Homework #3 Report

[CSIC30108] Computer Graphics 2022

Student ID: 311605004 **Name:** 劉子齊 **Date:** 2022.12.05

1. Introduction:

In this project, our goal is to implement a program capable in performing several rendering effects, such as skybox, grayscale filter, and edge detection filter, based on the provided template with OpenGL and GLSL. According to the figure provided by the specification attached below, the final result of our implementation should contain the skybox, 3 cubes, 2 mugs, and a wooden floor, also the corresponding rendering effects.

2. Implementation Details:

First of all, as shown in the following figures, I started with implementing the skybox mode. As shown in the following figure, I first created a model and manually and set the box positions by making use of the position data specified in the variable named skyboxVerices.

```
99    m = new Model();
100
101   for (int i = 0; i < 108;  i++) {
102     m->positions.push_back(skyboxVertices[i]);
103   }
104
105   m->textures.push_back(createCubemap(blueSkyboxfaces));
106
107   m->numVertex = 108;
108   //m->drawMode = GL_QUADS;
109   attachSkyboxVAO(m);
110   ctx.models.push_back(m);
```

After setting up the skybox mode, I started working on the texture loading of the skybox. Different from the texture of other objects, there's 6 faces for the skybox's texture. After obtaining the texture from their corresponding files, I generated the textures and bind them to the cube map. Then I loaded the texture images in through the stbi_load function, also aligned them to their right position through binding the corresponding position to these textures. Afterwards, I set the MIN & MAG filter to linear and wrap them to the mode of clamping to the edge before returning the texture ID.

```
125   GLuint textureID;
126   glGenTextures(1, &textureID);
127   glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
128
129   int width, height, nrChannels;
130
131   for (int i = 0; i < 6; i++) {
132     unsigned char* data = stbi_load(faces[i], &width, &height, &nrChannels, 0);
133
134     if (data) {
135       glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
136       stbi_image_free(data);
137     } else {
138       std::cout << "Cubemap tex failed to load at path: " << faces[i] << std::endl;
139       // stbi_image_free(data);
140     }
141   }
142
143   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
144   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
145   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
146   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
147   glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
148
149   return textureID;
```

With the textures set, I started creating the VAO and the VBO for the skybox. I bonded the vertex array to the skyboxVAO and bonded the buffer data to the skyboxVBO, which implementation is shown in the following figure.

```
46    GLuint* skyboxVAO = new GLuint[1];
47
48    glGenVertexArrays(1, skyboxVAO);
49    glBindVertexArray(skyboxVAO[0]);
50    model->vao = skyboxVAO[0];
```

```
51
52    GLuint skyboxVBO[1];
53    glGenBuffers(1, skyboxVBO);
54    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO[0]);
55    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), model->positions.data(), GL_STATIC_DRAW);
56    glEnableVertexAttribArray(0);
57    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
58
59    glBindBuffer(GL_ARRAY_BUFFER, 0);
60    glBindVertexArray(0);
```

Then I moved onto the implementation of the skybox.cpp, which is shown in the figure below. The main loop of my SkyboxProgram starts with disabling the depth mask in the beginning following with loading the vao of the skybox. Afterwards, I passed the projection matrix, the view matrix, and the skybox shader to the glsl shader. As all the things are passed properly, I drew the skybox with glDrawArrays, also I enable the depth mask in the end.

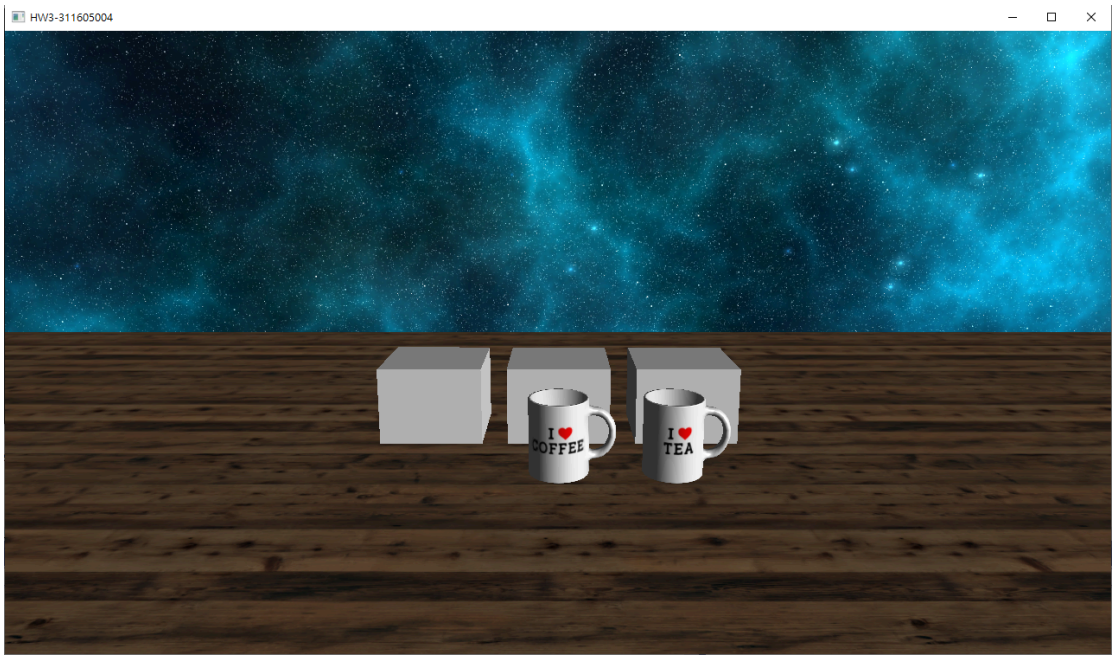
```
22    glDepthMask(GL_FALSE);
23    glBindVertexArray(model->vao);
24
25    const float* p = ctx->camera->getProjectionMatrix();
26    GLint pmatLoc = glGetUniformLocation(programId, "Projection");
27    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);
28
29    const float* v = glm::value_ptr(glm::mat4(glm::mat3(ctx->camera->getViewMatrixGLM())));
30    GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
31    glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);
32
33
34    glActiveTexture(GL_TEXTURE0);
35    glBindTexture(GL_TEXTURE_CUBE_MAP, model->textures[ctx->skybox->textureIndex]);
36    glUniform1i(glGetUniformLocation(programId, "skybox"), 0);
37    glDrawArrays(model->drawMode, 0, model->numVertex);
38    glDepthMask(GL_TRUE);
39
40    glUseProgram(0);
```

Afterwards, I implemented the glsl shaders as the following figures. For the vertex shader, I set the TexCoord as the position, and the multiplication of Projection, ViewMatrix, and position in the form of 4D vector as the gl_Position. For the fragment shader, I set the color with skybox and TexCoord by the texture function.

```
1   #version 430
2
3   layout(location = 0) in vec3 position;
4
5   out vec3 TexCoord;
6
7   uniform mat4 Projection;
8   uniform mat4 ViewMatrix;
9
10  // TODO#1-2: vertex shader / fragment shader
11  // 1. properly set gl_Position and TexCoord in vertex shader
12  // 2. properly set color with skybox texture and input from vertex shader
13
14  void main()
15  {
16          TexCoord = position;
17          gl_Position = Projection * ViewMatrix * vec4(position, 1.0);
18  }
```

```
1   #version 430
2
3   in vec3 TexCoord;
4
5   out vec4 color;
6
7   uniform samplerCube skybox;
8
9   void main()
10  {
11          color = texture(skybox, TexCoord);
12  }
```

By the implementations explained above, we would obtain the result in the figure below, which we can see the skybox and the 3 cubes, 2 mugs, and a plane with their corresponding texture.



After finishing the skybox, I started implementing the shadow program. I started implementing the shadow.cpp by generating the frame buffer and the depth map for the shadow program, which implementation is shown in the figure below.

First, I generated the frame buffer and store it to the depthMapFBO. Then I generated the texture of the depth map and store it to the shadowMapTexture in ctx, also I set up the parameters of the texture properly to make sure all the things are just right. Hereafter, I bonded the texture to the depth buffer of the frame buffer, and disable the I/O of the color buffer.

```
38    glGenFramebuffers(1, &depthMapFBO);
39
40    glGenTextures(1, &(ctx->shadowMapTexture));
41    glBindTexture(GL_TEXTURE_2D, ctx->shadowMapTexture);
42    glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_MAP_SIZE, SHADOW_MAP_SIZE, 0, GL_DEPTH_COMPONENT, GL
43
44    glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
45    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
46    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
47    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
48    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
49
```

```
50    glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
51    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, ctx->shadowMapTexture, 0);
52
53    glDrawBuffer(GL_NONE);
54    glReadBuffer(GL_NONE);
55
56    glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

After generating the frame buffer, I started rendering the depth map with shader. At the beginning, I changed the viewport to the size of the depth map, and I bonded out the frame buffer implemented previously. Afterwards, I conducted the calculation of the light view matrix.

For the light view matrix, I first changed the original perspective projection of directional light into orthogonal projection through the glm::ortho function. Then I obtained the "fake" position of the directional light through multiplying the light direction by −10. Hereafter, I obtained the light view through the glm::lookAt function and multiply it to the light projection matrix we just obtained then we can get the light view matrix.

Next, I render all the scene models as usual. Finally, I restored the viewport back to the screen size, which I obtained from the OpenGLContext::getWidth and OpenGLContext::getHeight, and the frame buffer back to 0. The implementation is shown in the figure below.

```
77    glViewport(0, 0, SHADOW_MAP_SIZE, SHADOW_MAP_SIZE);
78    glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
79    glClear(GL_DEPTH_BUFFER_BIT);
80
81    int obj_num = (int)ctx->objects.size();
82
83    for (int i = 0; i < obj_num; i++) {
84
85      int modelIndex = ctx->objects[i]->modelIndex;
86      Model* model = ctx->models[modelIndex];
87
88      glBindVertexArray(model->vao);
89
90      const float* m = glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
91      GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
92      glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);
93
94      GLfloat near_plane = 1.0f;
95      GLfloat far_plane = 7.5f;
96
97      glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
98      glm::vec3 fake_LightPos = ctx->lightDirection * -10.0f;
99      glm::mat4 lightView = glm::lookAt(fake_LightPos, glm::vec3(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
100     glm::mat4 LightViewMatrix = lightProjection * lightView;
101
```

```
    const float* lvm = glm::value_ptr(LightViewMatrix);
    GLint lvmatLoc = glGetUniformLocation(programId, "LightViewMatrix");
    glUniformMatrix4fv(lvmatLoc, 1, GL_FALSE, lvm);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);
    glDrawArrays(model->drawMode, 0, model->numVertex);
  }

  glBindFramebuffer(GL_FRAMEBUFFER, 0);

  glViewport(0, 0, OpenGLContext::getWidth(), OpenGLContext::getHeight());
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glUseProgram(0);
```

Then we move onto the shadow light program, the whole shadowLight.cpp is basically the same to light.cpp, except for several parts I added, which implementations are shown below. For the light view matrix and the fake light position, their calculation is exactly the same as I just mentioned above. Besides, we need to pass the shadow map to the shader here, which passing method is basically the same as passing original textures.

```
    GLfloat near_plane = 1.0f;
    GLfloat far_plane = 7.5f;

    glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
    glm::vec3 fake_LightPos = ctx->lightDirection * -10.0f;
    glm::mat4 lightView = glm::lookAt(fake_LightPos, glm::vec3(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    glm::mat4 LightViewMatrix = lightProjection * lightView;

    const float* lvm = glm::value_ptr(LightViewMatrix);
    mmatLoc = glGetUniformLocation(programId, "LightViewMatrix");
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, lvm);

    GLint eable_shadowLoc = glGetUniformLocation(programId, "enableShadow");
    glUniform1i(eable_shadowLoc, ctx->enableShadow);

    GLint fakeLightPosition_Loc = glGetUniformLocation(programId, "fakeLightPos");
    glUniform3fv(fakeLightPosition_Loc, 1, glm::value_ptr(fake_LightPos));
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, ctx->shadowMapTexture);
    glUniform1i(glGetUniformLocation(programId, "shadowMap"), 1);
```

Next, I moved onto the glsl shader of the shadow light. For the vertex shader, the calculation of each of the output variables and the variables passed to the fragment shader are shown in the figure below.

```
void main() {
        gl_Position = Projection * ViewMatrix * ModelMatrix * vec4(position, 1.0);
        FragPos = vec3(ModelMatrix * vec4(position, 1.0));
        Normal = mat3(TIModelMatrix) * normal;
        TexCoord = texCoord;
        LightFragPost = LightViewMatrix * vec4(FragPos, 1.0);
}
```

For the fragment shader, I implemented the shadow calculation function. As the calculation of the shadow, I first transform the clip-space coordinates in the range of [-w, w] to [-1, 1] by dividing the x, y, and z component by the w component of the vector.

Afterwards, since the depth from the depth map is in the range [0,1] and I also want to use projCoords to sample from the depth map, I transform the NDC coordinates to the range [0,1].

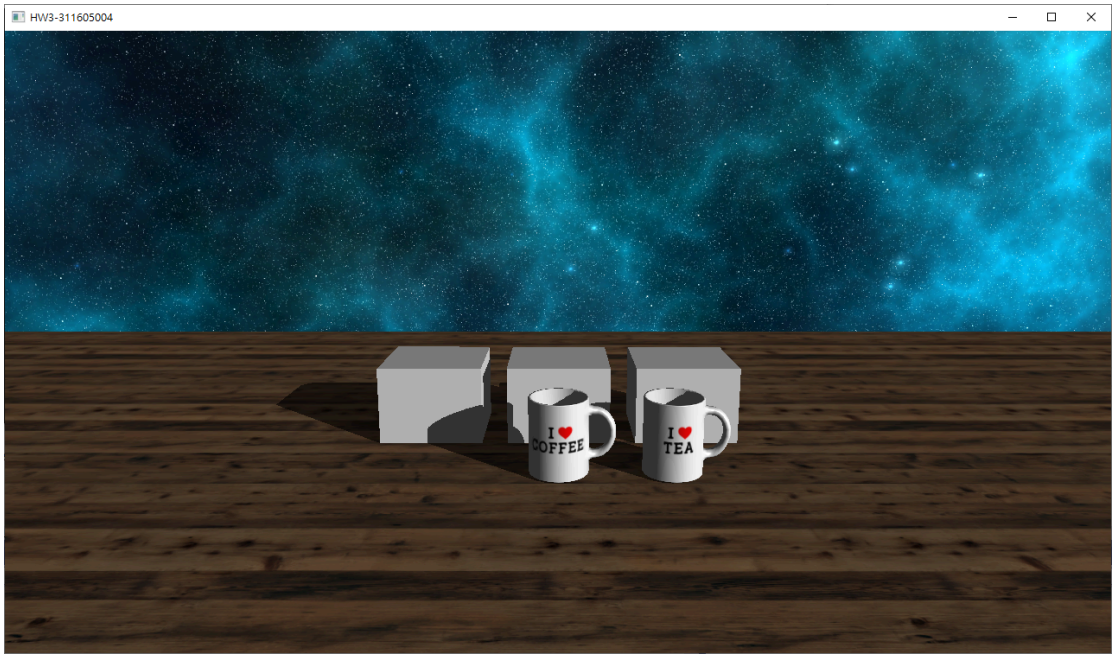
Hereafter, I sampled the depth map from projCoords directly correspond to the transformed NDC coordinates from the first render pass, which provided the closest depth from the light's point of view.

To get the current depth at this fragment I retrieve the projected vector's z coordinate which equals the depth of this fragment from the light's perspective.

Finally, I made a comparison which checks whether the current depth is greater than closest depth and if so, the fragment is in shadow.

```
29  float ShadowCalculation(){
30      float bias = 0.002;
31      // TODO
32
33      vec3 projCoords = LightFragPost.xyz / LightFragPost.w;
34      projCoords = projCoords * 0.5 + 0.5;
35
36      float closestDepth = texture(shadowMap, projCoords.xy).r;
37      float currentDepth = projCoords.z;
38
39      float shadow = currentDepth-bias > closestDepth  ? 1.0 : 0.0;
40
41      if(currentDepth > 1.0){
42          shadow = 0.0;
43      }
44
45      return shadow;
46  }
```

By the implementations explained above, we should obtain the result in the figure below, which we can see the skybox and the objects with their corresponding texture and shadow.



With the shadow rendering done, I started to work on the filters performed advanced shading, such as edge detection and grayscale, which I started with the filter program. First, I designed the quad. vertices data for the filter program NDC. I separated the data into two parts, which are position and texcoord. Then I generated the framebuffer, the quadVAO, and the quadVBO for the filter through the similar process as I mentioned earlier, which implementation is shown in the figure below.

```
13    float quadVertices_pos[] = { -1.0f,  1.0f, 0.0f,
14                                      -1.0f, -1.0f, 0.0f,
15                                       1.0f, -1.0f, 0.0f,
16
17                                      -1.0f,  1.0f,  0.0f,
18                                       1.0f,  -1.0f, 0.0f,
19                                       1.0f,   1.0f, 0.0f};
20
21    float quadVertices_tex[] = {0.0f, 1.0f,
22                                    0.0f, 0.0f,
23                                    1.0f, 0.0f,
24
25                                    0.0f, 1.0f,
26                                    1.0f, 0.0f,
27                                    1.0f, 1.0f  };
28
29    glGenFramebuffers(1, &filterFBO);
30
31    glGenVertexArrays(1, &quadVAO);
32    glGenBuffers(2, quadVBO);
33    glBindVertexArray(quadVAO);
34
35    glBindBuffer(GL_ARRAY_BUFFER, quadVBO[0]);
36    glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices_pos), &quadVertices_pos, GL_STATIC_DRAW);
37    glEnableVertexAttribArray(0);
38    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
39
```

```
40      glBindBuffer(GL_ARRAY_BUFFER, quadVBO[1]);
41      glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices_tex), &quadVertices_tex, GL_STATIC_DRAW);
42      glEnableVertexAttribArray(1);
43      glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void *)0);
44
45      glBindBuffer(GL_ARRAY_BUFFER, 0);
46      glBindVertexArray(0);
47
48      updateFrameBuffer(OpenGLContext::getWidth(), OpenGLContext::getHeight());
49  }
```

Afterwards, we move onto generating color buffer and depth buffer for the frame buffer. First, we generate the color buffer texture and store it in the colorBuffer. Since this update function will be triggered whenever the window is resized, we then set the texture size to the same size as the screen. Besides, we set the MIN filter and the MAG filter to linear. Hereafter, I generated a render buffer, bonded it in and stored it into the rboDepth. Also, I set the size of the render buffer to the same size as the screen. Finally, I attached the color buffer and the rboDepth back to the frame buffer. The implementation is shown in the figure below.

```
72      glBindFramebuffer(GL_FRAMEBUFFER, filterFBO);
73
74      glGenTextures(1, &colorBuffer);
75      glBindTexture(GL_TEXTURE_2D, colorBuffer);
76
77      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
78      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
79      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
80
81      glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, colorBuffer, 0);
82
83      glBindTexture(GL_TEXTURE_2D, 0);
84
85      GLuint rboDepth;
86      glGenRenderbuffers(1, &rboDepth);
87      glBindRenderbuffer(GL_RENDERBUFFER, rboDepth);
88      glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, SCR_WIDTH, SCR_HEIGHT);
89      glBindRenderbuffer(GL_RENDERBUFFER, 0);
90
91      glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, rboDepth);
92
93      if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE) {
94        std::cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!" << std::endl;
95      }
96      glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

In the doMainLoop, I passed the VAO, the edge detection enabling flag, the gray scale enabling flag, and the color buffer to the glsl shader and render them, which implementation is shown in the figure below.

```
110       glBindFramebuffer(GL_FRAMEBUFFER, 0);
111       glBindVertexArray(quadVAO);
112
```

```
    GLint enableEdgeDetection_Loc = glGetUniformLocation(programId, "enableEdgeDetection");
    glUniform1i(enableEdgeDetection_Loc, ctx->enableEdgeDetection);

    GLint eanbleGrayscale_Loc = glGetUniformLocation(programId, "eanbleGrayscale");
    glUniform1i(eanbleGrayscale_Loc, ctx->eanbleGrayscale);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, colorBuffer);
    glUniform1i(glGetUniformLocation(programId, "colorBuffer"), 0);
    glDrawArrays(GL_TRIANGLES, 0, 6);
```

For the fragment shader of the filter, I divided it into several parts, which are shown in the following figure.

If the gray scale filter is enabled, I will mix the color obtained from the color buffer and the texcoord through the texture function with the ratio (0.2126 : 0.7152 : 0.0722).

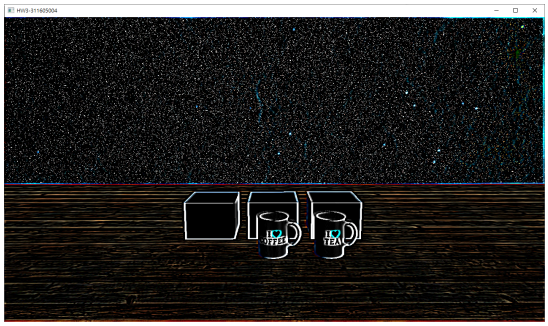
If the edge detection filter is enabled, I will add each offset to the current texture coordinate when sampling and multiply these texture values with the weighted kernel values that we add together, which offset and kernel values are pre-defined.

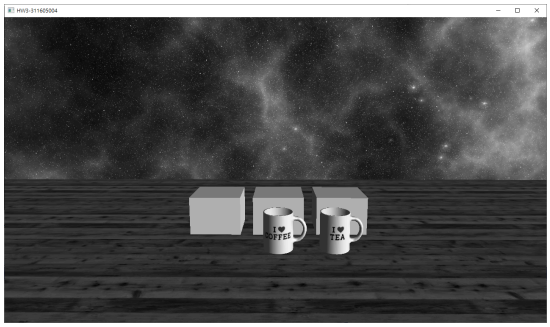
If both of them are enabled, I'll first conduct the edge detection filter then the gray scale filter. If none of them were enabled, I'll simply return the original color.

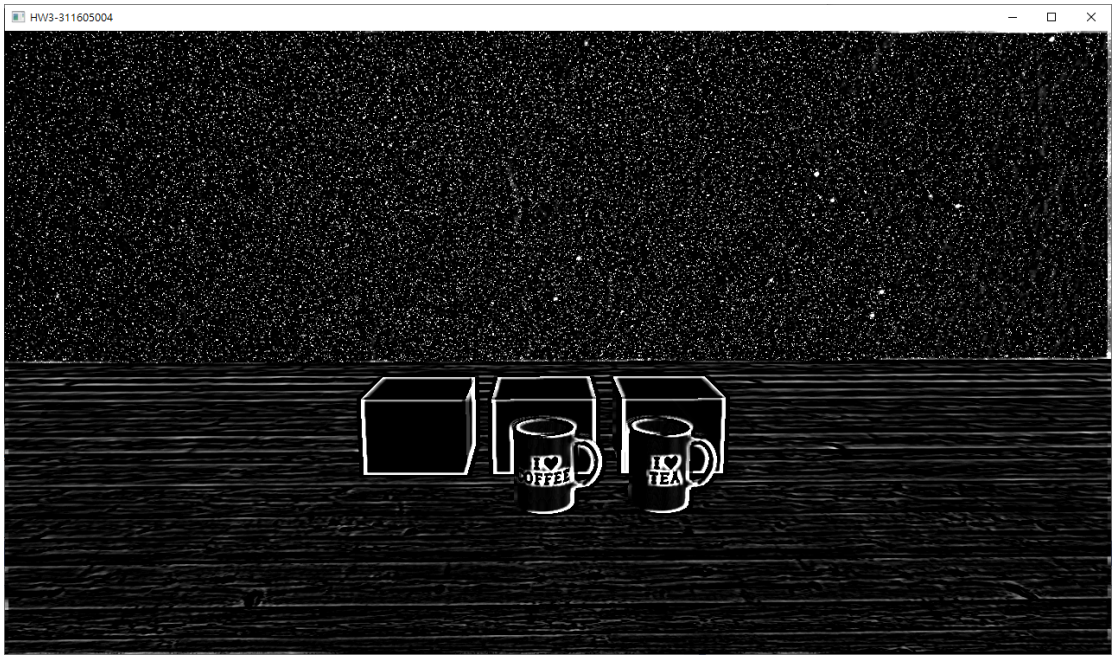
```
    if((eanbleGrayscale == 1) && (enableEdgeDetection == 1)){
        vec3 sampleTex[9];
        for(int i = 0; i < 9; i++){
            sampleTex[i] = vec3(texture(colorBuffer, TexCoord.st + offsets[i]));
        }

        vec3 col = vec3(0.0);
        for(int i = 0; i < 9; i++){
            col += sampleTex[i] * kernel[i];
        }

        color = vec4(col, 1.0);
        float average = 0.2126 * color.r + 0.7152 * color.g + 0.0722 * color.b;
        color = vec4(average, average, average, 1.0);
    }

    else if(eanbleGrayscale == 1){
        color = texture(colorBuffer, TexCoord);
        float average = 0.2126 * color.r + 0.7152 * color.g + 0.0722 * color.b;
        color = vec4(average, average, average, 1.0);
    }

    else if(enableEdgeDetection == 1){
        vec3 sampleTex[9];
        for(int i = 0; i < 9; i++){
            sampleTex[i] = vec3(texture(colorBuffer, TexCoord.st + offsets[i]));
        }

```

```
63          vec3 col = vec3(0.0);
64          for(int i = 0; i < 9; i++){
65              col += sampleTex[i] * kernel[i];
66          }
67
68          color = vec4(col, 1.0);
69      }
70
71      else{
72          color = texture(colorBuffer, TexCoord);
73      }
```

When all the parts explained above were implemented correctly, we can get the results shown in the following figures, which are the results of the edge detection filter, the gray scale filter, and both of them.







3. Problems Encountered

- **Rendering the skybox**

  In the very beginning of this project, I got some problem drawing the skybox, since the way how we add the skybox mode kind of confuses me in the very beginning. Besides, I kept setting the draw mode of the skybox into "GL_QUADS", which made the skybox to be extremely strange. However, after studying relevant references, this problem was resolved.

- **Relation between shadow & shadow light**

  Although the teaching assistant explained this thoroughly when going through the specification, this still bothered me a lot when implementing shadow rendering. When implementing shadow rendering, my screen always resulted in black screen or else my shadow seems weird. This really stuck me for days. However, I finally came through it after studying numerous references.

- **Usage of each OpenGL functions and their orders**

  This is probably the greatest obstacle of this project. Not only the parameters in each, but even the order of each functions affects the rendering result. It isn't a big deal if there's still rendering output, but it can be really despair if there's nothing for us to observe when the whole screen is black. However, after days of hard working, I finally solved all the problems.