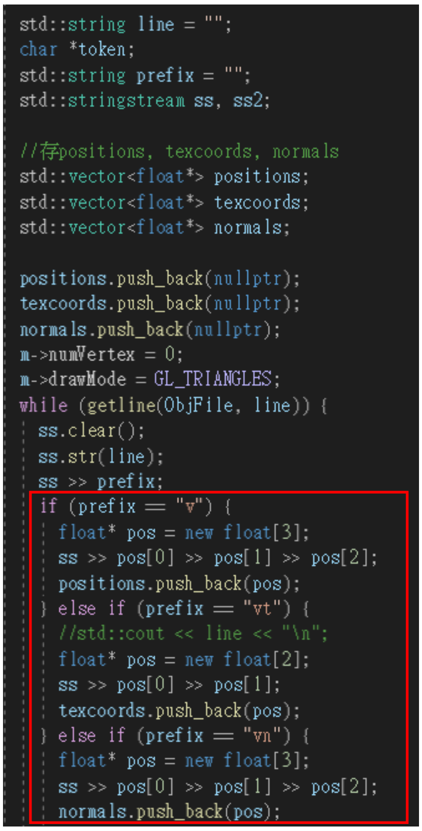
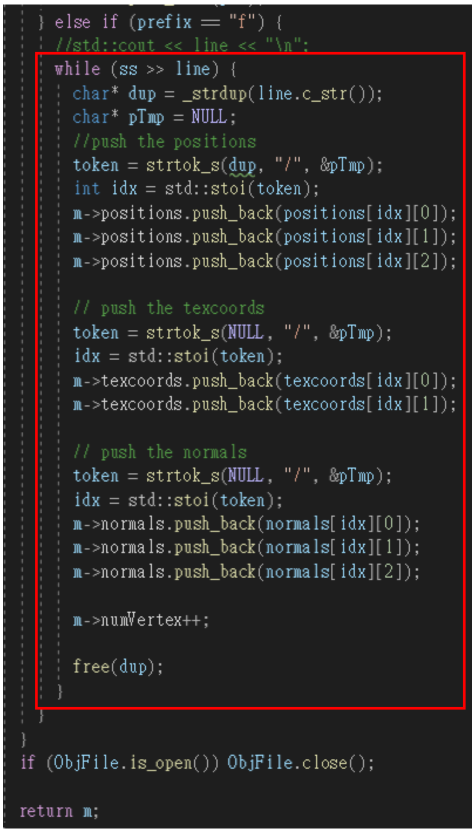
HW2

TASK 1:



```
std::string line = "";
char *token;
std::string prefix = "";
std::stringstream ss, ss2;

//存positions, texcoords, normals
std::vector<float*> positions;
std::vector<float*> texcoords;
std::vector<float*> normals;

positions.push_back(nullptr);
texcoords.push_back(nullptr);
normals.push_back(nullptr);
m->numVertex = 0;
m->drawMode = GL_TRIANGLES;
while (getline(ObjFile, line)) {
    ss.clear();
    ss.str(line);
    ss >> prefix;
    if (prefix == "v") {
        float* pos = new float[3];
        ss >> pos[0] >> pos[1] >> pos[2];
        positions.push_back(pos);
    } else if (prefix == "vt") {
        //std::cout << line << "\n";
        float* pos = new float[2];
        ss >> pos[0] >> pos[1];
        texcoords.push_back(pos);
    } else if (prefix == "vn") {
        float* pos = new float[3];
        ss >> pos[0] >> pos[1] >> pos[2];
        normals.push_back(pos);
```

← 分別把 positions, texcoords, normals 存進 vector 裡
一列為一個點的座標或方向

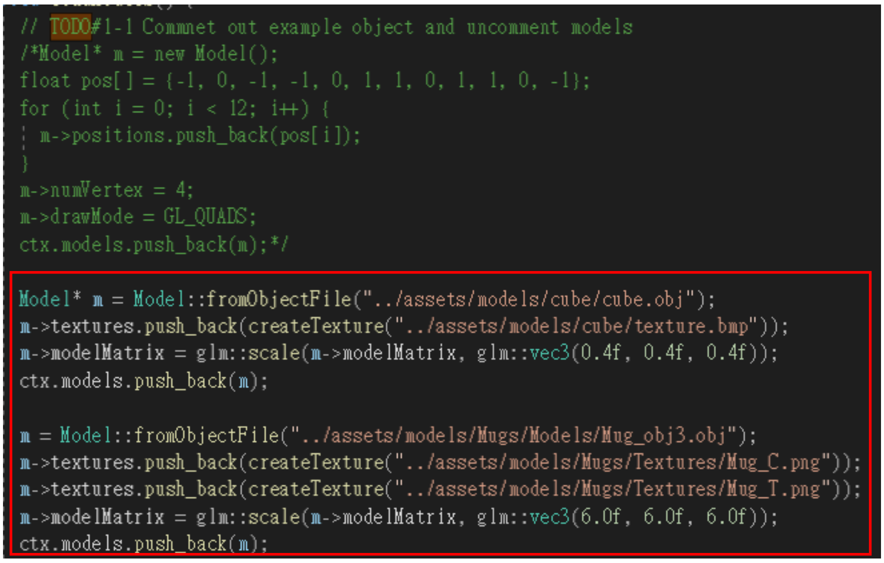


```
    } else if (prefix == "f") {
        //std::cout << line << "\n";
        while (ss >> line) {
            char* dup = _strdup(line.c_str());
            char* pTmp = NULL;
            //push the positions
            token = strtok_s(dup, "/", &pTmp);
            int idx = std::stoi(token);
            m->positions.push_back(positions[idx][0]);
            m->positions.push_back(positions[idx][1]);
            m->positions.push_back(positions[idx][2]);

            // push the texcoords
            token = strtok_s(NULL, "/", &pTmp);
            idx = std::stoi(token);
            m->texcoords.push_back(texcoords[idx][0]);
            m->texcoords.push_back(texcoords[idx][1]);

            // push the normals
            token = strtok_s(NULL, "/", &pTmp);
            idx = std::stoi(token);
            m->normals.push_back(normals[idx][0]);
            m->normals.push_back(normals[idx][1]);
            m->normals.push_back(normals[idx][2]);

            m->numVertex++;

            free(dup);
        }
    }
}
if (ObjFile.is_open()) ObjFile.close();

return m;
```

← 分別從 vector 取出 positions, texcoords, normals 並 push 到 model

格式為 index: (positions/ texcoords/ normals)

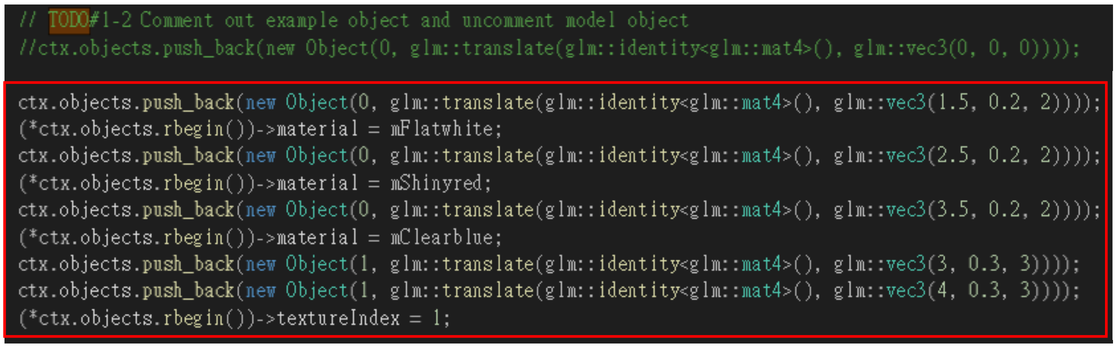
Task 1-1:



```
// TODO#1-1 Commnet out example object and uncomment models
/*Model* m = new Model();
float pos[] = {-1, 0, -1, -1, 0, 1, 1, 0, 1, 1, 0, -1};
for (int i = 0; i < 12; i++) {
  m->positions.push_back(pos[i]);
}
m->numVertex = 4;
m->drawMode = GL_QUADS;
ctx.models.push_back(m);*/

Model* m = Model::fromObjectFile("../assets/models/cube/cube.obj");
m->textures.push_back(createTexture("../assets/models/cube/texture.bmp"));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(0.4f, 0.4f, 0.4f));
ctx.models.push_back(m);

m = Model::fromObjectFile("../assets/models/Mugs/Models/Mug_obj3.obj");
m->textures.push_back(createTexture("../assets/models/Mugs/Textures/Mug_C.png"));
m->textures.push_back(createTexture("../assets/models/Mugs/Textures/Mug_T.png"));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(6.0f, 6.0f, 6.0f));
ctx.models.push_back(m);
```

← 創建立方體與杯子物件

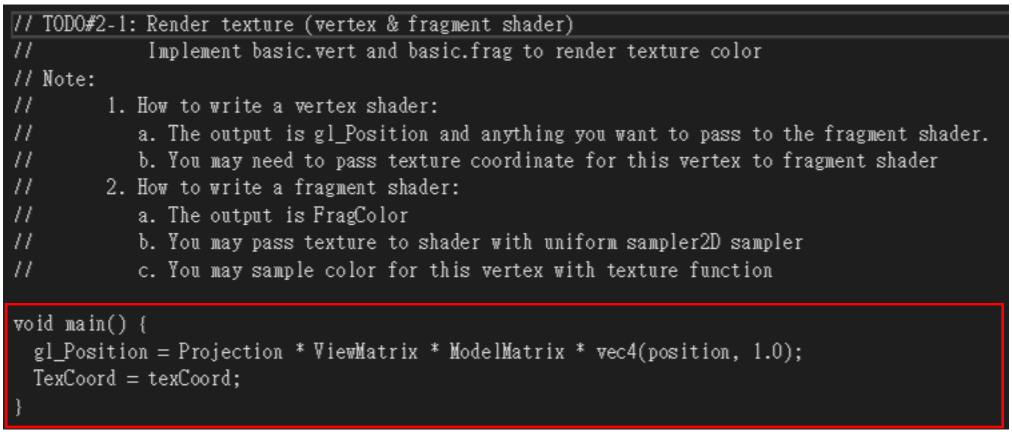
TASK 1-2:



```
// TODO#1-2 Comment out example object and uncomment model object
//ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(0, 0, 0))));

ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(1.5, 0.2, 2))));
(*ctx.objects.rbegin())->material = mFlatwhite;
ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(2.5, 0.2, 2))));
(*ctx.objects.rbegin())->material = mShinyred;
ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(), glm::vec3(3.5, 0.2, 2))));
(*ctx.objects.rbegin())->material = mClearblue;
ctx.objects.push_back(new Object(1, glm::translate(glm::identity<glm::mat4>(), glm::vec3(3, 0.3, 3))));
ctx.objects.push_back(new Object(1, glm::translate(glm::identity<glm::mat4>(), glm::vec3(4, 0.3, 3))));
(*ctx.objects.rbegin())->textureIndex = 1;
```

← 將物件放到場景中，並將第二個杯子的材質設定為Tea

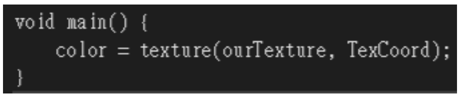
TASK 2-1(Vertex shader):



```
// TODO#2-1: Render texture (vertex & fragment shader)
//           Implement basic.vert and basic.frag to render texture color
// Note:
//       1. How to write a vertex shader:
//          a. The output is gl_Position and anything you want to pass to the fragment shader.
//          b. You may need to pass texture coordinate for this vertex to fragment shader
//       2. How to write a fragment shader:
//          a. The output is FragColor
//          b. You may pass texture to shader with uniform sampler2D sampler
//          c. You may sample color for this vertex with texture function

void main() {
  gl_Position = Projection * ViewMatrix * ModelMatrix * vec4(position, 1.0);
  TexCoord = texCoord;
}
```

← 計算座標並將材質座標傳到fragment shader

TASK 2-1(Fragment shader):



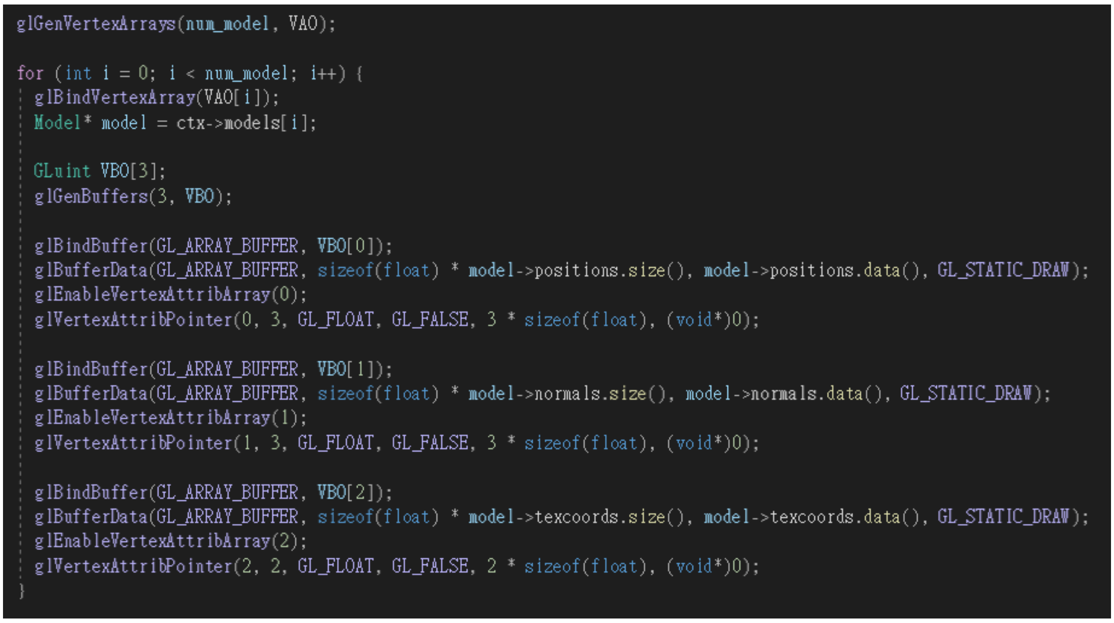
```
void main() {
    color = texture(ourTexture, TexCoord);
}
```

← 取得材質的顏色

TASK 2-2:

生成 VertexArray -> 對每個 model 生成 buffer -> 綁定 buffer -> 給 buffer 資料

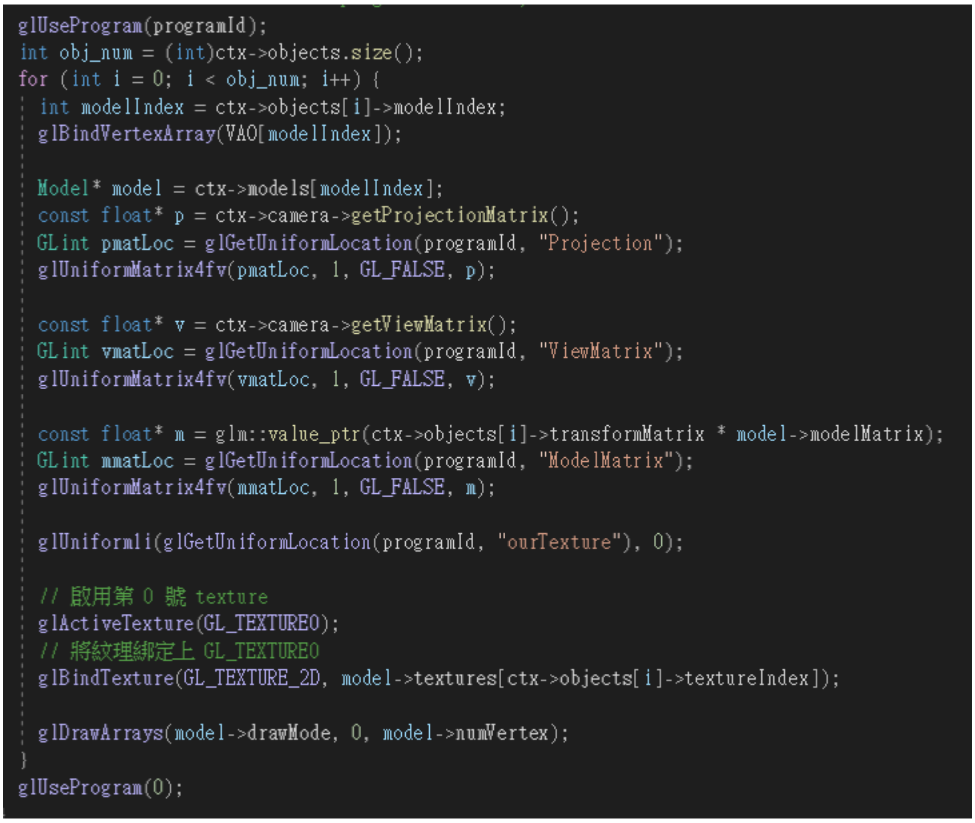
- VBO 分別對應到 position, normal, texcoord



```
glGenVertexArrays(num_model, VAO);

for (int i = 0; i < num_model; i++) {
  glBindVertexArray(VAO[i]);
  Model* model = ctx->models[i];

  GLuint VBO[3];
  glGenBuffers(3, VBO);

  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, VBO[1]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

  glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), model->texcoords.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(2);
  glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
}
```

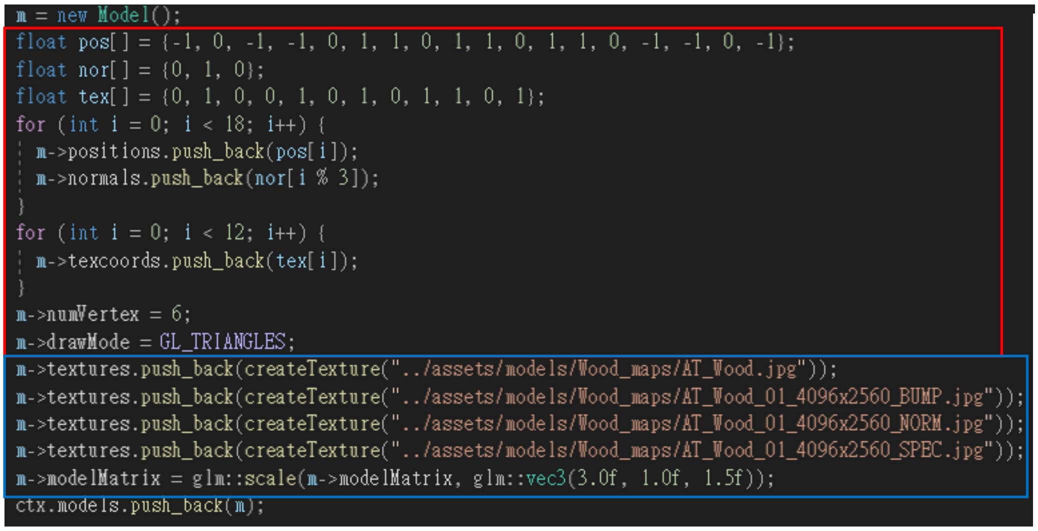
TASK 2-3:

對於每個物件，給定 shader 裡的 projection, view, model matrix 與 texture sampler，並綁定材質。



```
glUseProgram(programId);
int obj_num = (int)ctx->objects.size();
for (int i = 0; i < obj_num; i++) {
  int modelIndex = ctx->objects[i]->modelIndex;
  glBindVertexArray(VAO[modelIndex]);

  Model* model = ctx->models[modelIndex];
  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);

  glUniform1i(glGetUniformLocation(programId, "ourTexture"), 0);

  // 啟用第 0 號 texture
  glActiveTexture(GL_TEXTURE0);
  // 將紋理綁定上 GL_TEXTURE0
  glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);

  glDrawArrays(model->drawMode, 0, model->numVertex);
}
glUseProgram(0);
```

TASK 3-1:



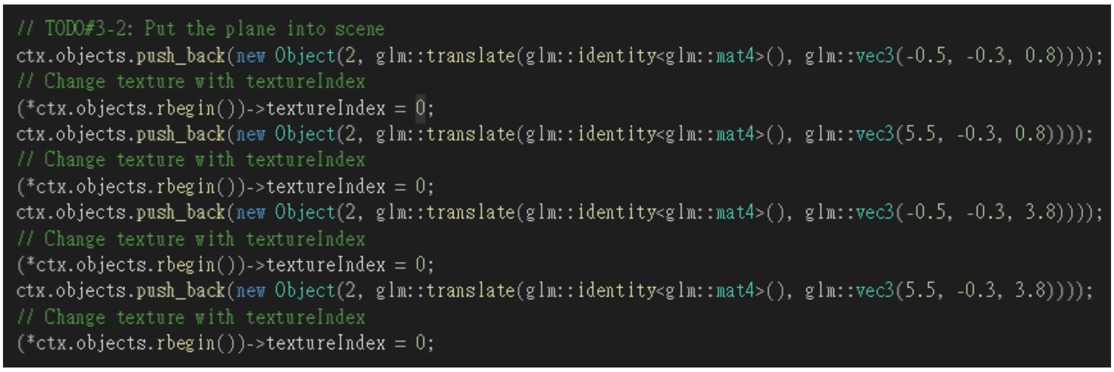
```
m = new Model();
float pos[] = {-1, 0, -1, -1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, -1, -1, 0, -1};
float nor[] = {0, 1, 0};
float tex[] = {0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1};
for (int i = 0; i < 18; i++) {
  m->positions.push_back(pos[i]);
  m->normals.push_back(nor[i % 3]);
}
for (int i = 0; i < 12; i++) {
  m->texcoords.push_back(tex[i]);
}
m->numVertex = 6;
m->drawMode = GL_TRIANGLES;
m->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood.jpg"));
m->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood_01_4096x2560_BUMP.jpg"));
m->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood_01_4096x2560_NORM.jpg"));
m->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood_01_4096x2560_SPEC.jpg"));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(3.0f, 1.0f, 1.5f));
ctx.models.push_back(m);
```

← 設定平面的 position, normal, texcoord 並 push 到 model

← 設定平面的材質選項並 push 到 model

TASK 3-2:

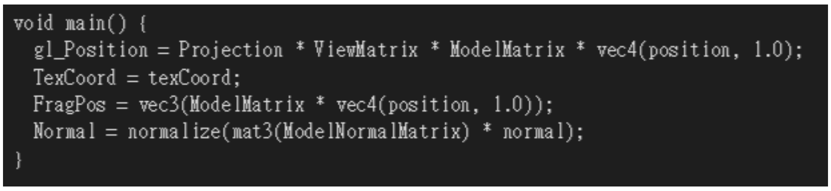
將平面放到場景中並設定材質



```
// TODO#3-2: Put the plane into scene
ctx.objects.push_back(new Object(2, glm::translate(glm::identity<glm::mat4>(), glm::vec3(-0.5, -0.3, 0.8))));
// Change texture with textureIndex
(*ctx.objects.rbegin())->textureIndex = 0;
ctx.objects.push_back(new Object(2, glm::translate(glm::identity<glm::mat4>(), glm::vec3(5.5, -0.3, 0.8))));
// Change texture with textureIndex
(*ctx.objects.rbegin())->textureIndex = 0;
ctx.objects.push_back(new Object(2, glm::translate(glm::identity<glm::mat4>(), glm::vec3(-0.5, -0.3, 3.8))));
// Change texture with textureIndex
(*ctx.objects.rbegin())->textureIndex = 0;
ctx.objects.push_back(new Object(2, glm::translate(glm::identity<glm::mat4>(), glm::vec3(5.5, -0.3, 3.8))));
// Change texture with textureIndex
(*ctx.objects.rbegin())->textureIndex = 0;
```

TASK 4-1(Vertex shader):

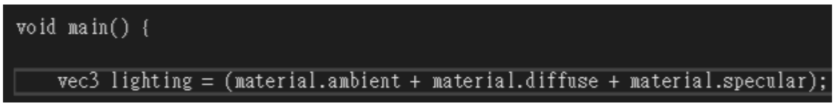
計算座標並將材質座標, FragPos, normal 傳至 Fragment shader



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

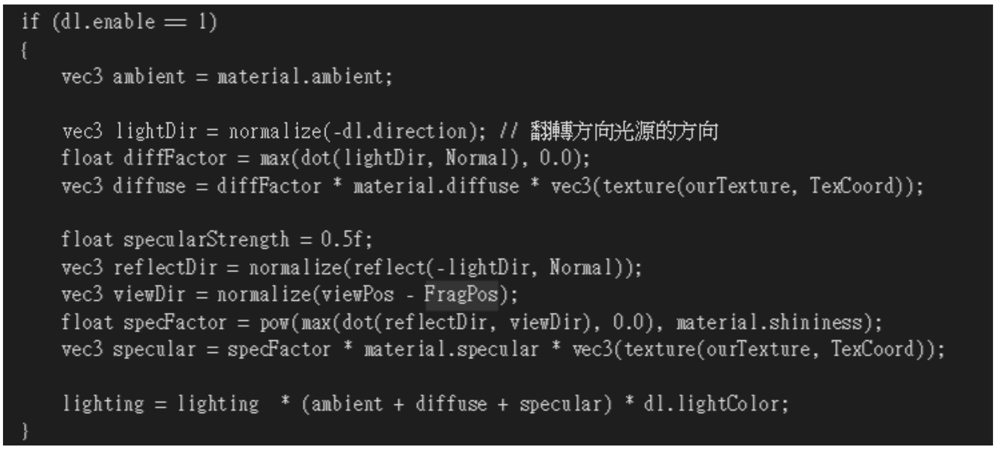
TASK 4-1(Fragment shader):

設定基本光



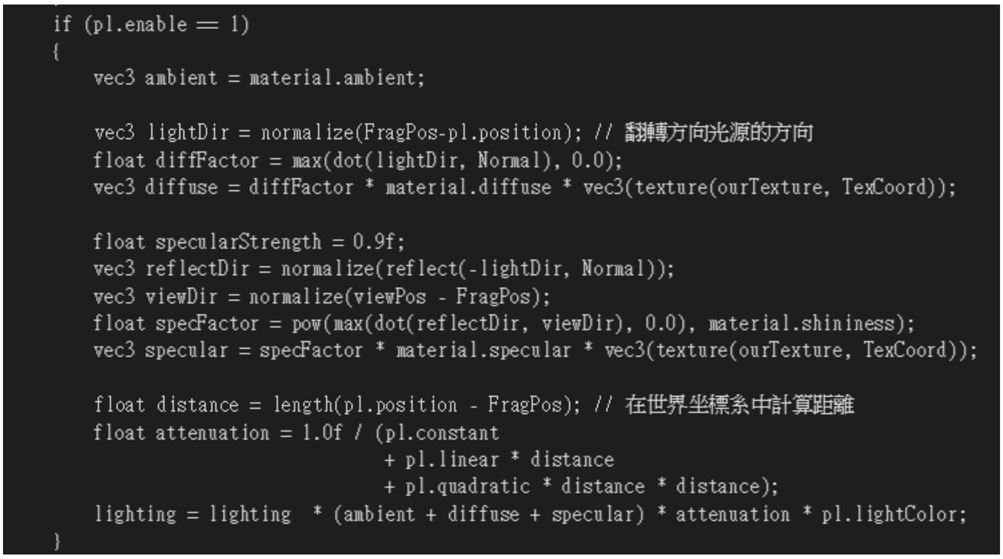
```
void main() {

    vec3 lighting = (material.ambient + material.diffuse + material.specular);
```

計算 Direction light 下的光



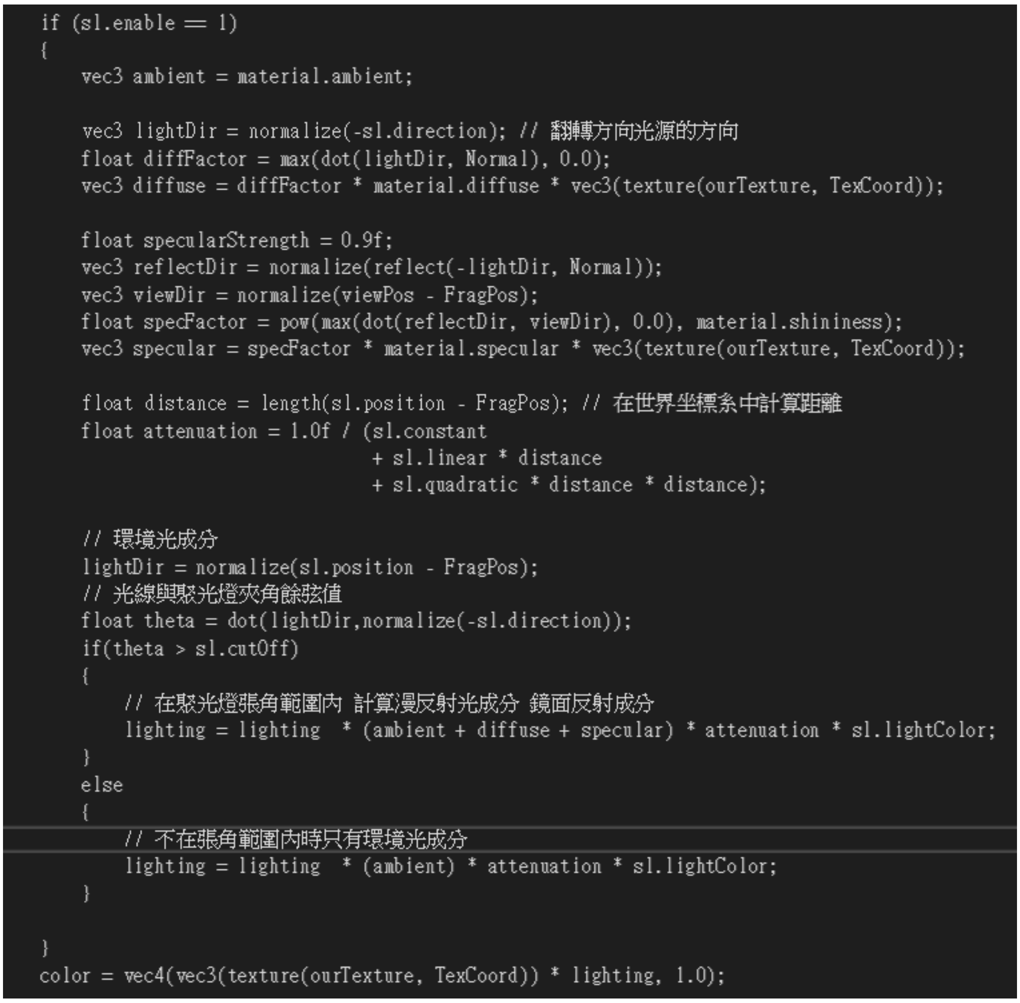
```
if (dl.enable == 1)
{
    vec3 ambient = material.ambient;

    vec3 lightDir = normalize(-dl.direction); // 翻轉方向光源的方向
    float diffFactor = max(dot(lightDir, Normal), 0.0);
    vec3 diffuse = diffFactor * material.diffuse * vec3(texture(ourTexture, TexCoord));

    float specularStrength = 0.5f;
    vec3 reflectDir = normalize(reflect(-lightDir, Normal));
    vec3 viewDir = normalize(viewPos - FragPos);
    float specFactor = pow(max(dot(reflectDir, viewDir), 0.0), material.shininess);
    vec3 specular = specFactor * material.specular * vec3(texture(ourTexture, TexCoord));

    lighting = lighting  * (ambient + diffuse + specular) * dl.lightColor;
}
```

計算 Point light 下的光



```
    if (pl.enable == 1)
    {
        vec3 ambient = material.ambient;

        vec3 lightDir = normalize(FragPos-pl.position); // 翻轉方向光源的方向
        float diffFactor = max(dot(lightDir, Normal), 0.0);
        vec3 diffuse = diffFactor * material.diffuse * vec3(texture(ourTexture, TexCoord));

        float specularStrength = 0.9f;
        vec3 reflectDir = normalize(reflect(-lightDir, Normal));
        vec3 viewDir = normalize(viewPos - FragPos);
        float specFactor = pow(max(dot(reflectDir, viewDir), 0.0), material.shininess);
        vec3 specular = specFactor * material.specular * vec3(texture(ourTexture, TexCoord));

        float distance = length(pl.position - FragPos); // 在世界坐標系中計算距離
        float attenuation = 1.0f / (pl.constant
                                   + pl.linear * distance
                                   + pl.quadratic * distance * distance);
        lighting = lighting  * (ambient + diffuse + specular) * attenuation * pl.lightColor;
    }
```

計算 Spot light 下的光

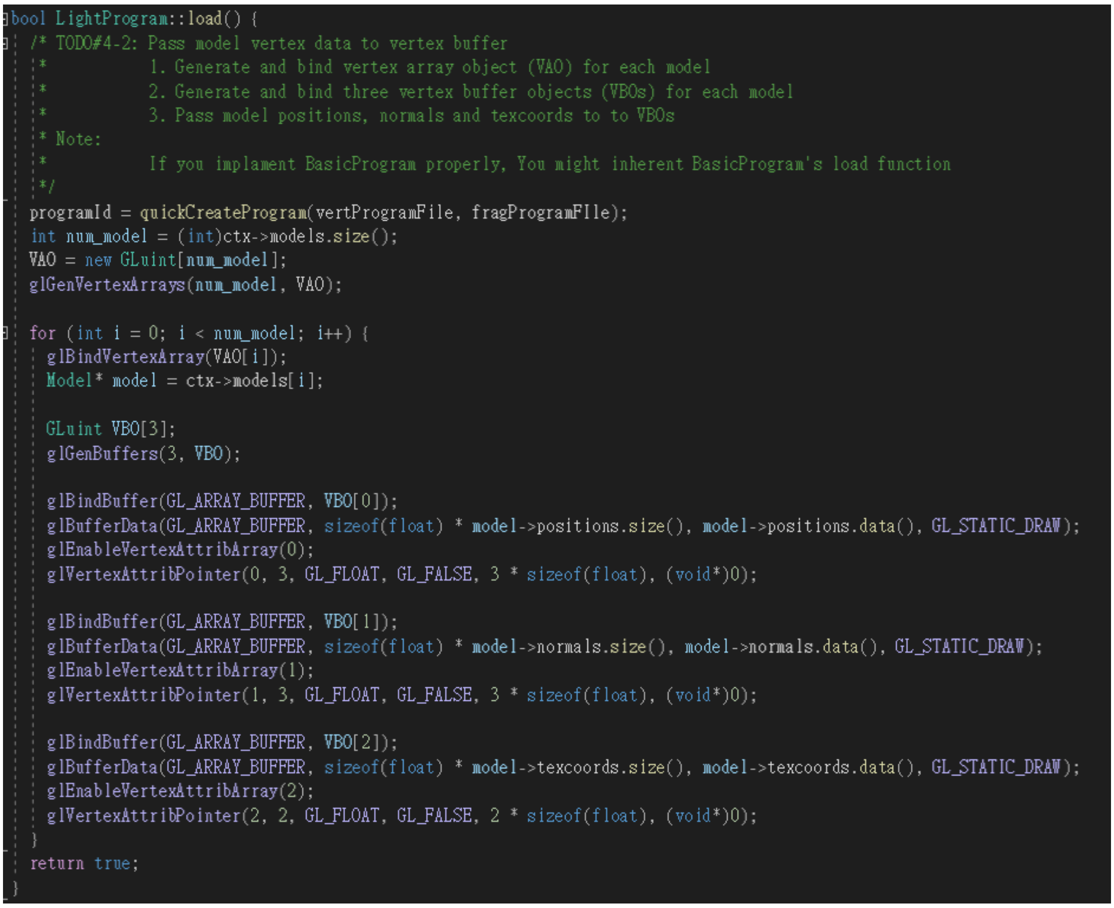


```
if (sl.enable == 1)
{
    vec3 ambient = material.ambient;

    vec3 lightDir = normalize(-sl.direction); // 翻轉方向光源的方向
    float diffFactor = max(dot(lightDir, Normal), 0.0);
    vec3 diffuse = diffFactor * material.diffuse * vec3(texture(ourTexture, TexCoord));

    float specularStrength = 0.9f;
    vec3 reflectDir = normalize(reflect(-lightDir, Normal));
    vec3 viewDir = normalize(viewPos - FragPos);
    float specFactor = pow(max(dot(reflectDir, viewDir), 0.0), material.shininess);
    vec3 specular = specFactor * material.specular * vec3(texture(ourTexture, TexCoord));

    float distance = length(sl.position - FragPos); // 在世界坐標系中計算距離
    float attenuation = 1.0f / (sl.constant
                              + sl.linear * distance
                              + sl.quadratic * distance * distance);

    // 環境光成分
    lightDir = normalize(sl.position - FragPos);
    // 光線與聚光燈夾角餘弦值
    float theta = dot(lightDir,normalize(-sl.direction));
    if(theta > sl.cutOff)
    {
        // 在聚光燈張角範圍內 計算漫反射光成分 鏡面反射成分
        lighting = lighting  * (ambient + diffuse + specular) * attenuation * sl.lightColor;
    }
    else
    {
        // 不在張角範圍內時只有環境光成分
        lighting = lighting  * (ambient) * attenuation * sl.lightColor;
    }

}
color = vec4(vec3(texture(ourTexture, TexCoord)) * lighting, 1.0);
```

TASK 4-2:

生成 VertexArray -> 對每個 model 生成 buffer -> 綁定 buffer -> 給 buffer 資料

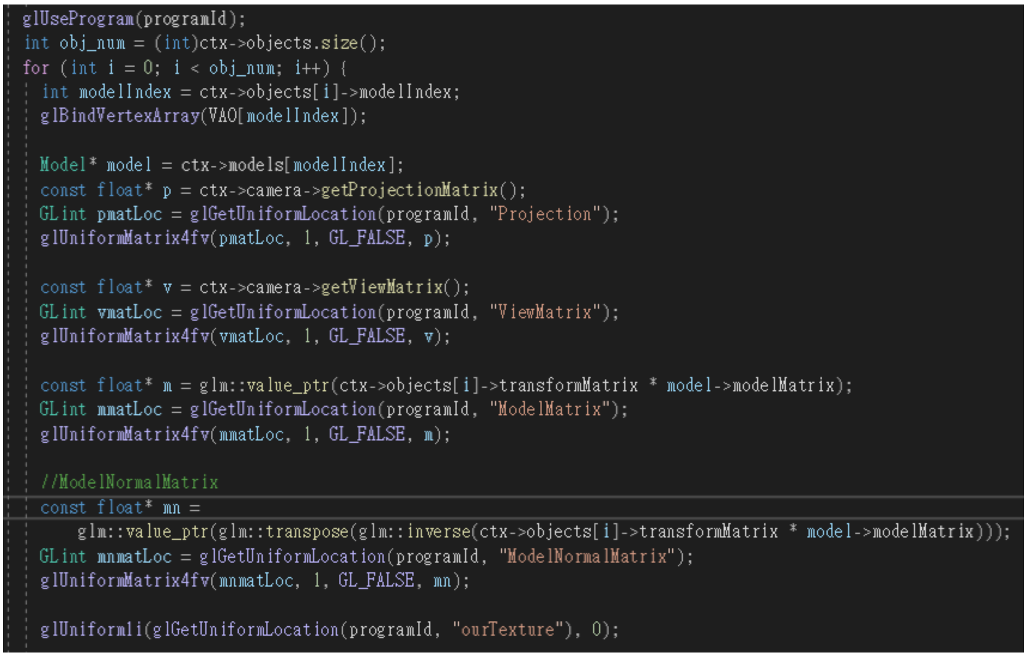
- VBO 分別對應到 position, normal, texcoord



```
bool LightProgram::load() {
  /* TODO#4-2: Pass model vertex data to vertex buffer
   *           1. Generate and bind vertex array object (VAO) for each model
   *           2. Generate and bind three vertex buffer objects (VBOs) for each model
   *           3. Pass model positions, normals and texcoords to to VBOs
   * Note:
   *           If you implament BasicProgram properly, You might inherent BasicProgram's load function
   */
  programId = quickCreateProgram(vertProgramFile, fragProgramFIle);
  int num_model = (int)ctx->models.size();
  VAO = new GLuint[num_model];
  glGenVertexArrays(num_model, VAO);

  for (int i = 0; i < num_model; i++) {
    glBindVertexArray(VAO[i]);
    Model* model = ctx->models[i];

    GLuint VBO[3];
    glGenBuffers(3, VBO);

    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, VBO[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), model->texcoords.data(), GL_STATIC_DRAW);
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
  }
  return true;
}
```

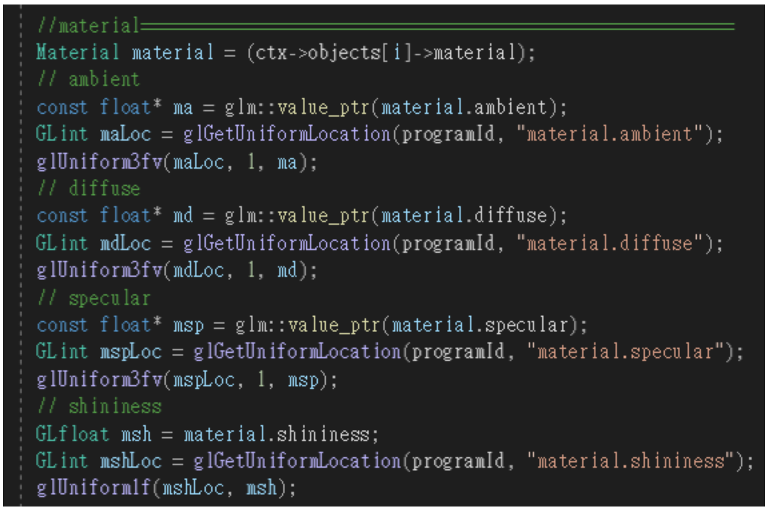
TASK 4-3:

對於每個物件，給定 shader 裡的 projection, view, model, modelNormal matrix 與 texture sampler，並綁定材質。



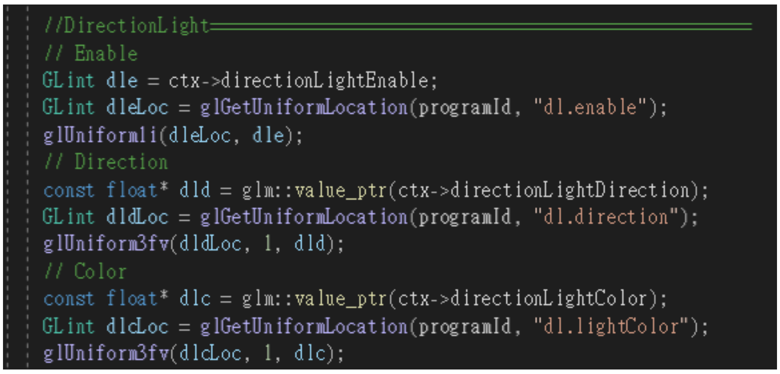
```
glUseProgram(programId);
int obj_num = (int)ctx->objects.size();
for (int i = 0; i < obj_num; i++) {
  int modelIndex = ctx->objects[i]->modelIndex;
  glBindVertexArray(VAO[modelIndex]);

  Model* model = ctx->models[modelIndex];
  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);

  //ModelNormalMatrix
  const float* mn =
      glm::value_ptr(glm::transpose(glm::inverse(ctx->objects[i]->transformMatrix * model->modelMatrix)));
  GLint mnmatLoc = glGetUniformLocation(programId, "ModelNormalMatrix");
  glUniformMatrix4fv(mnmatLoc, 1, GL_FALSE, mn);

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

給 shader 物體的材質特性



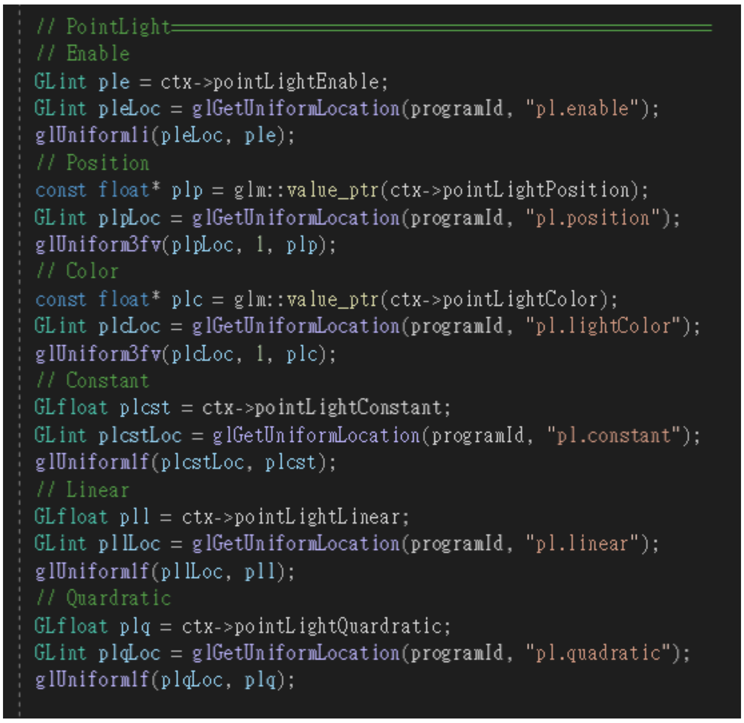
```
//material==================================================
Material material = (ctx->objects[i]->material);
// ambient
const float* ma = glm::value_ptr(material.ambient);
GLint maLoc = glGetUniformLocation(programId, "material.ambient");
glUniform3fv(maLoc, 1, ma);
// diffuse
const float* md = glm::value_ptr(material.diffuse);
GLint mdLoc = glGetUniformLocation(programId, "material.diffuse");
glUniform3fv(mdLoc, 1, md);
// specular
const float* msp = glm::value_ptr(material.specular);
GLint mspLoc = glGetUniformLocation(programId, "material.specular");
glUniform3fv(mspLoc, 1, msp);
// shininess
GLfloat msh = material.shininess;
GLint mshLoc = glGetUniformLocation(programId, "material.shininess");
glUniform1f(mshLoc, msh);
```

給 shader DirectionLight 的參數



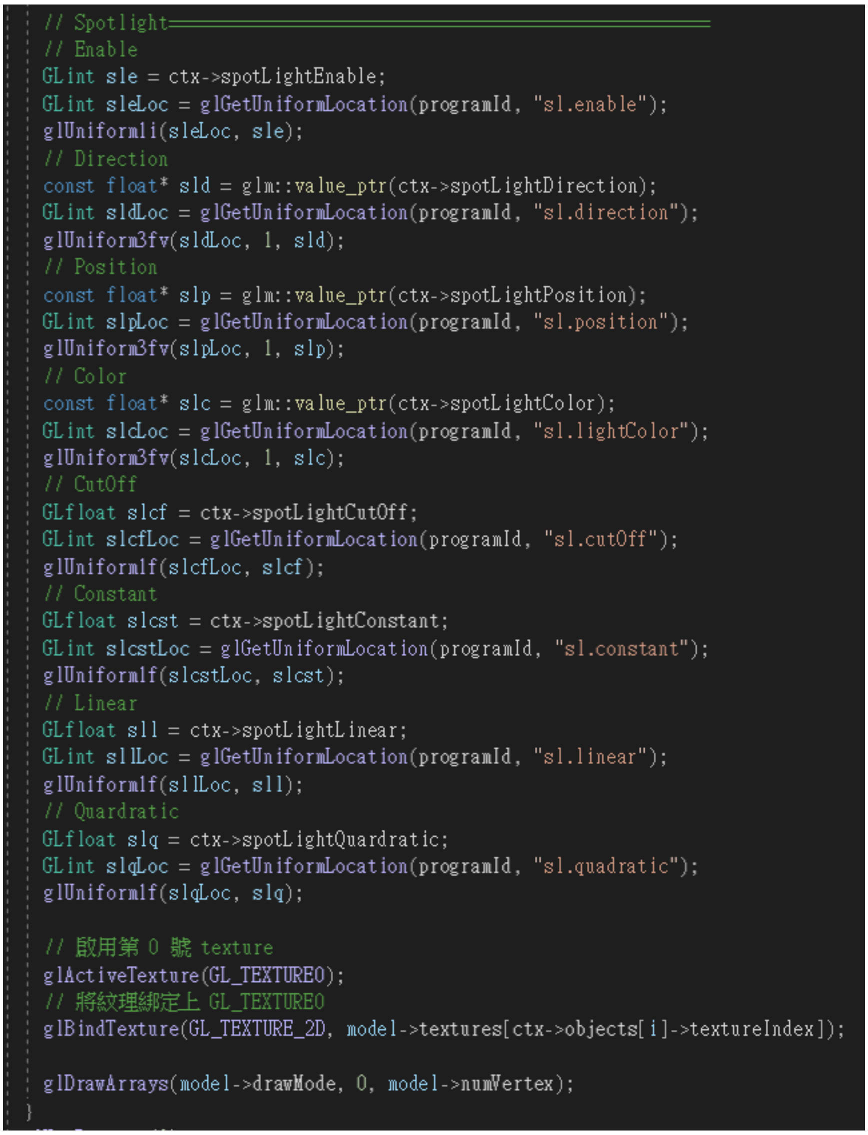
```
//DirectionLight==============================================
// Enable
GLint dle = ctx->directionLightEnable;
GLint dleLoc = glGetUniformLocation(programId, "dl.enable");
glUniform1i(dleLoc, dle);
// Direction
const float* dld = glm::value_ptr(ctx->directionLightDirection);
GLint dldLoc = glGetUniformLocation(programId, "dl.direction");
glUniform3fv(dldLoc, 1, dld);
// Color
const float* dlc = glm::value_ptr(ctx->directionLightColor);
GLint dlcLoc = glGetUniformLocation(programId, "dl.lightColor");
glUniform3fv(dlcLoc, 1, dlc);
```

給 shader PointLight 的參數



```
// PointLight==================================================
// Enable
GLint ple = ctx->pointLightEnable;
GLint pleLoc = glGetUniformLocation(programId, "pl.enable");
glUniform1i(pleLoc, ple);
// Position
const float* plp = glm::value_ptr(ctx->pointLightPosition);
GLint plpLoc = glGetUniformLocation(programId, "pl.position");
glUniform3fv(plpLoc, 1, plp);
// Color
const float* plc = glm::value_ptr(ctx->pointLightColor);
GLint plcLoc = glGetUniformLocation(programId, "pl.lightColor");
glUniform3fv(plcLoc, 1, plc);
// Constant
GLfloat plcst = ctx->pointLightConstant;
GLint plcstLoc = glGetUniformLocation(programId, "pl.constant");
glUniform1f(plcstLoc, plcst);
// Linear
GLfloat pll = ctx->pointLightLinear;
GLint pllLoc = glGetUniformLocation(programId, "pl.linear");
glUniform1f(pllLoc, pll);
// Quardratic
GLfloat plq = ctx->pointLightQuardratic;
GLint plqLoc = glGetUniformLocation(programId, "pl.quadratic");
glUniform1f(plqLoc, plq);
```

給 shader Spotlight 的參數



```
    // Spotlight==================================================
    // Enable
    GLint sle = ctx->spotLightEnable;
    GLint sleLoc = glGetUniformLocation(programId, "sl.enable");
    glUniform1i(sleLoc, sle);
    // Direction
    const float* sld = glm::value_ptr(ctx->spotLightDirection);
    GLint sldLoc = glGetUniformLocation(programId, "sl.direction");
    glUniform3fv(sldLoc, 1, sld);
    // Position
    const float* slp = glm::value_ptr(ctx->spotLightPosition);
    GLint slpLoc = glGetUniformLocation(programId, "sl.position");
    glUniform3fv(slpLoc, 1, slp);
    // Color
    const float* slc = glm::value_ptr(ctx->spotLightColor);
    GLint slcLoc = glGetUniformLocation(programId, "sl.lightColor");
    glUniform3fv(slcLoc, 1, slc);
    // CutOff
    GLfloat slcf = ctx->spotLightCutOff;
    GLint slcfLoc = glGetUniformLocation(programId, "sl.cutOff");
    glUniform1f(slcfLoc, slcf);
    // Constant
    GLfloat slcst = ctx->spotLightConstant;
    GLint slcstLoc = glGetUniformLocation(programId, "sl.constant");
    glUniform1f(slcstLoc, slcst);
    // Linear
    GLfloat sll = ctx->spotLightLinear;
    GLint sllLoc = glGetUniformLocation(programId, "sl.linear");
    glUniform1f(sllLoc, sll);
    // Quardratic
    GLfloat slq = ctx->spotLightQuardratic;
    GLint slqLoc = glGetUniformLocation(programId, "sl.quadratic");
    glUniform1f(slqLoc, slq);

    // 啟用第 0 號 texture
    glActiveTexture(GL_TEXTURE0);
    // 將紋理綁定上 GL_TEXTURE0
    glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);

    glDrawArrays(model->drawMode, 0, model->numVertex);
}
```

Problems you encountered:

- 少看到 model.cpp 也有 TODO 1 而覺得怎麼沒有顯示。
- 漏看到 keyCallback 的部分，所以不知道 basic 需要按 2 才有效果，以至於嘗試了數天才發現。