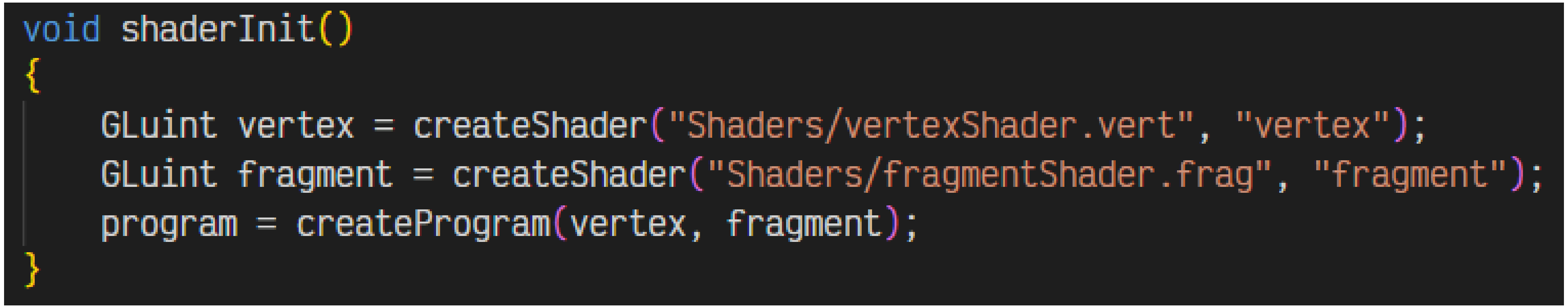
Introduction to Computer Graphics HW2:GLSL Explanation

0711239 李勝維

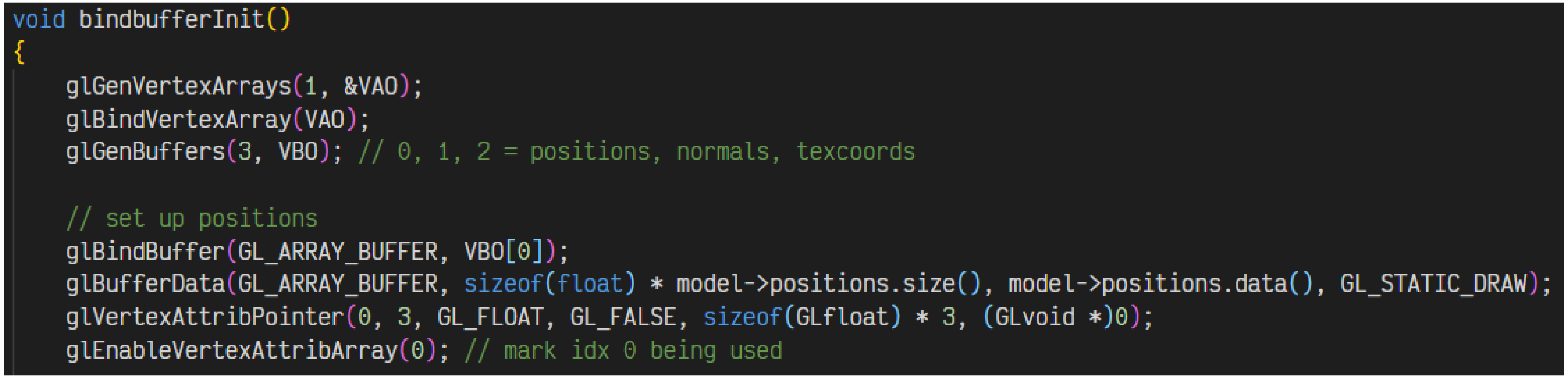
In main.cpp

Step 1 Create Program


```
void shaderInit()
{
    GLuint vertex = createShader("Shaders/vertexShader.vert", "vertex");
    GLuint fragment = createShader("Shaders/fragmentShader.frag", "fragment");
    program = createProgram(vertex, fragment);
}
```

- 使用助教包好的”createShader”來載入並compile vertex shader及fragment shader
- 使用助教包好的”createProgram”將兩個shader attach到program並link program

Step 2-1 Create VAO & setup VBO[0]

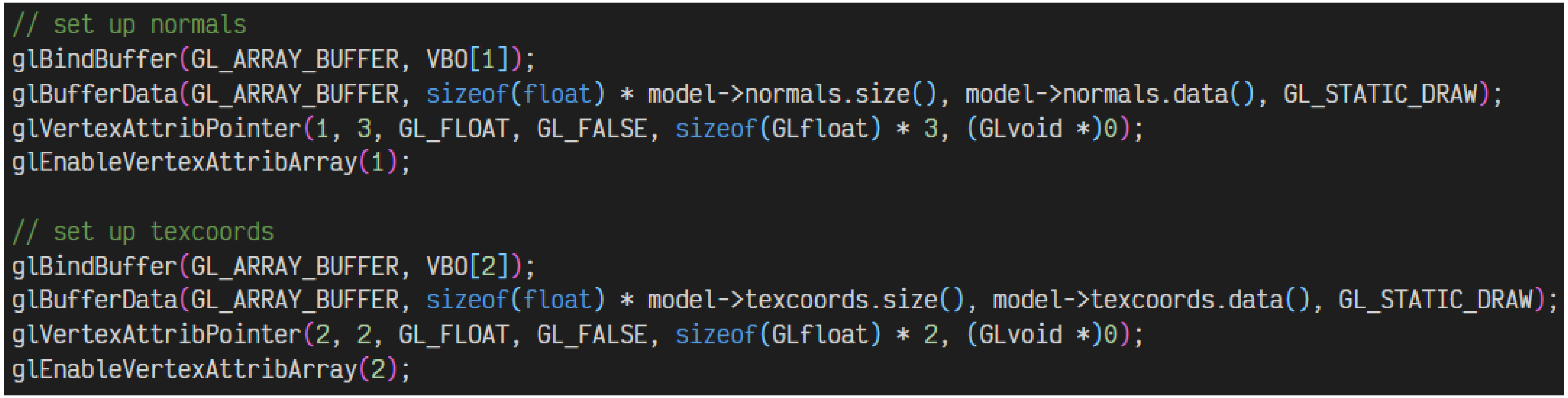


```
void bindbufferInit()
{
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(3, VBO); // 0, 1, 2 = positions, normals, texcoords

    // set up positions
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), model->positions.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GLfloat) * 3, (GLvoid *)0);
    glEnableVertexAttribArray(0); // mark idx 0 being used
```

- 使用“glGenVertexArrays()”和“glBindVertexArray()”來掛載VAO(之後bind的VBO都會在這個VAO裡面)，並用“glGenBuffers”生成3個等等要用的VBO
- VBO[0]代表的是positions，將資料由”glBufferData()”傳入並用”glVertexAttribPointer()”設定index和stride等參數
- 最後用”glEnableVertexAttribArray(0)”來讓GLSL知道location 0有東西

Step 2-2 Setup VBO[1] & VBO[2]

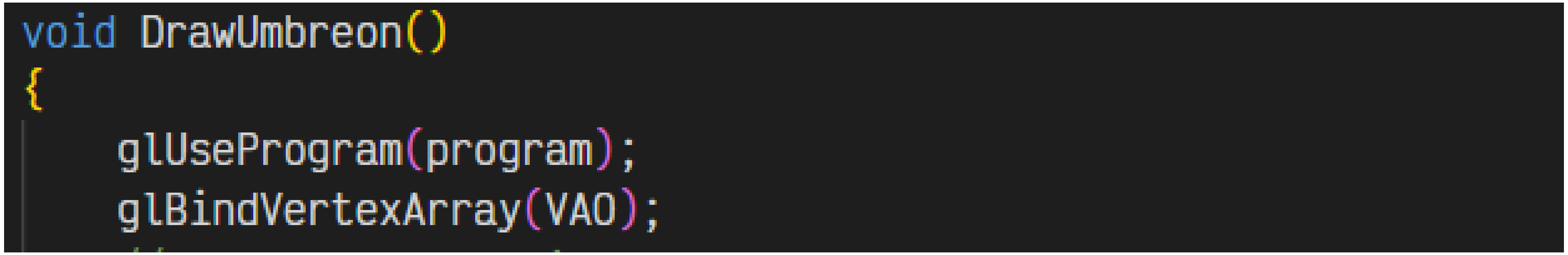


```
// set up normals
glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(GLfloat) * 3, (GLvoid *)0);
glEnableVertexAttribArray(1);

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

- VBO[1]代表了normals，VBO[2]代表了texture coordinates
- 不同的是VBO[2]每個vertex只有2個float(2D)而非3個，因此glVertexAttribPointer的參數也要有所不同
- 完成了VAO和VBO的設定，在最後將其unbind(截圖沒有，在最後一行下面)

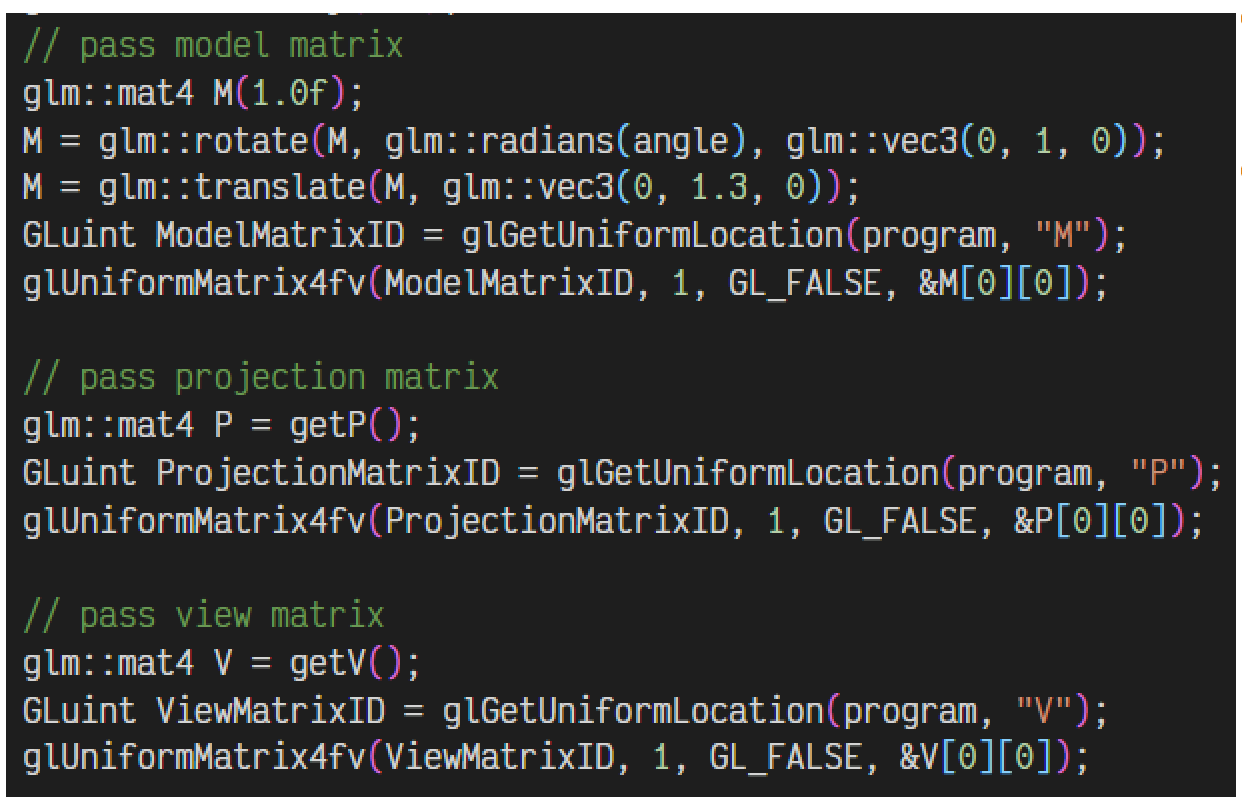
Step 3-1 Before drawing Umbreon



```
void DrawUmbreon()
{
    glUseProgram(program);
    glBindVertexArray(VAO);
```

- 呼叫” glUseProgram()” ，在此之後的任何繪製都會先去shader處理，可以上色、計算phong refelction model…
- 呼叫” glBindVertexArray()” ，這樣shader可以直接從VAO和它裡面的VBO抓vertex資料

Step 3-2 Pass matrixes by Uniform



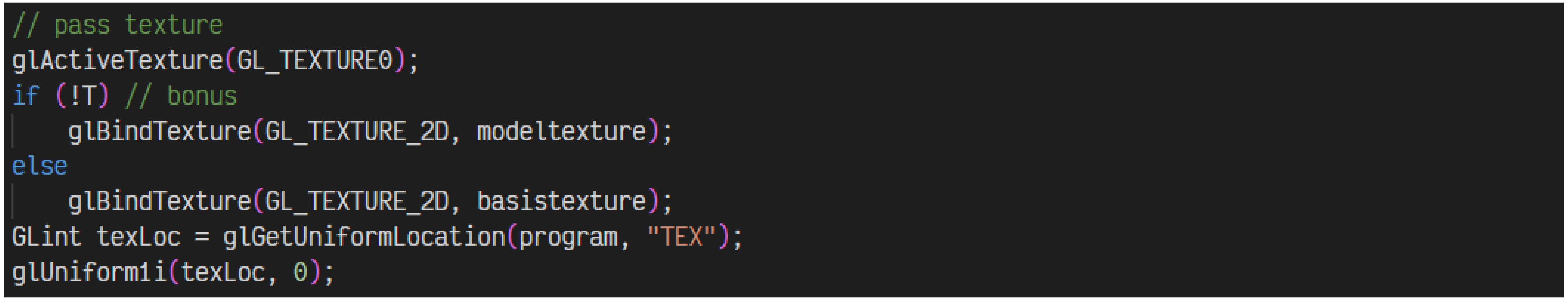
```
// pass model matrix
glm::mat4 M(1.0f);
M = glm::rotate(M, glm::radians(angle), glm::vec3(0, 1, 0));
M = glm::translate(M, glm::vec3(0, 1.3, 0));
GLuint ModelMatrixID = glGetUniformLocation(program, "M");
glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE, &M[0][0]);

// pass projection matrix
glm::mat4 P = getP();
GLuint ProjectionMatrixID = glGetUniformLocation(program, "P");
glUniformMatrix4fv(ProjectionMatrixID, 1, GL_FALSE, &P[0][0]);

// pass view matrix
glm::mat4 V = getV();
GLuint ViewMatrixID = glGetUniformLocation(program, "V");
glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE, &V[0][0]);
```

- 先由計算或getP/getV得到要傳入的matrix
- 使用”glGetUniformLocation()” 來從名字得到該Uniform的位址，並使用” glUniformMatrix4fv()” 將欲傳入資料的pointer傳入(此處為傳入4x4 matrix的pointer)

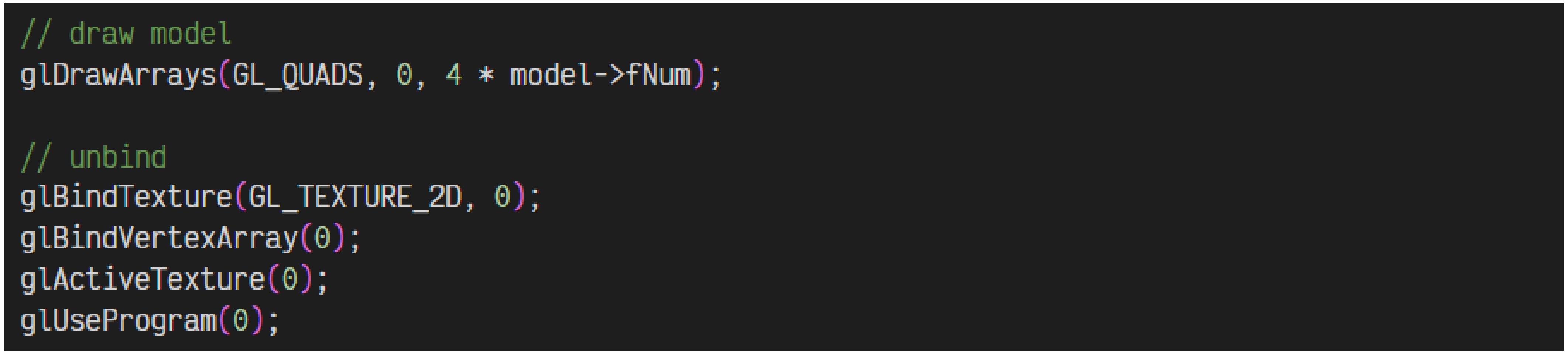
Step 3-3 Pass Texture(sampler2D)



```
// pass texture
glActiveTexture(GL_TEXTURE0);
if (!T) // bonus
    glBindTexture(GL_TEXTURE_2D, modeltexture);
else
    glBindTexture(GL_TEXTURE_2D, basistexture);
GLint texLoc = glGetUniformLocation(program, "TEX");
glUniform1i(texLoc, 0);
```

- T為一boolean值，根據T來選擇傳入的texture為modeltexture或basistexture（這邊是我的bonus，非本次作業必須，正常應傳入modelbasis）
- 同樣使用”glGetUniformLocation()”來得到“TEX”的位址
- 不同的是這次傳入的只是傳入texture id，也就是整數0，因此使用”glUniform1i()”

Step 4 Draw Umbreon



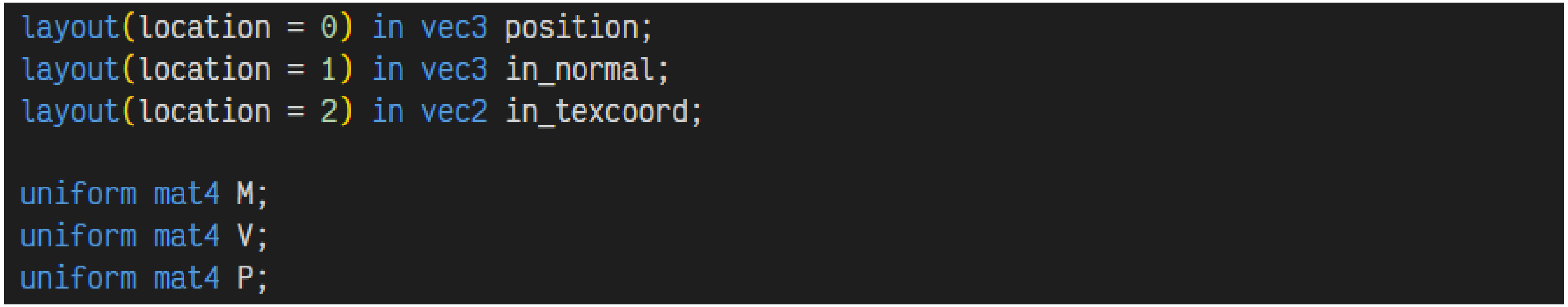
```
// draw model
glDrawArrays(GL_QUADS, 0, 4 * model->fNum);

// unbind
glBindTexture(GL_TEXTURE_2D, 0);
glBindVertexArray(0);
glActiveTexture(0);
glUseProgram(0);
```

- 使用” glDrawArrays()” 畫出每個primitive，繪製前會經過shader program的運算。而model->fNum會等同於vertex數/4，也就是model->positions.size()/3/4
- 畫完之後unbind乾淨

In vertexShader.vert

Part 1 Input Data

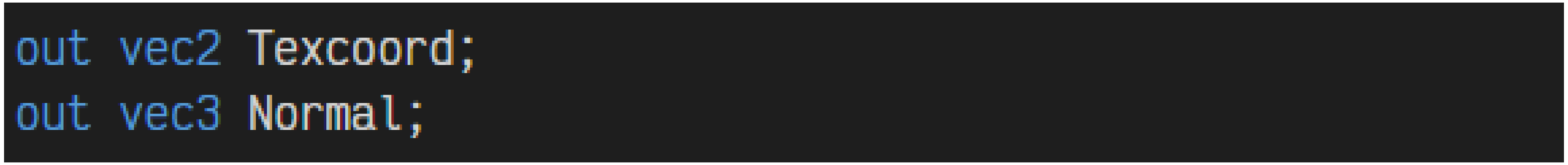


```
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 in_normal;
layout(location = 2) in vec2 in_texcoord;

uniform mat4 M;
uniform mat4 V;
uniform mat4 P;
```

- 由VBO讀入資料，在main.cpp裡面定義的VBO[0],VBO[1],VBO[2]分別為position、normal、texture coordinate，對應到三組vector
- “接”由main.cpp存入的Uniform，分別為model, viewport, projection matrix

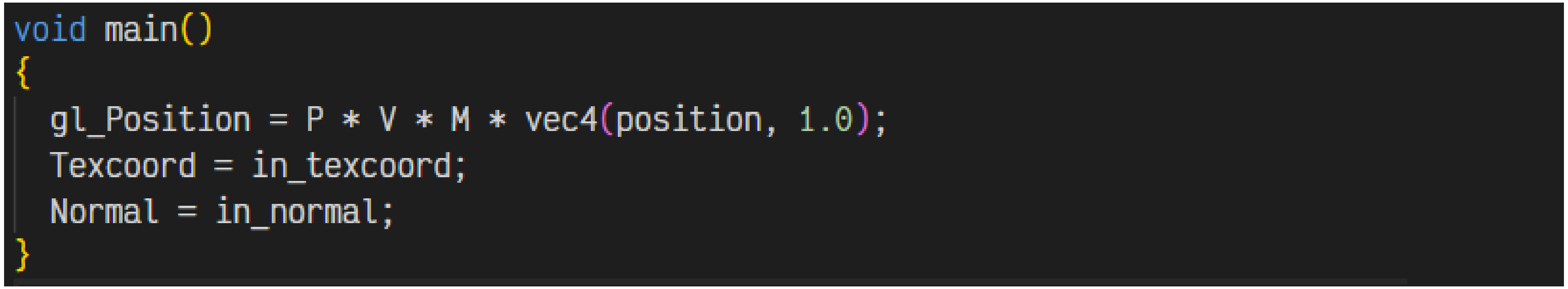
Part 2 Output data



```
out vec2 Texcoord;
out vec3 Normal;
```

- 這裡的out會直接傳給fragment shader，而要傳的有texture coordinate和normal兩組vector

Part 3 main function

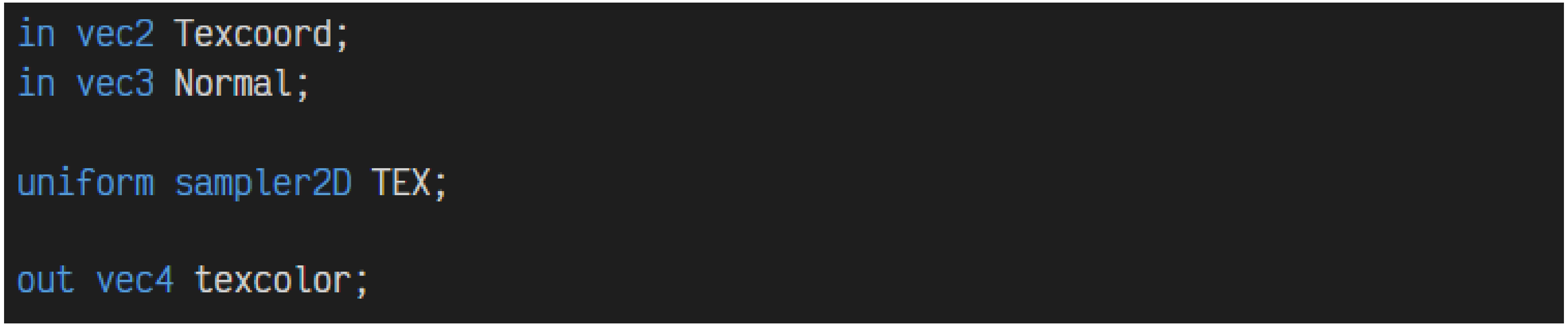


```
void main()
{
  gl_Position = P * V * M * vec4(position, 1.0);
  Texcoord = in_texcoord;
  Normal = in_normal;
}
```

- 根據上課所學，先將3維的座標轉換為4維齊次座標系並以正確順序乘上MVP三矩陣得到gl_Position，即為該vertex最終的座標
- 將texture coordinate和normal繼續傳給fragment shader

In fragmentShader.frag

Part 1 Input / Output data



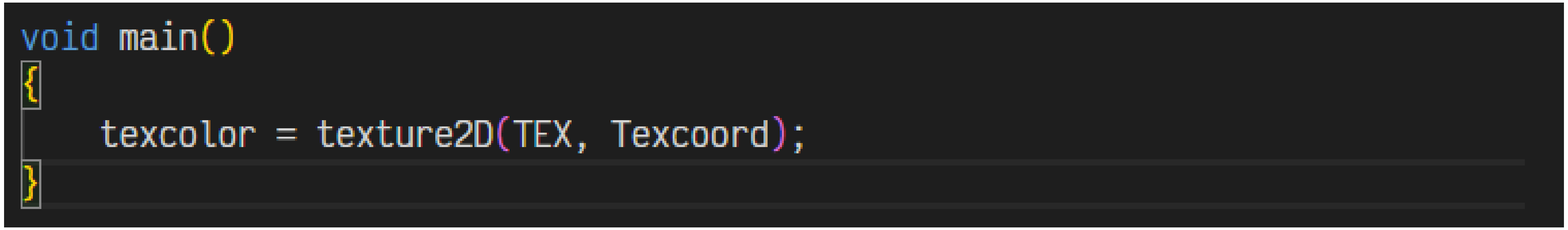
```
in vec2 Texcoord;
in vec3 Normal;

uniform sampler2D TEX;

out vec4 texcolor;
```

- Texcoord和Normal為vertex shader傳入的texture coordinate和normal
- “接”由main.cpp存入的Uniform，為texture id
- 輸出為該fragment(pixel)的顏色值，一個4維vector，代表了RGB和不透明度

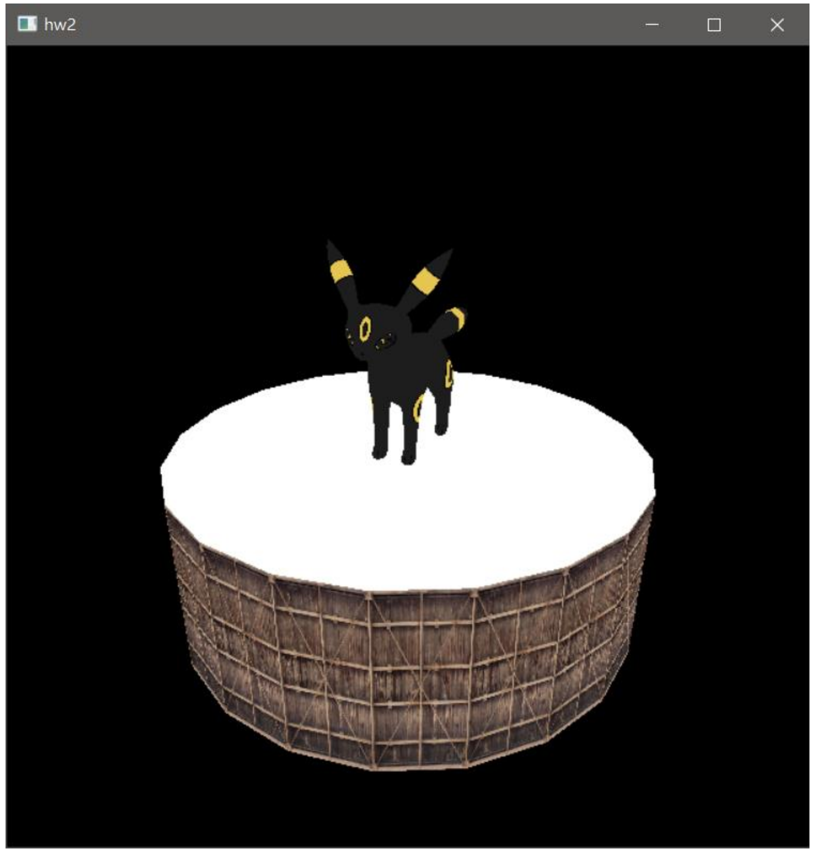
Part 2 main function



```
void main()
{
    texcolor = texture2D(TEX, Texcoord);
}
```

- 使用”texture2D()”來將材質資料(Umbreon.jpg)在1024*(Texcoord.x, Texcoord.y)的顏色提取出來並輸出

Done !



Problems I’ve met

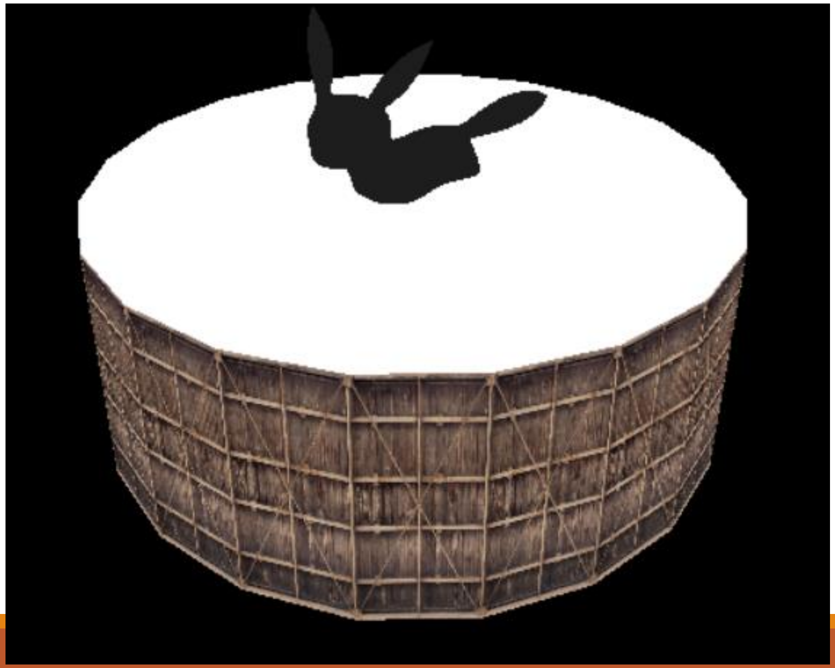
Texture Coordinate

- 我一直以為Texture Coordinate是從左上(0,0)到右下(1,1)，但其實是左下(0,0)到右上(1,1)
- 投影片裡面其實有寫，助教對不起我眼睛有問題

變數名稱

- 起初，在vertexShader中的out vec2 Texcoord的變數名稱為out vec2 out_texcoord
- 奇怪的是變數名稱前面加上out_好像會把事情搞砸，讓整個結果爛掉，而我debug了3個小時才發現到原來是變數名稱的問題，我寫code也才花了1小時，真的搞死人

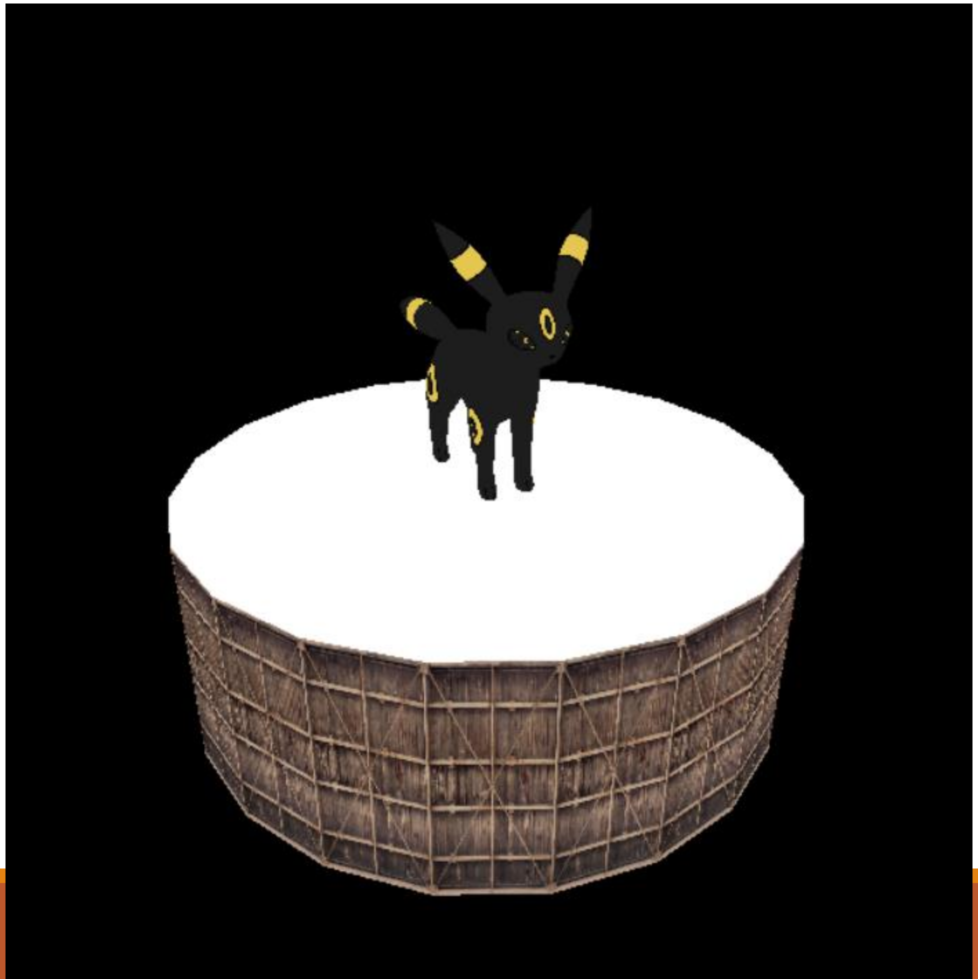
把變數名稱開頭加上out_，結果如下：（我還是不懂是什麼原理）



Bonus

可變換伊布的材質貼圖 (利用fragment shader實現)

原本：



按下space鍵：