Computer Graphics Assignment #2

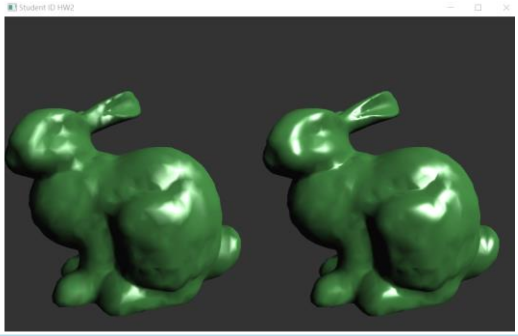
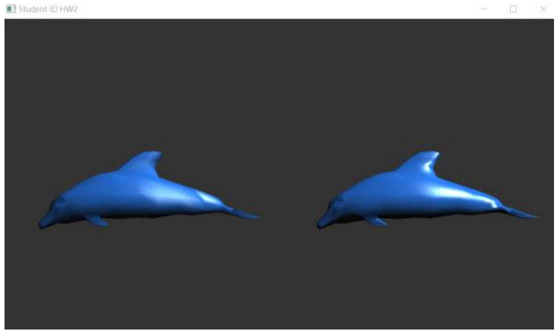
107062313 黃寶萱

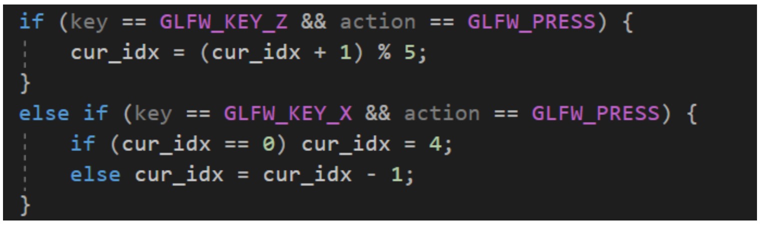
A. Grading Policy

Item	Done
Directional light	✓
Point light	✓
Spot light	✓
Per-pixel lighting / Per-vertex lighting	✓
Side-by-side viewport	✓
Switch lights & models	✓
Dynamic light position, cutoff, shininess	✓
Report	✓

B. Keyboard : Z/X

(與作業一做法相同)

cur_idx = 0	cur_idx = 4
	

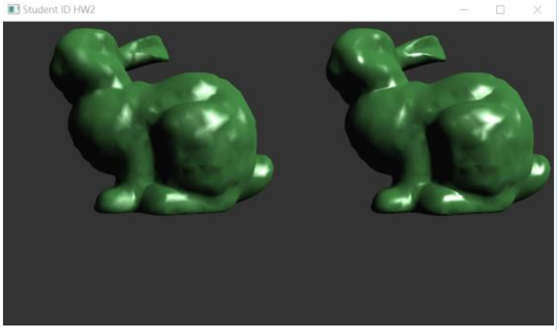


```
if (key == GLFW_KEY_Z && action == GLFW_PRESS) {
    cur_idx = (cur_idx + 1) % 5;
}
else if (key == GLFW_KEY_X && action == GLFW_PRESS) {
    if (cur_idx == 0) cur_idx = 4;
    else cur_idx = cur_idx - 1;
}
```

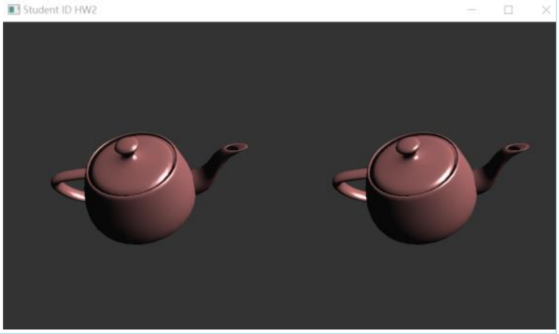
C. Keyboard : T / R / S

(與作業一做法相同)

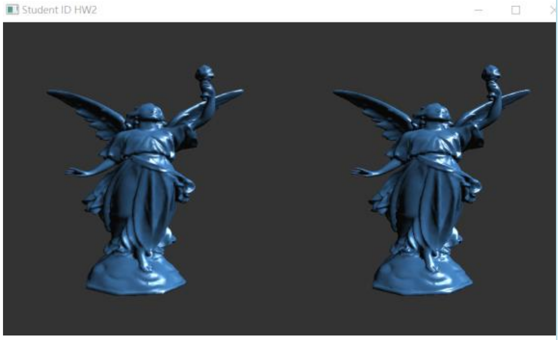
Translation mode



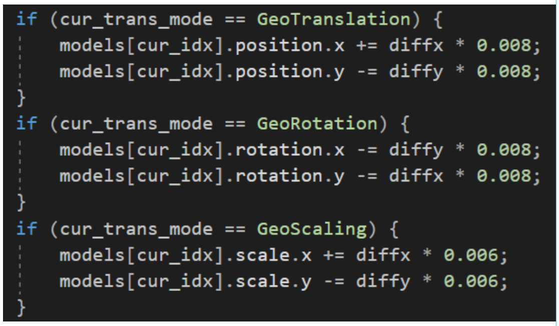
Rotation mode



Scale mode

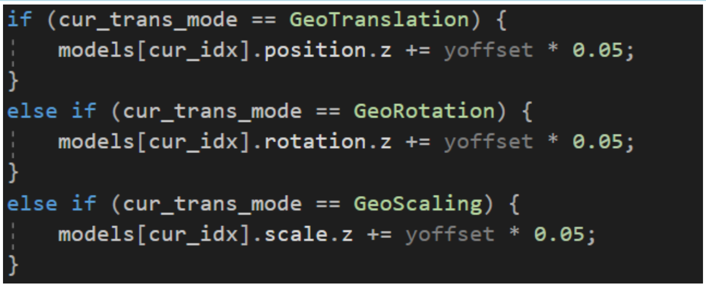


Cursor_pos_callback



```
if (cur_trans_mode == GeoTranslation) {
    models[cur_idx].position.x += diffx * 0.008;
    models[cur_idx].position.y -= diffy * 0.008;
}
if (cur_trans_mode == GeoRotation) {
    models[cur_idx].rotation.x -= diffy * 0.008;
    models[cur_idx].rotation.y -= diffx * 0.008;
}
if (cur_trans_mode == GeoScaling) {
    models[cur_idx].scale.x += diffx * 0.006;
    models[cur_idx].scale.y -= diffy * 0.006;
}
```

Scroll_callback

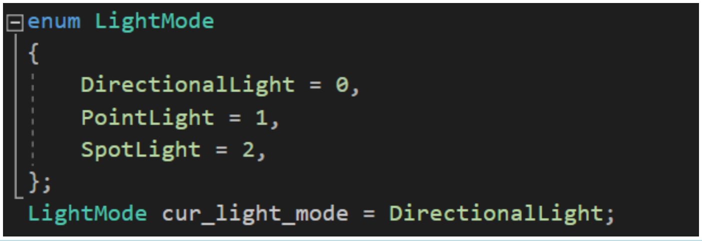


```
if (cur_trans_mode == GeoTranslation) {
    models[cur_idx].position.z += yoffset * 0.05;
}
else if (cur_trans_mode == GeoRotation) {
    models[cur_idx].rotation.z += yoffset * 0.05;
}
else if (cur_trans_mode == GeoScaling) {
    models[cur_idx].scale.z += yoffset * 0.05;
}
```

D. Keyboard : L

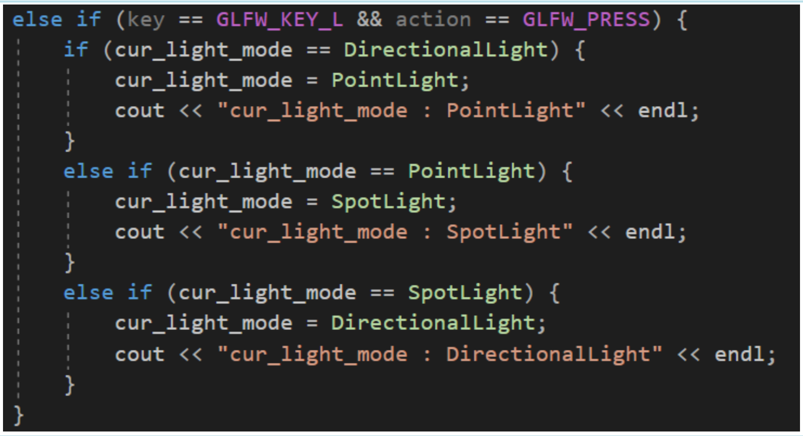
參考 *TransMode* 的作法新增一個負責記錄 light mode 的 class，並利用變數 *cur_light_mode* 紀錄當下使用的 light mode，由鍵盤 L 鍵在三種 light mode 之間切換。

Define light mode



```
enum LightMode
{
    DirectionalLight = 0,
    PointLight = 1,
    SpotLight = 2,
};
LightMode cur_light_mode = DirectionalLight;
```

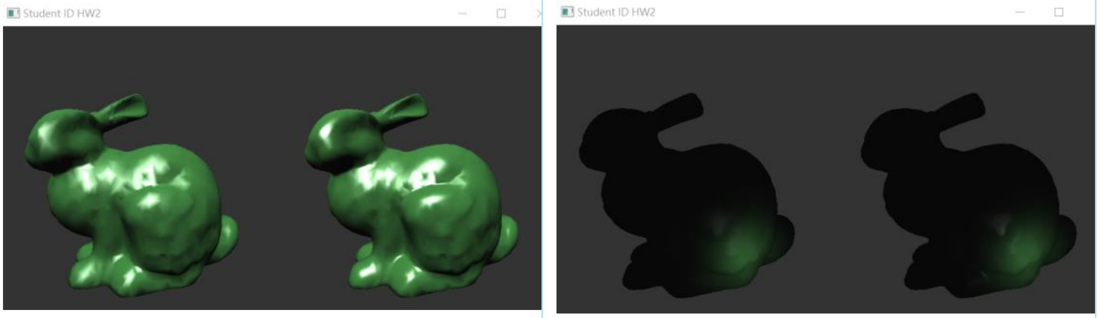
KeyCallback



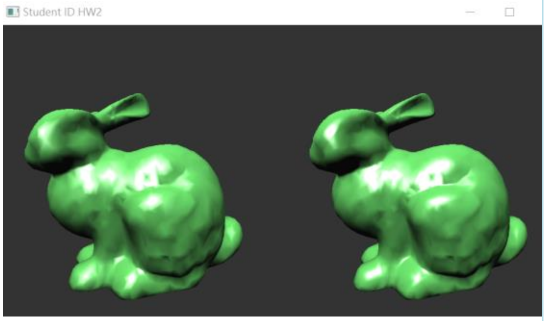
```
else if (key == GLFW_KEY_L && action == GLFW_PRESS) {
    if (cur_light_mode == DirectionalLight) {
        cur_light_mode = PointLight;
        cout << "cur_light_mode : PointLight" << endl;
    }
    else if (cur_light_mode == PointLight) {
        cur_light_mode = SpotLight;
        cout << "cur_light_mode : SpotLight" << endl;
    }
    else if (cur_light_mode == SpotLight) {
        cur_light_mode = DirectionalLight;
        cout << "cur_light_mode : DirectionalLight" << endl;
    }
}
```

E. Keyboard : K

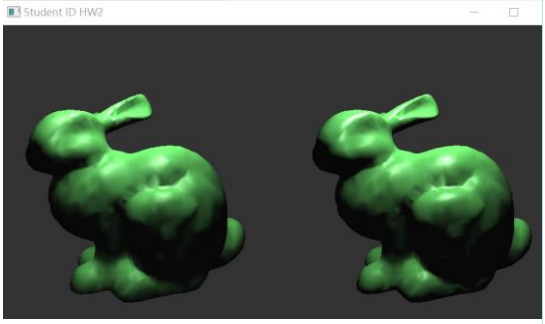
Change light's position



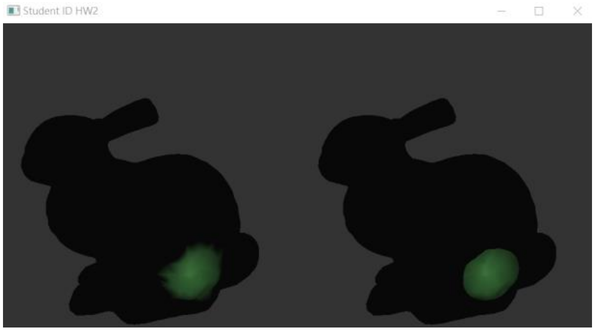
Scroll_callback (directional light)

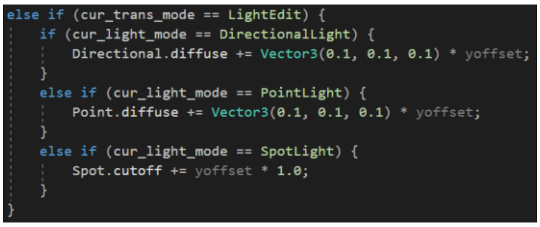
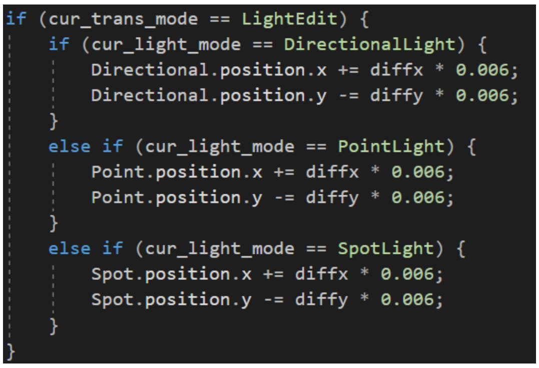


Scroll_callback (point light)



Scroll_callback (spot light)



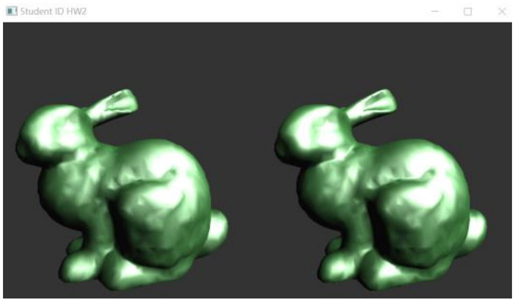
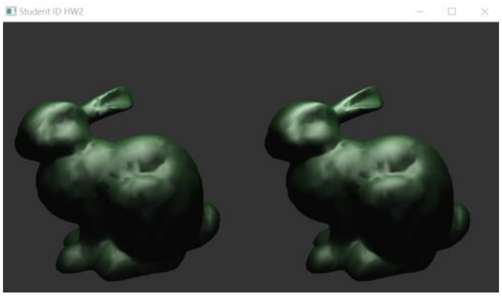
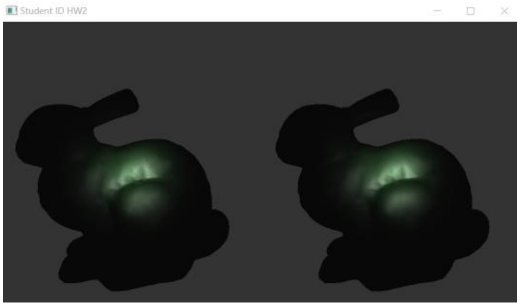
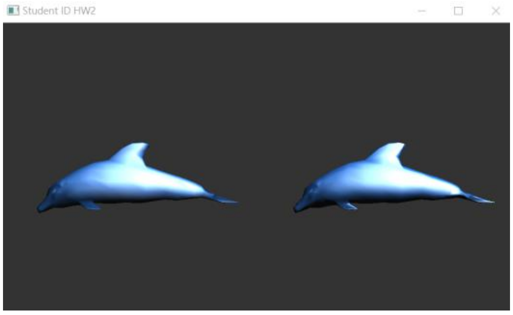
Scroll_callback	Cursor_callback
	

```
else if (cur_trans_mode == LightEdit) {
    if (cur_light_mode == DirectionalLight) {
        Directional.diffuse += Vector3(0.1, 0.1, 0.1) * yoffset;
    }
    else if (cur_light_mode == PointLight) {
        Point.diffuse += Vector3(0.1, 0.1, 0.1) * yoffset;
    }
    else if (cur_light_mode == SpotLight) {
        Spot.cutoff += yoffset * 1.0;
    }
}
```

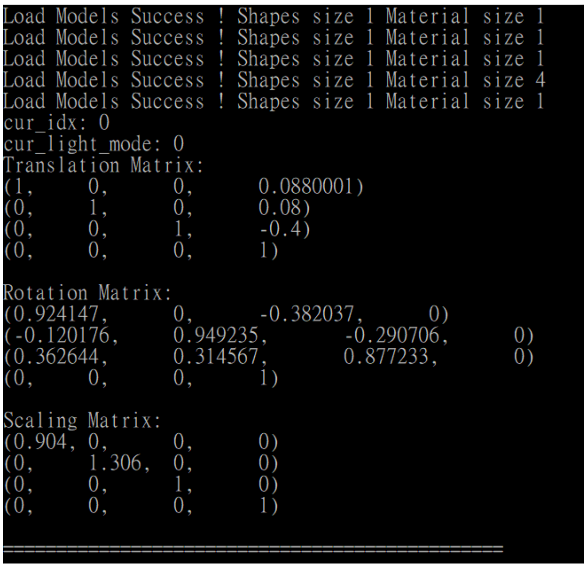
```
if (cur_trans_mode == LightEdit) {
    if (cur_light_mode == DirectionalLight) {
        Directional.position.x += diffx * 0.006;
        Directional.position.y -= diffy * 0.006;
    }
    else if (cur_light_mode == PointLight) {
        Point.position.x += diffx * 0.006;
        Point.position.y -= diffy * 0.006;
    }
    else if (cur_light_mode == SpotLight) {
        Spot.position.x += diffx * 0.006;
        Spot.position.y -= diffy * 0.006;
    }
}
```

F. Keyboard : J

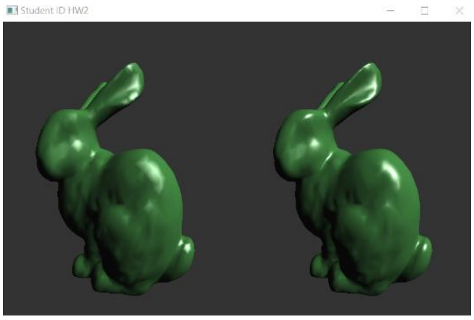
新增一個全域變數 *float shininess* 紀錄當下的 shininess value，便可以 apply 到所有的 model 以及不同的 light mode。當滾動滑鼠滾軸時更新 shininess，傳送到 vertex shader 與 fragment shader 即可看到 shininess 應用在不同的 light mode 上。

Directional light	Point light
	
Spot light	**Other model**
	

G. Keyboard : I

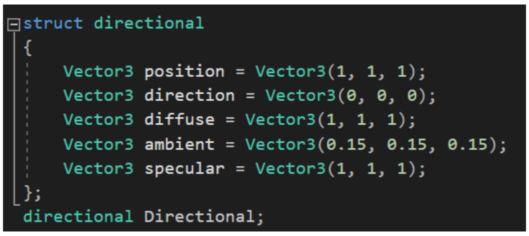


```
Load Models Success ! Shapes size 1 Material size 1
Load Models Success ! Shapes size 1 Material size 1
Load Models Success ! Shapes size 1 Material size 1
Load Models Success ! Shapes size 1 Material size 4
Load Models Success ! Shapes size 1 Material size 1
cur_idx: 0
cur_light_mode: 0
Translation Matrix:
(1,      0,      0,      0.0880001)
(0,      1,      0,      0.08)
(0,      0,      1,      -0.4)
(0,      0,      0,      1)

Rotation Matrix:
(0.924147,      0,      -0.382037,      0)
(-0.120176,     0.949235,       -0.290706,      0)
(0.362644,      0.314567,       0.877233,       0)
(0,      0,      0,      1)

Scaling Matrix:
(0.904, 0,      0,      0)
(0,      1.306, 0,      0)
(0,      0,      1,      0)
(0,      0,      0,      1)

===============================================
```

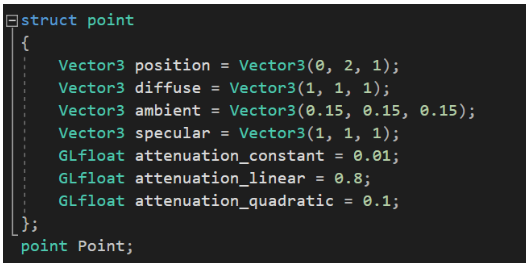


H. Define 3 types of light

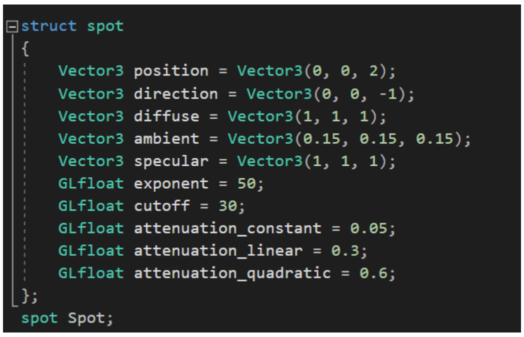
定義三個 class 分別紀錄 directional/point/spot light 的 attribute。



```
struct directional
{
    Vector3 position = Vector3(1, 1, 1);
    Vector3 direction = Vector3(0, 0, 0);
    Vector3 diffuse = Vector3(1, 1, 1);
    Vector3 ambient = Vector3(0.15, 0.15, 0.15);
    Vector3 specular = Vector3(1, 1, 1);
};
directional Directional;
```



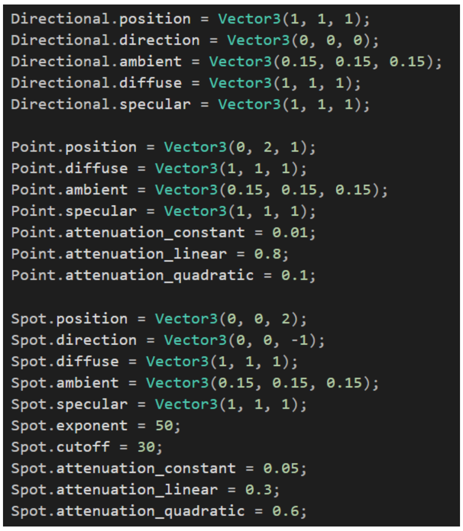
```
struct point
{
    Vector3 position = Vector3(0, 2, 1);
    Vector3 diffuse = Vector3(1, 1, 1);
    Vector3 ambient = Vector3(0.15, 0.15, 0.15);
    Vector3 specular = Vector3(1, 1, 1);
    GLfloat attenuation_constant = 0.01;
    GLfloat attenuation_linear = 0.8;
    GLfloat attenuation_quadratic = 0.1;
};
point Point;
```



```
struct spot
{
    Vector3 position = Vector3(0, 0, 2);
    Vector3 direction = Vector3(0, 0, -1);
    Vector3 diffuse = Vector3(1, 1, 1);
    Vector3 ambient = Vector3(0.15, 0.15, 0.15);
    Vector3 specular = Vector3(1, 1, 1);
    GLfloat exponent = 50;
    GLfloat cutoff = 30;
    GLfloat attenuation_constant = 0.05;
    GLfloat attenuation_linear = 0.3;
    GLfloat attenuation_quadratic = 0.6;
};
spot Spot;
```

I. Initialize parameters

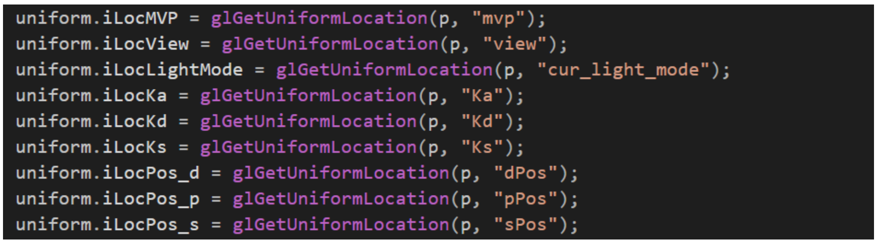
在 *initParameter()* 中設定這些 attribute 的初始值。



```
Directional.position = Vector3(1, 1, 1);
Directional.direction = Vector3(0, 0, 0);
Directional.ambient = Vector3(0.15, 0.15, 0.15);
Directional.diffuse = Vector3(1, 1, 1);
Directional.specular = Vector3(1, 1, 1);

Point.position = Vector3(0, 2, 1);
Point.diffuse = Vector3(1, 1, 1);
Point.ambient = Vector3(0.15, 0.15, 0.15);
Point.specular = Vector3(1, 1, 1);
Point.attenuation_constant = 0.01;
Point.attenuation_linear = 0.8;
Point.attenuation_quadratic = 0.1;

Spot.position = Vector3(0, 0, 2);
Spot.direction = Vector3(0, 0, -1);
Spot.diffuse = Vector3(1, 1, 1);
Spot.ambient = Vector3(0.15, 0.15, 0.15);
Spot.specular = Vector3(1, 1, 1);
Spot.exponent = 50;
Spot.cutoff = 30;
Spot.attenuation_constant = 0.05;
Spot.attenuation_linear = 0.3;
Spot.attenuation_quadratic = 0.6;
```

J. Set shader

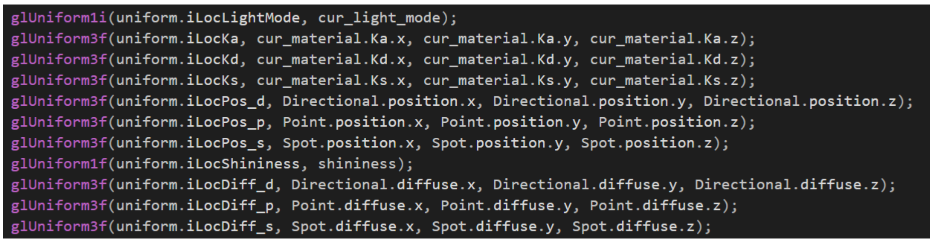
由於這些 attribute 參數需要傳送到 shader 做計算，因此先在 *setShaders()* 中定義變數的對應關係，以下為部分截圖。



```
uniform.iLocMVP = glGetUniformLocation(p, "mvp");
uniform.iLocView = glGetUniformLocation(p, "view");
uniform.iLocLightMode = glGetUniformLocation(p, "cur_light_mode");
uniform.iLocKa = glGetUniformLocation(p, "Ka");
uniform.iLocKd = glGetUniformLocation(p, "Kd");
uniform.iLocKs = glGetUniformLocation(p, "Ks");
uniform.iLocPos_d = glGetUniformLocation(p, "dPos");
uniform.iLocPos_p = glGetUniformLocation(p, "pPos");
uniform.iLocPos_s = glGetUniformLocation(p, "sPos");
```

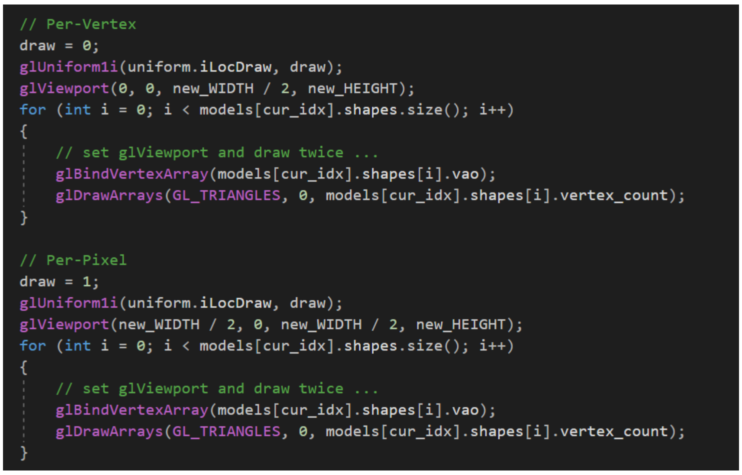
K. Render Scene

在 *RenderScene()* 中利用 *glUniformXX* 將 uniform 變數傳送到 shader，以下為部分截圖。



```
glUniform1i(uniform.iLocLightMode, cur_light_mode);
glUniform3f(uniform.iLocKa, cur_material.Ka.x, cur_material.Ka.y, cur_material.Ka.z);
glUniform3f(uniform.iLocKd, cur_material.Kd.x, cur_material.Kd.y, cur_material.Kd.z);
glUniform3f(uniform.iLocKs, cur_material.Ks.x, cur_material.Ks.y, cur_material.Ks.z);
glUniform3f(uniform.iLocPos_d, Directional.position.x, Directional.position.y, Directional.position.z);
glUniform3f(uniform.iLocPos_p, Point.position.x, Point.position.y, Point.position.z);
glUniform3f(uniform.iLocPos_s, Spot.position.x, Spot.position.y, Spot.position.z);
glUniform1f(uniform.iLocShininess, shininess);
glUniform3f(uniform.iLocDiff_d, Directional.diffuse.x, Directional.diffuse.y, Directional.diffuse.z);
glUniform3f(uniform.iLocDiff_p, Point.diffuse.x, Point.diffuse.y, Point.diffuse.z);
glUniform3f(uniform.iLocDiff_s, Spot.diffuse.x, Spot.diffuse.y, Spot.diffuse.z);
```

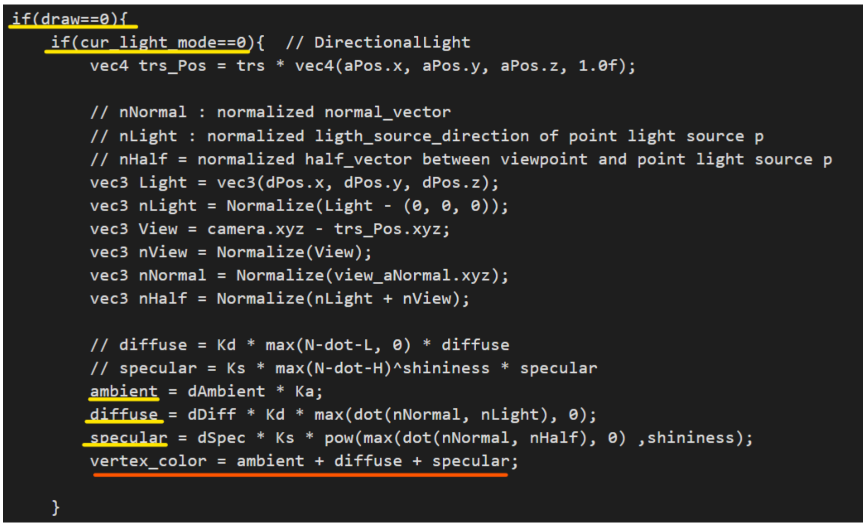
利用變數 *draw* 代表為 per-vertex 或 per-pixel，在畫 model 之前需先將 *draw* 值傳送到 shader，並且定義 viewport 視窗(左半邊為 per-vertex, 右半邊為 per-pixel)



```
// Per-Vertex
draw = 0;
glUniform1i(uniform.iLocDraw, draw);
glViewport(0, 0, new_WIDTH / 2, new_HEIGHT);
for (int i = 0; i < models[cur_idx].shapes.size(); i++)
{
    // set glViewport and draw twice ...
    glBindVertexArray(models[cur_idx].shapes[i].vao);
    glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
}

// Per-Pixel
draw = 1;
glUniform1i(uniform.iLocDraw, draw);
glViewport(new_WIDTH / 2, 0, new_WIDTH / 2, new_HEIGHT);
for (int i = 0; i < models[cur_idx].shapes.size(); i++)
{
    // set glViewport and draw twice ...
    glBindVertexArray(models[cur_idx].shapes[i].vao);
    glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
}
```

L. Vertex Shader

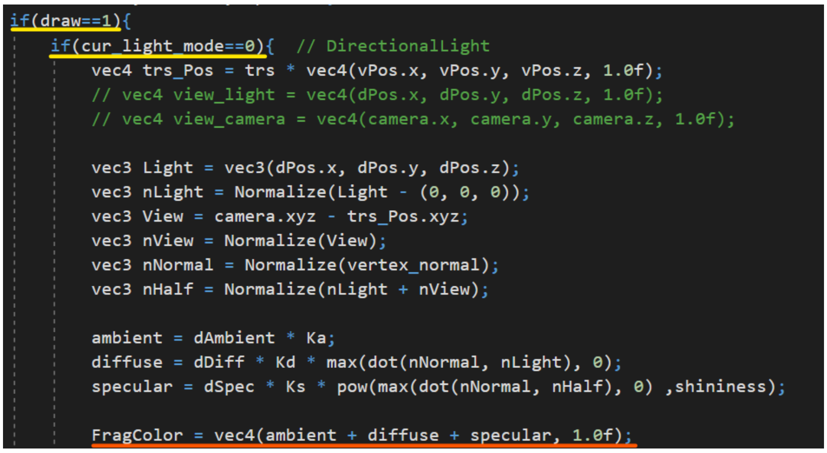
在 vertex shader 中計算 per-vertex lighting，當 draw==0 代表為 per-vertex lighting，再根據 cur_light_mode 分為 directional/point/spot light 計算方式。參考講義第 28, 29, 47 頁的公式分別計算出 ambient、diffuse、specular 三個值，加總後 assign 給 vertex_color 傳送到 fragment shader。



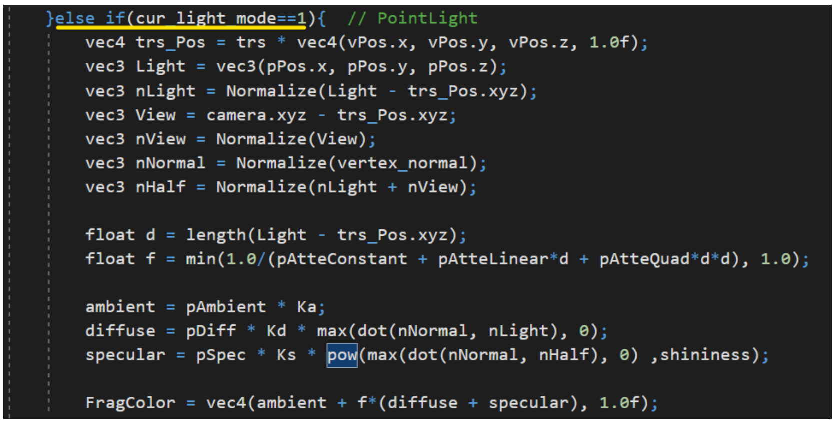
```
if(draw==0){
    if(cur_light_mode==0){  // DirectionalLight
        vec4 trs_Pos = trs * vec4(aPos.x, aPos.y, aPos.z, 1.0f);

        // nNormal : normalized normal_vector
        // nLight : normalized ligth_source_direction of point light source p
        // nHalf = normalized half_vector between viewpoint and point light source p
        vec3 Light = vec3(dPos.x, dPos.y, dPos.z);
        vec3 nLight = Normalize(Light - (0, 0, 0));
        vec3 View = camera.xyz - trs_Pos.xyz;
        vec3 nView = Normalize(View);
        vec3 nNormal = Normalize(view_aNormal.xyz);
        vec3 nHalf = Normalize(nLight + nView);

        // diffuse = Kd * max(N-dot-L, 0) * diffuse
        // specular = Ks * max(N-dot-H)^shininess * specular
        ambient = dAmbient * Ka;
        diffuse = dDiff * Kd * max(dot(nNormal, nLight), 0);
        specular = dSpec * Ks * pow(max(dot(nNormal, nHalf), 0) ,shininess);
        vertex_color = ambient + diffuse + specular;

    }
```

M. Fragment Shader

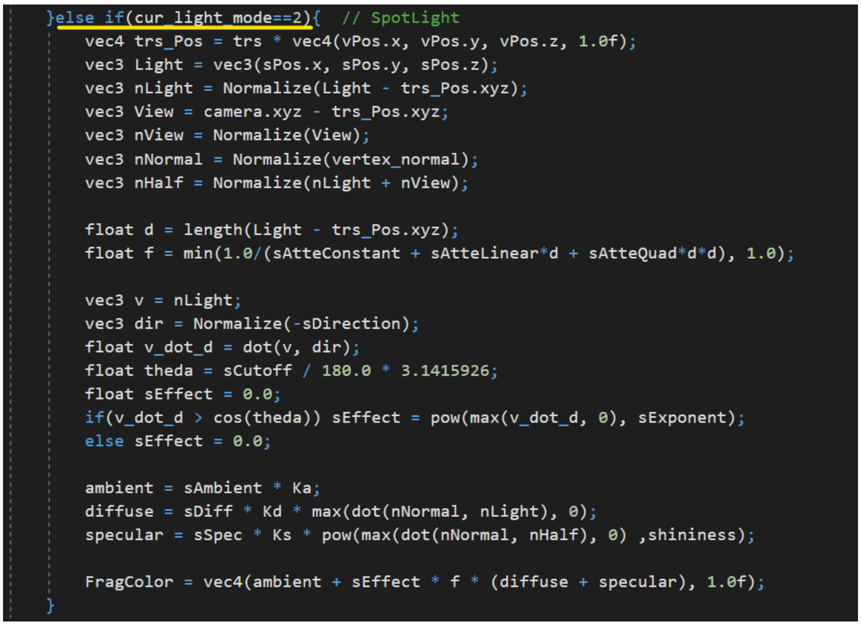
作法與 vertex shader 中的計算相似，但 fragment shader 是計算 per-pixel lighting，因此判斷 draw==1 再根據 cur_light_mode 分為 directional / point / spot light



```
if(draw==1){
    if(cur_light_mode==0){  // DirectionalLight
        vec4 trs_Pos = trs * vec4(vPos.x, vPos.y, vPos.z, 1.0f);
        // vec4 view_light = vec4(dPos.x, dPos.y, dPos.z, 1.0f);
        // vec4 view_camera = vec4(camera.x, camera.y, camera.z, 1.0f);

        vec3 Light = vec3(dPos.x, dPos.y, dPos.z);
        vec3 nLight = Normalize(Light - (0, 0, 0));
        vec3 View = camera.xyz - trs_Pos.xyz;
        vec3 nView = Normalize(View);
        vec3 nNormal = Normalize(vertex_normal);
        vec3 nHalf = Normalize(nLight + nView);

        ambient = dAmbient * Ka;
        diffuse = dDiff * Kd * max(dot(nNormal, nLight), 0);
        specular = dSpec * Ks * pow(max(dot(nNormal, nHalf), 0) ,shininess);

        FragColor = vec4(ambient + diffuse + specular, 1.0f);
```



```
    }else if(cur_light_mode==1){  // PointLight
        vec4 trs_Pos = trs * vec4(vPos.x, vPos.y, vPos.z, 1.0f);
        vec3 Light = vec3(pPos.x, pPos.y, pPos.z);
        vec3 nLight = Normalize(Light - trs_Pos.xyz);
        vec3 View = camera.xyz - trs_Pos.xyz;
        vec3 nView = Normalize(View);
        vec3 nNormal = Normalize(vertex_normal);
        vec3 nHalf = Normalize(nLight + nView);

        float d = length(Light - trs_Pos.xyz);
        float f = min(1.0/(pAtteConstant + pAtteLinear*d + pAtteQuad*d*d), 1.0);

        ambient = pAmbient * Ka;
        diffuse = pDiff * Kd * max(dot(nNormal, nLight), 0);
        specular = pSpec * Ks * pow(max(dot(nNormal, nHalf), 0) ,shininess);

        FragColor = vec4(ambient + f*(diffuse + specular), 1.0f);
```



```
}else if(cur_light_mode==2){  // SpotLight
    vec4 trs_Pos = trs * vec4(vPos.x, vPos.y, vPos.z, 1.0f);
    vec3 Light = vec3(sPos.x, sPos.y, sPos.z);
    vec3 nLight = Normalize(Light - trs_Pos.xyz);
    vec3 View = camera.xyz - trs_Pos.xyz;
    vec3 nView = Normalize(View);
    vec3 nNormal = Normalize(vertex_normal);
    vec3 nHalf = Normalize(nLight + nView);

    float d = length(Light - trs_Pos.xyz);
    float f = min(1.0/(sAtteConstant + sAtteLinear*d + sAtteQuad*d*d), 1.0);

    vec3 v = nLight;
    vec3 dir = Normalize(-sDirection);
    float v_dot_d = dot(v, dir);
    float theda = sCutoff / 180.0 * 3.1415926;
    float sEffect = 0.0;
    if(v_dot_d > cos(theda)) sEffect = pow(max(v_dot_d, 0), sExponent);
    else sEffect = 0.0;

    ambient = sAmbient * Ka;
    diffuse = sDiff * Kd * max(dot(nNormal, nLight), 0);
    specular = sSpec * Ks * pow(max(dot(nNormal, nHalf), 0) ,shininess);

    FragColor = vec4(ambient + sEffect * f * (diffuse + specular), 1.0f);
}
```