HW3

TODO#1-1: Load skybox cubemap texture:

```
GLuint createCubemap(char faces[6][30]) {
    /* TODO#1-1: Load skybox cubemap texture
    *           1. Get right, left, bottom, top, front, back texture filename from parameter
    *           2. ref: https://learnopengl.com/Advanced-OpenGL/Cubemaps
    *           2. Genetare texture and bind to cube map
    *           3. Load image with stbi_load and bind to correct cubemap position
    *           4. Set MIN/MAG filter to linear and wrap to "clamp to edge"
    *           5. return the texture id
    * Note: for stbi_load, you can refer to createTexture above
    */
    GLuint texture;
    int width, height, nrChannels;
    unsigned char* data;
    //right, left, bottom, top, front, back
    GLenum DIRECT_FOR_GL_TEXTURE_CUBE_MAP[] = {GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
                                               GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
                                               GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z};

    glGenTextures(1, &texture);
    //GL_TEXTURE_CUBE_MAP
    glBindTexture(GL_TEXTURE_CUBE_MAP, texture);
    //right, left, bottom, top, front, back
    for (unsigned int i = 0; i < 6; i++) {
        data = stbi_load(faces[i], &width, &height, &nrChannels, 0);
        glTexImage2D(DIRECT_FOR_GL_TEXTURE_CUBE_MAP[i], 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
        stbi_image_free(data);
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);

    printf("createCubemap: %d\n", texture);
    return texture;
}
```

Skybox 的方向面的參數

Skybox 的各方向面的讀檔

參數設定

TODO#1-1: Add skybox mode:

```
m = new Model();
for (int i = 0; i < 108; i++) {
    m->positions.push_back(skyboxVertices[i]);
}
char faces[6][30];
strcpy_s(faces[0], "../assets/skybox/right.png");
strcpy_s(faces[1], "../assets/skybox/left.png");
strcpy_s(faces[2], "../assets/skybox/top.png");
strcpy_s(faces[3], "../assets/skybox/bot.png");
strcpy_s(faces[4], "../assets/skybox/front.png");
strcpy_s(faces[5], "../assets/skybox/back.png");

m->textures.push_back(createCubemap(faces));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(100.0f, 100.0f, 100.0f));

m->numVertex = 36;
m->drawMode = GL_TRIANGLES;

attachSkyboxVAO(m);

ctx.models.push_back(m);
```

給點座標

給材質檔案位置

TODO#1-2: Render skybox with shader:

```
const float* p = ctx->camera->getProjectionMatrix();
GLint pmatLoc = glGetUniformLocation(programId, "Projection");
glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);

const float* v = glm::value_ptr(ctx->camera->getViewMatrixGLM() * model->modelMatrix);
GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);


glUniform1i(glGetUniformLocation(programId, "skybox"), 0);
```

將變數傳給 shader

```
glBindVertexArray(model->vao);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, model->textures[ctx->skybox->textureIndex]);
```

Bind & Active Texture

```
glDepthMask(GL_FALSE);
glDrawArrays(model->drawMode, 0, model->numVertex);
glDepthMask(GL_TRUE);
```

畫 skybox

TODO#1-2: vertex shader / fragment shader

vertex shader

```
void main()
{
  vec4 pos = Projection * ViewMatrix * vec4(position, 1.0);
  gl_Position = vec4(pos.x, pos.y, pos.w, pos.w);
  TexCoord = vec3(position.x, position.y, position.z);
}
```

計算 gl_Position

傳 TexCoord 到 fragment shader

fragment shader

```
void main()
{
    color = texture(skybox, TexCoord);
}
```

取得材質 color

TODO#2-1 Generate frame buffer and depth map for shadow program

```
//1. Generate frame buffer and store to depthMapFBO
glGenFramebuffers(1, &depthMapFBO);
```

Generate frame buffer and store to depthMapFBO

```
//2. Generate depthmap texture and store to ctx->shadowMapTexture
const GLuint SHADOW_WIDTH = SHADOW_MAP_SIZE, SHADOW_HEIGHT = SHADOW_MAP_SIZE;
GLuint depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
ctx->shadowMapTexture = depthMap;
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
```

Generate depthmap texture and store to ctx->shadowMapTexture

```
//3. properly setup depthmap's texture paremters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
//4. bind texture to framebuffer's depth buffer and disable color buffer read and write
GLfloat borderColor[] = {1.0, 1.0, 1.0, 1.0};
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

bind texture to framebuffer's depth buffer and disable color buffer read and write

TODO#2-2: Render depth map with shader

```
//1. Change viewport to depth map size
const GLuint SHADOW_WIDTH = SHADOW_MAP_SIZE, SHADOW_HEIGHT = SHADOW_MAP_SIZE;
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
```

Change viewport to depth map size

```
//2. Bind out framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
```

Bind out framebuffer

```
//3. A directional light doesn't have a source position.
//However, for shadown map we need to render the scene from a light's perspective
//and thus render the scene from a position somewhere along the lines of the light direction.
//The position of the light need to be "ctx->lightDirection * -10.0f"
//4. Properly set LightViewMatrix(projection matrix * view matrux)
GLfloat near_plane = 1.0f, far_plane = 7.5f;
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
glm::mat4 lightView = glm::lookAt(ctx->lightDirection * -10.0f, glm::vec3(0.0f), glm::vec3(1.0));
glm::mat4 lightSpaceMatrix = lightProjection * lightView;
const float* v = glm::value_ptr(lightSpaceMatrix);
GLint vmatLoc = glGetUniformLocation(programId, "LightViewMatrix");
glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);
glCullFace(GL_FRONT);
```

Properly set LightViewMatrix(projection matrix * view matrux)

```
//5. Render all scene models as usual
int obj_num = (int)ctx->objects.size();
for (int i = 0; i < obj_num; i++) {
    int modelIndex = ctx->objects[i]->modelIndex;
    Model* model = ctx->models[modelIndex];
    const float* m =
        glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
    GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);
    glBindVertexArray(model->vao);
    glDrawArrays(model->drawMode, 0, model->numVertex);
    glBindVertexArray(0);
}
glCullFace(GL_BACK);
```

Render all scene models as usual

```
//6. restore viewport and framebuffer(get screen size from OpenGLContext::getWidth, OpenGLContext::getHeight)
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glViewport(0, 0, OpenGLContext::getWidth(), OpenGLContext::getHeight());
```

restore viewport and framebuffer(get screen size from OpenGLContext::getWidth, OpenGLContext::getHeight)

TODO#2-3: Render scene with shadow mapping

- Copy from LightProgram

```
int obj_num = (int)ctx->objects.size();

for (int i = 0; i < obj_num; i++) {
  int modelIndex = ctx->objects[i]->modelIndex;
  Model* model = ctx->models[modelIndex];
  glBindVertexArray(model->vao);

  const float* p = ctx->camera->getProjectionMatrix();
  GLint pmatLoc = glGetUniformLocation(programId, "Projection");
  glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);

  const float* v = ctx->camera->getViewMatrix();
  GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
  glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);

  const float* m = glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
  GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
  glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);

  glm::mat4 TIMatrix = glm::transpose(glm::inverse(model->modelMatrix));
  const float* ti = glm::value_ptr(TIMatrix);
  mmatLoc = glGetUniformLocation(programId, "TIModelMatrix");
  glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, ti);

  const float* vp = ctx->camera->getPosition();
  mmatLoc = glGetUniformLocation(programId, "viewPos");
  glUniform3f(mmatLoc, vp[0], vp[1], vp[2]);
```

```
// LightViewMatrix
GLfloat near_plane = 1.0f, far_plane = 7.5f;
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
glm::mat4 lightView = glm::lookAt(ctx->lightDirection * -10.0f, glm::vec3(0.0f), glm::vec3(1.0));
glm::mat4 lightSpaceMatrix = lightProjection * lightView;
const float* lvm = glm::value_ptr(lightSpaceMatrix);
GLint lvmatLoc = glGetUniformLocation(programId, "LightViewMatrix");
glUniformMatrix4fv(lvmatLoc, 1, GL_FALSE, lvm);
// fakeLightPos
const float* flp = glm::value_ptr(ctx->lightDirection * -10.0f);
GLint flpLoc = glGetUniformLocation(programId, "fakeLightPos");
glUniform3f(flpLoc, lvm[0], lvm[1], lvm[2]);
// shadowMap
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, ctx->shadowMapTexture);
glUniform1i(glGetUniformLocation(programId, "shadowMap"), 1);
// enableShadow
GLint eSmatLoc = glGetUniformLocation(programId, "enableShadow");
glUniform1i(eSmatLoc, ctx->enableShadow);
```

Pass LightViewMatrix, fakeLightPos, shadowMap, enableShadow to shader program

```
glUniform3fv(glGetUniformLocation(programId, "dl.direction"), 1, glm::value_ptr(ctx->lightDirection));
glUniform3fv(glGetUniformLocation(programId, "dl.ambient"), 1, glm::value_ptr(ctx->lightAmbient));
glUniform3fv(glGetUniformLocation(programId, "dl.diffuse"), 1, glm::value_ptr(ctx->lightDiffuse));
glUniform3fv(glGetUniformLocation(programId, "dl.specular"), 1, glm::value_ptr(ctx->lightSpecular));

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);
glUniform1i(glGetUniformLocation(programId, "ourTexture"), 0);
glDrawArrays(model->drawMode, 0, model->numVertex);
```

TODO#2-4: shadow-enabled shader with single direct light

vertex shader

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

fragment shader: 計算是否在陰影裡

```
float ShadowCalculation()
{
    float bias = 0.002;
    // TODO
    // perform perspective divide
    vec3 projCoords = LightFragPost.xyz / LightFragPost.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragPosLight as coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    if (currentDepth > 1.0)
        currentDepth = 1.0;
    // check whether current frag pos is in shadow
    float shadow = closestDepth + bias < currentDepth  ? 1.0 : 0.0;

    return shadow;
}
```

TODO#3-1: Generate Framebuffer and VAO/VBO for filter

```
float quadVertices[] = {
    -1.0f, 1.0f, 0.0f, 1.0f,
    -1.0f, -1.0f, 0.0f, 0.0f,
    1.0f, -1.0f, 1.0f, 0.0f,

    -1.0f, 1.0f, 0.0f, 1.0f,
    1.0f, -1.0f, 1.0f, 0.0f,
    1.0f, 1.0f, 1.0f, 1.0f
};
```

設定 quadVertices

```
glGenVertexArrays(1, &quadVAO);
glGenBuffers(1, &(quadVBO[0]));
glBindVertexArray(quadVAO);
glBindBuffer(GL_ARRAY_BUFFER, quadVBO[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices), &quadVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(float), (void*)0);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(float), (void*)(2 * sizeof(float)));
```

設定 VAO/VBO

並傳位置到 shader

```
updateFrameBuffer(OpenGLContext::getWidth(), OpenGLContext::getHeight());
```

TODO#3-1: generate color/depth buffer for frame buffer

```
glGenFramebuffers(1, &filterFBO);
glBindFramebuffer(GL_FRAMEBUFFER, filterFBO);

glGenTextures(1, &colorBuffer);
glBindTexture(GL_TEXTURE_2D, colorBuffer);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, colorBuffer, 0);
```

Generate color buffer texture and store in colorBuffer

Set texture size to SCR_WIDTH*SCR_HEIGHT

Set MIN/MAG filter to linear

```
unsigned int rbo;
glGenRenderbuffers(1, &rbo);
glBindRenderbuffer(GL_RENDERBUFFER, rbo);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, SCR_WIDTH, SCR_HEIGHT);
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, rbo);
```

Generate/Bind a render buffer inad store in rboDepth

Set Render buffer size to SCR_WIDTH*SCR_HEIGHT

Attach colorBuffer and rboDepth to filterFBO

```
if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
  std::cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!" << std::endl;

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

TODO#3-1: pass VAO, enableEdgeDetection, eanbleGrayscale, colorBuffer to shader and render

```
// enableEdgeDetection
GLint eEDLoc = glGetUniformLocation(programId, "enableEdgeDetection");
glUniform1i(eEDLoc, ctx->enableEdgeDetection);
// eanbleGrayscale
GLint eGLoc = glGetUniformLocation(programId, "eanbleGrayscale");
glUniform1i(eGLoc, ctx->eanbleGrayscale);
// colorBuffer
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, colorBuffer);
glUniform1i(glGetUniformLocation(programId, "colorBuffer"), 0);

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

pass VAO, enableEdgeDetection, eanbleGrayscale, colorBuffer to shader

```
glBindVertexArray(quadVAO);
glBindTexture(GL_TEXTURE_2D, colorBuffer);
glDrawArrays(GL_TRIANGLES, 0, 6);
```

render

TODO#3-2: apply filter to color

```
void main()
{
    color = vec4(vec3(texture(colorBuffer, TexCoord)), 1.0);
    if (enableEdgeDetection == 1)
    {
        vec3 sampleText[9];                                          邊緣運算
        for(int i=0; i < 9;++i)
        {
            sampleText[i] = vec3(texture(colorBuffer, TexCoord.st + offsets[i]));
        }
        // 利用權值求最終紋理顏色
        vec3 result = vec3(0.0);
        for(int i=0; i < 9;++i)
        {
            result += sampleText[i] * kernel[i];
        }
        color = vec4(result, 1.0);
    }
    if (eanbleGrayscale == 1)
    {                                                                灰階運算
        float average = 0.2126 * color.r + 0.7152 * color.g + 0.0722 * color.b;
        color = vec4(average, average, average, 1.0);
    }
}
```

Problems you encountered

- Task2 的 shadow 與 shadowLight 區分不清楚，誤以為在 shadow 就要畫出陰影

BONUS:

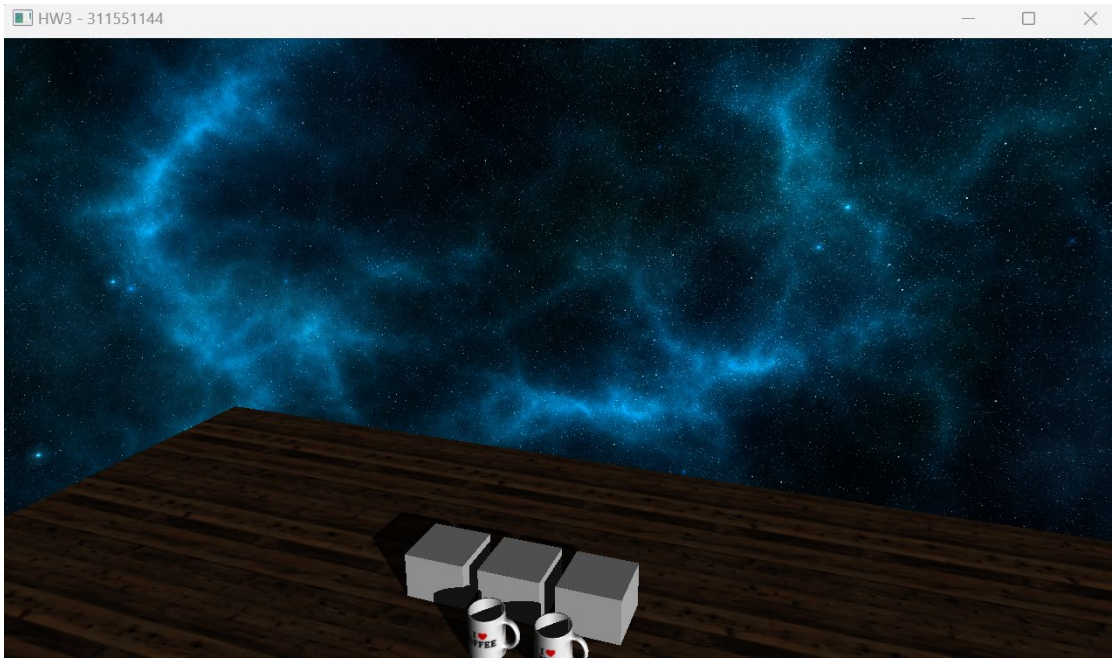
使用按鍵 H&J 調整 Gamma 做 Gamma Correction

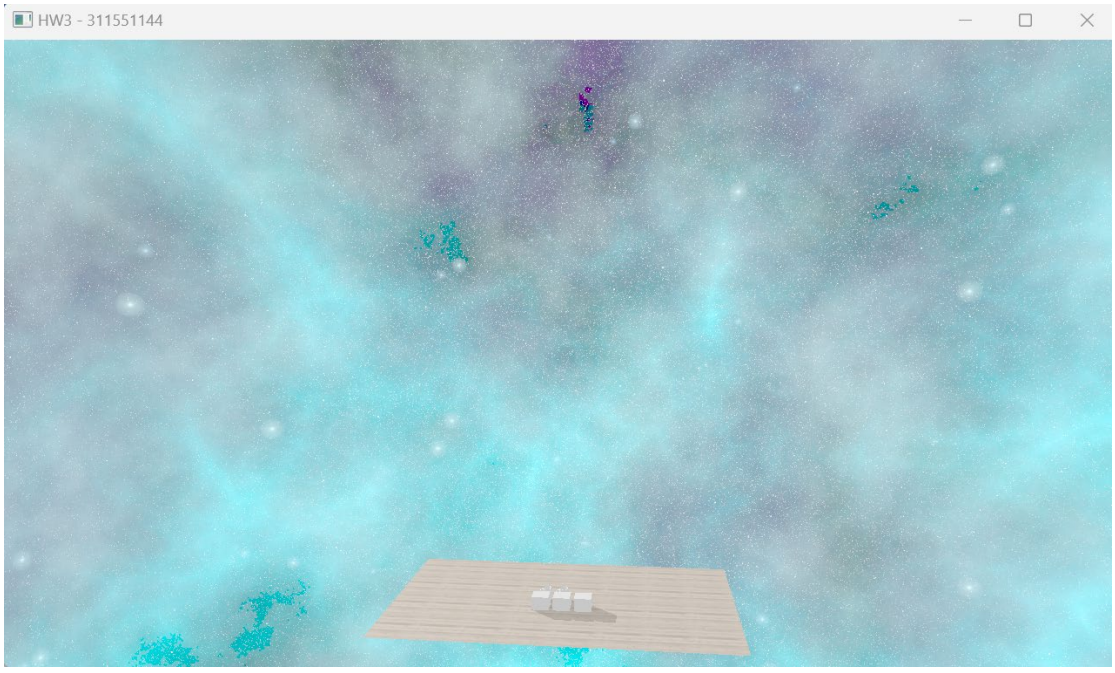
P.S 調到過曝可以找到星星(?)

```
color = texture(colorBuffer, TexCoord);
float R = color.r, G = color.g, B = color.b;
color = vec4(pow(R, gamma), pow(G, gamma), pow(B, gamma), 1.0);
```

```
GLint gammaLoc = glGetUniformLocation(programId, "gamma");
glUniform1f(gammaLoc, ctx->gamma);
```

```
if (action == GLFW_REPEAT) {
  switch (key) {
    case GLFW_KEY_H:
      ctx.gamma += 0.05f;
      break;
    case GLFW_KEY_J:
      ctx.gamma -= 0.05f;
      if (ctx.gamma < 0.0f) {
        ctx.gamma = 0.0f;
      }
      break;
```


HW3 - 311551144


HW3 - 311551144