Writeup: Homework #2

Malcolm Riley

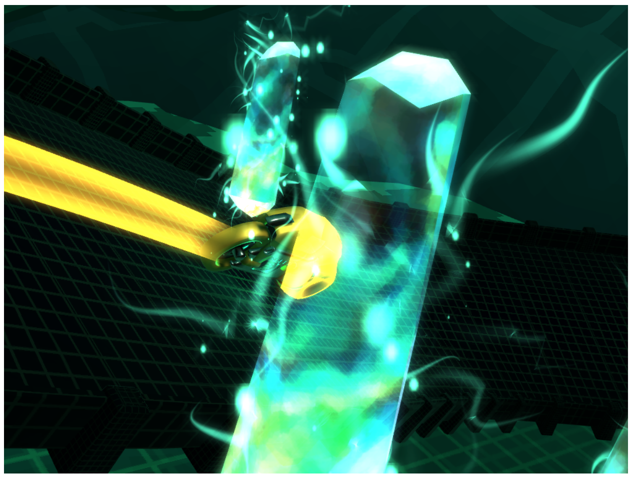
May 12, 2019

Introduction

As with the previous assignment, all subtasks of this assignment have been implemented as a single Unity Scene. No third-party Unity packages were used to create this scene. All code and scene assets including models and sounds were created by myself for the purpose of this assignment.

The controls necessary for interacting with the scene are depicted onscreen, but the mouse can be used to look in a particular direction, and the shift key can be pressed to zoom towards the vehicle and slow time.

A. Tron Filter



Bloom Filter

To implement the bloom filter, I created a second "VFX Camera" object. This camera's culling mask is set so that it only draws objects on the "Transparent FX" render layer, and the main camera is set to not render objects on that layer. The "VFX Camera" is furthermore configured to output to an intermediate `RenderTexture`. This `RenderTexture` is subsequently blended additively to the main scene render in the `CameraVFX` class, via the `CameraVFXBloom` shader. This shader performs a simple iterative box blur on the `RenderTexture` derived from the "Transparent FX" layer, adding the result to the primary lookup and main texture.

While this approach does not use emission properties in any way, a benefit is that it is trivially easy to add bloom to a particular object: simply switch the object's render layer to "Transparent FX". An obvious shortcoming is that since the "Transparent FX" layer is additively blended to the main scene, objects on this layer will appear transparent and emissive, and will not be properly occluded by main scene geometry (since a separate depth buffer is used for each camera's render target). These shortcomings were acceptable in the limited case of this assignment.

Rim Shader

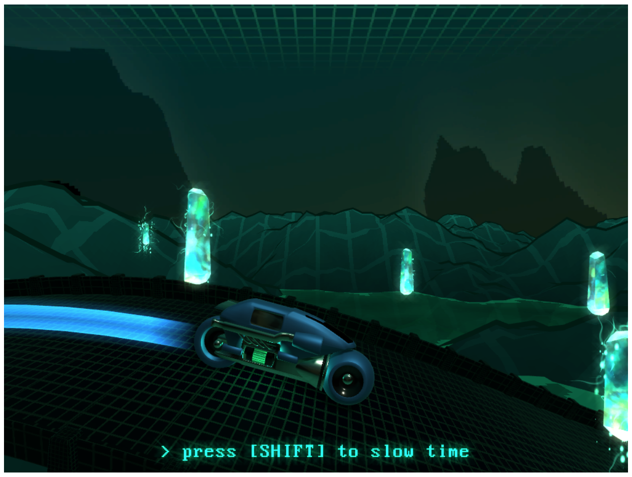
The rim shader was implemented as a custom material applied to the desired mesh. The result was achieved via the simple dot product of the world space normal and world space view vector, multiplied by the desired color and an arbitrary intensity factor.

To achieve the extra credit requirement, the stencil buffer was employed in the `GlowOverlay` and `VFXOccluder` shaders. While admittedly, the name is a bit of a misnomer, the stencil operation in the `VFXOccluder` shader permits the `GlowOverlay` render to persist whenever the two meshes occupy the same screen space, and the `VFXOccluder`-rendered mesh is in front of the `GlowOverlay`-enabled mesh.

Links

Camera VFX Adder: `CameraVFX.cs`
Bloom Material Shader: `CameraVFXBloon.shader` Rim Shader: `GlowOverlay.shader`
Rim Shader Complement Occluder: `VFXOccluder.shader`

B. Outdoor 3D Scene



Height Map

The terrain heightmap deformation was implemented as a vertex-displacement shader applied to a flat plane mesh. This shader samples the red channel from an arbitrary image, and offsets the vertex by this channel value times an arbitrary “Height” property.

Water

The water shader samples the skybox cubemap in order to give the appearance of reflecting the sky. To produce the pixelated “edge foam” effect, the shader samples from the same heightmap texture as the terrain – beyond the threshold “Height” property, the water fragment shader draws a lighter color. This threshold is varied slightly over time for a subtle motion effect.

Although the shader does not technically meet the extra-credit requirement of refracting objects *beneath* the water’s surface, the shader does employ the `refract()` function add limited apparent motion to the water surface.

Links

Heightmap Mesh Deformation Shader: `TerrainDisplace.shader`
Cubemap-Reflecting/Refracting Water Shader: `WaterShader.shader`