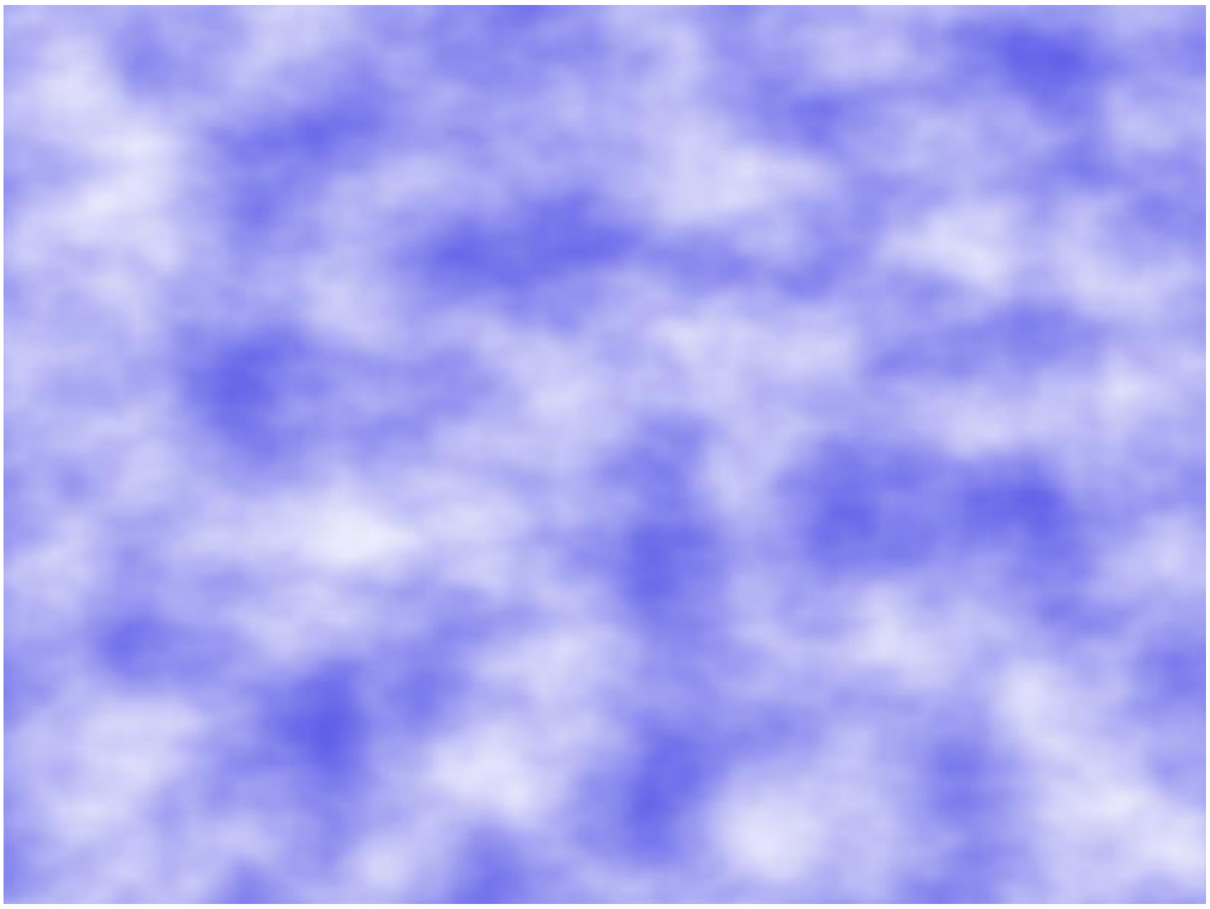
Cloud effect

The end result:



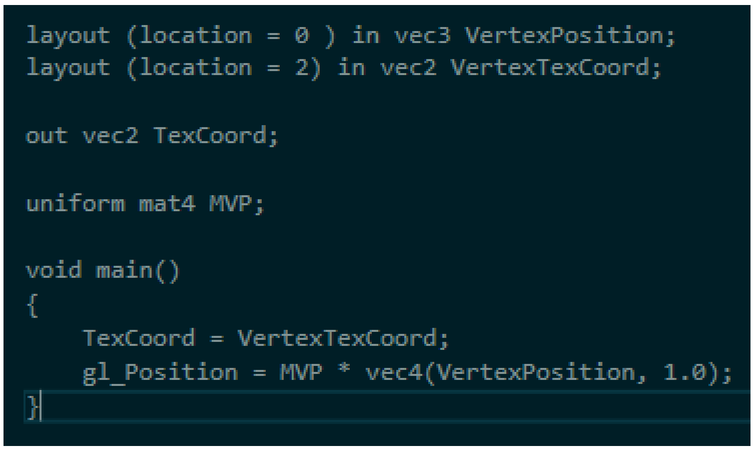
To create a texture that resembles a sky with clouds, we can use the noise values as a blending factor between the sky colour and the cloud colour. As clouds usually have large-scale structure, it makes sense to use low-octave noise. However, the large-scale structure often has higher frequency variations,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

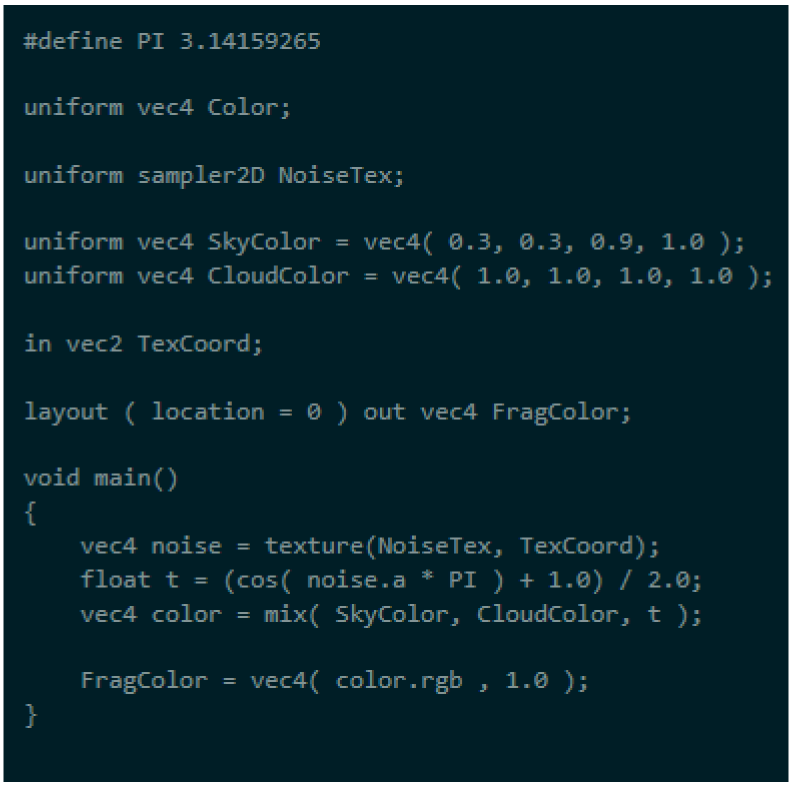
Next, we mix the sky colour and the cloud colour using the value of t. The result is used as the final output fragment colour.

Vertex shader:



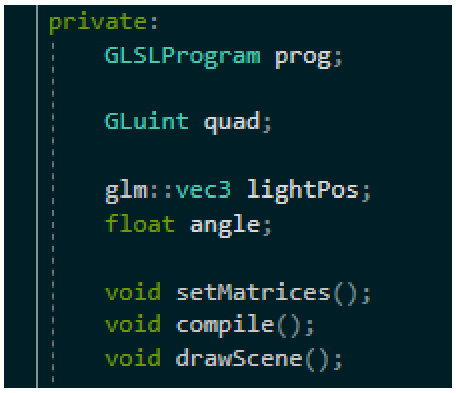
```
layout (location = 0 ) in vec3 VertexPosition;
layout (location = 2) in vec2 VertexTexCoord;

out vec2 TexCoord;

uniform mat4 MVP;

void main()
{
    TexCoord = VertexTexCoord;
    gl_Position = MVP * vec4(VertexPosition, 1.0);
}
```

Fragment shader:



```
#define PI 3.14159265

uniform vec4 Color;

uniform sampler2D NoiseTex;

uniform vec4 SkyColor = vec4( 0.3, 0.3, 0.9, 1.0 );
uniform vec4 CloudColor = vec4( 1.0, 1.0, 1.0, 1.0 );

in vec2 TexCoord;

layout ( location = 0 ) out vec4 FragColor;

void main()
{
    vec4 noise = texture(NoiseTex, TexCoord);
    float t = (cos( noise.a * PI ) + 1.0) / 2.0;
    vec4 color = mix( SkyColor, CloudColor, t );

    FragColor = vec4( color.rgb , 1.0 );
}
```

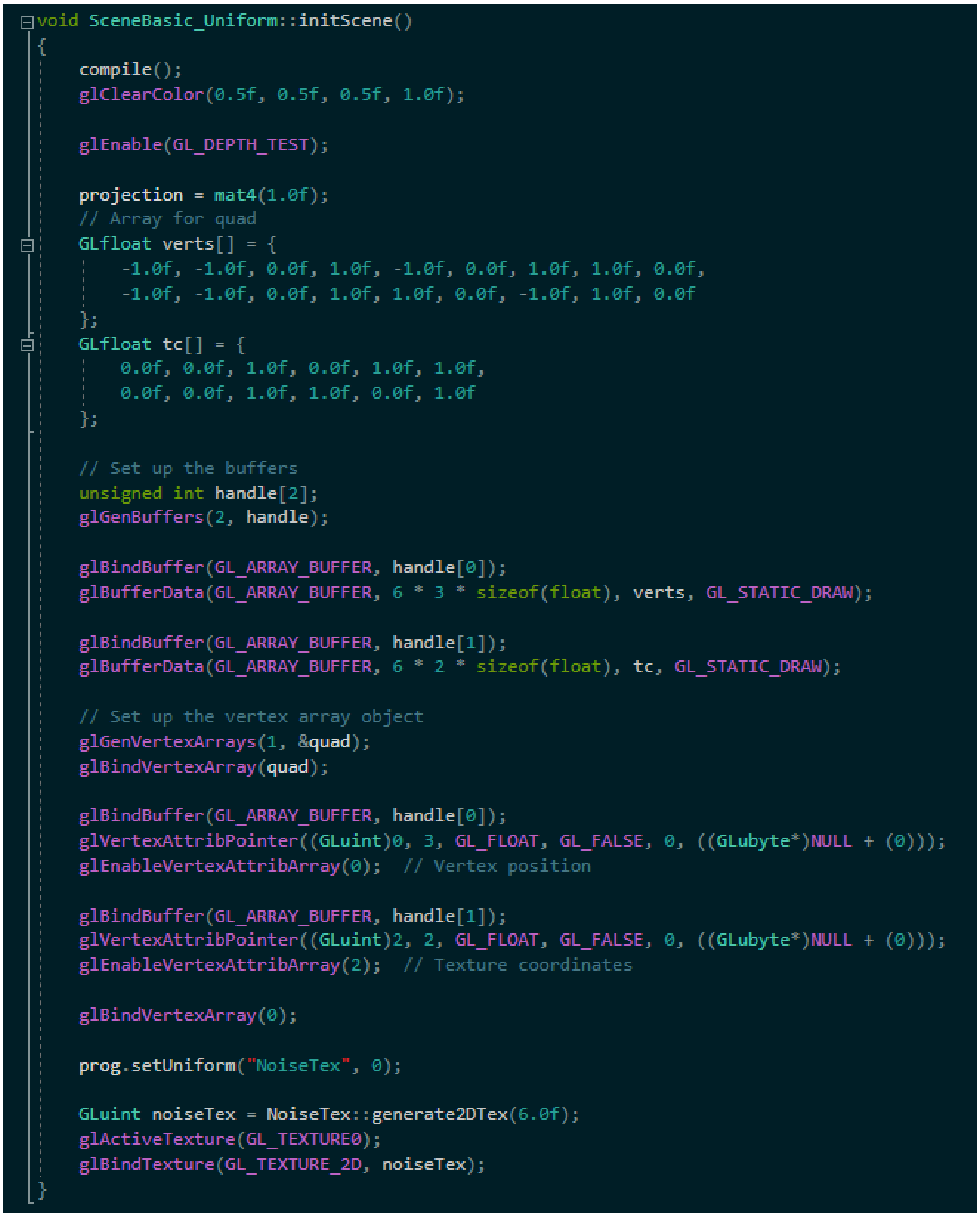
scenebasic_uniform.h:



```
private:
    GLSLProgram prog;

    GLuint quad;

    glm::vec3 lightPos;
    float angle;

    void setMatrices();
    void compile();
    void drawScene();
```

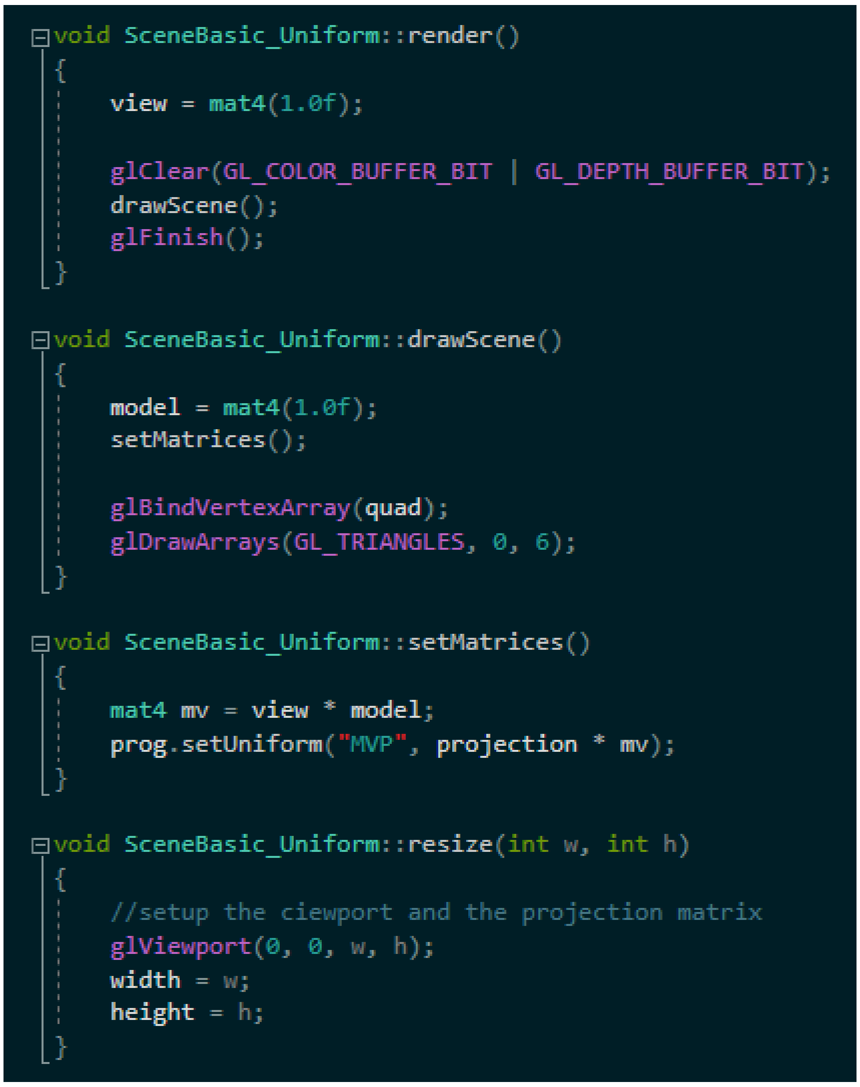
scenebasic_uniform.cpp:

For initScene():



```cpp
void SceneBasic_Uniform::initScene()
{
    compile();
    glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

    glEnable(GL_DEPTH_TEST);

    projection = mat4(1.0f);
    // Array for quad
    GLfloat verts[] = {
        -1.0f, -1.0f, 0.0f, 1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f,
        -1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f, -1.0f, 1.0f, 0.0f
    };
    GLfloat tc[] = {
        0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f
    };

    // Set up the buffers
    unsigned int handle[2];
    glGenBuffers(2, handle);

    glBindBuffer(GL_ARRAY_BUFFER, handle[0]);
    glBufferData(GL_ARRAY_BUFFER, 6 * 3 * sizeof(float), verts, GL_STATIC_DRAW);

    glBindBuffer(GL_ARRAY_BUFFER, handle[1]);
    glBufferData(GL_ARRAY_BUFFER, 6 * 2 * sizeof(float), tc, GL_STATIC_DRAW);

    // Set up the vertex array object
    glGenVertexArrays(1, &quad);
    glBindVertexArray(quad);

    glBindBuffer(GL_ARRAY_BUFFER, handle[0]);
    glVertexAttribPointer((GLuint)0, 3, GL_FLOAT, GL_FALSE, 0, ((GLubyte*)NULL + (0)));
    glEnableVertexAttribArray(0);  // Vertex position

    glBindBuffer(GL_ARRAY_BUFFER, handle[1]);
    glVertexAttribPointer((GLuint)2, 2, GL_FLOAT, GL_FALSE, 0, ((GLubyte*)NULL + (0)));
    glEnableVertexAttribArray(2);  // Texture coordinates

    glBindVertexArray(0);

    prog.setUniform("NoiseTex", 0);

    GLuint noiseTex = NoiseTex::generate2DTex(6.0f);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, noiseTex);
}
```

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

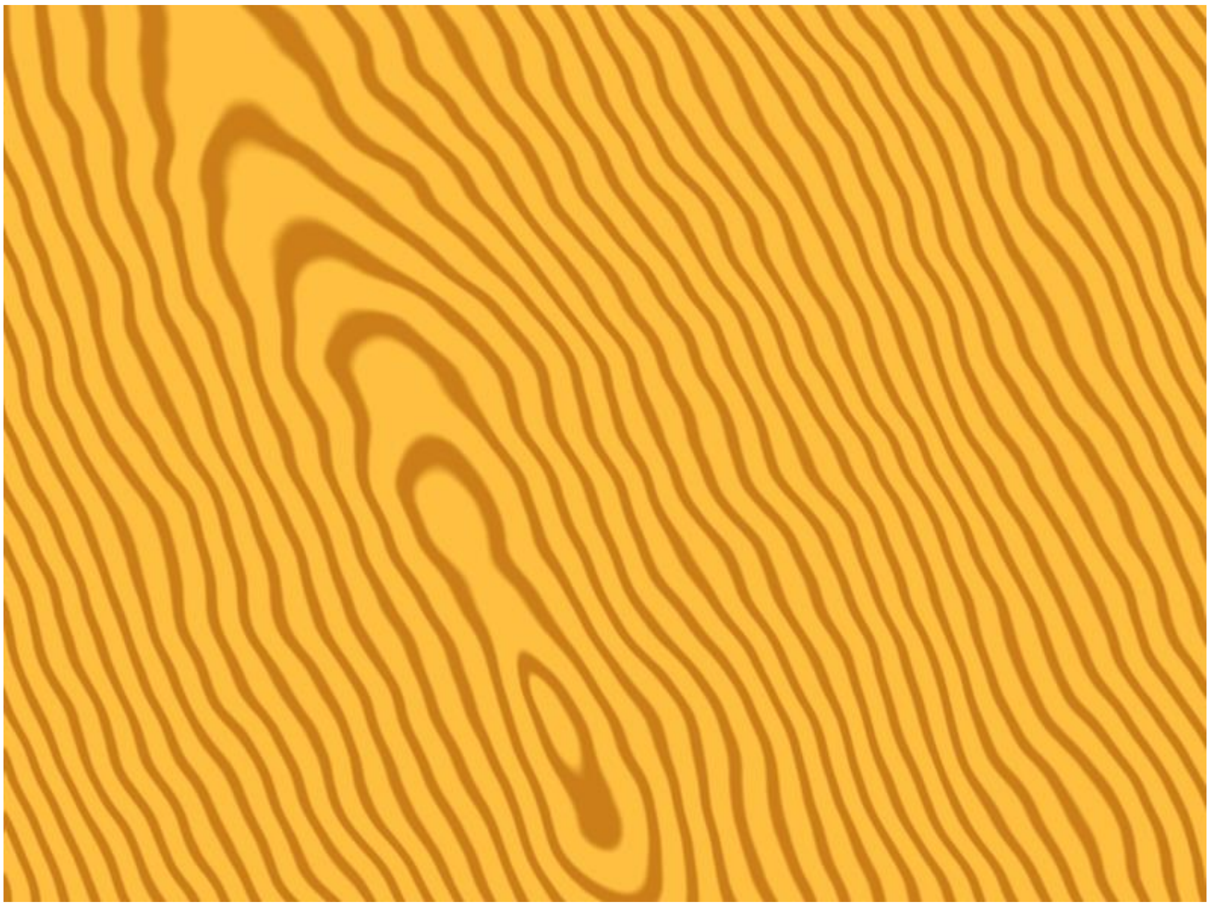


```
void SceneBasic_Uniform::render()
{
    view = mat4(1.0f);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    drawScene();
    glFinish();
}

void SceneBasic_Uniform::drawScene()
{
    model = mat4(1.0f);
    setMatrices();

    glBindVertexArray(quad);
    glDrawArrays(GL_TRIANGLES, 0, 6);
}

void SceneBasic_Uniform::setMatrices()
{
    mat4 mv = view * model;
    prog.setUniform("MVP", projection * mv);
}

void SceneBasic_Uniform::resize(int w, int h)
{
    //setup the ciewport and the projection matrix
    glViewport(0, 0, w, h);
    width = w;
    height = h;
}
```

That’s it.

Wood grain effect

The end result:



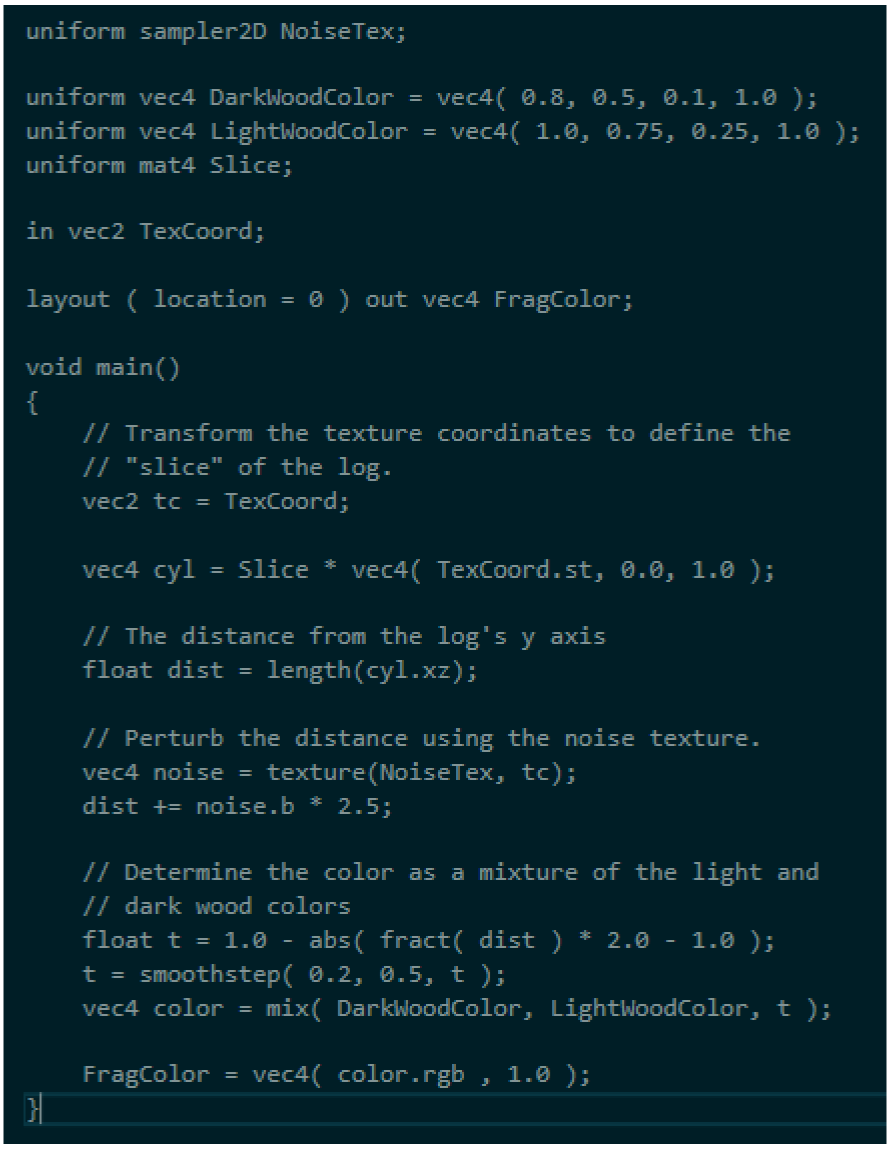
To create the look of wood, we can start by creating a virtual "log" with perfectly cylindrical growth rings. Then, we'll take a slice of the log and perturb the growth rings using noise from our noise texture.

Vertex shader:

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

Fragment shader:

Use this fragment shader:



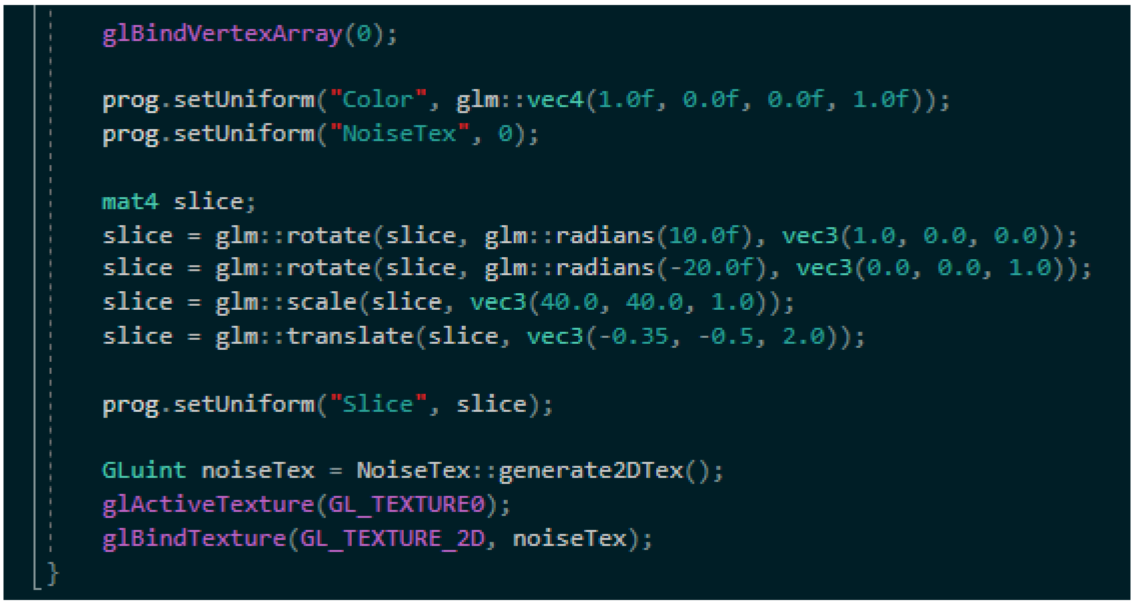
```
uniform sampler2D NoiseTex;

uniform vec4 DarkWoodColor = vec4( 0.8, 0.5, 0.1, 1.0 );
uniform vec4 LightWoodColor = vec4( 1.0, 0.75, 0.25, 1.0 );
uniform mat4 Slice;

in vec2 TexCoord;

layout ( location = 0 ) out vec4 FragColor;

void main()
{
    // Transform the texture coordinates to define the
    // "slice" of the log.
    vec2 tc = TexCoord;

    vec4 cyl = Slice * vec4( TexCoord.st, 0.0, 1.0 );

    // The distance from the log's y axis
    float dist = length(cyl.xz);

    // Perturb the distance using the noise texture.
    vec4 noise = texture(NoiseTex, tc);
    dist += noise.b * 2.5;

    // Determine the color as a mixture of the light and
    // dark wood colors
    float t = 1.0 - abs( fract( dist ) * 2.0 - 1.0 );
    t = smoothstep( 0.2, 0.5, t );
    vec4 color = mix( DarkWoodColor, LightWoodColor, t );

    FragColor = vec4( color.rgb , 1.0 );
}
```

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:



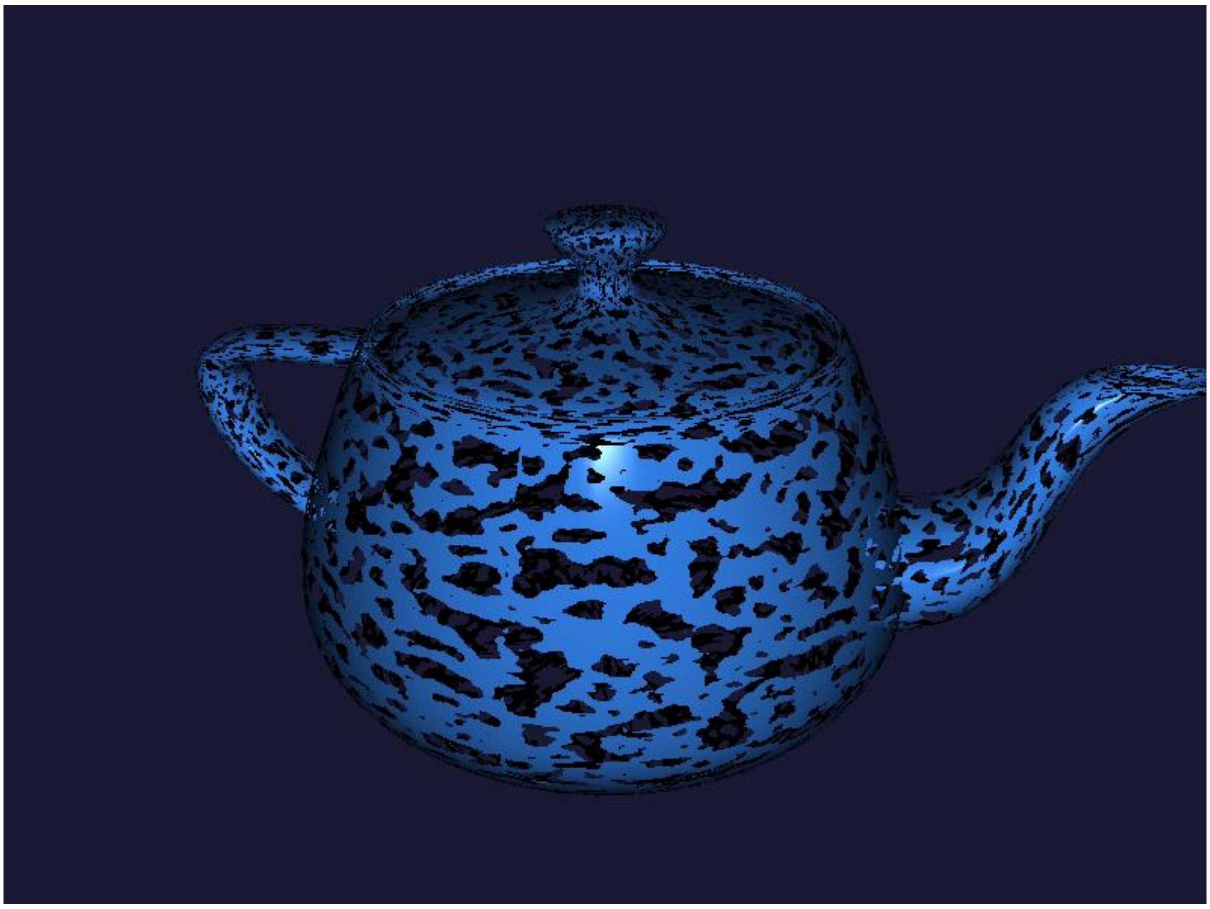
```
    glBindVertexArray(0);

    prog.setUniform("Color", glm::vec4(1.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("NoiseTex", 0);

    mat4 slice;
    slice = glm::rotate(slice, glm::radians(10.0f), vec3(1.0, 0.0, 0.0));
    slice = glm::rotate(slice, glm::radians(-20.0f), vec3(0.0, 0.0, 1.0));
    slice = glm::scale(slice, vec3(40.0, 40.0, 1.0));
    slice = glm::translate(slice, vec3(-0.35, -0.5, 2.0));

    prog.setUniform("Slice", slice);

    GLuint noiseTex = NoiseTex::generate2DTex();
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, noiseTex);
}
```

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

That’s it.

Disintegration effect

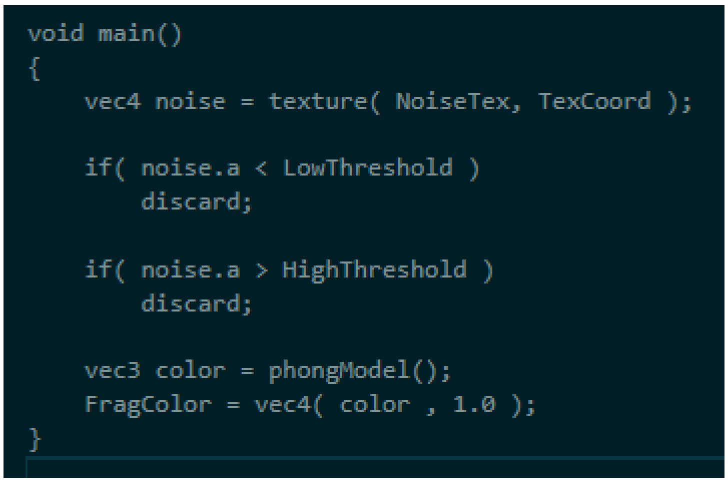
The end result:



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

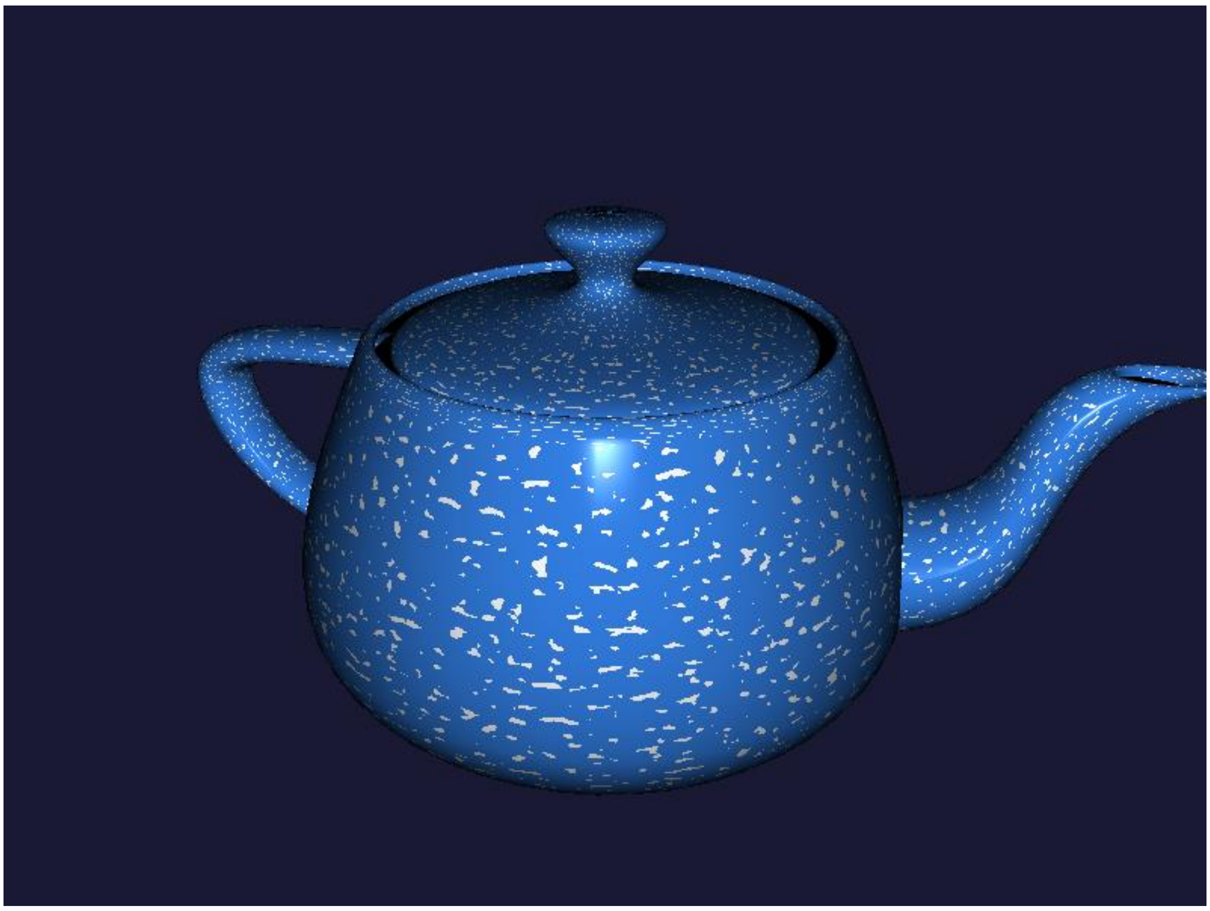
Fragment shader:



```
void main()
{
    vec4 noise = texture( NoiseTex, TexCoord );

    if( noise.a < LowThreshold )
        discard;

    if( noise.a > HighThreshold )
        discard;

    vec3 color = phongModel();
    FragColor = vec4( color , 1.0 );
}
```

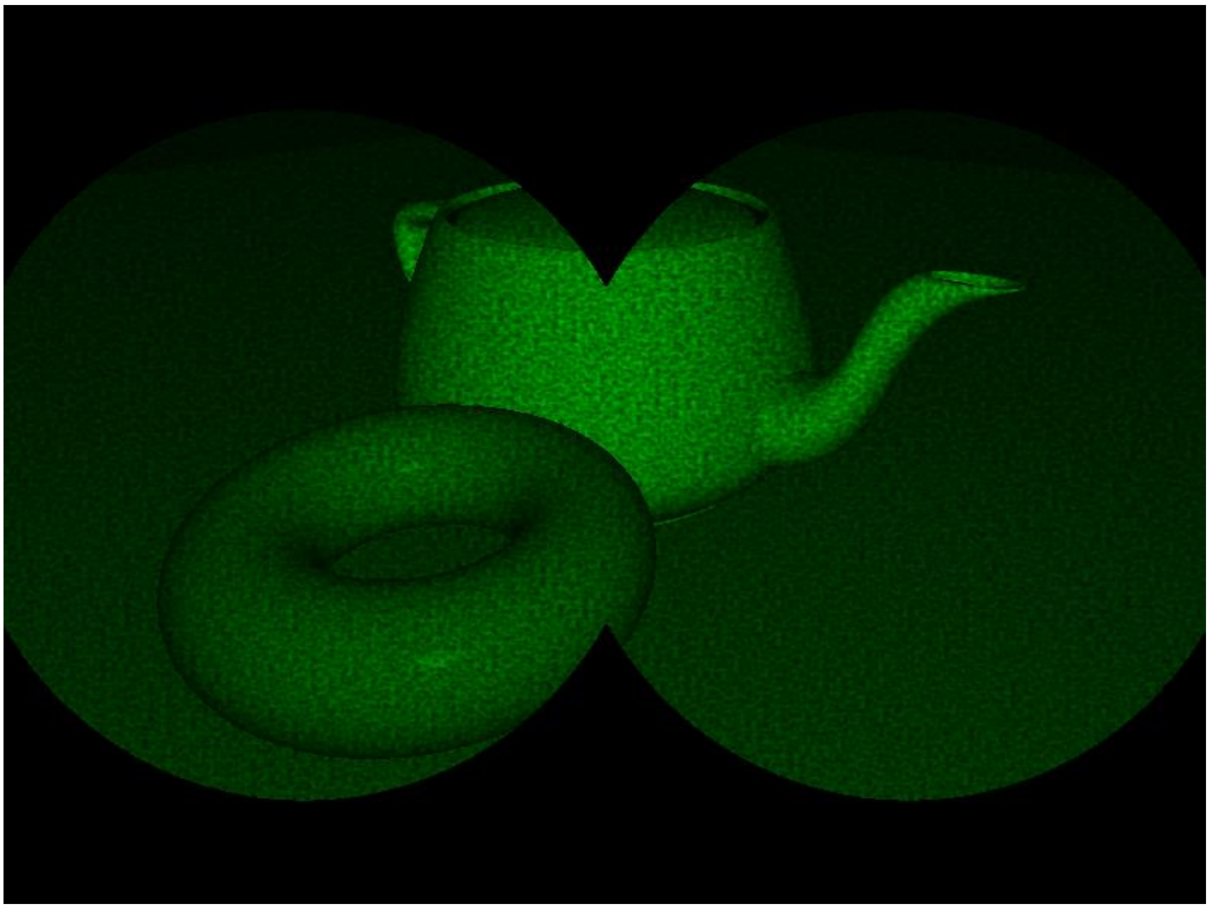
That’s it.

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



Night vision effect

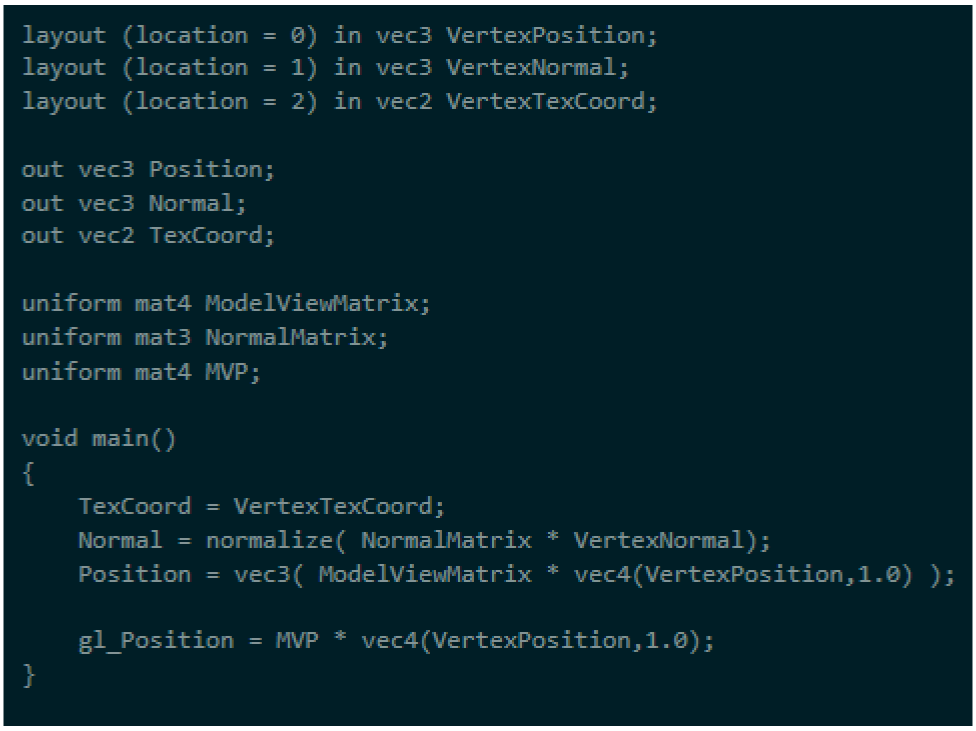
The end result:



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.

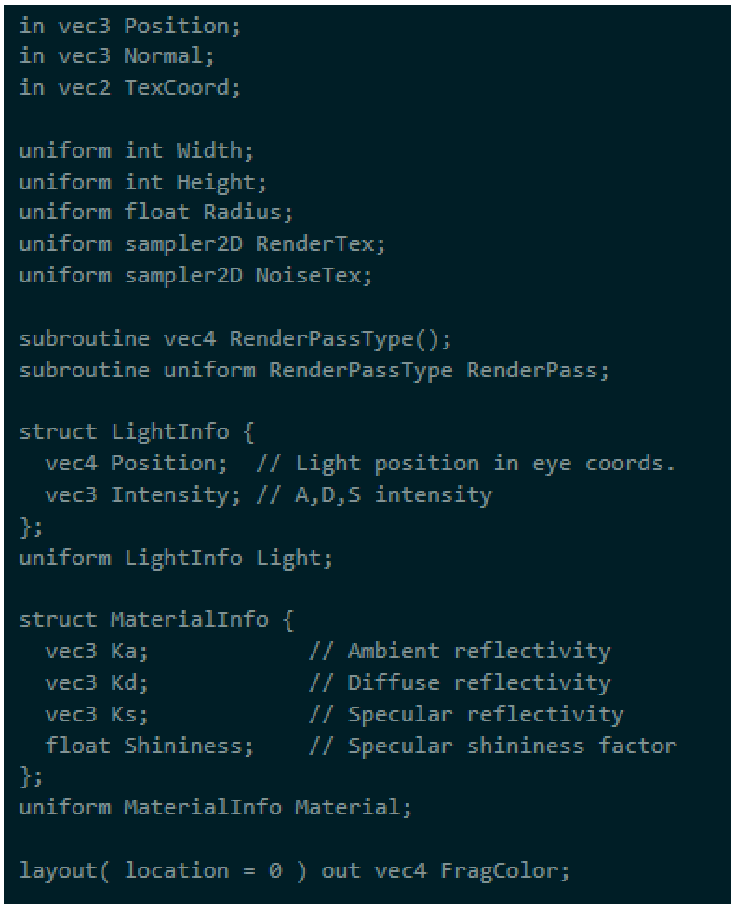
We'll apply the night-vision effect as a second pass to the rendered scene. The first pass will render the scene to a texture and the second pass will apply the night-vision effect.

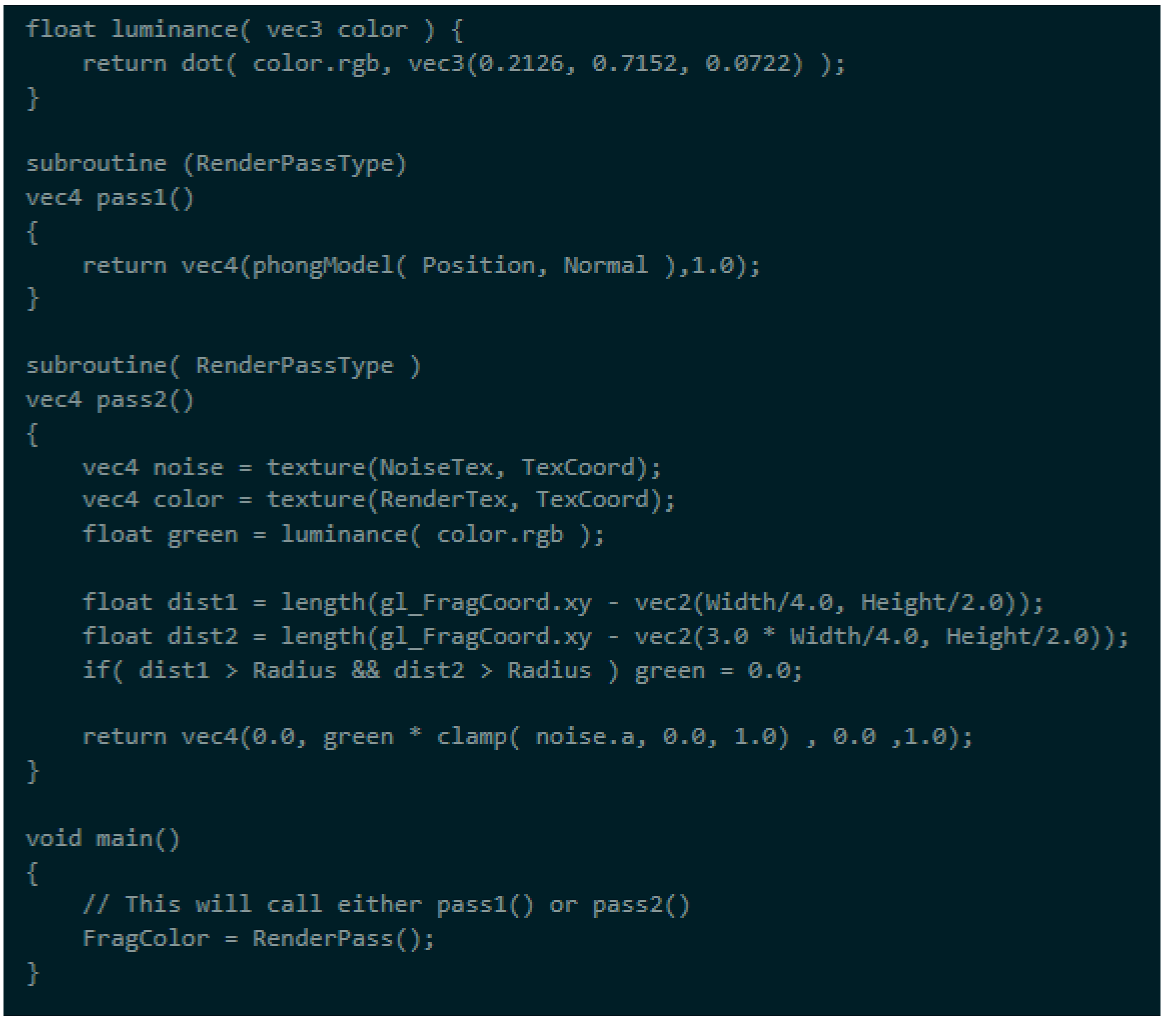
Vertex shader:


```
layout (location = 0) in vec3 VertexPosition;
layout (location = 1) in vec3 VertexNormal;
layout (location = 2) in vec2 VertexTexCoord;

out vec3 Position;
out vec3 Normal;
out vec2 TexCoord;

uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform mat4 MVP;

void main()
{
    TexCoord = VertexTexCoord;
    Normal = normalize( NormalMatrix * VertexNormal);
    Position = vec3( ModelViewMatrix * vec4(VertexPosition,1.0) );

    gl_Position = MVP * vec4(VertexPosition,1.0);
}
```

Fragment shader:

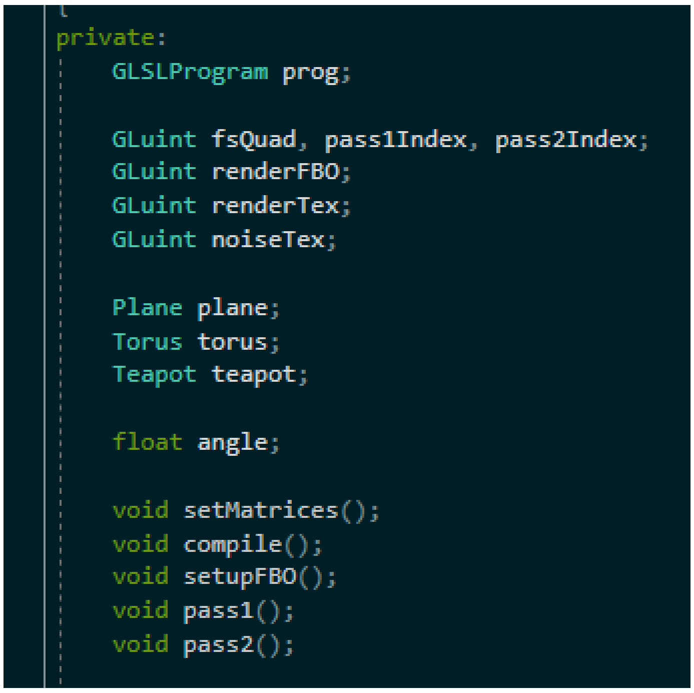
Use this fragment shader. Keep in mind that I didn’t screenshot the phongModel implementation, you can do that yourself.


```
in vec3 Position;
in vec3 Normal;
in vec2 TexCoord;

uniform int Width;
uniform int Height;
uniform float Radius;
uniform sampler2D RenderTex;
uniform sampler2D NoiseTex;

subroutine vec4 RenderPassType();
subroutine uniform RenderPassType RenderPass;

struct LightInfo {
  vec4 Position;  // Light position in eye coords.
  vec3 Intensity; // A,D,S intensity
};
uniform LightInfo Light;

struct MaterialInfo {
  vec3 Ka;            // Ambient reflectivity
  vec3 Kd;            // Diffuse reflectivity
  vec3 Ks;            // Specular reflectivity
  float Shininess;    // Specular shininess factor
};
uniform MaterialInfo Material;

layout( location = 0 ) out vec4 FragColor;
```



```
float luminance( vec3 color ) {
    return dot( color.rgb, vec3(0.2126, 0.7152, 0.0722) );
}

subroutine (RenderPassType)
vec4 pass1()
{
    return vec4(phongModel( Position, Normal ),1.0);
}

subroutine( RenderPassType )
vec4 pass2()
{
    vec4 noise = texture(NoiseTex, TexCoord);
    vec4 color = texture(RenderTex, TexCoord);
    float green = luminance( color.rgb );

    float dist1 = length(gl_FragCoord.xy - vec2(Width/4.0, Height/2.0));
    float dist2 = length(gl_FragCoord.xy - vec2(3.0 * Width/4.0, Height/2.0));
    if( dist1 > Radius && dist2 > Radius ) green = 0.0;

    return vec4(0.0, green * clamp( noise.a, 0.0, 1.0) , 0.0 ,1.0);
}

void main()
{
    // This will call either pass1() or pass2()
    FragColor = RenderPass();
}
```

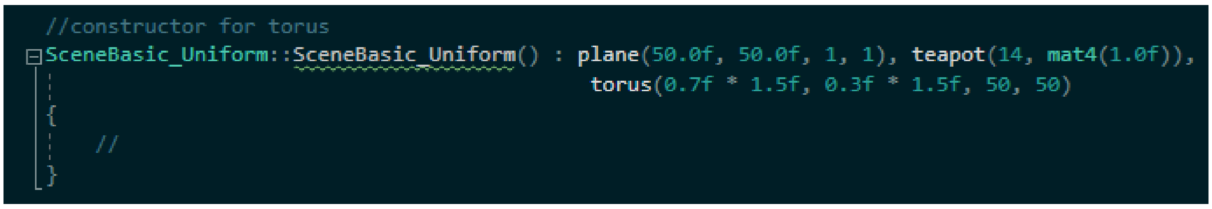
scenebasic_uniform.h:



```
{
private:
    GLSLProgram prog;

    GLuint fsQuad, pass1Index, pass2Index;
    GLuint renderFBO;
    GLuint renderTex;
    GLuint noiseTex;

    Plane plane;
    Torus torus;
    Teapot teapot;

    float angle;

    void setMatrices();
    void compile();
    void setupFBO();
    void pass1();
    void pass2();
```

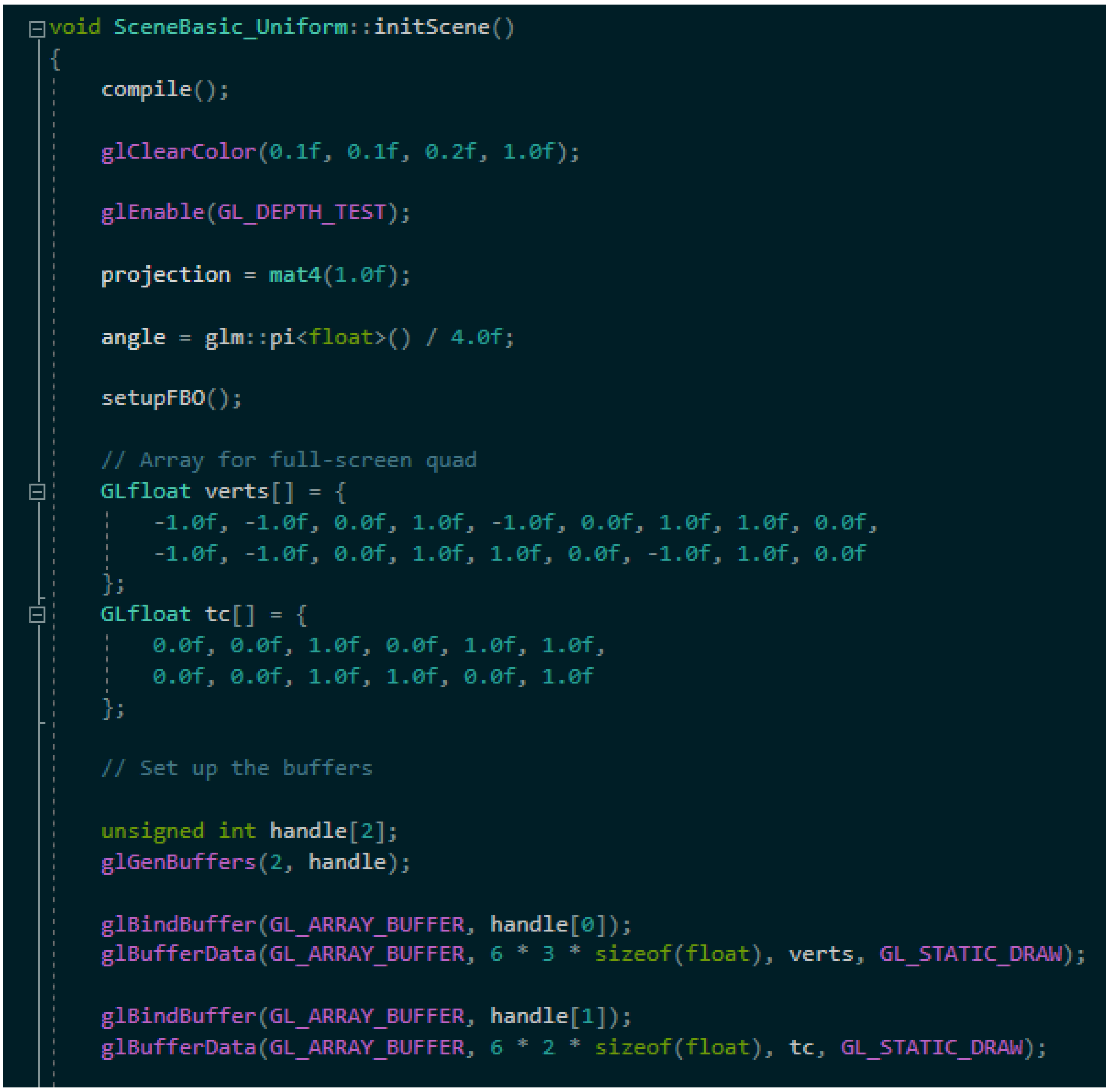
scenebasic_uniform.cpp:

For constructor use this:

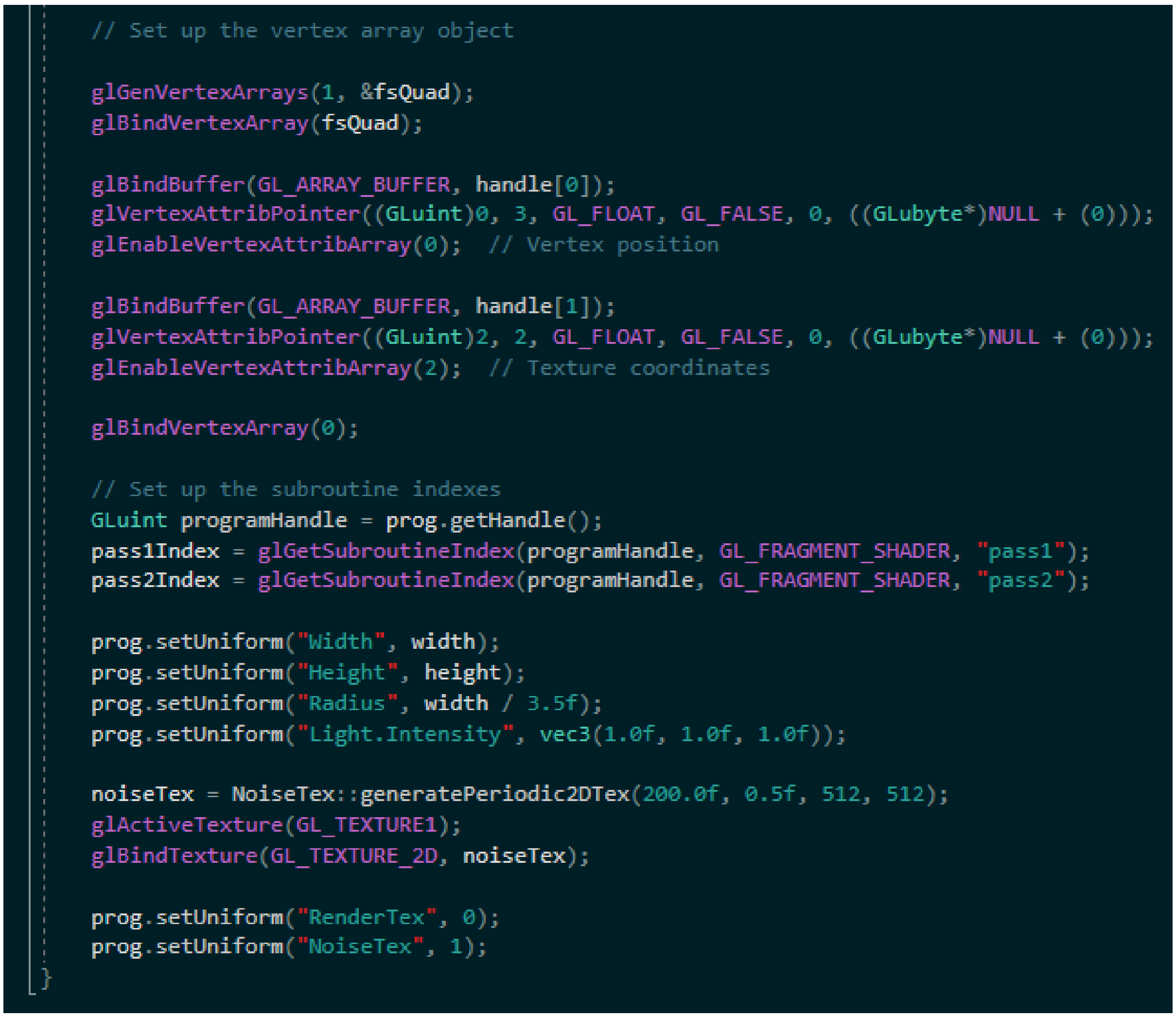


```
//constructor for torus
SceneBasic_Uniform::SceneBasic_Uniform() : plane(50.0f, 50.0f, 1, 1), teapot(14, mat4(1.0f)),
                                           torus(0.7f * 1.5f, 0.3f * 1.5f, 50, 50)
{
    //
}
```

For initScene():

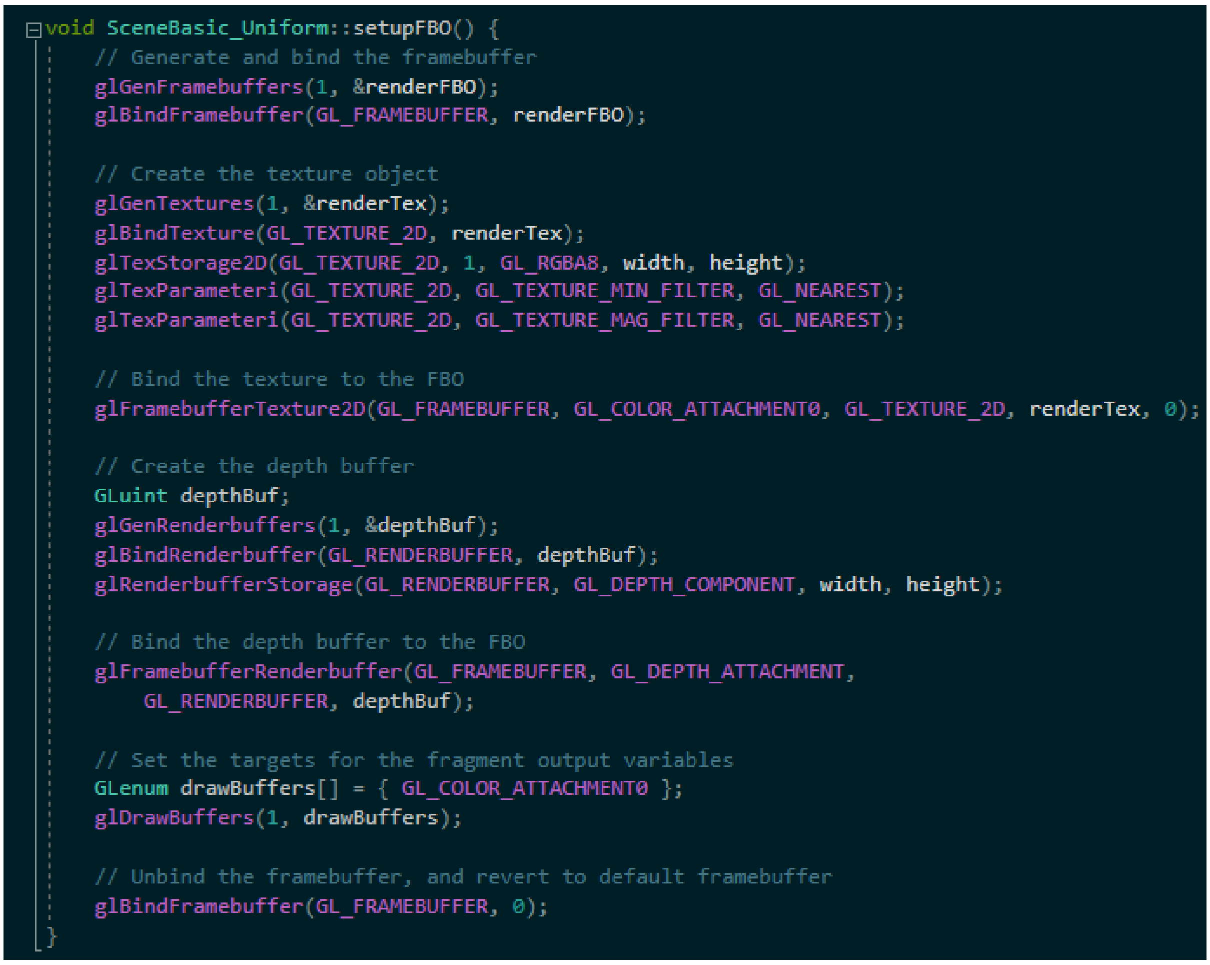


```
void SceneBasic_Uniform::initScene()
{
    compile();

    glClearColor(0.1f, 0.1f, 0.2f, 1.0f);

    glEnable(GL_DEPTH_TEST);

    projection = mat4(1.0f);

    angle = glm::pi<float>() / 4.0f;

    setupFBO();

    // Array for full-screen quad
    GLfloat verts[] = {
        -1.0f, -1.0f, 0.0f, 1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f,
        -1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f, -1.0f, 1.0f, 0.0f
    };
    GLfloat tc[] = {
        0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f
    };

    // Set up the buffers

    unsigned int handle[2];
    glGenBuffers(2, handle);

    glBindBuffer(GL_ARRAY_BUFFER, handle[0]);
    glBufferData(GL_ARRAY_BUFFER, 6 * 3 * sizeof(float), verts, GL_STATIC_DRAW);

    glBindBuffer(GL_ARRAY_BUFFER, handle[1]);
    glBufferData(GL_ARRAY_BUFFER, 6 * 2 * sizeof(float), tc, GL_STATIC_DRAW);
```



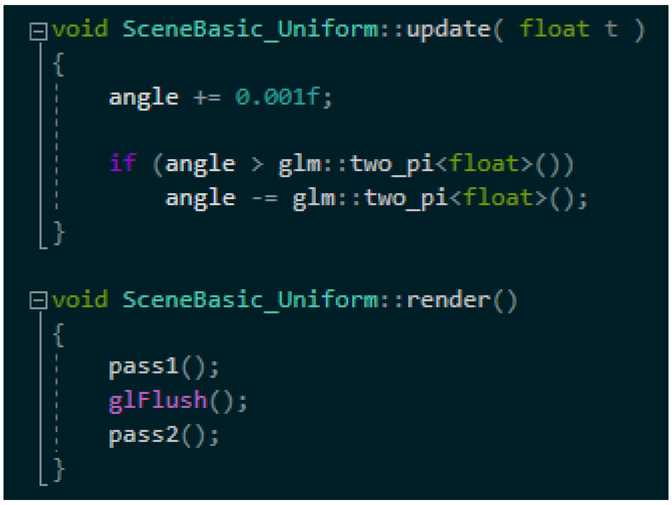
```
    // Set up the vertex array object

    glGenVertexArrays(1, &fsQuad);
    glBindVertexArray(fsQuad);

    glBindBuffer(GL_ARRAY_BUFFER, handle[0]);
    glVertexAttribPointer((GLuint)0, 3, GL_FLOAT, GL_FALSE, 0, ((GLubyte*)NULL + (0)));
    glEnableVertexAttribArray(0);  // Vertex position

    glBindBuffer(GL_ARRAY_BUFFER, handle[1]);
    glVertexAttribPointer((GLuint)2, 2, GL_FLOAT, GL_FALSE, 0, ((GLubyte*)NULL + (0)));
    glEnableVertexAttribArray(2);  // Texture coordinates

    glBindVertexArray(0);

    // Set up the subroutine indexes
    GLuint programHandle = prog.getHandle();
    pass1Index = glGetSubroutineIndex(programHandle, GL_FRAGMENT_SHADER, "pass1");
    pass2Index = glGetSubroutineIndex(programHandle, GL_FRAGMENT_SHADER, "pass2");

    prog.setUniform("Width", width);
    prog.setUniform("Height", height);
    prog.setUniform("Radius", width / 3.5f);
    prog.setUniform("Light.Intensity", vec3(1.0f, 1.0f, 1.0f));

    noiseTex = NoiseTex::generatePeriodic2DTex(200.0f, 0.5f, 512, 512);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, noiseTex);

    prog.setUniform("RenderTex", 0);
    prog.setUniform("NoiseTex", 1);
}
```

For setupFBO():



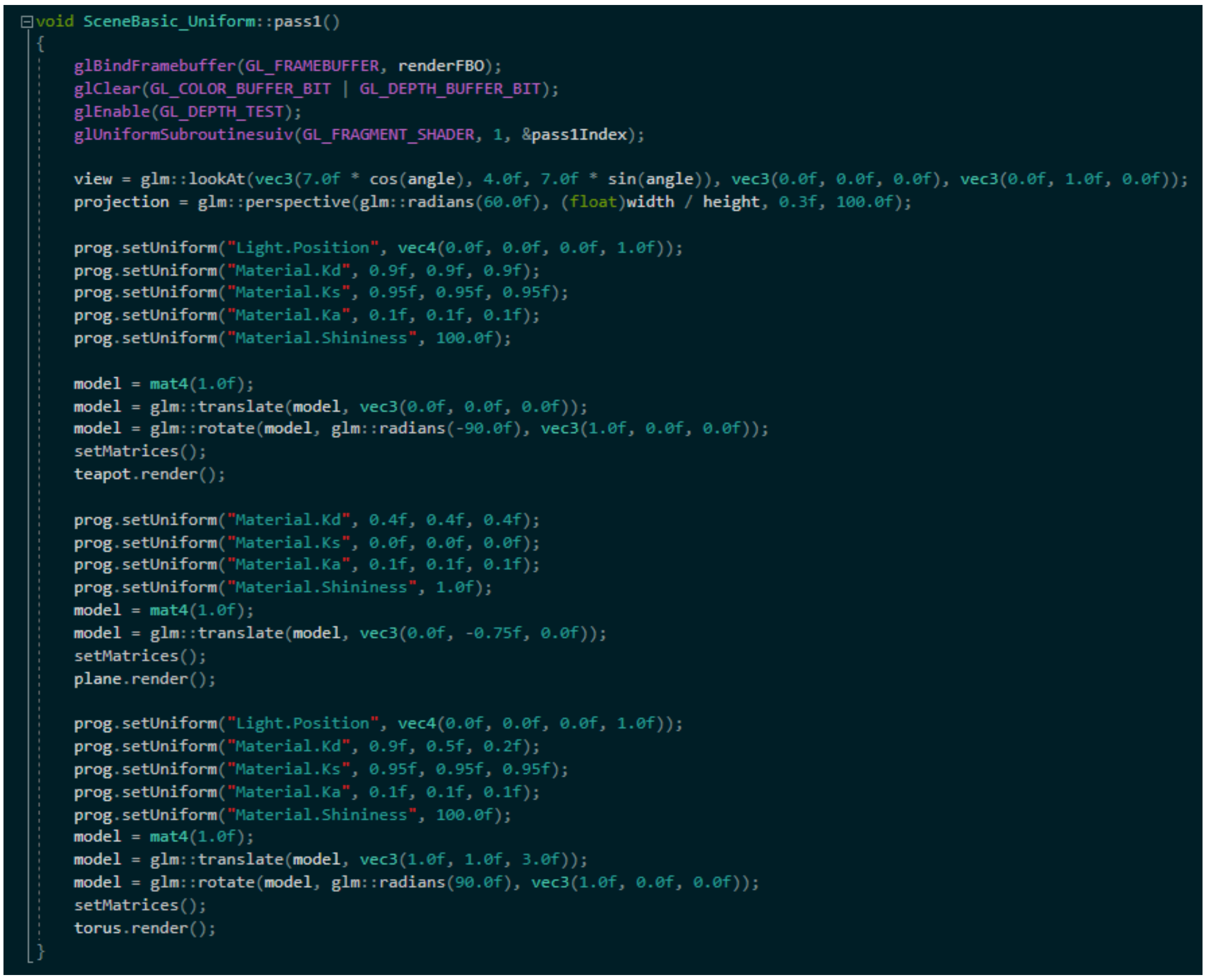
```
void SceneBasic_Uniform::setupFBO() {
    // Generate and bind the framebuffer
    glGenFramebuffers(1, &renderFBO);
    glBindFramebuffer(GL_FRAMEBUFFER, renderFBO);

    // Create the texture object
    glGenTextures(1, &renderTex);
    glBindTexture(GL_TEXTURE_2D, renderTex);
    glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGBA8, width, height);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    // Bind the texture to the FBO
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, renderTex, 0);

    // Create the depth buffer
    GLuint depthBuf;
    glGenRenderbuffers(1, &depthBuf);
    glBindRenderbuffer(GL_RENDERBUFFER, depthBuf);
    glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, width, height);

    // Bind the depth buffer to the FBO
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
        GL_RENDERBUFFER, depthBuf);

    // Set the targets for the fragment output variables
    GLenum drawBuffers[] = { GL_COLOR_ATTACHMENT0 };
    glDrawBuffers(1, drawBuffers);

    // Unbind the framebuffer, and revert to default framebuffer
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

For update() and render():



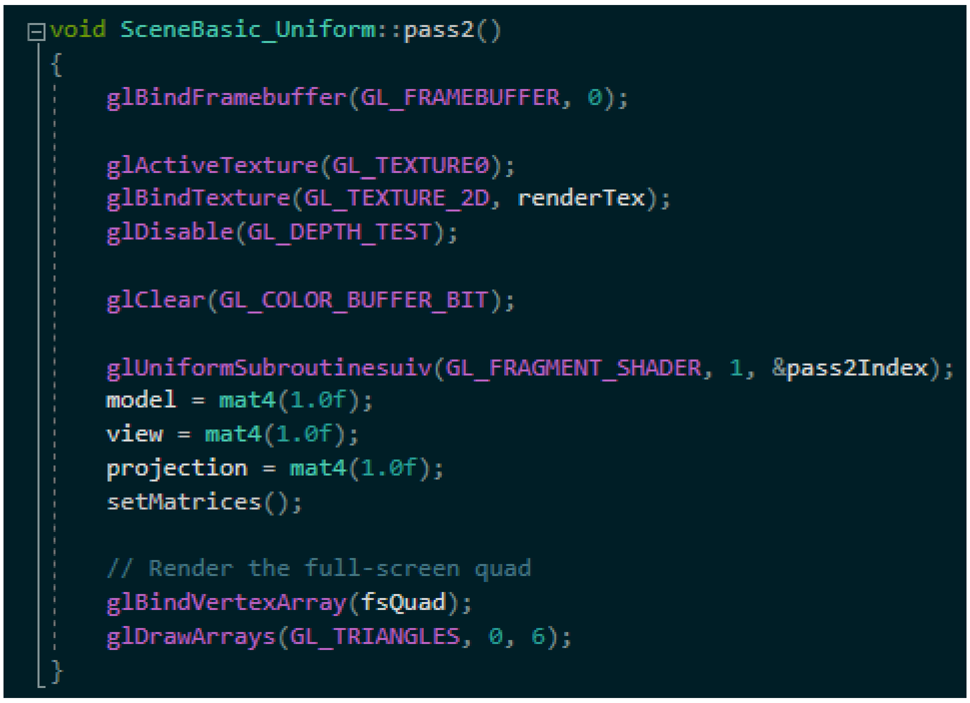
```
void SceneBasic_Uniform::update( float t )
{
    angle += 0.001f;

    if (angle > glm::two_pi<float>())
        angle -= glm::two_pi<float>();
}

void SceneBasic_Uniform::render()
{
    pass1();
    glFlush();
    pass2();
}
```

For pass1():



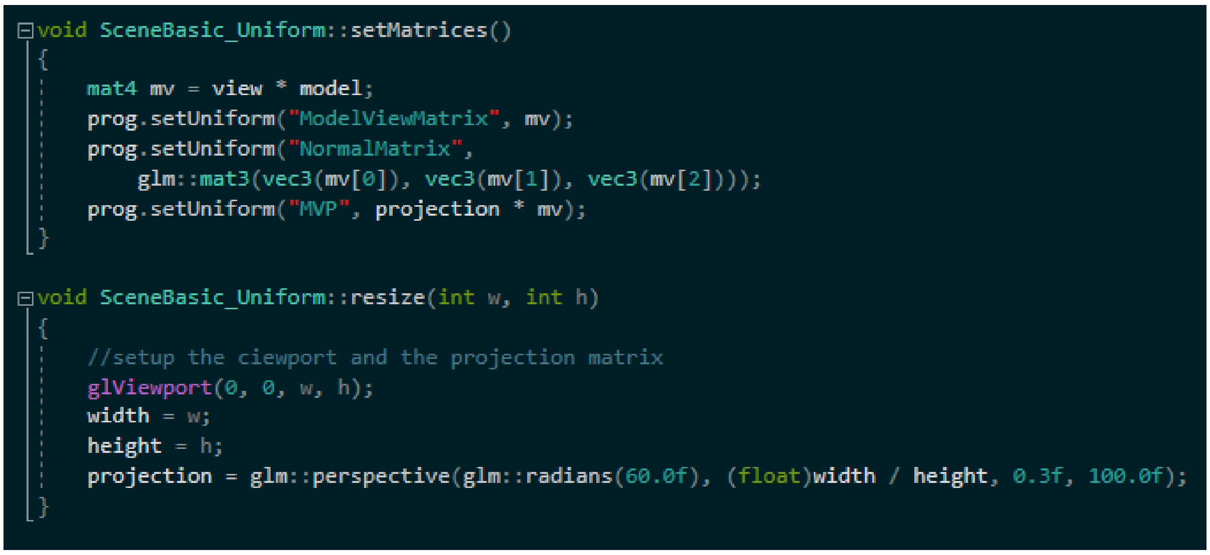
```
void SceneBasic_Uniform::pass1()
{
    glBindFramebuffer(GL_FRAMEBUFFER, renderFBO);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glUniformSubroutinesuiv(GL_FRAGMENT_SHADER, 1, &pass1Index);

    view = glm::lookAt(vec3(7.0f * cos(angle), 4.0f, 7.0f * sin(angle)), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));
    projection = glm::perspective(glm::radians(60.0f), (float)width / height, 0.3f, 100.0f);

    prog.setUniform("Light.Position", vec4(0.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("Material.Kd", 0.9f, 0.9f, 0.9f);
    prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
    prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
    prog.setUniform("Material.Shininess", 100.0f);

    model = mat4(1.0f);
    model = glm::translate(model, vec3(0.0f, 0.0f, 0.0f));
    model = glm::rotate(model, glm::radians(-90.0f), vec3(1.0f, 0.0f, 0.0f));
    setMatrices();
    teapot.render();

    prog.setUniform("Material.Kd", 0.4f, 0.4f, 0.4f);
    prog.setUniform("Material.Ks", 0.0f, 0.0f, 0.0f);
    prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
    prog.setUniform("Material.Shininess", 1.0f);
    model = mat4(1.0f);
    model = glm::translate(model, vec3(0.0f, -0.75f, 0.0f));
    setMatrices();
    plane.render();

    prog.setUniform("Light.Position", vec4(0.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("Material.Kd", 0.9f, 0.5f, 0.2f);
    prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
    prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
    prog.setUniform("Material.Shininess", 100.0f);
    model = mat4(1.0f);
    model = glm::translate(model, vec3(1.0f, 1.0f, 3.0f));
    model = glm::rotate(model, glm::radians(90.0f), vec3(1.0f, 0.0f, 0.0f));
    setMatrices();
    torus.render();
}
```

For pass2():



```
void SceneBasic_Uniform::pass2()
{
    glBindFramebuffer(GL_FRAMEBUFFER, 0);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, renderTex);
    glDisable(GL_DEPTH_TEST);

    glClear(GL_COLOR_BUFFER_BIT);

    glUniformSubroutinesuiv(GL_FRAGMENT_SHADER, 1, &pass2Index);
    model = mat4(1.0f);
    view = mat4(1.0f);
    projection = mat4(1.0f);
    setMatrices();

    // Render the full-screen quad
    glBindVertexArray(fsQuad);
    glDrawArrays(GL_TRIANGLES, 0, 6);
}
```

For setMatrices() and resize():



```
void SceneBasic_Uniform::setMatrices()
{
    mat4 mv = view * model;
    prog.setUniform("ModelViewMatrix", mv);
    prog.setUniform("NormalMatrix",
        glm::mat3(vec3(mv[0]), vec3(mv[1]), vec3(mv[2])));
    prog.setUniform("MVP", projection * mv);
}

void SceneBasic_Uniform::resize(int w, int h)
{
    //setup the ciewport and the projection matrix
    glViewport(0, 0, w, h);
    width = w;
    height = h;
    projection = glm::perspective(glm::radians(60.0f), (float)width / height, 0.3f, 100.0f);
}
```

That’s it.