Colossal Order Technical Art Test Task

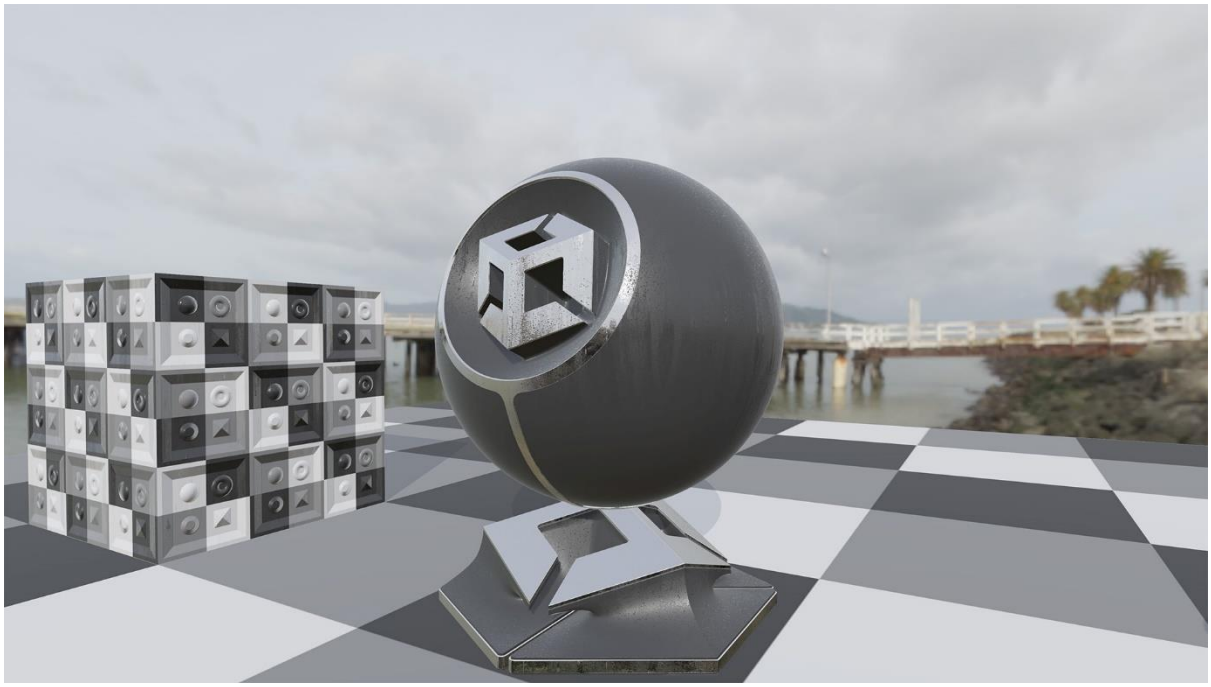

Figure 1 View from the CalibrationScene. The cube on the left is meant to show how the effect blends with normal maps. Unity shader ball is meant to showcase the effect on a complex mesh.

Main shader

The shader can be found in **Assets > Shaders > Rain** folder.

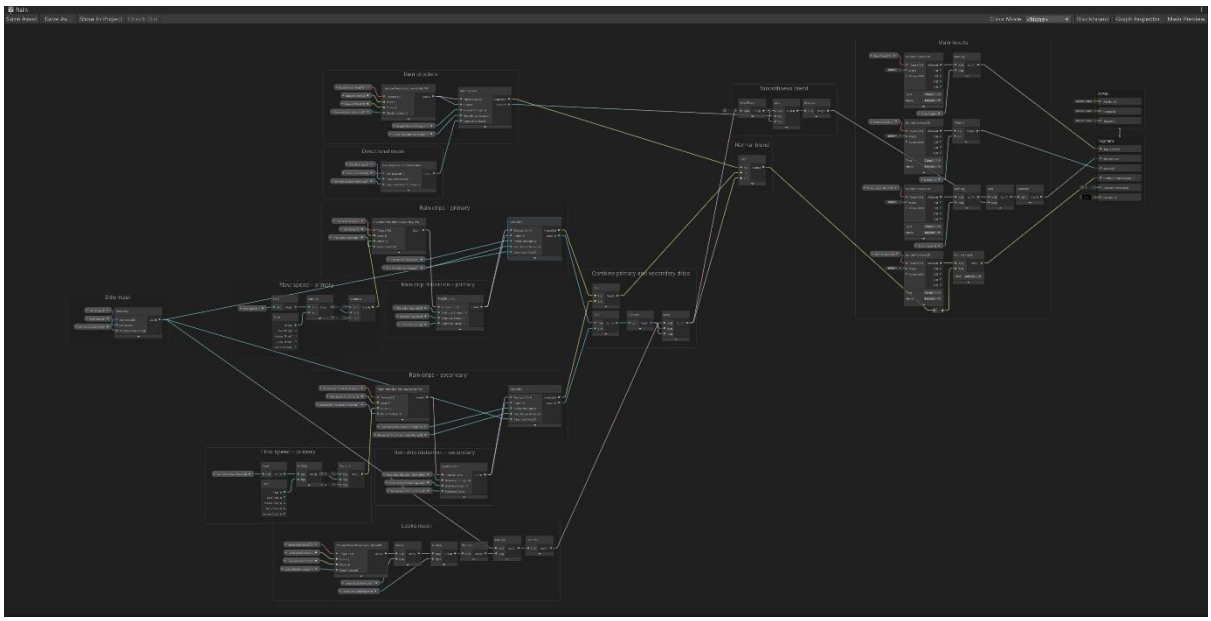

Figure 2 Network of Shader Graph nodes in the Rain shader.

Disclaimer

The entire effect is controlled by a single shader. Even though the task description says that it needs to be driven both by the VFX Graph and Shader Graph, I decided against it for the following reasons:

- Using VFX Graph to achieve the dripping effect would require the use of a dedicated render target (a render texture). Updating the render texture at every frame can be costly, although it depends on multiple factors, such as resolution.
- In addition, this would require a dedicated camera to capture the particle effect and project render it to a render target. Each additional camera in Unity has a performance overhead.
- This approach would also require the effect to be always present in the game environment.

However, there's one big advantage for using VFX Graph to achieve the effect, and that is procedural variation. Achieving the same level of variation using shaders alone would require an overly complex shader with multiple texture samples.

Features

- Works on all geometry, thanks to cheap tri-planar projection.
- The effect is projected in world space. This ensures scale consistency across objects, as it does not rely on object UVs.
- Rain drips are projected only on the sides of the objects. Droplets are projected uniformly.
- Rain drips and droplets can be adjusted by using directional masks.
- Leaks, droplets, and drips can all be adjusted individually.
- The entire effect is an overlay, and it respects base texture inputs, including normal maps.

Shader structure

Main inputs

This category holds the base PBR inputs that define the look of the object.

Quick toggles

Allows users to toggle rain drips and droplets quickly.

Leaks

This category has the grunge mask input. This input is controllable by the user. It influences the smoothness and normals of the rain droplets, and base material inputs.

Drip directional mask

This mask allows users to procedurally control where rain drips and the grunge leaks effect appears on the object.

Rain drips

Rain drips effect is driven by two texture samples. This approach was inspired by Sebastien Lagarde's article on rain effects in Remember Me. Each texture is sampled at a different scale and offset to create variation. Offset is controlled by the time node.

Users can also control the intensity of the rain drop distortion. It's driven by a hard-coded cloudy noise texture.

Droplet directional mask

This allows users to control where the static rain droplets appear on the object.

Rain droplets

Rain droplets effect is driven by a single grayscale texture. It's projected uniformly in world space.

VFX Graph

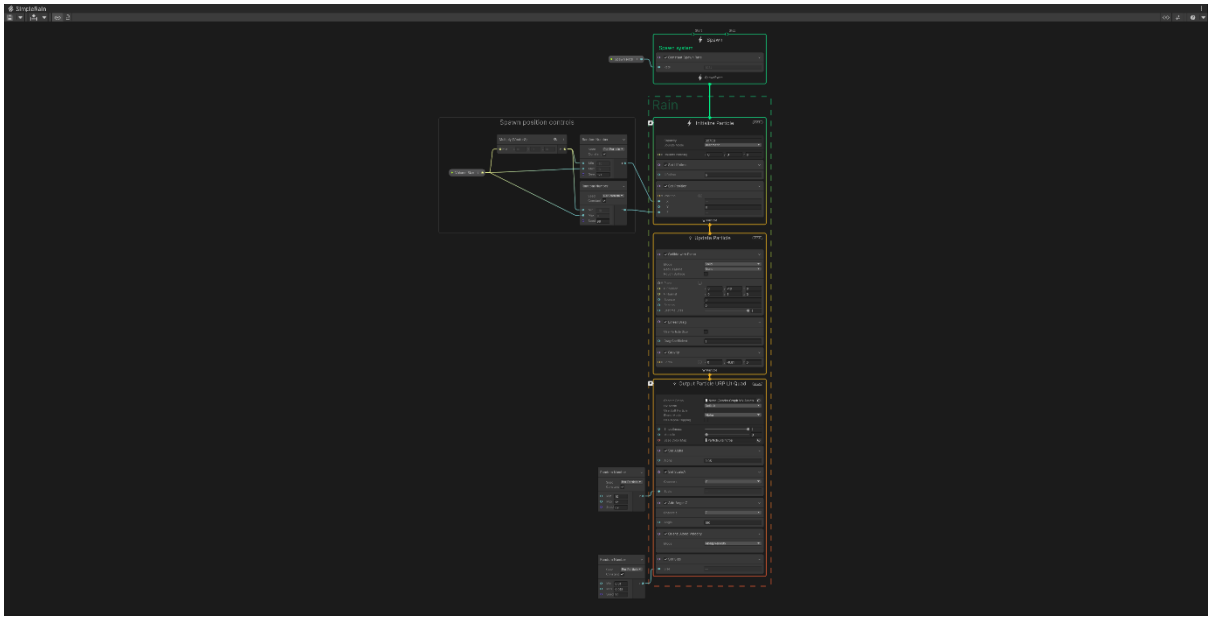

Figure 3 Rain particle effect in VFX Graph.

As mentioned in the Disclaimer section of the document, I didn't use VFX Graph to create a procedural render target for the rain effect. However, I did create a simple rain particle system to simulate rain. Alpha blended particles are spawned from a volume and are affected by gravity and drag. Particles are removed once they intersect the plane.

Users can control the spawn rate and the volume size from the Inspector window.

It's disappointing that the VFX Graph in URP still doesn't support the normal map input, as this would have greatly improved the look of the rain particles.

Textures

Here are all the textures used in the shader:

- Rain droplet mask (Condensation_Height.tif) courtesy of Add Your Light.
- Grunge leaks mask (grungeleaks-variant.jpg) courtesy of Adobe.
- Rain drip mask (Condensation_Raindrips_Height.tif).
- Cloud noise mask (Clouds_Grayscale.tif).

VFX Graph particle effect uses a single texture (Particle_Raindrop.tif).

All above-mentioned textures are grayscale, and all of them (aside from the rain particle texture), were save imported in linear space. Applying a gamma curve to mask textures can have a detrimental effect when used in shaders.

All textures were compressed using the BC7 compression to prevent pixelated artifacts from appearing. However, this didn't help that much when reconstructing normal maps from grayscale textures.

The number of textures grew as I worked on the effect. Initially, I thought I'd be able to only use a single texture for the rain drips. However, the effect wasn't convincing enough with the drips alone, and I ended up adding a couple more textures for rain droplets and leaks.

Normal map generation

Figure 4 Pixelated normals are especially visible on rain droplets. They are visible when viewed closely.

To save on memory, I opted to not use normal maps, and instead reconstruct them from grayscale textures. This was a mistake for the following reasons:

- Normal maps reconstructed from grayscale textures ended up looking pixelated.
- There was no improvement in memory usage. On the contrary, when using BC7 compression for grayscale images, the file size was the same BC7 compression for normal maps.

Limitations

- Due to time constraints, I was unable to do any performance tests. Thus, performance implications of this shader are unknown. Instead, I had to rely on intuition and best practices to make educated guesses.
- While the tri-planar projection used in the shader is cheap performance-wise, it results in visible seams when viewed very closely.
- Rain drips require two texture samples and a noise map to create variation. Textures tend to be the biggest culprits for poor shader performance, as they increase memory pressure on the GPU.
- Rain droplets and drips can appear in occluded areas which can break the immersion.
- Normal map pixelation is visible due to the reasons described in the “Normal map generation” part of this document.

Possible improvements

- Consider using a more robust tri-planar projection method. While I don’t know how to implement such functionality from scratch, Shader Graph appears to have a tri-planar projection node which uses three texture samples to ensure smooth transitions.
- Use texture packing. As mentioned previously, this shader relies on four grayscale textures. They could have easily fit in a single RGBA texture.
- Use dedicated normal maps. This would have prevented issues with normal map reconstruction from grayscale images.
- Implement animated droplets and project them on the Y-facing normals.
- Implement more robust wetness controls for base inputs. Now, rain smoothness is added to the base material smoothness. While it can still look convincing, having dedicated wetness and porosity could improve the look of the effect.
- Find a way to get rid of the secondary texture sample for rain drips.
- Create occlusion maps to remove the effect from the occluded areas.
- Conduct performance tests regularly to see how the shader performs.