# Advanced 3D Game Development

CA1: Rendering Foundations Report

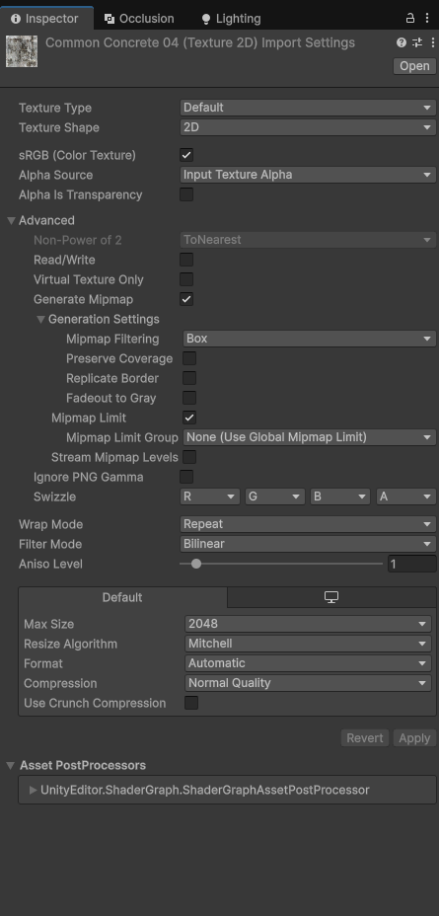
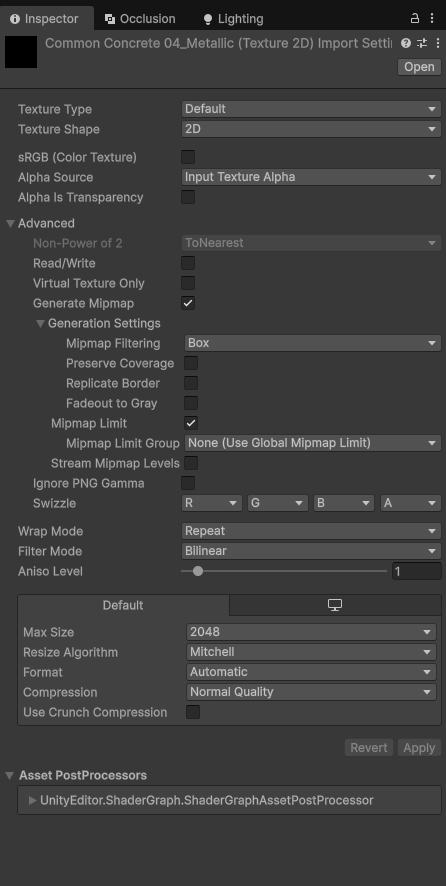
Name: Sam Lyster Cummins

Student Number: 20102256

YouTube Link: <https://youtu.be/tcuLzIU3FyA>

## 1. PBR Texture Import Settings

In Unity's PBR workflow, textures need different import settings depending on whether they represent visual colors or technical data. Getting these settings right is important for accurate rendering.

**Color Maps (Albedo/Base Color)**

Albedo textures contain the actual RGB colors we see on surfaces. These are created in sRGB color space, which is how cameras capture images and how monitors display them. Unity's rendering pipeline works in linear space for accurate lighting calculations, so the sRGB checkbox needs to be enabled. This tells Unity to convert the texture from sRGB to linear space during import.

**Data Maps (Metallic, Roughness, AO)**

Textures like Metallic, Roughness, and Ambient Occlusion store values for shader calculations rather than visual colors. A metallic map value of 0.5 means '50% metallic', it's a number for the renderer to use, not a gray color to display. These textures are used in linear space and need to stay linear, so sRGB must be disabled If you turn sRGB on it messes up the values and surfaces look too rough or shiny.

**Normal Maps**

Normal maps store surface bumps as directional vectors in RGB. The texture type needs to be set to Normal Map so Unity compresses them properly. Like data maps, sRGB stays off or it breaks the directions.

## Why This Matters

Correct import settings ensure materials respond accurately to lighting. Wrong settings break the accuracy that PBR is based on, materials might look less metallic, surfaces become too rough, or colors are off.

**2. Shader Graph: Dissolve/Burn-Out Effect**

## The dissolve shader makes objects gradually disappear with a glowing burn edge. The effect uses exposed parameters that get animated by a C# script with MaterialPropertyBlock.

## Shader Graph Evidence

## Complete Shader Graph Overview

## 

## Material Inspector - Exposed Parameters

## 

The shader samples noise and compares it to a threshold to figure out what parts dissolve. It makes a threshold window by subtracting edge width from dissolve amount which creates a band where pixels transition. Inside the band it blends the edge colors and multiplies by emission strength for the glow. Edge width changes how thick it looks and the colors let you pick the style. As dissolve goes 0 to 1 it moves across automatically.

The DissolveEffect.cs script animates it with MaterialPropertyBlock. I used MaterialPropertyBlock because it changes values without making new materials which keeps things fast. The script has play, reset, and replay and can start from keyboard or automatically.

## 3. Dusk Lighting Setup

## Labeled Lights – Left Side of Warehouse

## 

## Labeled Lights – Right Side of Warehouse

## 

## Labeled Lights – Birds eye view of Warehouse

## 

## The warehouse scene uses dusk lighting with a low-angle directional key light (Key\_DuskSun) and gradient ambient lighting to establish the time of day.

## Helper lights are positioned to create motivated lighting. Four practical work lights are evenly spaced along the warehouse ceiling, each using volumetric fog and emissive bulbs to create visible beams. Four interior fill lights are positioned between the practical lights to reinforce the impression that the fill illumination comes from those fixtures rather than appearing unmotivated. Two rim lights are placed on opposite sides of the warehouse and angled inward to provide silhouette separation regardless of player position. A central reflection probe with box projection encapsulates the entire warehouse interior.

## The practical work lights use warm white with soft shadows and the imported volumetric fog asset to create visible beams with emissive bulb materials. The fill lights use cool blue without shadows for performance. The rim lights also skip shadows since they primarily provide edge highlights rather than scene illumination. Volumetric fog is also applied to the directional sun light to create atmospheric rays entering through warehouse windows.

## Also for an extra touch, I added a particle system which slowly emits dust drifting across the warehouse, making the light beams more visible throughout the space and add to the abandoned industrial setting.

## I made deliberate trade-offs for performance. Most helper lights don't cast shadows since they primarily provide visibility rather than primary illumination, saving shadow map overhead. The volumetric fog is limited to specific lights rather than global fog to maintain frame rate while keeping the atmospheric effect where it's most visible.

## 4. Post-Processing Volume

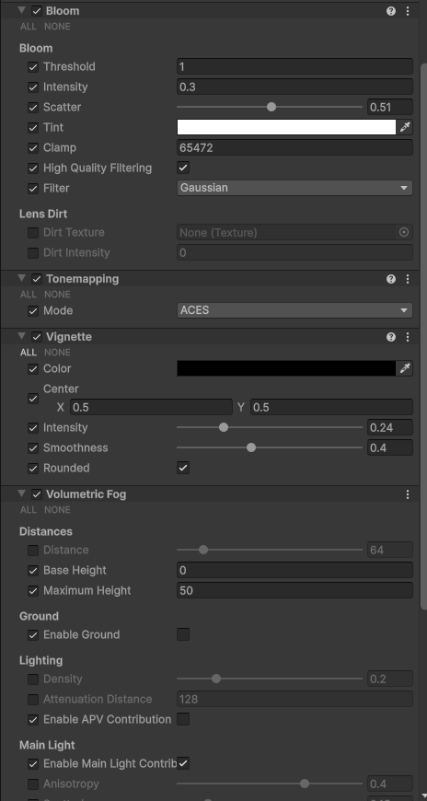
## Before Global Post-Processing

## 

## After Global Post-Processing

## 

Global Post-Processing Overrides



The scene uses a Global Volume with ACES tonemapping, vignette, and bloom configured for the dusk warehouse atmosphere.

ACES tonemapping handles the contrast between bright sunlight and dark warehouse areas without clipping highlights. Bloom at threshold 1.0 and intensity 0.3 makes the emissive light bulbs glow subtly. Vignette with intensity 0.24 and smoothness 0.4 draws focus toward the center gameplay area.

Volumetric fog is an imported asset applied to the directional sun light and practical work lights to create visible light beams. This makes the lighting feel more atmospheric and helps show where light is coming from.

Together these create the dusk warehouse atmosphere while keeping visibility clear for gameplay.

## 5. Unreal Engine 5 Comparison

## Dissolve Material – Unreal Engine

## 

## Texture Workflow

## The barrel textures imported into Unreal without issues. The base color and normal maps worked the same in both engines. The main difference was the metallic texture. Unity uses smoothness (which is inverted roughness) in the alpha channel, while Unreal expects actual roughness data. For the barrel, I connected the metallic texture's red channel to both the Metallic and Roughness inputs in Unreal, which worked fine since the texture was PBR format.

## Material Editor Differences

## Unreal's Material Editor and Unity's Shader Graph are very similar. The main terminology differences were "Texture Sample" instead of "Sample Texture 2D" and "Scalar Parameter" instead of "Float". The math nodes (Multiply, Subtract, etc.) work identically between engines.

## The biggest structural difference was handling transparency. Unity uses an Alpha Clip Threshold property that's built into the shader settings. Unreal requires you to manually set the material's Blend Mode to Masked and connect your dissolve logic to the Opacity Mask output. It's more explicit but gives you better control over what's happening.

## Unity's Shader Graph has a procedural Simple Noise node that generates noise at runtime. Unreal doesn't have this, well that I know of, so you need to import a noise texture as an asset beforehand.

## Key Technical Differences

## Emission intensity works differently between engines. Unity uses lower values (0-5 range typically) while Unreal needs much higher values to get the same visual brightness. My EdgeIntensity parameter was set to 10.0 in Unreal but would be around 2.0 in Unity for similar results.

## Material instances also work differently. In Unity you can adjust material properties directly on objects. Unreal uses a parent-child system where you create a Material Instance from the base material first, then adjust parameters on the instance. It's an extra step but keeps things more organized.

## Production Standardization

## For a cross-engine pipeline, texture naming should be consistent, use "\_BaseColor" and "\_Roughness" instead of Unity's "\_Albedo" and "\_Smoothness". Texture authoring should use standard roughness maps rather than inverted smoothness, as this matches Unreal and most texture creation software like Substance Painter. Parameter names should be identical across engines so materials port more easily.

## 6. Assets

Assets used:

Volumetric Fog - <https://github.com/CristianQiu/Unity-URP-Volumetric-Light.git>

First Person Controller - <https://assetstore.unity.com/packages/essentials/starter-assets-firstperson-updates-in-new-charactercontroller-pa-196525>

Swat Character and Animations - <https://www.mixamo.com/>