp-builtin-feature-comparison.html

However, if you do decide to switch from the BIRP to URP, then you will have to manually convert and fix all the problems after switching pipelines. Luckily for us, Unity does provide an option that helps automate part of this process in that it tries to automatically convert basic materials from the BIRP to their URP counterparts. You can find this option under **Edit > Render Pipeline > Universal Render Pipeline > Upgrade Project Materials to URP Materials** .

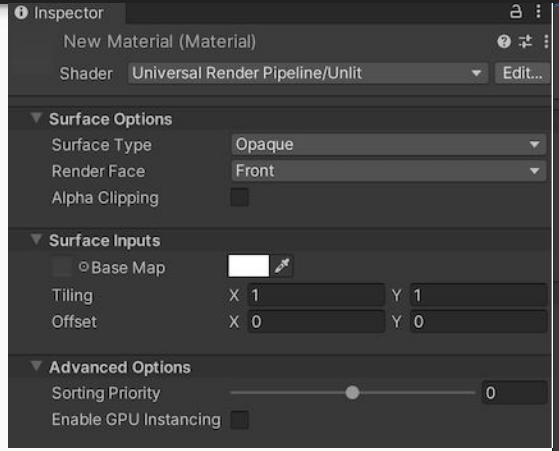
If you are using URP, then the basic “Default-Material” that you may be used to from BIRP will not work anymore because the shaders it uses are not compatible with URP. Instead URP uses its own set of standard shaders; two most commonly used Materials are the “Unlit” and “Lit” Materials which uses the URP Unlit and Lit shaders respectively.

For more info on the many URP’s standard shaders see : https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.8/manual/shaders-in-universalrp.html

Universal Render Pipeline

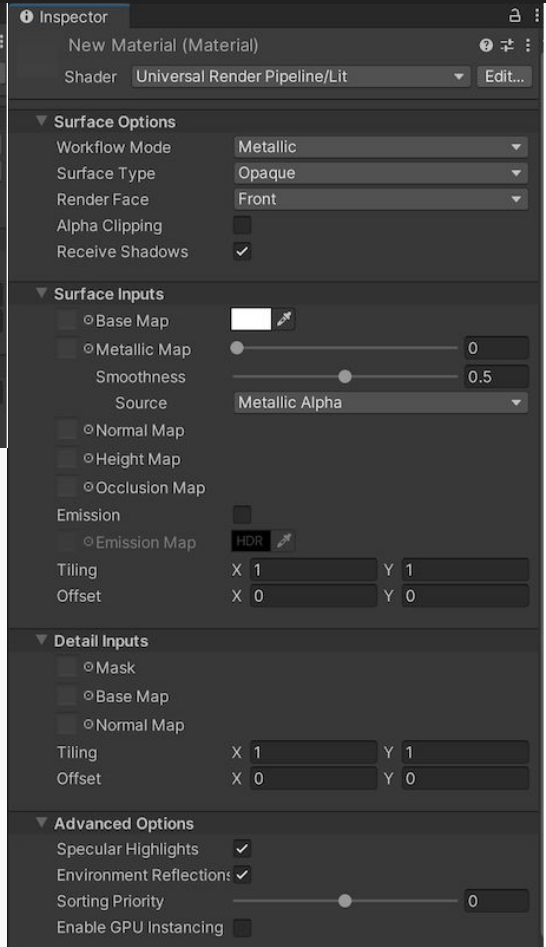
Unlit Shader

The Unlit Shader uses the most simple shading model in URP. Use this Shader for effects or unique objects in your visuals that don't need lighting. Because there are no time-consuming lighting calculations or lookups, this Shader is optimal for lower-end hardware.



Lit Shader

The Lit Shader uses the most computationally heavy shading model in URP. The Lit Shader lets you render real-world surfaces like stone, wood, glass, plastic, and metals in photo-realistic quality. Your light levels and reflections look lifelike and react properly across various lighting conditions, for example bright sunlight, or a dark cave.



Shader Graph

Only available with URP, Shader Graph lets you build shaders visually. Instead of writing code, you create and connect nodes in a graph framework. ShaderGraph gives instant feedback that reflects your changes, and it's simple enough for users who are new to shader creation.
For more about ShaderGraph see :
https://docs.unity3d.com/Packages/com.unity.shadergraph@10.8/manual/index.html

Post Processing

Post Processing is the process of applying filters and effects on the camera's image buffer after the camera finishes outputting the scene and lighting.

Unity provides a number of post-processing effects and full-screen effects that can greatly improve the appearance of your application with little set-up time. You can use these effects to simulate physical camera and film properties, or to create stylised visuals.

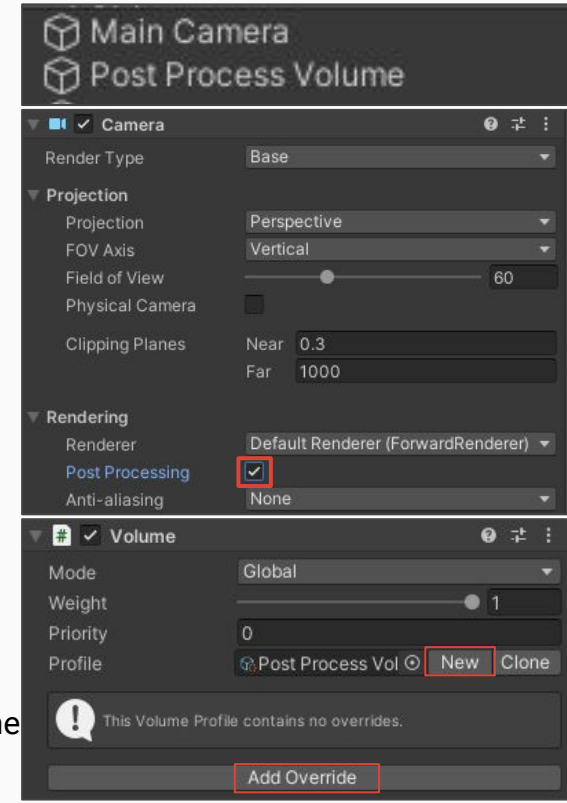
Which post-processing effects are available and how you apply them depend on which render pipeline you are using. A post-processing solution from one render pipeline is not compatible with other render pipelines.
For more information on compatibility and available effects per pipeline, see the Unity Manual : https://docs.unity3d.com/Manual/PostProcessingOverview.html

In these slides we will cover a few effects and focus on URP.
For a detailed look at all the different post processing effects using URP see:
https://www.youtube.com/watch?v=9tjYz6Ab0oc

Post Processing

To start using post processing in Unity using URP, we start by creating a **Post Processing Volume**.
To do this we simply:

- Create an empty gameobject in your scene and attach the Volume component to it.
- Set the post processing flag on the camera to true.
- Create the post processing volume asset for our Volume and refer to it.



Each Volume can either be global or have local boundaries.
The Volume component has a mode property which you can set to one of the following:

- **Global** : Makes the Volume have no boundaries and allow it to affect every Camera in the Scene. This is the default option.
- **Local** : Allows you to specify boundaries for the Volume so that the Volume only affects Cameras inside the boundaries. Add a Collider to the Volume's GameObject and use that to set the boundaries.

Link to Volume documentation :
https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/Volumes.html

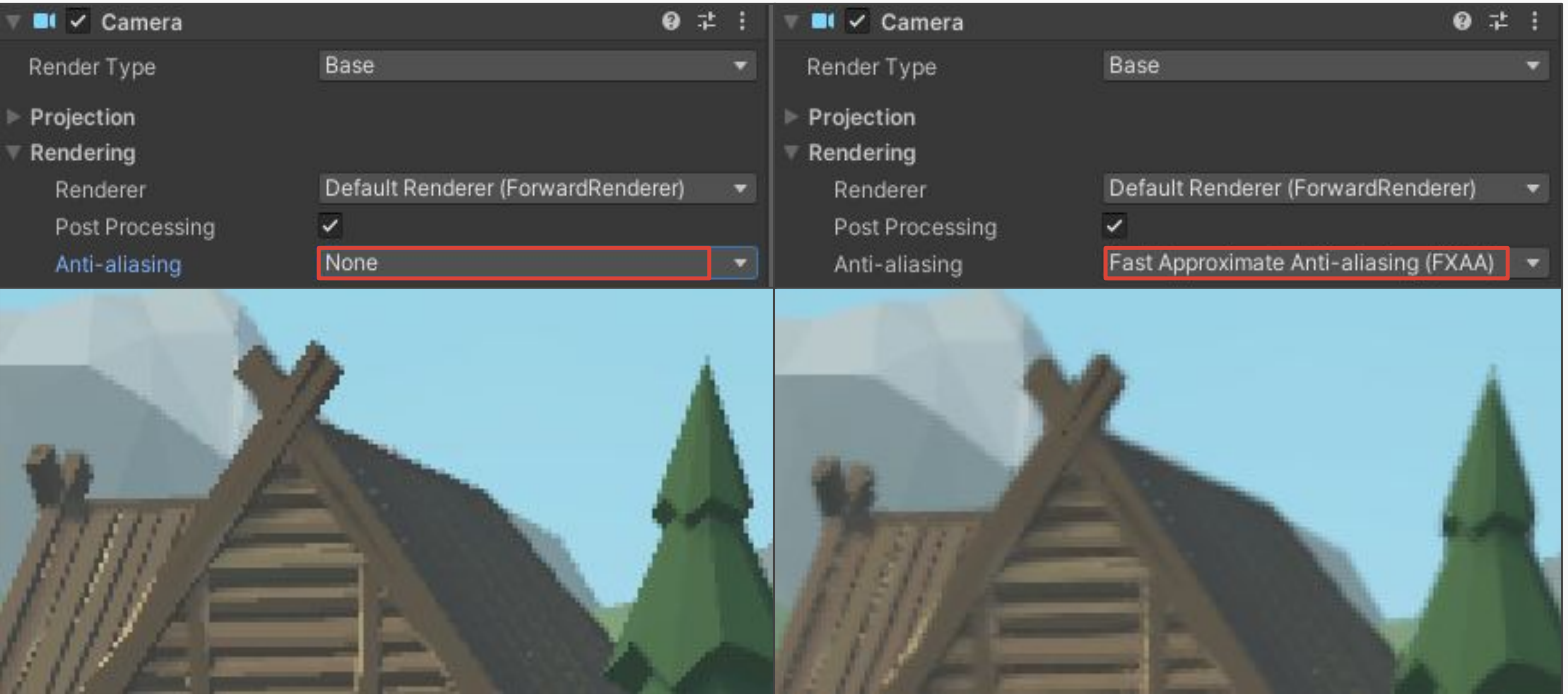
There are a couple post processing effects do not need a Volume and are applied directly on the Camera, although the majority are only available using the Volume component. You can add these effects simply by pressing the "**Add Override**" button in the inspector.

Post Processing

Anti-Aliasing

The Anti-aliasing effect softens the appearance of edges in your scene.

This is the simplest effect to add. It does not even require a Volume to be present.
It is simply applied by setting the Anti-Aliasing flag on the camera to true.

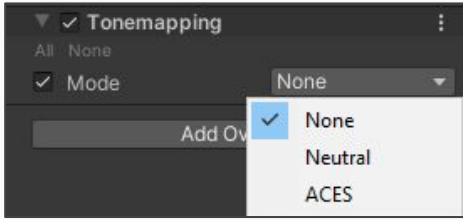


Post Processing

Tonemapping is the process of remapping the **HDR** values of an image to a new range of values. Its most common purpose is to make an image with a low dynamic range appear to have a higher range.

High dynamic range (HDR) is a technique that produces images with a greater dynamic range of luminosity than standard dynamic range (SDR) images, allowing for realistic depictions of color and brightness.

- In standard rendering, the red, green, and blue values of a pixel are stored using an 8-bit value between 0 and 1, where 0 represents zero intensity and 1 represents the maximum intensity for the display device. This limited range of values doesn't accurately reflect the way that we perceive light in real life and leads to unrealistic images when very bright or very dark elements are present.
- In HDR rendering, the pixel values are stored using floating point numbers. This allows for a much larger range of values, which more accurately represents the way that the human eye perceives color and brightness.



To add this effect to our Volume, click "**Add Override**" in the Volume inspector and add it.
We can use the **Mode** property to select a tone mapping algorithm to use for color grading.
The options are:

- **None**: Use this option if you do not want to apply tonemapping.
- **Neutral**: Use this option if you only want range-remapping with minimal impact on color hue & saturation. It is generally a good starting point for extensive color grading.
- **ACES**: Use this option to apply a close approximation of the reference ACES tonemapper, for a more cinematic look. It is more contrasted than Neutral, and has an effect on actual color hue & saturation. If you use this tonemapper, Unity does all the grading operations in the ACES color spaces, for optimal precision and results.
  https://www.oscars.org/science-technology/sci-tech-projects/aces

Post Processing

Vignette

This effect darkens the edges of an image. You can add this to apply a very nice dramatic effect to the output image and can be used to draw more attention to the center of the screen.

To add this effect, click "**Add Override**" in the Volume inspector and add it.
Properties:

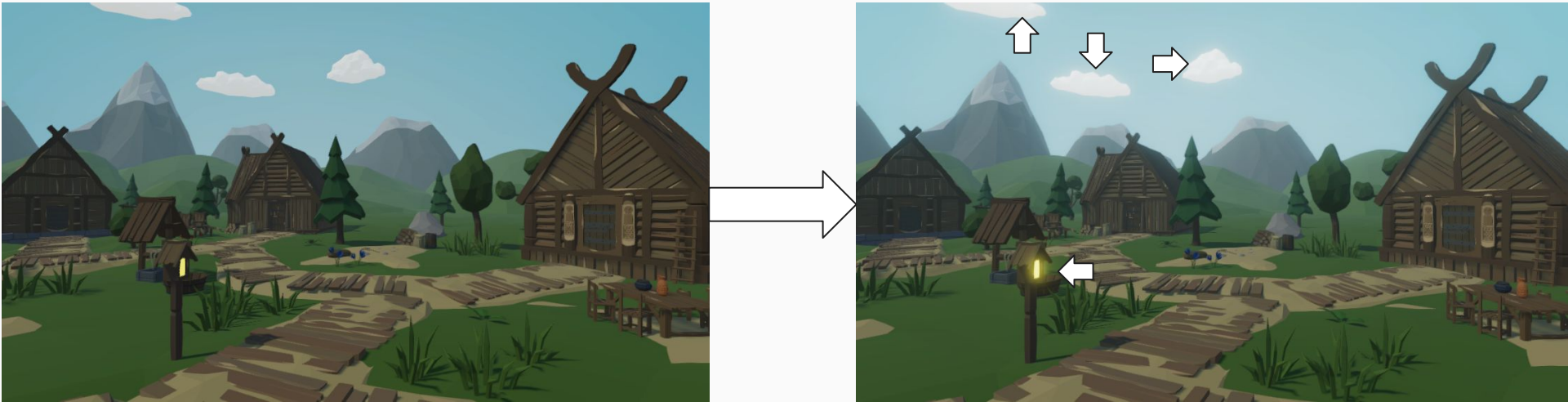
- **Color**: Use the color picker to set the color of the vignette.
- **Center**: Set the vignette center point. The center of the screen is [0.5, 0.5].
- **Intensity**: Set the strength of the vignette effect.
- **Smoothness**: Use the slider to set the smoothness of the vignette borders. Values range between 0.01 and 1. The higher the value, the smoother the vignette border. The default value is 0.2.
- **Rounded**: When enabled, the vignette is perfectly round. When disabled, the vignette matches the shape on the current aspect ratio.

Post Processing

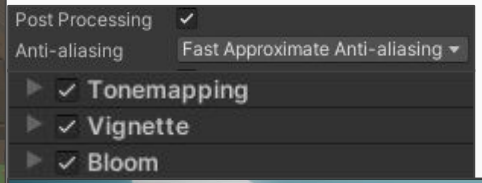
The **Bloom** effect allows bright areas in an image to bleed into the surrounding pixels making it seem like they are glowing creating the illusion of extremely bright light overwhelming the Camera.

To add this effect, click "**Add Override**" in the Volume inspector and add it.
The bloom effect has a few properties, although these three are the most prominent:

- **Threshold**: Set the gamma space brightness value at which URP applies Bloom. URP does not apply Bloom to any pixels in the Scene that have a brightness lower than this value. The minimum value is 0, where nothing is filtered. Default value is 0.9.
- **Intensity**: Set the strength of the Bloom filter, in a range from 0 to 1. The default is 0, which means that the Bloom is disabled.
- **Scatter**: Set the radius of the bloom effect in a range from 0 to 1. Higher values give a larger radius. The default value is 0.7.



Post Processing



Customizing the Editor

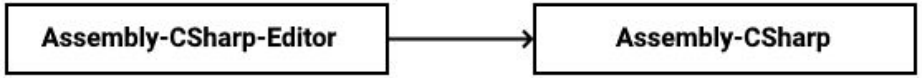
Unity lets you extend the editor with your own custom inspectors and Editor Windows and you can even define how custom properties are displayed in the inspector with custom Property Drawers.

The “Immediate Mode” GUI system (also known as **IMGUI**) is an entirely separate feature to Unity’s main GameObject-based UI System. IMGUI is a code-driven GUI system, and is mainly intended as a tool for programmers. It is driven by calls to the **OnGUI()** and similar functions which are called every time the Editor ticks (updates). IMGUI is mostly used for extending the Unity Editor itself (do not use this for your game’s GUI).

Rather than creating GameObjects, manually positioning them, and then writing a script that handles its functionality, IMGUI allows you to do everything at once with just a few lines of code or a single function call. However, there are some style/layout issues that come with IMGUI that can be quite tricky to debug due to lack of documentation or the functionality you are seeking is simply not built in. This is usually fine as style is not as important when developing tools for programmers.

For more information on what controls (buttons, labels, etc...) are available with IMGUI and how to implement them see the docs : https://docs.unity3d.com/Manual/GUIScriptingGuide.html

Remember, when creating Editor-Only scripts they should be within an Editor folder in your project so that they are excluded from the build of your game. Also, these scripts are automatically grouped up and placed within a separate Editor-Only Assembly.



We will look at creating some custom editor scripts together in the lab, although if you’re interested you can read more about extending the editor in the docs :
Unity Manual : https://docs.unity3d.com/Manual/ExtendingTheEditor.html
Scripting API : https://docs.unity3d.com/ScriptReference/GUI.html , https://docs.unity3d.com/ScriptReference/GUILayout.html
Editor-Only API : https://docs.unity3d.com/ScriptReference/EditorGUI.html , https://docs.unity3d.com/ScriptReference/EditorGUILayout.html

Tasks

- Create a basic URP project using the template provided by Unity.
- Open up the provided lab project and play with some post processing effects.
- Create a custom editor window that allows you to rename multiple objects in the scene hierarchy at once.
- Create a custom inspector for the Player class.
- Create a custom property drawer

Thank You

Link to Lab Drive

https://drive.google.com/drive/folders/1jgFTWCJ5JD-JnIY7n_L7s2n2vA1F9nNX?usp=sharing

OR

https://tinyurl.com/ta-comp476-daniel

Other Links

https://docs.unity3d.com/2020.3/Documentation/Manual/index.html

https://docs.unity3d.com/2020.3/Documentation/Manual/ExecutionOrder.html

https://docs.unity3d.com/Manual/RootMotion.html