Layered Explosion Shader

MARIANA ÁVALOS ARCE, Universidad Panamericana

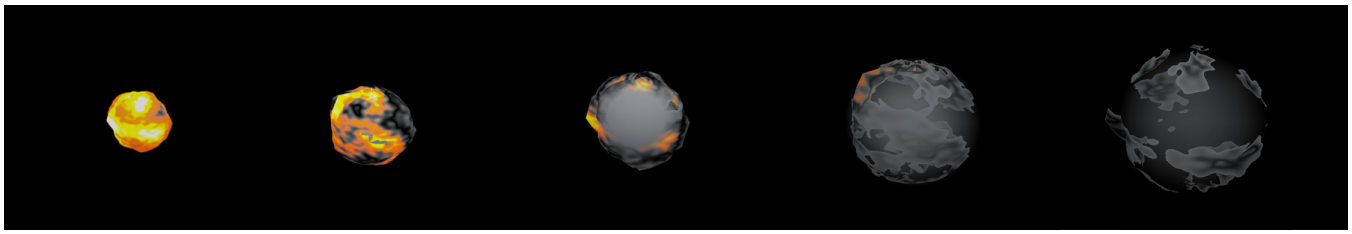

Fig. 1. Volumetric explosion shader in Unity 2018.4.4f1

The following document focuses mainly on the algorithm and logic followed in order to create the code for an explosion simulation, visualized in the game engine Unity, in the form of a shader applied to a sphere.

Additional Key Words and Phrases: shader, explosion, smoke, unity

ACM Reference Format:
Mariana Ávalos Arce. 2019. Layered Explosion Shader. *ACM Trans. Graph.* 1, 1, Article 1 (December 2019), 3 pages. https://doi.org/http://dx.doi.org/10.1145/8888888.7777777

1 INTRODUCTION

A shader is a program that calculates the specified light, darkness and color within a computer generated scene. In other words, a shader is a set of instructions for visual interpretation of geometry.

2 TEXTURES

The first thing we need is the basis geometry, which is a sphere. Then, we need to define three different textures: the color gradient texture, the displacement texture and the smoke noise texture.

The **color gradient texture** will be the one that defines which color the vertex will take, depending on its distance of displacement. For this, the user has an interface that lets her/him choose the key colors, and the script is be able to create an image with a gradient of those N colors.

Once the user defines which colors she/he wants in the gradient, then we divide the blank image into N parts, calling each part a **step**. The gradient can be either horizontal or vertical, but for this program, the gradient is horizontal. Once we know the step length, we loop through the image and on each of the N steps, the program

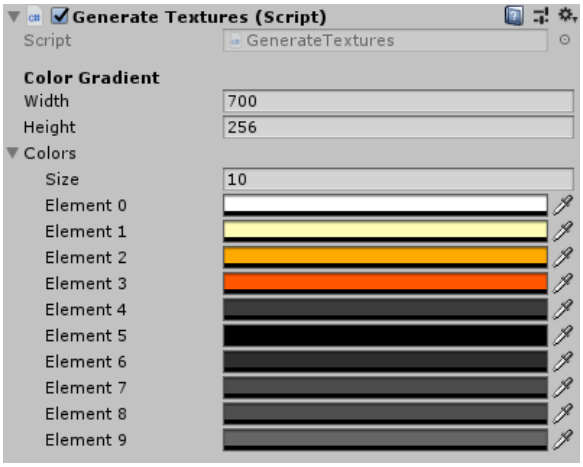

Fig. 2. User interface to choose N colors for the gradient

makes an interpolation of the current color and the current next color. The result is like the following:

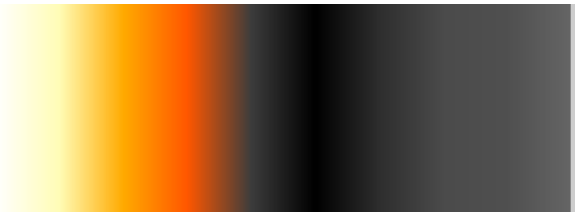

Fig. 3. Gradient texture result

When the explosion scales up, the vertices must be displaced with a noise value, so that they stop looking like a perfect sphere and instead they get the appearance of a rougher cloud. Unity has a built-in function for Perlin noise creation that can be used to fill in a texture, but using this for the explosion displacement looks extremely soft.

Author's address: Mariana Ávalos Arce, Faculty of Engineering, Universidad Panamericana, 0197495@up.edu.mx.

0730-0301/2019/12-ART1 $0.00
https://doi.org/http://dx.doi.org/10.1145/8888888.7777777

For the **displacement texture**, I decided to use the concept of **layered noise**. This created a more realistic result. **Layered noise** is having one final texture depending on more than one noise values. Perlin noise takes in the coordinate information of the current pixel, and a constant factor affecting the coordinate input is called **scale**, whereas a factor that multiplies *Perlin(x, y)* as a whole is called **strength**. For this, I created an object known as **Layer**, which had the customizable parameters, scale and strength.

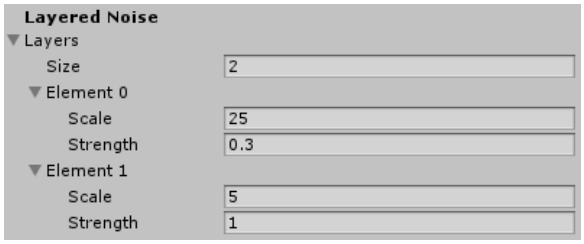

Fig. 4. User interface for N Layered Noise

Once the parameters are set for an array of N Layer objects, a blank texture is looped through, and for each pixel, I create its r,g,b values and loop through each layer. Here, the r,g,b values accumulate each of the resulting values for

```
(Perlin(x * scale, y * scale)) * strength
```

coming from each of the layers. After accumulating each layer value, the r,g,b are normalized and then the loop goes on with this for the rest of the pixels. This gives the impresion of a sharper Perlin Noise, because the strength and scale define more irregular forms.

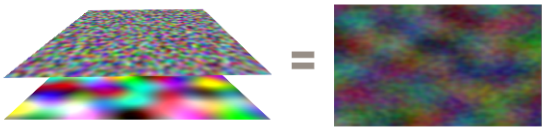

Fig. 5. Layered Noise Scheme

After various tests, the conclusion was that two layers were realistic enough and did not interfere much in the time performance of the texture, because the operations tend to be slow during the loop process.

The **smoke noise texture** was much simpler, because it required a simple Perlin Noise texture.

3 SEPARATE SHADERS

Once the textures are defined, the script must send them to different shaders: the explosion shader and the smoke shader. The explosion shader performs inside the vertex shader an extrusion of the vertices based on the rgb value coming from the displacement texture, by accumulating to the vertex position the multiplication of its normal and the displacement value. In the fragment shader, I mix the rgb values as well but also use another parameter, called the **depth**.

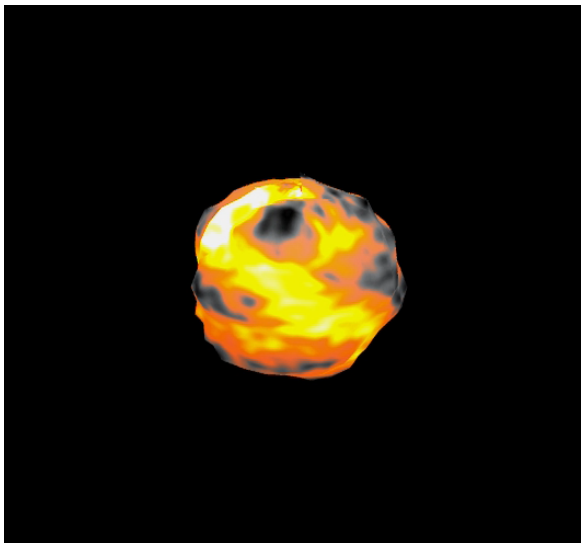

Fig. 6. Vertex displacement

I add the depth value to the past displacement value, and this will be the x coordinate to use for getting a color from the gradient texture. With the depth I also decide when does the pixel gets rendered or not, by using the clip() built-in function. The depth value grows with time, so the color will advance and the pixel discarding as well, making the explosion grow darker and start to disappear.

Now, the smoke. Once the explosion occurs, when it starts to disappear, I instantiate a smoke globe, that will grow with time as well. The growth parameters for this scaling are also customizable, so that the user can decide how the explosion looks. The shader for the smoke is made using the idea behind a rim shader, which involves the dot product of the normal and the view direction:

```
rim = saturate(dot(normalize(IN.viewDir), o.Normal));
```

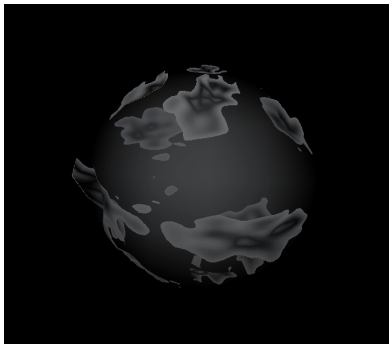

Fig. 7. Smoke and clipping of explosion

I use this value plus the value of the simple noise texture just mentioned, to create an alpha rim with noise. This needs to be taken carefully, because alpha tends to go negative and create weird looks. I also substract to this alpha value the time of creation coming from

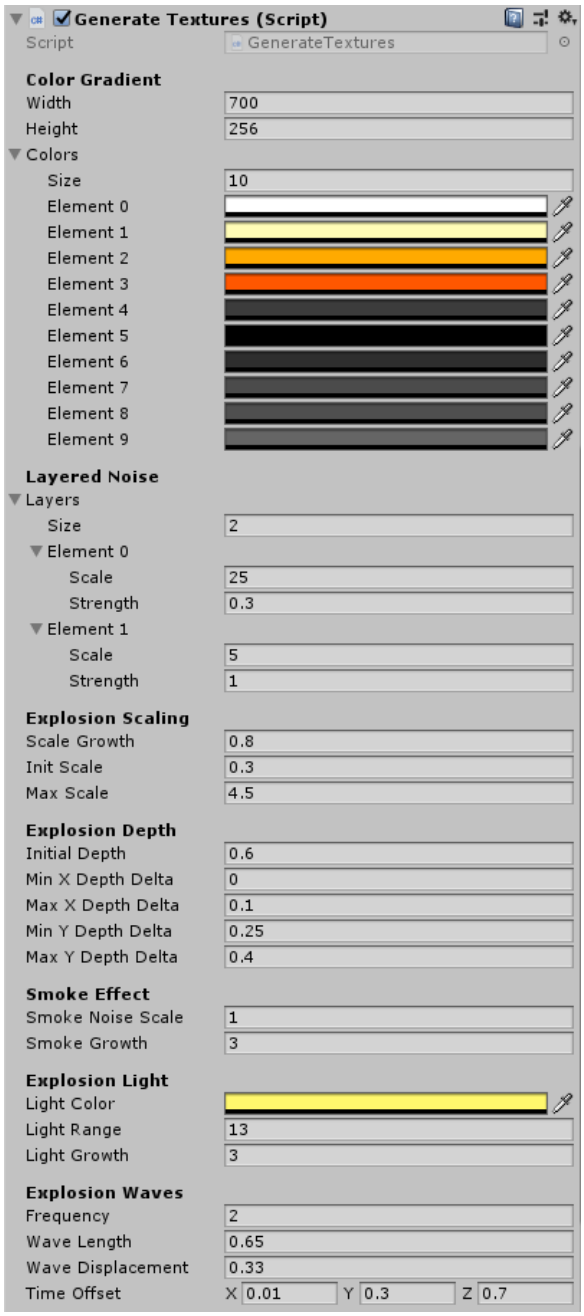

Fig. 10. Final user interface for explosion params

a script, so that the bigger the time, the less alpha. In this way, the smoke will fade soflty with time.

The last detail is a point light animation I added after the explosion and its smoke. This point light gets instantiated a little after the explosion occurs, and it grows its intensity over time to add realism. The light parameters and its color are also open to custom values.

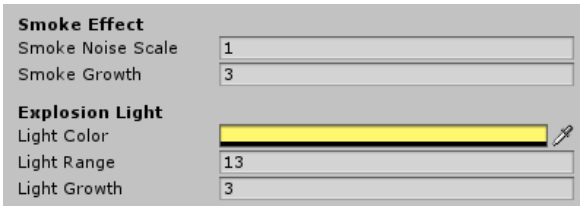

Fig. 8. User interface for additional parameters

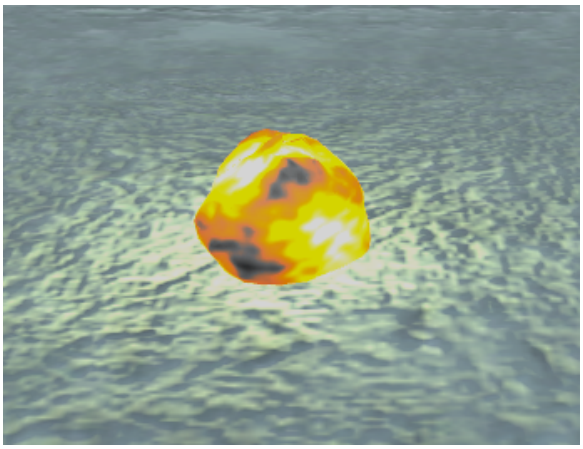

Fig. 9. Light growth

4 POST-PROCESSING

When an explosion occurs, smoke and light come out. But also the environment changes. I set the initial value of the rendered image at 0.6. Every time an explosion gets instantiated, the rendered image coming from the camera gets lighter and lighter until it reaches the value of 1, to go back again to 0.6, making the effect of enlightening every time an explosion progresses.

For this I also created a shader, that will process the camera rendered image of the scene. The shader's only job is to substract a certain amount to the rendered pixel color, which gets sent from a script handled by timers and collision detectors.