計算機圖學 作業三報告

103062234 張克齊

一、 The method of operating program:

執行程式前，要先確定在 CG_HW3 底下有 NormalModels 這個資料夾，裡面用來存放要讀取的.obj 檔；接著，要修改程式中所開的 filename 陣列和 file_size，若 filename 中寫了 3 個路徑的名稱，則要將 file_size 改成 3，如下圖所示，這樣就能順利執行程式。一開始為 perspective mode，按 o 可切換至 orthogonal mode，按 p 可切回去；而操作如同要求所述，也可按 h 鍵有詳細的按鍵說明。

```
char filename[][50] = { "NormalModels/High/dragon10KN.obj", "NormalModels/Medium/cow3KN.obj",
                        "NormalModels/Medium/bunny5KN.obj" };
int file_size = 3;
```

二、 Implementation and problems:

這次的重點為 Lighting 的實作，我的想法是在 main.cpp 中處理完所有變數的設定與改變，並把所有要傳的參數透過 uniform variable 傳入 shader；Shader 主要處理的是怎麼利用傳進來的參數算出結果，也就是公式的展現。因此，以下分為兩大部分來講解：

1. 主程式:

(1) OBJ 檔的讀取：

這部分的讀取跟前兩次有所不同，因為這次是讀他的 Normal vector 而非 Color，因此在讀取的時候可以利用助教提示的方法取得 Normal 的 index，如下圖所示。

```
// the index of each normal
int indc1 = OBJ->triangles[i].nindices[0];
int indc2 = OBJ->triangles[i].nindices[1];
int indc3 = OBJ->triangles[i].nindices[2];
normals[i*9 + 0] = OBJ->normals[indc1*3+0];
normals[i*9 + 1] = OBJ->normals[indc1*3+1];
normals[i*9 + 2] = OBJ->normals[indc1*3+2];
```

而讀 OBJ 的過程中也可以獲得 material 的參數 ambient, diffuse, specular，提示中說可以利用 group 的概念，但我在實作時發現因為讀進來的都是單色系，代表 group 中其實只有一項有值，因此我利用下圖的方法只取出第一項的參數傳入 Shader 中，這樣我們就得到所有 OBJ 檔提供的參數並都傳入 Shader 做後續的處理。

```
//Get material data here
glUniform4fv(iLocMAmbient, 1, OBJ->materials[1].ambient);
glUniform4fv(iLocMDiffuse, 1, OBJ->materials[1].diffuse);
glUniform4fv(iLocMSpecular, 1, OBJ->materials[1].specular);
```

(2) Lightsource 的設定與傳遞:

Lightsource 的設定中，首先我先在 setLightingSource() 中設定起始值和 3 種光源的參數，在這邊我開 lightsource[4]的 struct 陣列，0 代表 Directional Light，1 代表 Point Light，2 代表 Spot Light，3 代表初始狀態；接著就透過設定不同的 Loc 取得 shader 中 uniform 的變數並利用 glUniformXX() 寫入，XX 代表不同的型態，將所有一開始設定好的參數全數傳入 shader 中，圖示擷取部分的抓位置和寫入。

```
iLocLPPosition = glGetUniformLocation(p, "LightSource[1].position");
iLocLPAmbient = glGetUniformLocation(p, "LightSource[1].ambient");
iLocLPDiffuse = glGetUniformLocation(p, "LightSource[1].diffuse");
iLocLPSpecular = glGetUniformLocation(p, "LightSource[1].specular");
iLocLPca = glGetUniformLocation(p, "LightSource[1].constantAttenuation");
iLocLPla = glGetUniformLocation(p, "LightSource[1].linearAttenuation");
iLocLPqa = glGetUniformLocation(p, "LightSource[1].quadraticAttenuation");
```

```
glUniform4fv(iLocLPPosition, 1, lightsource[1].position);
glUniform4fv(iLocLPAmbient, 1, lightsource[1].ambient);
glUniform4fv(iLocLPDiffuse, 1, lightsource[1].diffuse);
glUniform4fv(iLocLPSpecular, 1, lightsource[1].specular);
glUniform1f(iLocLPca, lightsource[1].constantAttenuation);
glUniform1f(iLocLPla, lightsource[1].linearAttenuation);
glUniform1f(iLocLPqa, lightsource[1].quadraticAttenuation);
```

功能實現中，會改到 lightsource 中的參數，我的方法是直接更改原本存在主程式變數的值，改變後利用 glUniformXX()寫入，也就是 setLightingSource()我只有在一開始跑一遍，其他都是在更改後立即寫入 shader。圖中為增加 spot light EXP 的操作。

```
case GLUT_LEFT_BUTTON:
    lightsource[2].spotExponent += 5.0f;
    glUniform1f(iLocLSExp, lightsource[2].spotExponent);
    break;
```

(3) mode 控制:

因為這次的作業中會有疊加的效果，因此我的方法是有六種 mode 參數來控制各項的開關，1 代表開，0 代表關；分別為三種不同光及三種組成 Light 的元素，並將 6 種 mode 傳入 shader，在 shader 中控制開關時給的值，下圖為 6 種 Mode 的傳遞，one, two, three 代表三種光源，a 是 ambient，d 是 diffuse，s 是 specular。

```
GLint iLocModeOne, iLocModeTwo, iLocModeThree, iLocModeA, iLocModeD, iLocModeS;
int mode_one = 0;
int mode_two = 0;
int mode_three = 0;
int mode_a = 1; // ON
int mode_d = 1; // ON
int mode_s = 1; // ON
```

```
uniform int modeONE;
uniform int modeTWO;
uniform int modeTHREE;

uniform int modeA;
uniform int modeD;
uniform int modeS;
```

(4) Rotate:

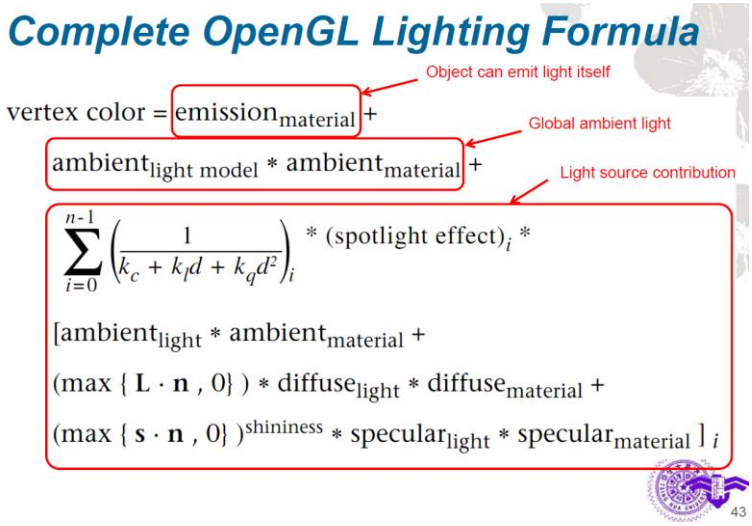
在旋轉的部分 Matrix 跟上次作業一樣，而因為我只有做 y 軸的旋轉因此只有一個 R，但是只做這樣會有一個問題，就是光源會跟著你的物體旋轉而非固定。因此，從助教得提示中，我們必須傳入 R 矩陣的 invert 跟一開始讀入的 normal vector 相乘才會抵消 R 的作用，可以讓光源固定在我們設的地方。突圍 R 的 invert 的傳遞和 normal vector 的改變，其中 M 為 y 軸旋轉矩陣，傳入方法跟傳 mvp 一樣要轉成 column major。

```
R = M.invert();  vec4 N = R * vec4(av3normal, 0.0);
```

```
GLfloat r[16];
// row-major ---> column-major
r[0] = R[0];   r[4] = R[1];    r[8] = R[2];     r[12] = R[3];
r[1] = R[4];   r[5] = R[5];    r[9] = R[6];     r[13] = R[7];
r[2] = R[8];   r[6] = R[9];    r[10] = R[10];   r[14] = R[11];
r[3] = R[12];  r[7] = R[13];   r[11] = R[14];   r[15] = R[15];
glUniformMatrix4fv(iLocR, 1, GL_FALSE, r);
```

2. Shader:

大方向是要實作出下圖 slide 中講的加總公式，而我分別討論三項作法。



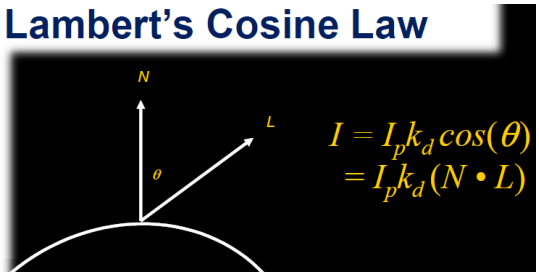
(1) Ambient:

Ambient 的部分比較單純，指的是環境光，代表及時在黑暗的環境下還是會有點微量的光，因此在實作時是固定的，不會因為不同的 lightsource 而改變，圖為我設定的 ambient，給的是灰色定值(0.7, 0.7, 0.7, 1)。

```
vv4ambient = Material.ambient * LightSource[0].ambient;
```

(2) Diffuse:

這個代表著漫反射的效果，也是主要顯示顏色的方法，跟方向有很大的影響，算法為講義寫的那樣，能夠反射出物體主要的顏色。其中，可利用 dot()算內積和利用 pow()算次方，並記得將指定的向量做 normalize()，像是 normal vector 和光源到點的向量。

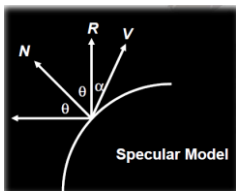


(3) Specular:

為光打在平滑物體上會產生如同鏡面的光澤，是不同的亮點，顏色為光的顏色(白)而非物體的顏色。作法也是參照講義即可完成，這邊要注意的是有關 V 的向量，他是人眼看的方向向量，因此要將做 perspective 時設定的眼睛位置傳入 shader 中(命名為 eye)，並與物體的位置相減取 normalize 就能得到 V 的向量。

$$I_s = I_p k_s \cos^n \alpha = I_p k_s (R_p \cdot V)^n$$

$$I = I_a k_a + f_p I_p (k_d (N \cdot L_p) + k_s (R_p \cdot V)^n)$$

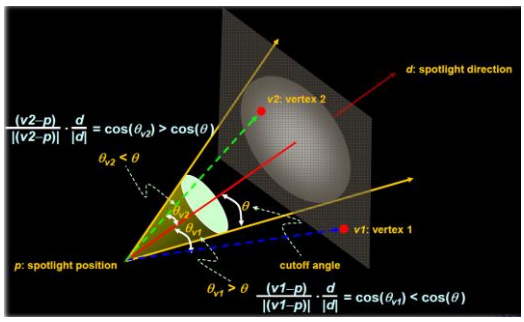


(4) Point & Spot Light adjusting:

做點光源時，要在 Diffuse 和 Specular 上乘上 fatt 的參數，才會得到是從一個點看出去的效果而非平行光，作法如同講義所述，其中 c1 對應的是傳入的 constantAttenuation，c2 對應的是 linearAttenuation，c3 對應的是 quadraticAttenuation，dL 則是點光源到該點的長度，可以利用 length()計算。

$$I = I_a k_a + f_{att} I_p k_d (N \bullet L) \qquad f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right)$$

而 Spotlight 的部分，除了要乘上 fatt 外，還要去算他的 spotlight effect，算法是要先判斷是否超出物體邊界，超出的話值為 0，若沒超出的話可以利用講義公式計算出 spotlight effect 效果並跟 fatt 一樣乘在 diffuse 和 specular 後的結果，就可以達到 spotlight 的效果。



Spotlight Effect =

- 1, if the light source is not a spotlight
- 0, if the light source is a spotlight but the vertex lies outside the cone of illumination produced by the spotlight
- Otherwise, spotlight effect = $(\max\{v \cdot d, 0\})^{spot\_exp}$
  - v is the unit vector from the spotlight to the vertex
  - d is the spotlight direction

3. Bonus:

(1) Per pixel mode ('f'): (15%)

原本的概念是每傳進一個 vertex 就對它進行光照的運算，而要把它改成 pixel 才進行運算，也就是在 fragment shader 才進行處理。我的作法是開兩組 (vert, frag) shader 來控制不同的 mode，切換的方法是在 setShaders() 中利用 if else 判斷要讀取哪組 shader，並在每次按下 f 鍵在 set 一遍，這樣可以讓我們傳入兩組 shader 的變數都是一樣的。在 shader 中，原本寫在 vertex shader 的運算全部移到 fragment shader 來做，而要在兩個 shader 間傳遞參數可以利用 varying 來宣告，這樣就能達到光澤更加圓滑的效果，圖為控制讀哪個 shader。

```
if (mode_f == 1) {
    vs = textFileRead("shader.vert");
    fs = textFileRead("shader.frag");
}
else {
    vs = textFileRead("sample.vert");
    fs = textFileRead("sample.frag");
}
```

(2) Change spot light to another kind ('c'): (5%)

這個部份其實比較簡單，就是要再多一個變數 (我取名 mode_c) 傳入 shader 來控制現在要是哪種光源，並且切換的每種光源都可以透過滑鼠的 hover 來改變光的 position，這樣就能順利切換。

三、 Other effort I have done:

在這次的作業中，因為有很多不同的組合和控制，所以我有適當的 Print 出現在的狀態和改變，按不同的按鍵也會有不同的敘述，可以讓我在 Debug 中更加清晰易懂，也可以讓其他人知道我現在做的控制，圖為一些印出的敘述。

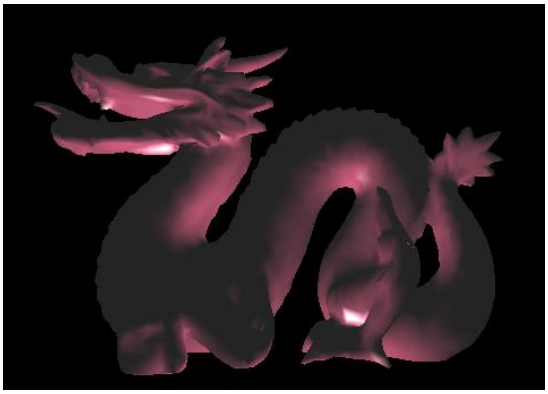
```
Press 'q'
Directional Light (First)
@@@ NOW: Directional Light ON, Point Light OFF, Spot Light OFF @@@


Press 'a'
=== NOW: Ambient OFF, Diffuse ON, Specular ON ===
Press 'd'
=== NOW: Ambient OFF, Diffuse ON, Specular OFF ===
Press 's'
=== NOW: Ambient OFF, Diffuse OFF, Specular OFF ===
Press 'e'
@@@ NOW: Directional Light ON, Point Light OFF, Spot Light ON @@@
Press 'c'
Press 'f'
Per pixel mode
Press 'o'
ORTHOGONAL mode
Press 'w'
@@@ NOW: Directional Light ON, Point Light ON, Spot Light ON @@@
```

四、 Results:

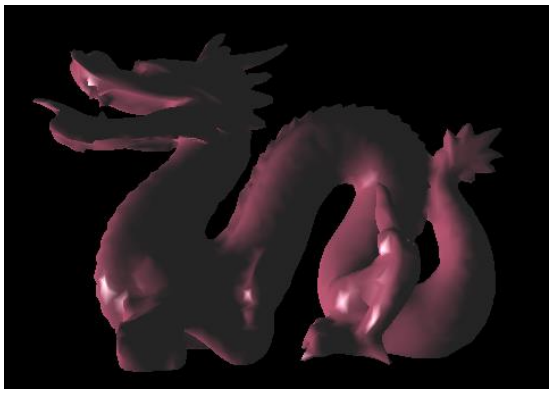
Directional Light(1)

Directional Light(2)

Directional Light(3)



Point light at (0,-1,-0.3)



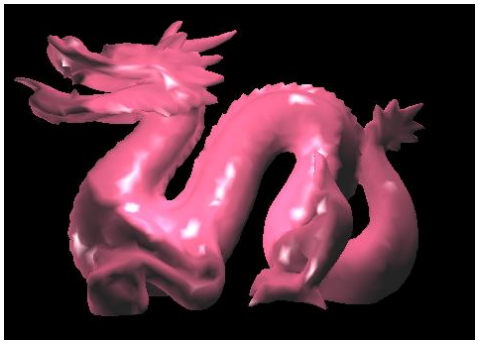
Point light at (-2.7,-1,0.3)

Spot light

Spot light(low spotCosCutff)

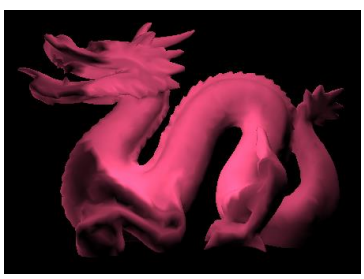
Spot light(low EXP)

Spot light(high EXP)



Blending for three kinds of light

Ambient



Diffuse

Specular