計算機圖學 HW3 Report

0. Task#0 更改視窗標題

把視窗標題改成"HW3 - 311552013"。

```
glfwSetWindowTitle(window, "HW3 - 311552013");
```

1. Task#1 天空盒Skybox

首先建立VAO和VBO，並綁定座標位置。

```
// create VAO
GLuint* VAO = new GLuint[1];
glGenVertexArrays(1, VAO);
glBindVertexArray(VAO[0]);
model->vao = VAO[0];

// create VBO
GLuint VBO[1];
glGenBuffers(1, VBO);

// bind buffer
glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(),
model->positions.data(), GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

接著載入skybox圖片，綁定目標由cube_map_target[6]所決定的，分別對應skybox的六個面。另外，MIN/MAG filter要設為GL_LINEAR，而wrap要設為GL_CLAMP_TO_EDGE。

```
GLuint texture_id;
glGenTextures(1, &texture_id);
glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id);

int cube_map_target[6] = {GL_TEXTURE_CUBE_MAP_POSITIVE_X,
                          GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
                          GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
                          GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
                          GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
                          GL_TEXTURE_CUBE_MAP_NEGATIVE_Z};

for (int i = 0; i < 6; i++) {
  int w, h, nc;
  unsigned char* data = stbi_load(faces[i], &w, &h, &nc, 0);
  if (data) {
    glTexImage2D(cube_map_target[i], 0, GL_RGB, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

```
    }
    else {
      std::cout << "stbi_load(): fail to load image." << std::endl;
    }
    stbi_image_free(data);
  }

  glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_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);
  return texture_id;
```

接著把參數傳給shader並繪出skybox。要注意這邊要讓skybox第一個被繪出，並且關閉depth writing，使得skybox出現在所有其他物體後面。另外，透過把view matrix轉成3x3再轉回4x4的做法，消除了view matrix中位移部分，使玩家看起來會一直處於skybox的中心點。

```
  // close depth when drawing skybox
  glDepthMask(GL_FALSE);

  glBindVertexArray(model->vao);
  setMat4("Projection", ctx->camera->getProjectionMatrix());

  // remove the translation section of transformation matrices
  setMat4("ViewMatrix",
glm::value_ptr(glm::mat4(glm::mat3(ctx->camera->getViewMatrixGLM()))));

  glBindTexture(GL_TEXTURE_CUBE_MAP, model->textures[ctx->skybox->textureIndex]);
  glDrawArrays(model->drawMode, 0, model->numVertex);
  glDepthMask(GL_TRUE);
```

最後進到shader，在vertex shader設定好gl_Position座標位置，並把貼圖座標傳給fragment shader，然後在fragment shader中繪出貼圖即完成。

```
    TexCoord = position;
    gl_Position = Projection * ViewMatrix * vec4(position, 1.0);
-------------------------------------------------------------------------------------------
    color = texture(skybox, TexCoord);
```

2. Task#2 陰影映射Shadow Mapping

首先建立一個frame buffer以及一個frame buffer texture用作depth map。

```
  // frame buffer
  glGenFramebuffers(1, &depthMapFBO);

  // frame buffer texture
  glGenTextures(1, &ctx->shadowMapTexture);
  glBindTexture(GL_TEXTURE_2D, ctx->shadowMapTexture);
```

```
  glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_MAP_SIZE, SHADOW_MAP_SIZE, 0,
GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
  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_CLAMP_TO_BORDER);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);

  // set border color
  GLfloat border_color[] = {1.0, 1.0, 1.0, 1.0};
  glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);

  // bind frame buffer
  glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);

  // store frame buffer texture
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
ctx->shadowMapTexture, 0);

  // disable color buffer read and write
  glDrawBuffer(GL_NONE);
  glReadBuffer(GL_NONE);

  // bind back to default frame buffer
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

因為視窗長寬和shadow map不一樣，因此計算時要先用glViewport()轉換，並綁定frame buffer。接著計算光源空間矩陣，傳參數給shader並繪出至depth map。最後記得要把view port和frame buffer調回default。

```
  // change view port to shadow map
  glViewport(0, 0, SHADOW_MAP_SIZE, SHADOW_MAP_SIZE);

  // bind frame buffer
  glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
  glClear(GL_DEPTH_BUFFER_BIT);

  // create light space matrix
  float near_plane = 1.0f;
  float far_plane = 7.5f;
  float ortho_size = 10.0f;
  glm::vec3 light_pos = ctx->lightDirection * (-10.0f);
  glm::mat4 lightProjection = glm::ortho(-ortho_size, ortho_size, -ortho_size, ortho_size,
near_plane, far_plane);
  glm::mat4 lightView = glm::lookAt(light_pos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
  glm::mat4 lightViewMatrix = lightProjection * lightView;

  // render all objects 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];
    glBindVertexArray(model->vao);
```

```
    setMat4("LightViewMatrix", glm::value_ptr(lightViewMatrix));
    setMat4("ModelMatrix", glm::value_ptr(ctx->objects[i]->transformMatrix *
model->modelMatrix));
    glDrawArrays(model->drawMode, 0, model->numVertex);
  }

  // change view port back
  glViewport(0, 0, OpenGLContext::getWidth(), OpenGLContext::getHeight());

  // bind back to default buffer
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

有了剛才的depth map，就可以開始繪製場景了。ShadowLightShader這邊幾本上跟LightShader一樣，差在多傳了陰影相關參數進去。

```
    // shadow light shader
    float near_plane = 1.0f;
    float far_plane = 7.5f;
    float ortho_size = 10.0f;
    glm::vec3 light_pos = ctx->lightDirection * (-10.0f);
    glm::mat4 lightProjection = glm::ortho(-ortho_size, ortho_size, -ortho_size,
ortho_size, near_plane, far_plane);
    glm::mat4 lightView = glm::lookAt(light_pos, glm::vec3(0.0f), glm::vec3(0.0, 1.0,
0.0));
    glm::mat4 lightViewMatrix = lightProjection * lightView;
    setMat4("LightViewMatrix", glm::value_ptr(lightViewMatrix));
    setVec3("fakeLightPos", glm::value_ptr(ctx->lightDirection * -10.0f));
    setInt("shadowMap", 1);
    setInt("enableShadow", ctx->enableShadow);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, ctx->shadowMapTexture);
```

在vertex shader中，除了計算基本的頂點、貼圖座標等之外，還要多計算一個在光源中的座標LightFragPost，使用世界頂點座標呈乘上剛剛計算好的光源空間矩陣即可得到。

最後在fragment shader中計算陰影。//TODO前兩行把座標範圍鎖定在0~1之間，讓該座標可以對應到depth map。接著取得最近點深度以及當前點的深度。最後，若該點位於指定範圍內且當前深度>最近點深度的話，就位於陰影之中，回傳1.0f，反之，位於陰影外，回傳0.0f。

另外，也使用了bias來解決shadow acne的問題。

```
  gl_Position = Projection * ViewMatrix * ModelMatrix * vec4(position, 1.0);
  TexCoord = texCoord;
  FragPos = vec3(ModelMatrix * vec4(position, 1.0));
  Normal = mat3(TIModelMatrix) * normal;
  LightFragPost = LightViewMatrix * vec4(FragPos, 1.0);
-------------------------------------------------------------------------------------
float ShadowCalculation()
{
    float bias = 0.002;
```

```
    // TODO
    vec3 temp = LightFragPost.xyz / LightFragPost.w;
    temp = temp * 0.5 + 0.5;
    float closest = texture(shadowMap, temp.xy).r;
    float current = temp.z;
    if(current>1.0)  // out of range
        return 0.0;
    if(current-bias>closest) // in shadow
        return 1.0;
    return 0.0;
}
```

3. Task#3 濾鏡Filter

首先一樣建立frame buffer、VAO、VBO並綁定。要注意的點是quad_pos是依照NDC的範圍(-1~1, -1~1, 0)來決定，而quad_tex則是一般貼圖的(0~1, 0~1)。

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

  // VAO
  glGenVertexArrays(1, &quadVAO);
  glBindVertexArray(quadVAO);

  // VBO
  // set position and texture coordinates
  std::vector quad_pos = {
      -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
  };
  std::vector quad_tex = {
      0.0f, 1.0f,
      0.0f, 0.0f,
      1.0f, 0.0f,
      0.0f, 1.0f,
      1.0f, 0.0f,
      1.0f, 1.0f
  };

  // generate VBO
  glGenBuffers(2, quadVBO);

  // position
  glBindBuffer(GL_ARRAY_BUFFER, quadVBO[0]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * quad_pos.size(), quad_pos.data(),
GL_STATIC_DRAW);
  glEnableVertexAttribArray(0);
```

```
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void *)0);

  // texture
  glBindBuffer(GL_ARRAY_BUFFER, quadVBO[1]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * quad_tex.size(), quad_tex.data(),
GL_STATIC_DRAW);
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void *)0);
```

接著建立與視窗同大小的frame buffer texture和render buffer。

```
  // frame buffer texture
  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);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, colorBuffer,
0);

  // render buffer
  GLuint rboDepth;
  glGenRenderbuffers(1, &rboDepth);
  glBindRenderbuffer(GL_RENDERBUFFER, rboDepth);
  glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, SCR_WIDTH, SCR_HEIGHT);
  glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER,
rboDepth);

  // check if the frame buffer is complete
  if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    std::cout << "The frame buffer is not complete!" << std::endl;
```

接著把參數傳給shader並繪出。

```
  // pass data to shader
  glBindFramebuffer(GL_FRAMEBUFFER, filterFBO);
  setInt("colorBuffer", 0);
  setInt("enableEdgeDetection", ctx->enableEdgeDetection);
  setInt("eanbleGrayscale", ctx->eanbleGrayscale);
  glBindFramebuffer(GL_FRAMEBUFFER, 0);

  // bind VAO
  glBindVertexArray(quadVAO);

  // bind texture and draw
  glBindTexture(GL_TEXTURE_2D, colorBuffer);
  glDrawArrays(GL_TRIANGLES, 0, 6);
```

最後在shader中，載入貼圖，並套用灰階和邊緣檢測的公式即可完成。

```
// get color from color buffer
color = texture(colorBuffer, TexCoord);

// edge detection
if(enableEdgeDetection==1){
    color = vec4(0.0f, 0.0f, 0.0f, 1.0f);
    vec3 kernel_color[9];
    for(int i=0;i<9;i++){
        vec3 sample_color = vec3(texture(colorBuffer, TexCoord+offsets[i]));
        color += vec4(sample_color*kernel[i], 0.0);
    }
}

// gray scale
if(eanbleGrayscale==1){
    float gray = color.r * 0.2126 + color.g * 0.7152 + color.b * 0.0722;
    color.r = gray;
    color.g = gray;
    color.b = gray;
}
```