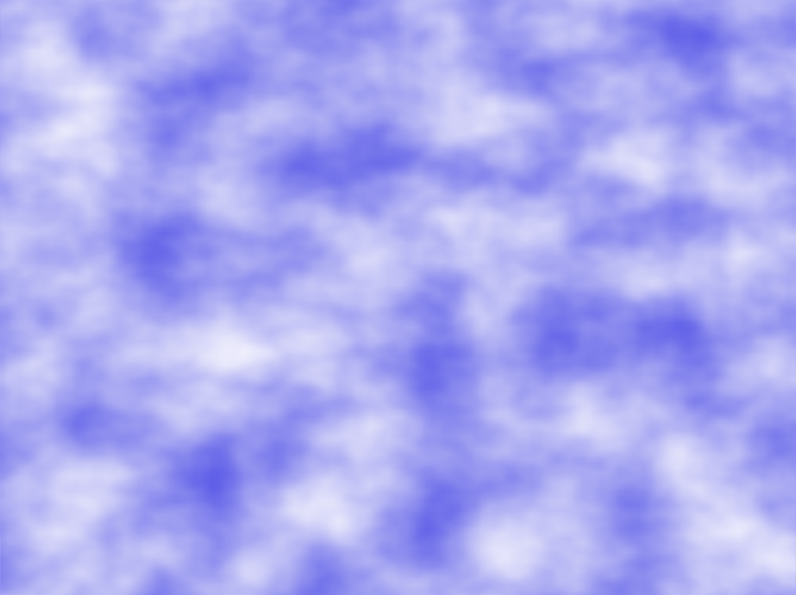
**Cloud effect**

**The end result:**



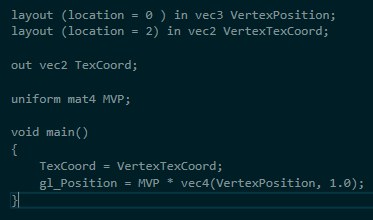
To create a texture that resembles a sky with clouds, we can use the noise values as a blending factor between the sky colour and the cloud colour. As clouds usually have large-scale structure, it makes sense to use low-octave noise. However, the large-scale structure often has higher frequency variations, so some contribution from higher octave noise may be desired.

We start by retrieving the noise value from the noise texture (the noise variable). The green channel contains two octave noises, so we use the value stored in that channel (noise.g). Feel free to try out other channels and determine what looks right to you.

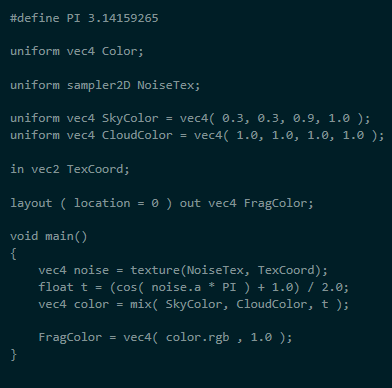
We use a cosine function to make a sharper transition between the cloud and sky colour. The noise value will be between zero and one, and the cosine of that value will range between -1 and 1, so we add 1.0 and divide by 2.0. The result that is stored in t should again range between zero and one. Without this cosine transformation, the clouds look a bit too spread out over the sky. However, if that is the desired effect, one could remove the cosine and just use the noise value directly.

Next, we mix the sky colour and the cloud colour using the value of t. The result is used as the final output fragment colour.

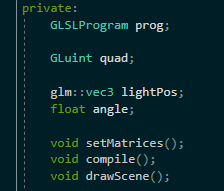
**Vertex shader:**



**Fragment shader:**



**scenebasic\_uniform.h:**

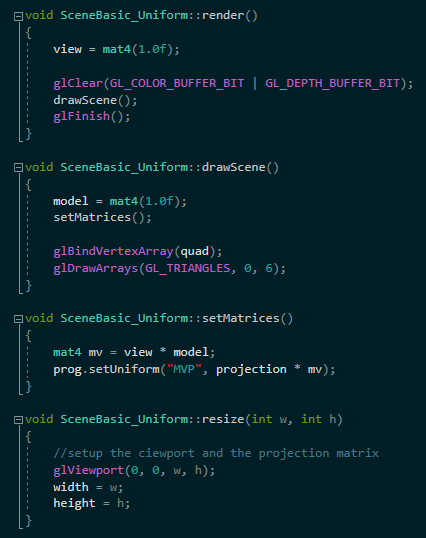


**scenebasic\_uniform.cpp:**

For initScene():



For render(), drawScene(), setMatrices() and resize():



That’s it.

**Wood grain effect**

**The end result:**



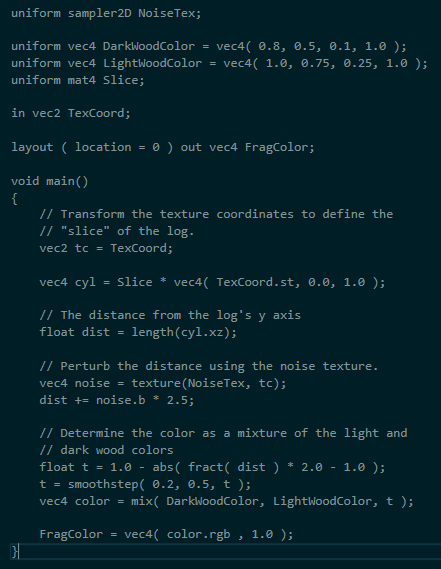
To create the look of wood, we can start by creating a virtual "log" with perfectly cylindrical growth rings. Then, we'll take a slice of the log and perturb the growth rings using noise from our noise texture.

**Vertex shader:**

Use the vertex shader from previous example “Cloud effect”.

**Fragment shader:**

Use this fragment shader:

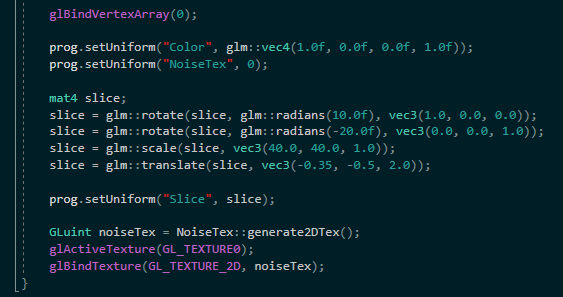


**scenebasic\_uniform.h:**

The same as in previous example “Cloud effect”.

**scenebasic\_uniform.cpp:**

The initScene() is identical to the previous example “Cloud effect” but the last bit under glBindVertexArray(0) is different:

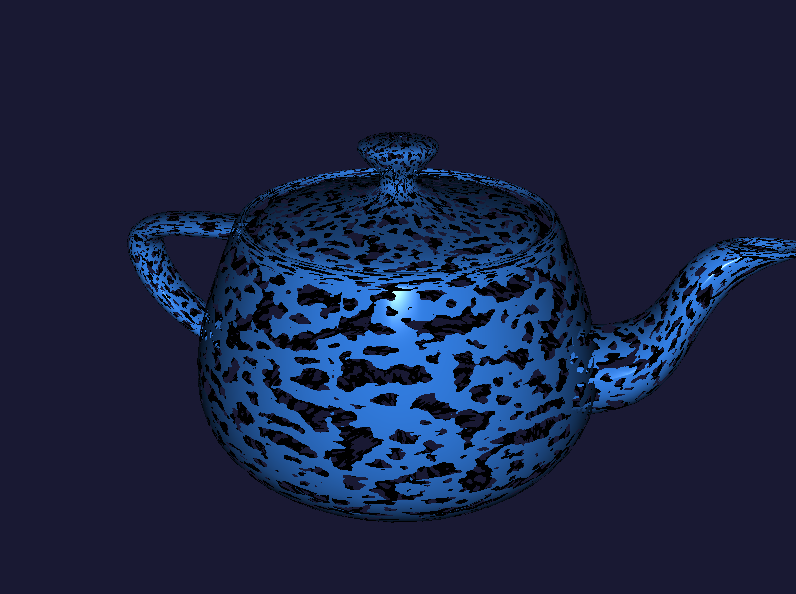


Everything else is similar the previous example “Cloud effect”.

That’s it.

**Disintegration effect**

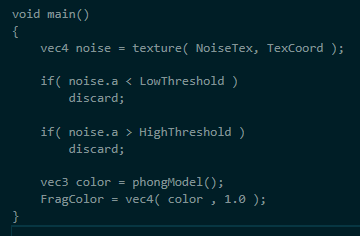
**The end result:**



It is straightforward to use the GLSL **discard** keyword in combination with noise to simulate erosion or decay. We can simply discard fragments that correspond to a noise value that is above or below a certain threshold.

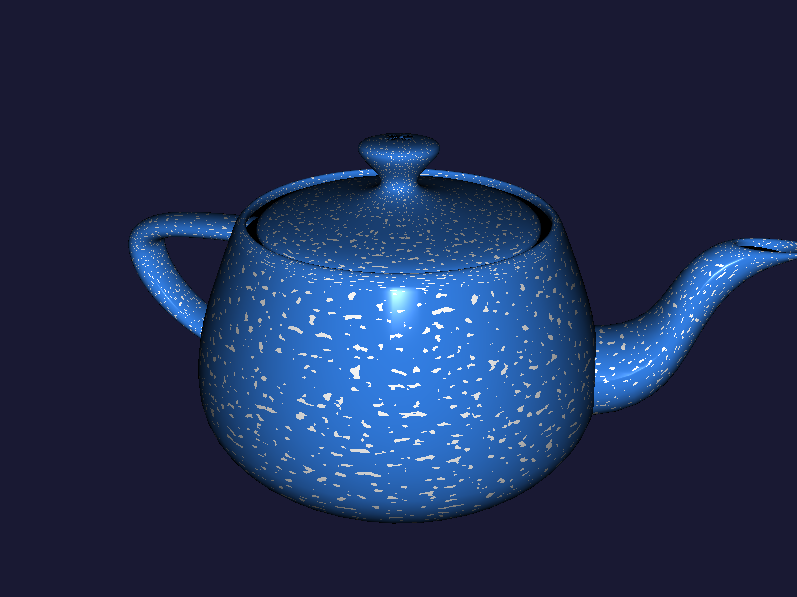
All we have to do is use the keyword discard in the fragment shader

**Fragment shader:**



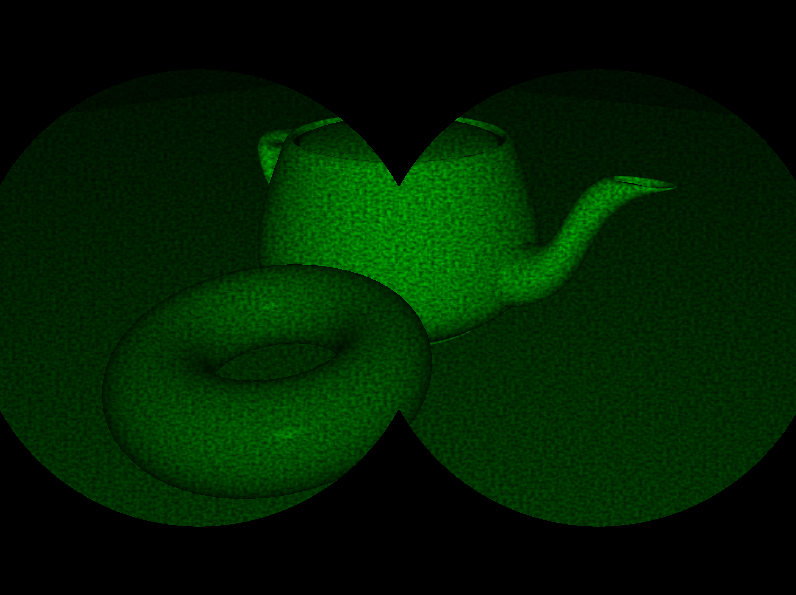
That’s it.

You can do the same for the paint splatter but don’t discard the fragments just coloured them into another colour:



**Night vision effect**

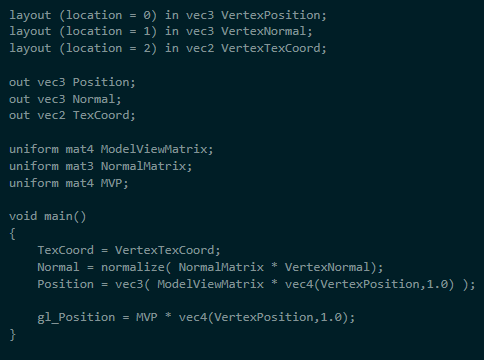
**The end result:**



We'll create the look of night-vision goggles with some noise thrown in to simulate some random static in the signal. We'll also outline the scene in the classic binocular view.

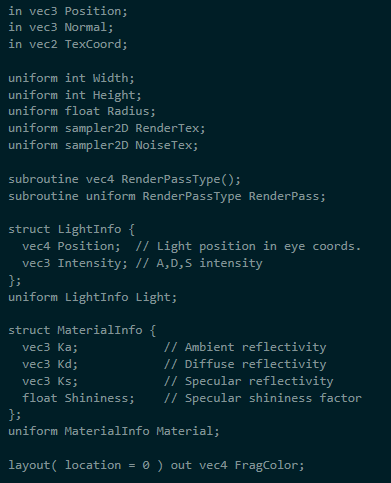
We'll apply the night-vision effect as a second pass to the rendered scene. The first pass will render the scene to a texture and the second pass will apply the night-vision effect.

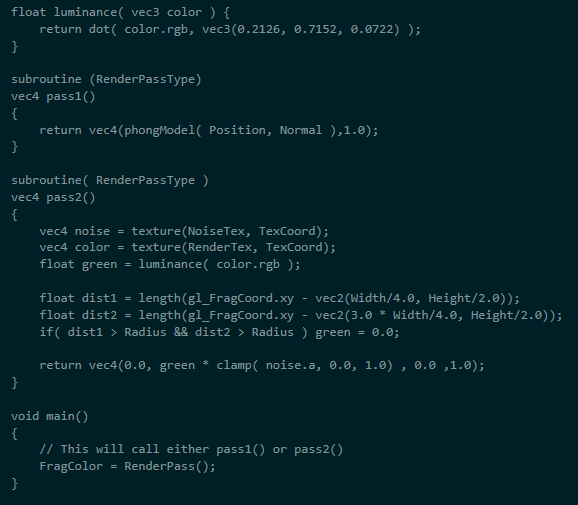
**Vertex shader:**



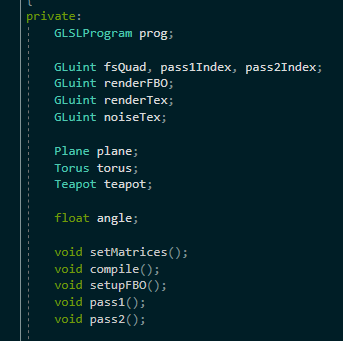
**Fragment shader:**

Use this fragment shader. Keep in mind that I didn’t screenshot the phongModel implementation, you can do that yourself.



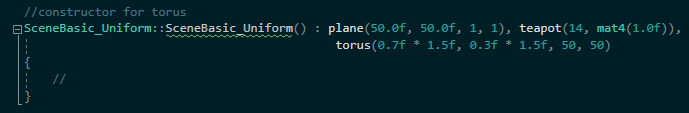


**scenebasic\_uniform.h:**

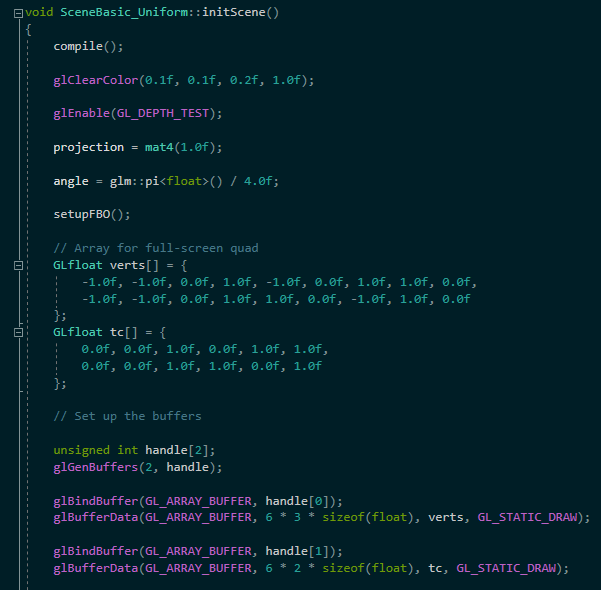


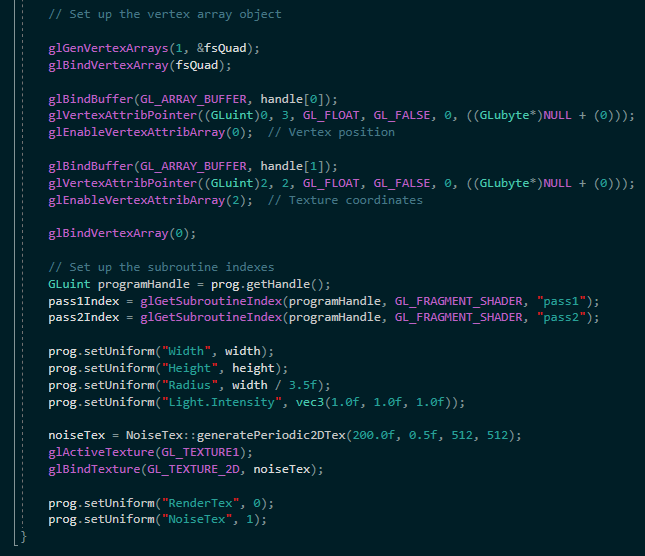
**scenebasic\_uniform.cpp:**

For constructor use this:

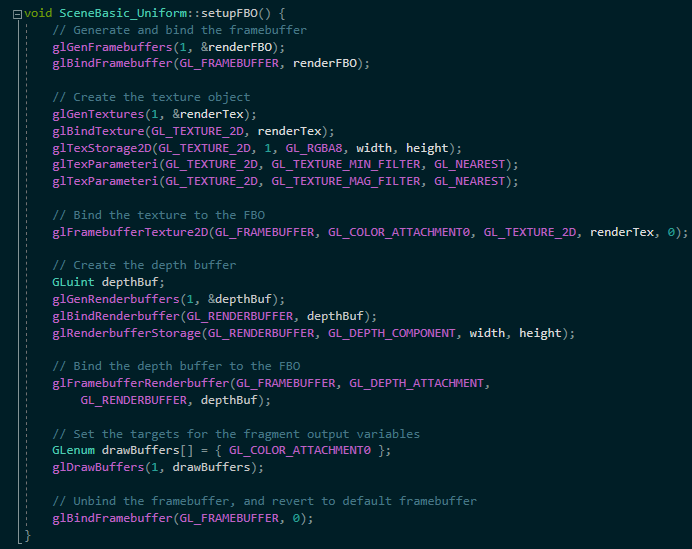


For initScene():

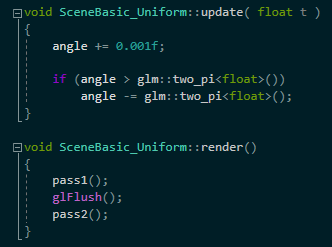




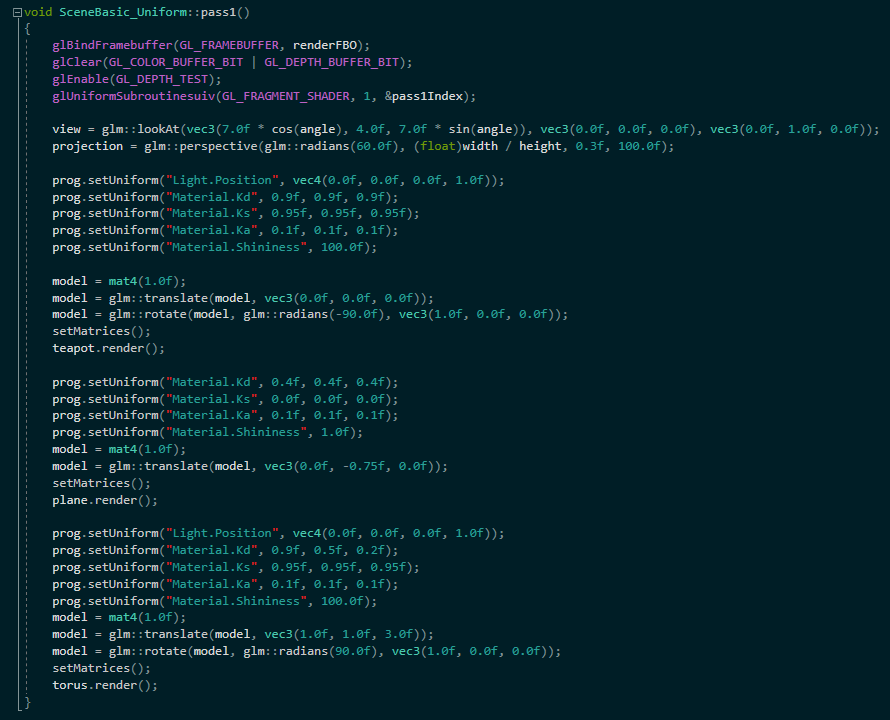
For setupFBO():



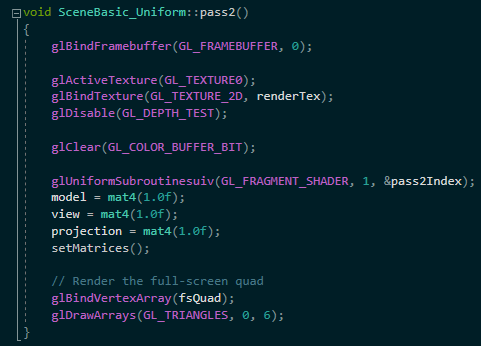
For update() and render():



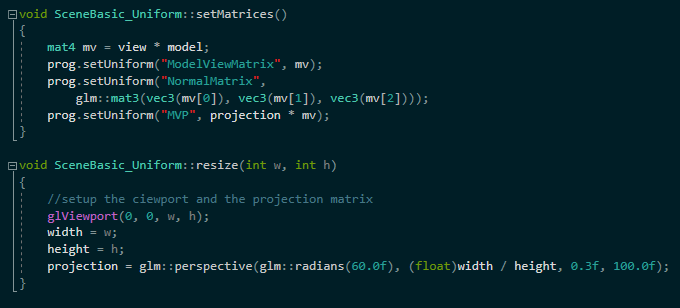
For pass1():



For pass2():



For setMatrices() and resize():



That’s it.