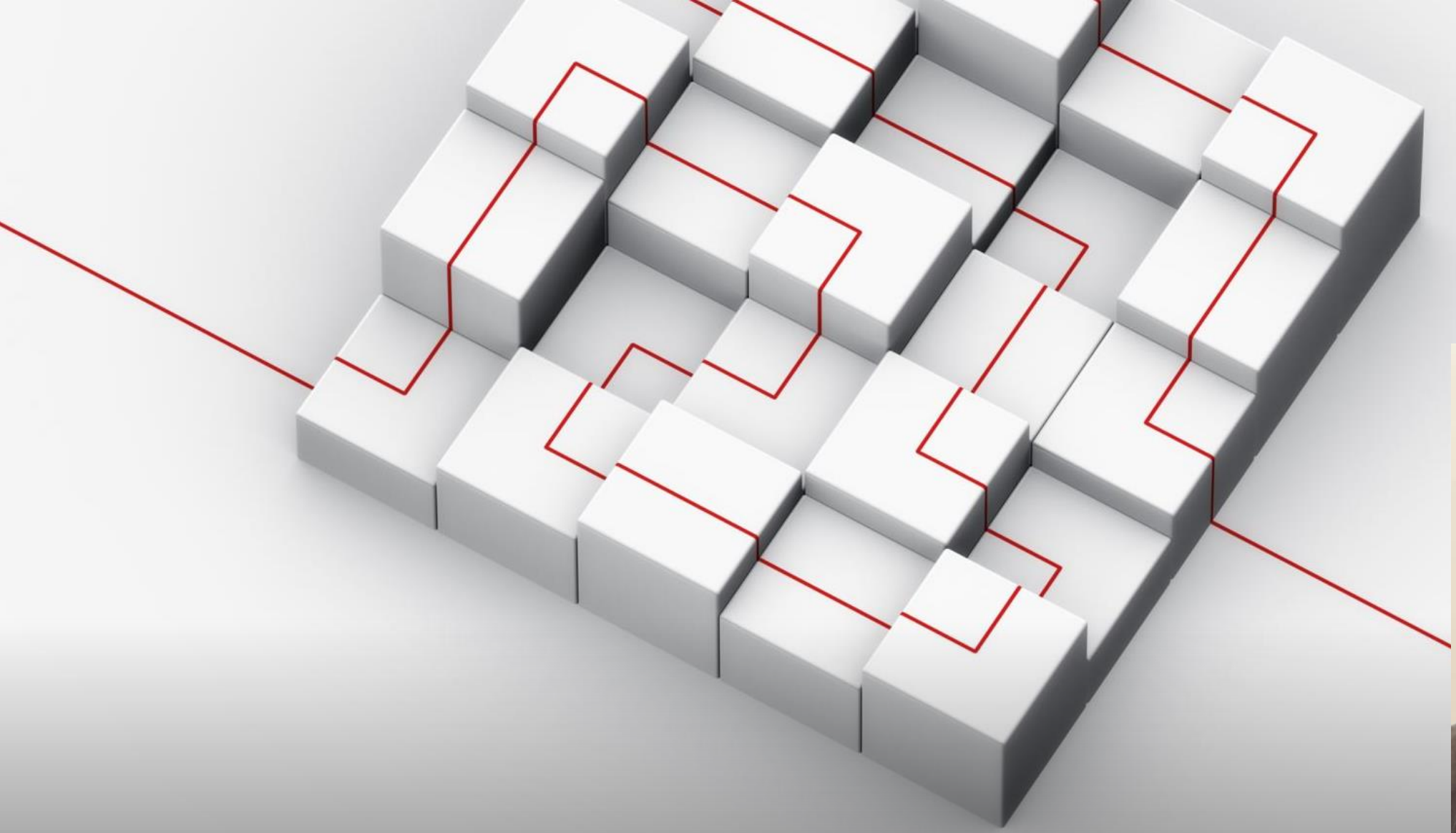


# CG – Project Progression 2025

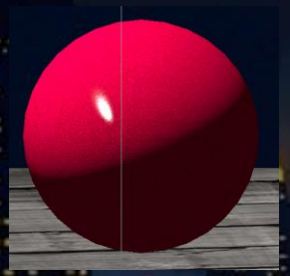
Evangelos Angelou

100876023

# Project Overview

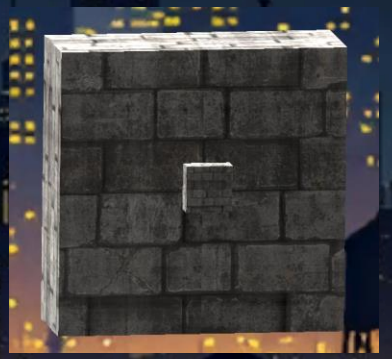
## Part 1: Base Scene

- 2.5D Obstacle Course with Dynamic Objects
- Playable Character and Win Condition



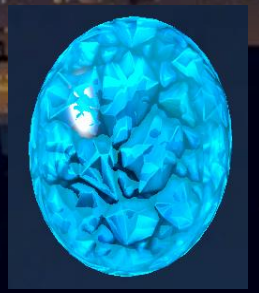
## Part 2: Illumination System

- 5 Toggleable Lighting Modes
- Triplanar texture mapping



## Part 3: Custom Shaders

- Normal Mapping
- Rim Lighting



## LIVE DEMO


# Base Scene

## Scene Design

- 2.5D low-poly obstacle course.
- Somber, relaxing atmosphere.
- 3D rain particle effects.
- Background Music - *Gymnopédie No. 1*

## Dynamic Elements

- Rotating central platform.
- Patrolling cube enemies.
- Floating collectible crystals.

## Character

- Shiny red ball (ideal for showing specular highlights)

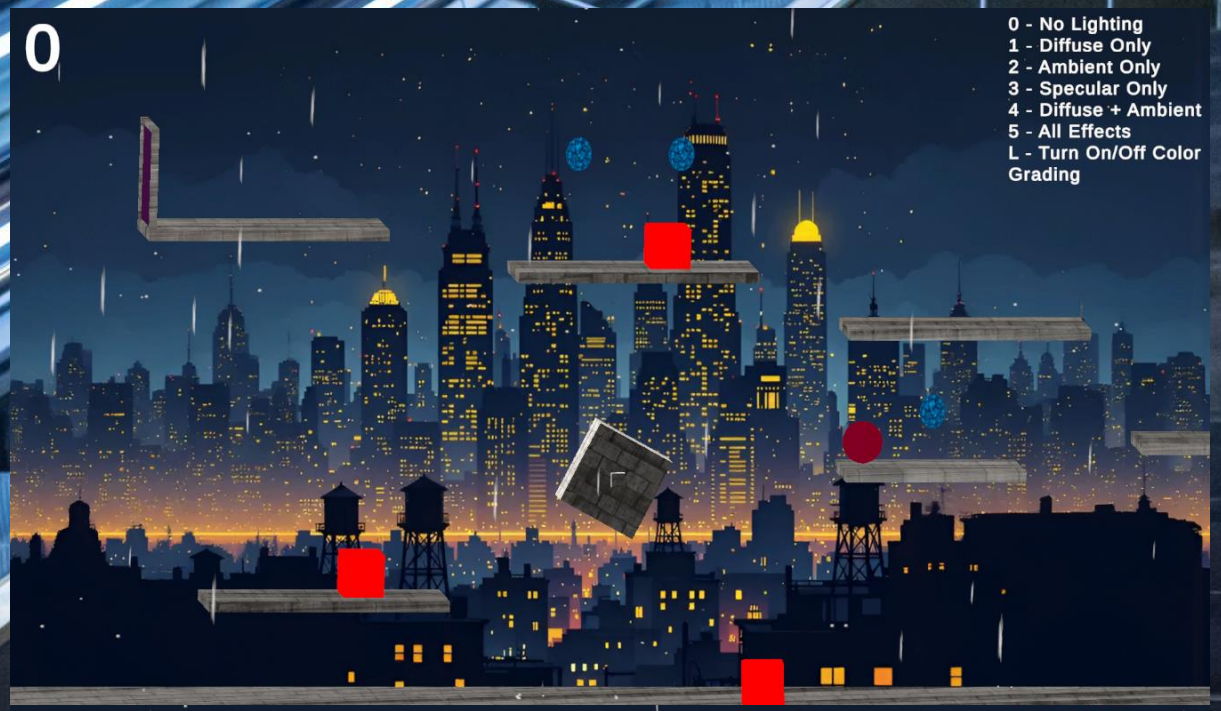
## Win Condition

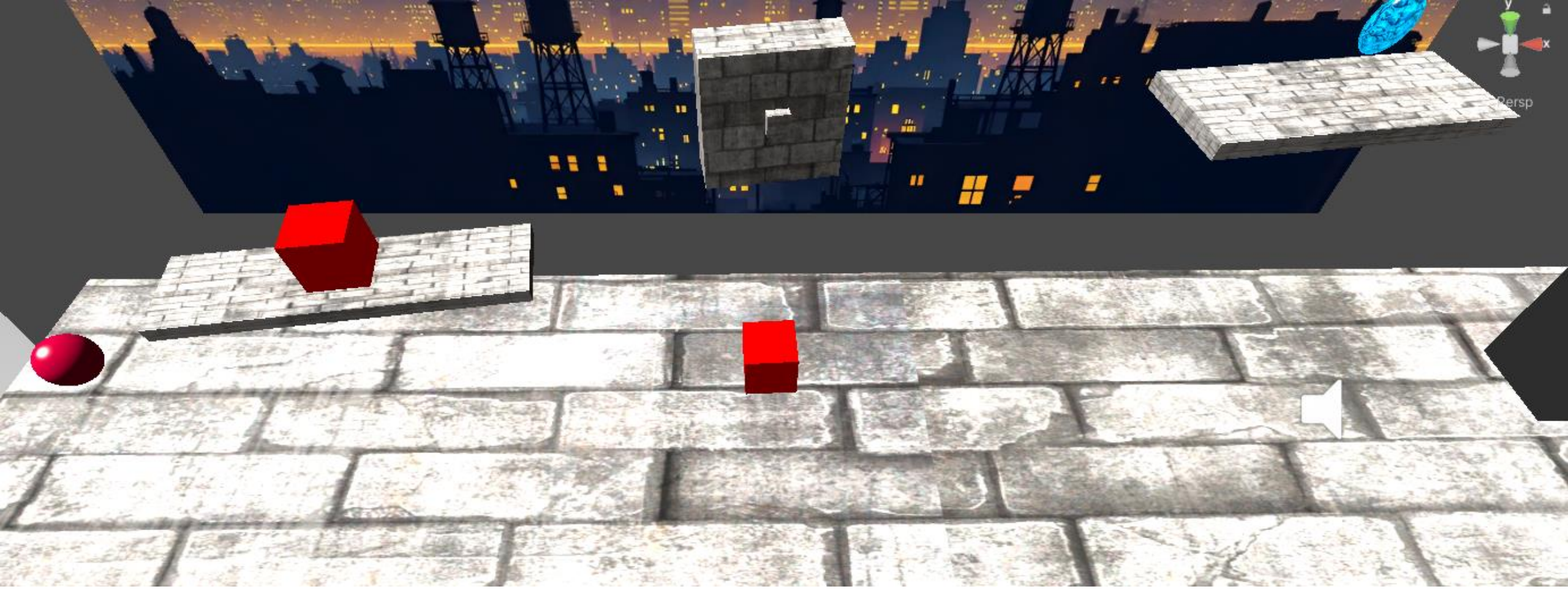
- Reach the end portal without getting hit by an enemy.

# Unified Lighting System

| Keybinds | Lighting Mode | Components |
|---|---|---|
| 0 | Albedo Only | Texture Only |
| 1 | Diffuse Only | Lambert (N*L) |
| 2 | Ambient Only | Ambient Light |
| 3 | Specular Only | Phong Specular |
| 4 | Diffuse + Ambient | Combined Diff + Amb |
| 5 | Full Lighting (Default) | All Components |

```
Shader "URP/Part2_UnifiedLighting"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _BumpMap ("Height/Bump Map", 2D) = "bump" {}
        _Color ("Color", Color) = (1,1,1,1)
        _AmbientStrength ("Ambient Strength", Range(0,1)) = 0.4
        _SpecularStrength ("Specular Strength", Range(0,1)) = 0.5
        _Shininess ("Shininess", Range(1,128)) = 32
        _TextureScale ("Texture Scale", Float) = 1.0
        _Mode ("Lighting Mode", Int) = 5
        _BumpStrength ("Bump Strength", Float) = 1.0
    }
```




# The Triplanar Mapping Challenge

## The Problem

- ***Standard UV Mapping Fails on Dynamic Objects***

**Traditional UV Mapping Issues**

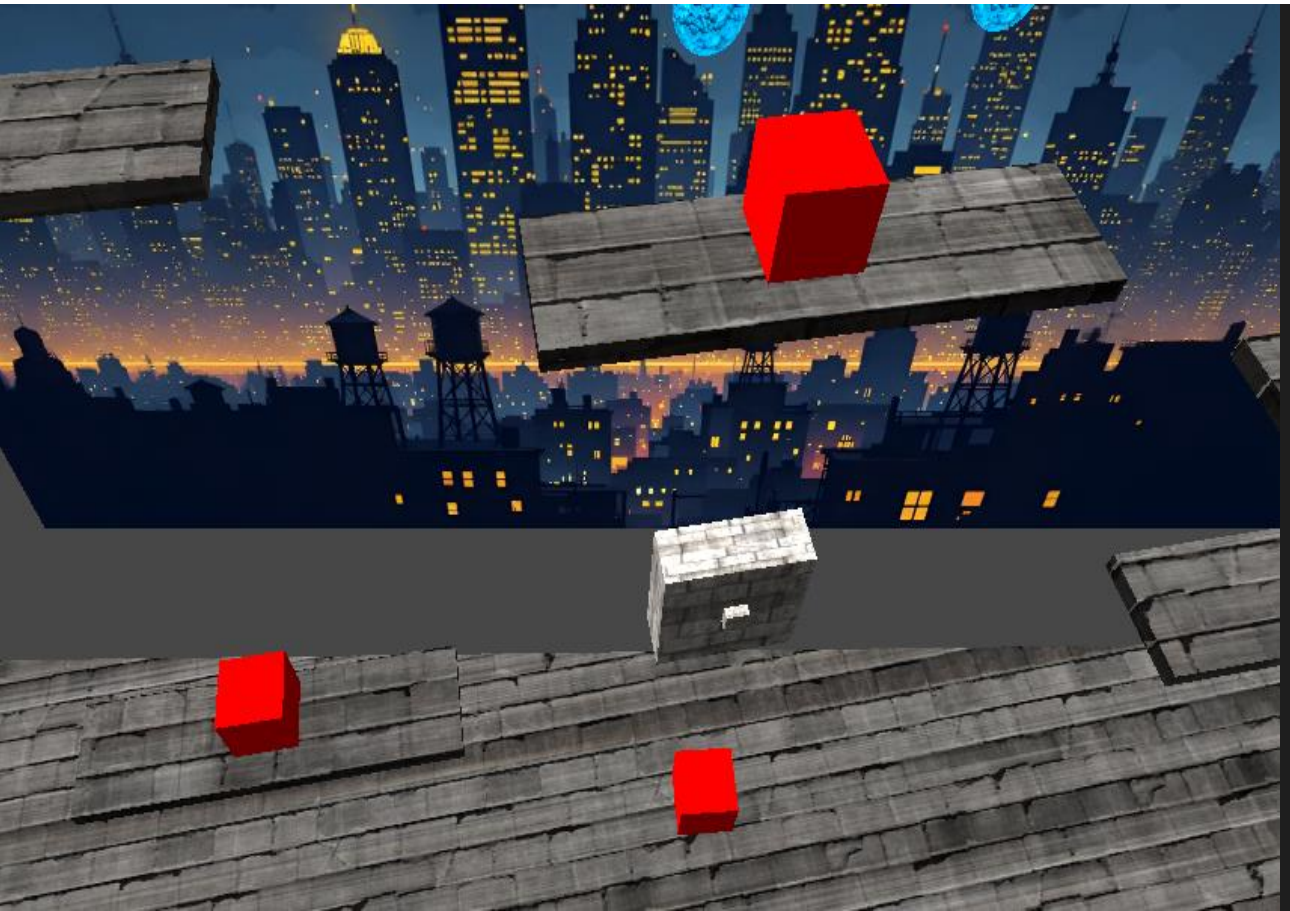
- Textures "slide" across moving objects as they rotate, Heavy distortion on objects with bad UV coordinates (such as the gems).
- Inconsistent texture scale across different models resulting in Stretching and warping on complex geometry.

# The Solution

## Object Space Triplanar Mapping

**How it Works:**

- Projects texture from 3 axes simultaneously (X, Y, Z)
- Blends projections based on surface normal direction.
- Uses object-space coordinates (locked to object).
- Texture moves, rotates, and scales with the object.



```
half3 Triplanar(TEXTURE2D(tex), SAMPLER(samplerTex), float3 worldPo
{
    float3 pos = RotateX(worldPos, radians(10));
    pos = RotateY(pos, radians(10));

    float3 blend = abs(normal);
    blend /= (blend.x + blend.y + blend.z + 1e-5);

    float2 uvX = pos.yz * scale;
    float2 uvY = pos.xz * scale;
    float2 uvZ = pos.xy * scale;

    half3 sampleX = SAMPLE_TEXTURE2D(tex, samplerTex, uvX).rgb;
    half3 sampleY = SAMPLE_TEXTURE2D(tex, samplerTex, uvY).rgb;
    half3 sampleZ = SAMPLE_TEXTURE2D(tex, samplerTex, uvZ).rgb;

    return sampleX * blend.x + sampleY * blend.y + sampleZ * blend.
}
```

# Colour Grading – Warm LUT

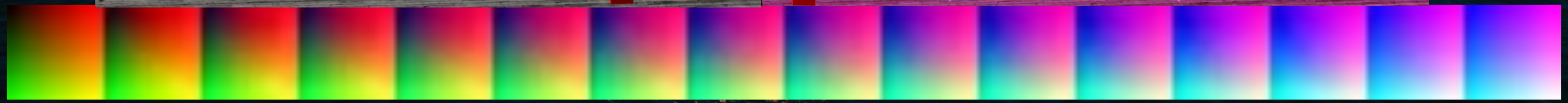
**What is a LUT?**

- 3D color transformation stored as a 2D texture.
- Maps input RGB values to output RGB values.
- Used in films and games for post-processing color grading.

**Why a Warm LUT?**

- Transforms the Scene from mellow to passionate. Warmer Colours mean urgency and intensity.
- Gives a "Cinematic" Post-Processed feel often seen in films.





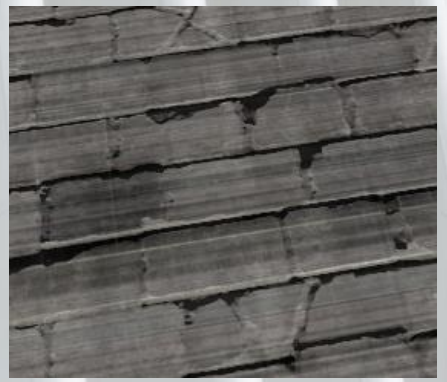
# Custom Shaders

## Normal Mapping

### Implementation Process:

1. Sample the Normal Map using Triplanar Projection.
2. Create TBN Matrix (Tangent-Bitangent-Normal)
3. Transform Normal from Tangent to World Space.
4. Use the new Normal in all lighting calculations.

**Without Scaling**



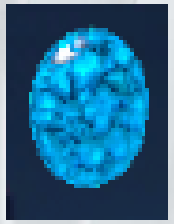
**With Scaling**

## Rim Lighting

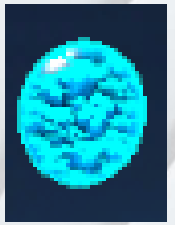
### Mathematical Formula:

```
half rim = 1.0 - saturate(dot(normal, viewDir));
rim = pow(rim, _RimPower);
half3 rimLight = _RimColor.rgb * rim * _RimIntensity;
```

- When viewing the surface from head-on: dot product is approx. 1, rim is approx. 0
- When viewing 90 degrees from an object: dot product is approx. 0, rim is approx. 1
- _RimPower controls how wide/tight the rim glow is, _Rim Intensity controls overall brightness and _RimColour tints the glow.



**Low Intensity**



**High Intensity**