Writeup: Homework #3

Malcolm Riley

May 26, 2019

Introduction

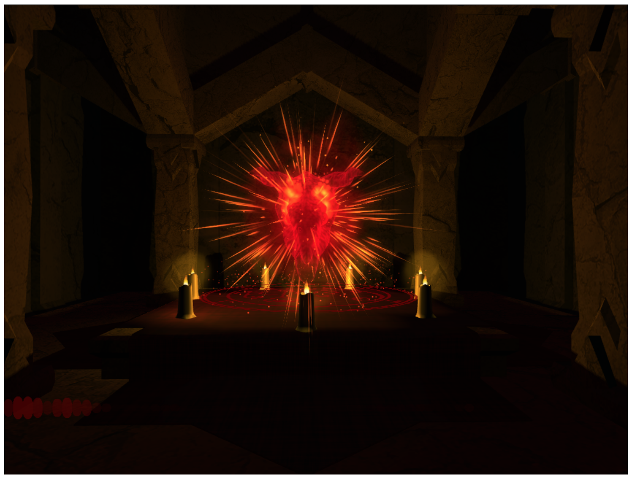
As with the previous assignment, all subtasks of this assignment have been implemented as a single Unity Scene. Code and 3D assets were created by myself for the purpose of this assignment.

The music and demon sigil image are not original. They were found on the internet at the sites below:

Music: `https://soundcloud.com/infiniteimprobability/ghost-year-zero-dark-synthwave-cove`
Beleth Seal: `https://commons.wikimedia.org/wiki/File:Beleth_seal.jpg`

There are no controls implemented for interacting with the scene.

A. Particle and Noise Scene



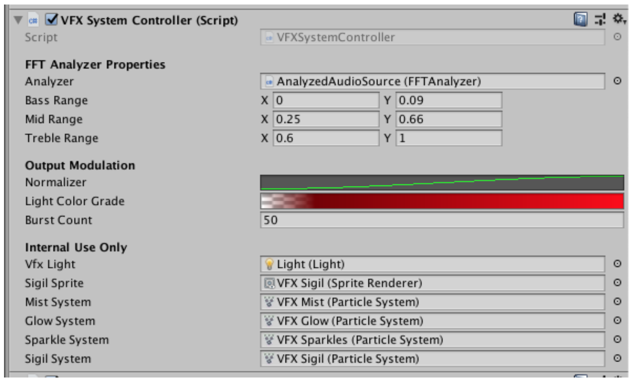
Naturalistic Particle System Phenomena

There are quite a few particle systems active in the main scene. The attempted naturalistic phenomena include smoke/mist, flames, or rays of light, and airborne cinders.

Noise Function Usage

Noise is used in this project in four capacities. First, the Unity Perlin noise method in `Mathf` is used to generate a 2D noise texture for application to the demon head model. This action is performed in `NoiseTexGenerator.cs`. Second, the same Unity method is employed to add slight camera drift over time in `CameraDrift.cs`. Third, the built-in Unity particle noise functionality is applied to several particle systems, notably the airborne cinders and the sigil particles. Finally, a static noise texture is applied to the red mist/smoke particles in `SmokeShader.shader` in order to emulate wispiness.

Inspector Property Manipulation



VFX System Controller (Script)

Script	VFXSystemController
FFT Analyzer Properties	
Analyzer	AnalyzedAudioSource (FFTAnalyzer)
Bass Range	X 0 Y 0.09
Mid Range	X 0.25 Y 0.66
Treble Range	X 0.6 Y 1
Output Modulation	
Normalizer	
Light Color Grade	
Burst Count	50
Internal Use Only	
Vfx Light	Light (Light)
Sigil Sprite	VFX Sigil (Sprite Renderer)
Mist System	VFX Mist (Particle System)
Glow System	VFX Glow (Particle System)
Sparkle System	VFX Sparkles (Particle System)
Sigil System	VFX Sigil (Particle System)

I make a habit of exposing as many properties in the Unity Inspector view as possible; for the sake of brevity I will only discuss one example.

The above image is a screenshot of the `VFXSystemController.cs` script in the Inspector view. As can be seen, there are several assignable reference fields marked "Internal Use Only". These are references to the various objects in the scene that the script controls.

In the "FFT Analyzer Properties" section, there are several `Vector2` instances that represent the various proportions that the `VFXSystemController.cs` will attempt to use in order to interpret bass, midrange, and treble from the `FFTAnalyzer.cs` referenced object. This decouples the interpretation of the analyzed spectrum data from the precise number of elements in the spectrum array – if the number of elements in the array is changed, the script will continue to work without edits. For instance, in the above image, the bottom 9% of the spectrum data array will be averaged and considered to be the "Bass Range", and the top 40% will be considered "Treble". This approach allows on-the-fly fine-tuning of the audio interpretation, for use in tailoring to specific songs.

The "Output Modulation" section features a normalization curve, color gradient, and integer property field for modifying the interpreted audio signal, coloring light and texture based on an input signal, and setting the maximum number of particles to be emitted in a audio-responsive burst. These features are discussed in greater detail in the next section; however they are mentioned here as an example of properties exposed in the Inspector for "controlling some parameters of the scene".

Dynamic Audio Response

The elements of the scene that respond to audio are all controlled in the `VFXSystemController.cs` script. The actual control occurs on lines 32-47:

- A point light is updated based on the interpreted bass-range audio signal. The light color is interpolated from the gradient object supplied in the Inspector, using the average detected bass signal. This causes a subtle light burst in the room.

- A particle system emits a number of particles based on the current-tick interpreted midrange signal. This causes a flurry of ray particles from the demon skull whenever the main melody picks up, or whenever a cymbal crash occurs.

- The color of a SpriteRenderer is updated using the same color interpolation as the point light. This causes the sigil to pulsate in time with the kick drum (for the most part).

- A second particle system emits a number of particles based on the current-tick interpreted treble signal. This causes a flurry of particles to emit from the floor when high-pitched sounds occur in the music.

B. Final Project Planning

Crepuscular Ray Methodology

There appear to be several different approaches to the implementation of Crepuscular Rays:

- **Screen-Space Post Procesing:** The stencil buffer is used by all "obscuring" elements of the scene to prepare a texture consisting of the distant light point obscured by the silhouettes of all near objects. This texture is then iteratively scaled outwards from the distant light point and blended additively with the scene in multiple steps. With enough iterations, this produces ray-like blurs of light surrounding the outlines of near objects. This method works best when the distant light is directly ahead of the viewer, and may result in artifacts if insufficient scaling steps are used. Additionally, the technique requires that the source of light is within the render target viewport.

- **Dummy Mesh Volumes:** A tapered cylindrical mesh is oriented lengthwise between the distant light source and a target area. A "Raylike" texture is applied to the rectangular part of the cylinder. The mesh is then blended additively with the scene. This method is incredibly simple to perform in most modern rendering engines but is quite recognizable as "fakery". Applying an animated texture can occasionally improve the effect.

- **Particles:** Stretched particles are emitted radially from the screen-space position of the distant light source. This technique is also incredibly simple to perform in most modern rendering images; however it may be difficult to coax the resulting "rays" into colliding with near objects in an appropriate manner. Furthermore, the effect does not produce firm boundaries between obscuring objects and the ideal light beams.

- **"Inverse" Shadow Volume Technique:** This is essentially the volumetric shadow operation, applied inversely. Draw rays from the light source towards each vertex of each mesh in view, constructing volumes from each identified silhouette edge. Instead of using these volumes to cast shadows, use these volumes to create conical rays of light. This technique can be quite slow if applied to all geometry in the scene, furthermore it is not terribly well suited to crepuscular rays through soft media such as clouds.

- **Full Volumetric Lighting:** By using a light-space shadow map in conjuction with the depth buffer, the entire scene is ray-traced inversely from the objects' shadow map towards the light source. The resulting volumes are blended additively with the scene, with each ray's contribution being sourced from the actual shadow hardness from the original shadow map. This technique is comparatively sophisticated to implement and requires a separate rendering pass (to determine scene lighting from the desired source) but allows for some performance tuning – to decrease performance cost, simply use a lower-resolution shadow map.

Group Members

I will be working with **Jan Yu** and **Anthony Medina**.