Real-time Volumetric Fog Simulation

Report 1

by

Lucia Tódová

## Introduction

The point of the first few weeks of this project was to learn how real-time rendering in Unity works. During this time, the goal was to go over multiple shader tutorials, read and study a few papers, sketch and prepare the project skeleton, create a simple scene and try and use a simple shader, ideally combining it with both a compute shader and a script.

This paper sums up what I have managed to do so far, what were the main issues I have encountered and in what direction I imagine the project to continue.

### Project setup

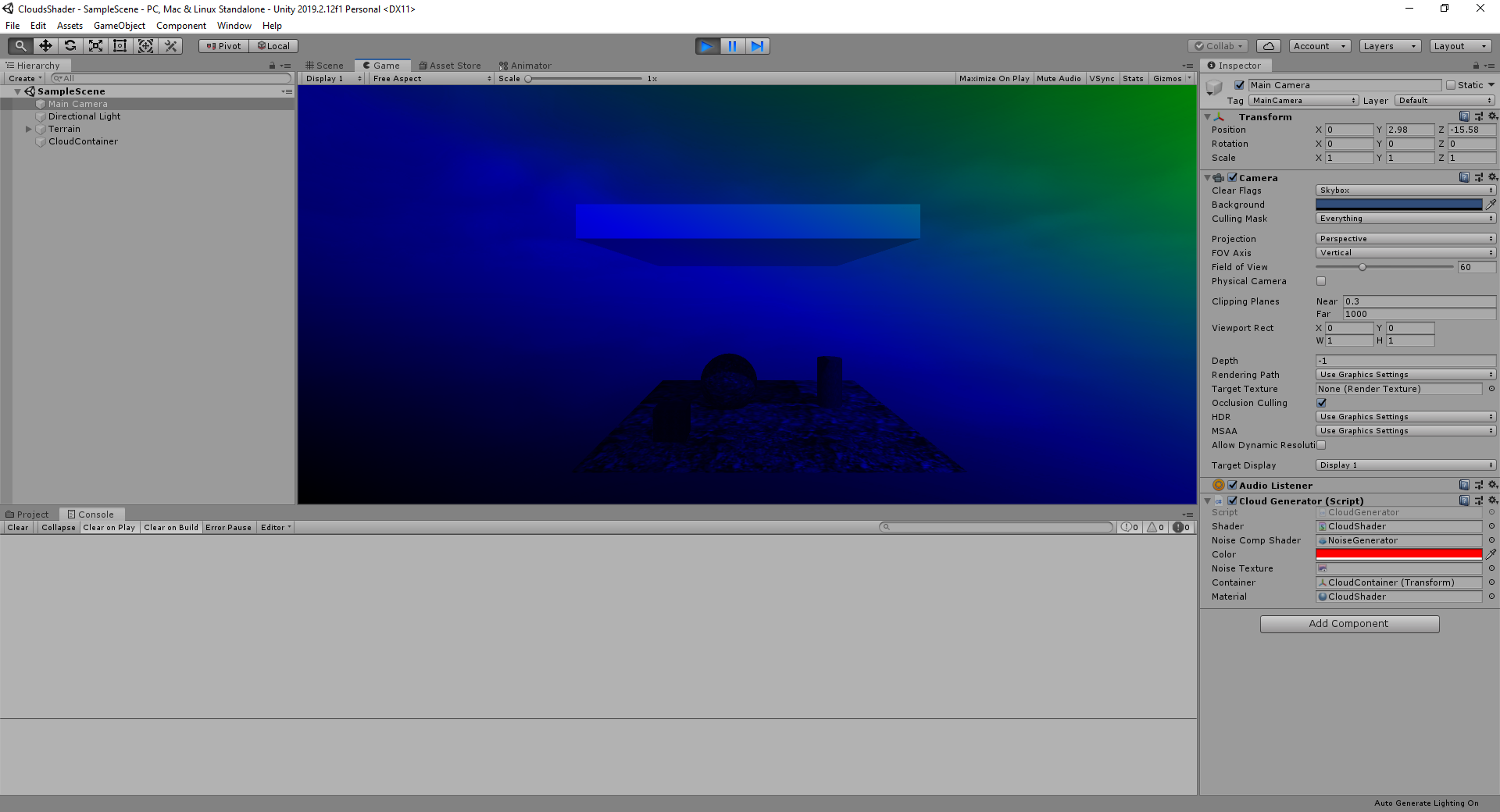
Although the project itself does not contain much code, I have setup a github project to follow its progress, which can be found here: <https://github.com/todoval/CloudsShader>.

The project contains the following:

* A simple scene with a few objects
* A script already applied to the camera, which is used for joining the individual shaders
* A compute shader, which will be used for noise generation
* A shader, which will be used for the actual cloud physics

### What has been done so far

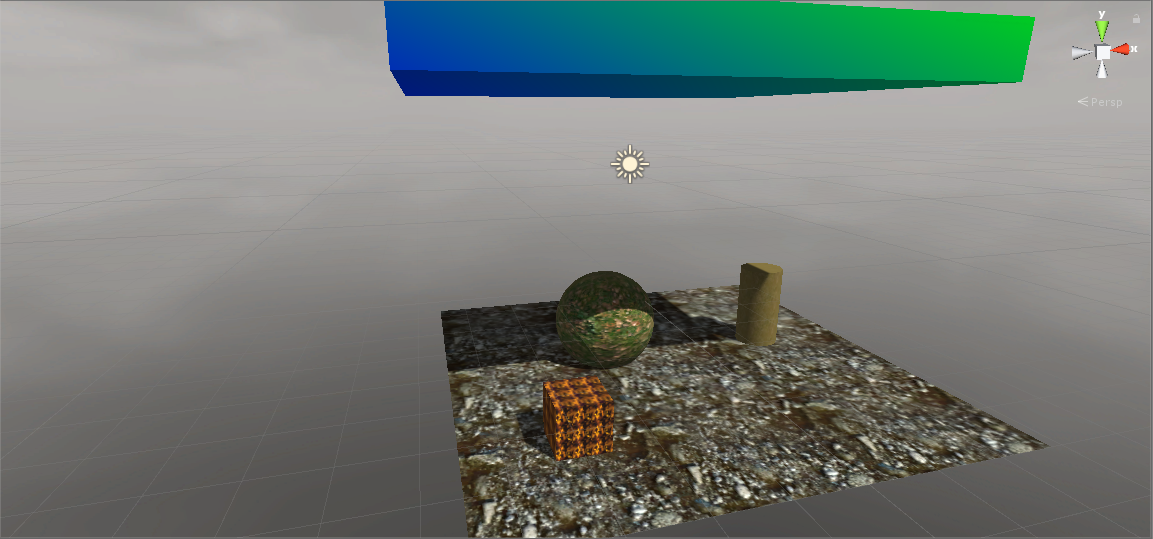
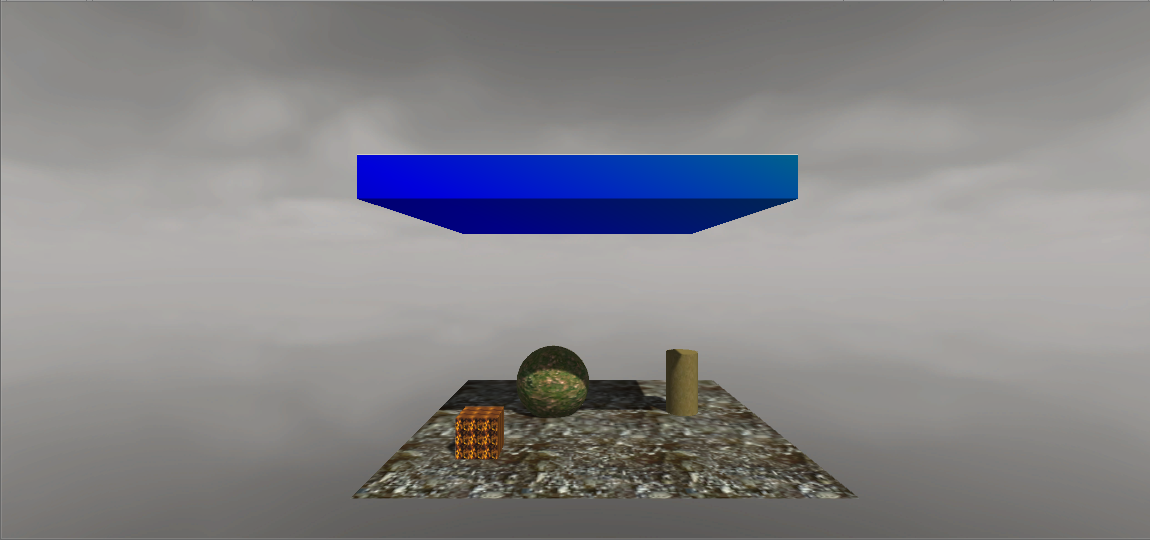
1. *Project setup* - I have created the scene, shaders and the cloud script. I also added a box object representing the cloud container, which will be used for the actual cloud rendering.
2. *Shader implementation -* I went through a few tutorials on different shaders, learned the basics of vertex and fragment shader and decided to use it for my project. At the beginning, the shader was solely used for tinting the camera view based on the position of the current vertex. I also applied it to a material manually, and then, also manually, I applied the material to an object for testing purposes.
3. *Compute shader implementation -* I tried to create a compute shader, which also provided tinting, and went on to learn the basics of RenderTexture, RWTexture3D, buffers and how they all work in Unity. I also created a script that dispatched the shader. I applied the script directly to the camera along with the shader, which allowed me to abstract from the usage of other objects.
4. *Script dispatching the shader* - I deleted the shader from the material on the test object and went on to set its parameters in the script. Therefore, the shader runs after the OnRenderImage() function in the script, which makes the compute shader + script + shader connection successful.
5. *Creation of a 2D texture from a color buffer* - So far, the only thing working for the combination of the three components was a solid color tint on the whole camera view. Although the solid color was represented as a 3D texture, I could not change it without further understanding of the compute shader and its threads. Therefore, I changed the texture back to 2D and created a color buffer that helped me with the implementation of a gradient in the compute shader. At that point, the tint was therefore a blueish gradient, as seen on the picture below.



1. *Container intersection* - The next step was to apply this texture to the cloud container that I created in the first step. As far as I am concerned, Unity cannot simply apply 3D textures to an object. This led to a realization that an algorithm for container intersection needs to be implemented. What this algorithm basically needs to do is to determine whether the current vector in the fragment shader is a part of the container. If it is, it applies the shader. If it’s not, it returns back the default texture of the shader.

### Container intersection

The container intersection is currently the part I am working on. The algorithm is based on raycasting - in the fragment shader, the vertex is projected to the camera view, a ray from the camera position to the vertex is cast, and the intersection point is computed. Although I tried to implement this with the help of a few open source projects and forums (e.g. <https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection> ), the code does not completely work as planned. Although the ray casting function returns the true/false values as expected, the texture mapped on the container is dependent on the camera view. This error can be clearly seen upon camera rotation as shown on the images below. Therefore, currently, the project is in the state of refactoring this raycasting function.



### Future steps

After the ray casting is resolved, the next few steps to execute until the next report would be the following:

1. change the 2D texture to 3D and map it to the container
2. create the Worley and Perlin noise in the compute shader
3. render the noise and make it user-friendly - let the user change parameters and possibly scale it

I do not expect the clouds to move and have any other physically-based abilities in the next report. However, the part after that (part 3 of the project) will focus on the actual shading and moving the clouds - on absorption, emission and other properties mentioned in the presentation.

### Setbacks

The biggest problem is definitely the lack of materials on Unity shaders. Therefore, most of the materials I use are from OpenGl or other shading languages, however, there are some inconsistencies that take a lot longer to resolve than would be expected. This was especially crucial during the project setup and the connection of the shaders, which looks simple enough but took over a week.