HW2 Report

main.cpp

先前作業一 transformation 所撰寫的部分在這次的 report 裡就沒有再特別提及，主要是針對 lighting 的部分進行說明。

1. Parameters

由於我們要將視窗分成兩個 viewport，所以用另外兩個變數來記錄當下的視窗大小，去改變 RenderScene()時 glViewport 裡的位置參數。

```
int window_width = 800;
int window_height = 800;
```

針對 lighting 的部分，將需要用到的 local parameter，包含 ambient diffuse specular lighting 所需要用到的係數、位置等資訊包在 uniform structure 裡。

```
struct Uniform
{
    GLint iLocMVP;
    GLint iLocKa;
    GLint iLocKd;
    GLint iLocKs;
    GLint iLocmv;
    GLint iLocView;
    GLint iLocCamera;
    GLint iLocShininess;


    GLint iLocI_d;
    GLint iLocI_p;
    GLint iLocI_s;


    GLint iLocPos_d;
    GLint iLocPos_p;
    GLint iLocPos_s;


    GLuint iLocper_vertex;
    GLuint iLocLight_id;
    GLuint iLocSpot_cutoff;

};
Uniform uniform;
```

接著在全域根據助教作業規範，針對不同的 lighting，給定其 diffuse intensity 的值和 light position。

```
//Diffuse
Vector3 I_d = Vector3(1.0f, 1.0f, 1.0f);
Vector3 I_p = Vector3(1.0f, 1.0f, 1.0f);
Vector3 I_s = Vector3(1.0f, 1.0f, 1.0f);

//Light Position
Vector3 lightPos_d = Vector3(1.0f, 1.0f, 1.0f);
Vector3 lightPos_p = Vector3(0.0f, 2.0f, 1.0f);
Vector3 lightPos_s = Vector3(0.0f, 0.0f, 2.0f);
```

最後利用 cur_light_id 紀錄目前切換到的是哪種 lighting 以及利用 per_vertex 當作 flag，用在 render 不同的 viewport 做使用。Shininess 及 spotlight cutoff 也是根據作業規範給定初始值。

```
int cur_idx = 0; // represent which model should be rendered now
int cur_light_id = 0; //0:directional,1:point,2:spot
int per_vertex = 0; //per-vertex shading

float shininess = 64.0f;
float spot_cutoff = 30;
```

2. 在 ChangeSize()中更新目前視窗大小，並改變 model 的 aspect ratio。

```
// Call back function for window reshape
void ChangeSize(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
    window_width = width;
    window_height = height;
    // [TODO] change your aspect ratio
    proj.aspect = (float)width / (float)height;
    setPerspective();
}
```

3. 在 RenderScene()裡，除了原本的 mvp matrix 以外，設另外一個 matrix mv 儲存 model transform matrix 的值，在 vertex shader 中會使用到。

```
// Render function for display rendering
void RenderScene(void) {
    // clear canvas
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);

    Matrix4 T, R, S;
    // [DONE] update translation, rotation and scaling
    T = translate(models[cur_idx].position);
    R = rotate(models[cur_idx].rotation);
    S = scaling(models[cur_idx].scale);

    Matrix4 MVP;
    Matrix4 mv;
    GLfloat mvp[16];

    // [DONE] multiply all the matrix
    MVP = project_matrix * view_matrix * T * R * S;

    //mv = view_matrix * T * R * S;
    mv = T * R * S;
    // row-major ---> column-major
    setGLMatrix(mvp, MVP);
```

接著利用 glUniform 不同型別的 function 將所有產生 lighting 效果的參數傳到 vertex shader，包含 mvp 矩陣、model transformation 矩陣、view matrix、view position、diffusion、light position、目前 lighting 種類的 id 以及 shininess 和 spotlight cutoff 的值。

```
// use uniform to send mvp to vertex shader
glUniformMatrix4fv(uniform.iLocMVP, 1, GL_FALSE, mvp);
glUniformMatrix4fv(uniform.iLocmv, 1, GL_FALSE, mv.getTranspose());
glUniformMatrix4fv(uniform.iLocView, 1, GL_FALSE, view_matrix.getTranspose());
glUniform3f(uniform.iLocCamera, main_camera.position.x, main_camera.position.y, main_camera.position.z);

glUniform3f(uniform.iLocI_d, I_d.x, I_d.y, I_d.z);
glUniform3f(uniform.iLocI_p, I_p.x, I_p.y, I_p.z);
glUniform3f(uniform.iLocI_s, I_s.x, I_s.y, I_s.z);

glUniform3f(uniform.iLocPos_d, lightPos_d.x, lightPos_d.y, lightPos_d.z);
glUniform3f(uniform.iLocPos_p, lightPos_p.x, lightPos_p.y, lightPos_p.z);
glUniform3f(uniform.iLocPos_s, lightPos_s.x, lightPos_s.y, lightPos_s.z);

glUniform1f(uniform.iLocShininess, shininess);
glUniform1i(uniform.iLocLight_id, cur_light_id);
glUniform1f(uniform.iLocSpot_cutoff, spot_cutoff);
```

為了將兩個模型畫在旁邊，所以 set 兩次 viewport 並將寬度設為視窗大小的一半。第一個 for 迴圈畫的是視窗左半邊的圖且 lighting 的方式為 per-vertex lighting。第二個 for 迴圈畫的是視窗右半邊的圖且 lighting 的方式為 per-pixel lighting。

```
for (int i = 0; i < models[cur_idx].shapes.size(); i++)
{
    // set glViewport and draw twice ...
    per_vertex = 1;
    glUniform1i(uniform.iLocper_vertex, per_vertex);
    glBindVertexArray(models[cur_idx].shapes[i].vao);
    glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
    glUniform3fv(uniform.iLocKa, 1, &(models[cur_idx].shapes[i].material.Ka[0]));
    glUniform3fv(uniform.iLocKd, 1, &(models[cur_idx].shapes[i].material.Kd[0]));
    glUniform3fv(uniform.iLocKs, 1, &(models[cur_idx].shapes[i].material.Ks[0]));


}
glViewport(0, 0, window_width / 2, window_height);

for (int i = 0; i < models[cur_idx].shapes.size(); i++)
{
    // set glViewport and draw twice ...
    per_vertex = 0;
    glUniform1i(uniform.iLocper_vertex, per_vertex);
    glBindVertexArray(models[cur_idx].shapes[i].vao);
    glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
    glUniform3fv(uniform.iLocKa, 1, &(models[cur_idx].shapes[i].material.Ka[0]));
    glUniform3fv(uniform.iLocKd, 1, &(models[cur_idx].shapes[i].material.Kd[0]));
    glUniform3fv(uniform.iLocKs, 1, &(models[cur_idx].shapes[i].material.Ks[0]));


}
glViewport(window_width / 2, 0, window_width / 2, window_height);
```

4. 在按鍵操作的部分，除了原先的操作外，利用按鍵 L 可以切換不同的 lighting 種類，id = 0 為 directional light, id = 1 為 point light, id = 2 為 spot light。按鍵 K 會切換至 LightEdit mode，,按鍵 J 則會切換至 ShininessEdit mode。

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    // [TODO] Call back function for keyboard
    if (key == GLFW_KEY_Z && action == GLFW_PRESS) { ... }
    else if (key == GLFW_KEY_X && action == GLFW_PRESS) { ... }
    else if (key == GLFW_KEY_T && action == GLFW_PRESS) { ... }
    else if (key == GLFW_KEY_S && action == GLFW_PRESS) { ... }
    else if (key == GLFW_KEY_R && action == GLFW_PRESS) { ... }
    else if (key == GLFW_KEY_L && action == GLFW_PRESS) {/* switch between directional/point/spot light */
        cur_light_id += 1;
        if (cur_light_id > 2) {
            cur_light_id = 0;
        }
        cout << cur_light_id << endl;
    }
    else if (key == GLFW_KEY_K && action == GLFW_PRESS) {/* switch to light editing mode*/
        cur_trans_mode = LightEdit;
    }
    else if (key == GLFW_KEY_J && action == GLFW_PRESS) {/* switch to shininess editinig mode */
        cur_trans_mode = ShininessEdit;
    }
}
```

5. 滑鼠滾輪操作的部分，在 LightEdit mode 的情況下，滾動滑鼠滾輪會影響 directional light 和 point light 的 diffuse intensity，所以根據 yoffset 的值更新當前 lighting 種類的 diffuse intensity，弱勢 spot light 則會根據 yoffset 的值調整其 cutoff 的大小。在 ShininessEdit mode 情況下，滾動滑鼠滾輪會影響三種 lighting 的 shininess。

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    // [TODO] scroll up positive, otherwise it would be negtive
    if (cur_trans_mode == GeoTranslation){ ... }
    else if (cur_trans_mode == GeoRotation){ ... }
    else if (cur_trans_mode == GeoScaling){ ... }
    else if (cur_trans_mode == LightEdit) {
        if (cur_light_id == 0)
        {
            I_d = I_d + Vector3(1, 1, 1)*0.5*yoffset;
        }
        else if (cur_light_id == 1)
        {
            I_p = I_p + Vector3(1, 1, 1)*0.5*yoffset;
        }
        else if (cur_light_id == 2)
        {
            spot_cutoff += yoffset;
        }
    }
    else if (cur_trans_mode == ShininessEdit) {
        shininess += 1.5*yoffset;
    }
}
```

6. 滑鼠操作的部分，在 LightEdit mode 的情況下，按住滑鼠並移動游標會分別影響三種 lighting 的 light position。

```
static void cursor_pos_callback(GLFWwindow* window, double xpos, double ypos)
{
    // [TODO] cursor position callback function
    float xoffset, yoffset;
    xoffset = xpos - starting_press_x;
    yoffset = starting_press_y - ypos;
    starting_press_x = xpos;
    starting_press_y = ypos;

    if (mouse_pressed) {
        if (cur_trans_mode == GeoTranslation){ ... }
        else if (cur_trans_mode == GeoRotation){ ... }
        else if (cur_trans_mode == GeoScaling){ ... }
        else if (cur_trans_mode == LightEdit) {
            if (cur_light_id == 0)
            {
                lightPos_d.x += 0.01*xoffset;
                lightPos_d.y += 0.01*yoffset;
            }
            else if (cur_light_id == 1)
            {
                lightPos_p.x += 0.01*xoffset;
                lightPos_p.y += 0.01*yoffset;
            }
            else if (cur_light_id == 2)
            {
                lightPos_s.x += 0.01*xoffset;
                lightPos_s.y += 0.01*yoffset;
            }
        }
    }
}
```

7. 在 setShaders()函數中，使用 glGetUniformLocation 的方法定位賦值，讓這些值在不同的 shaders 中可以使用，所以將先前 uniform structure 裡的參數利用 glGetUniformLocation()進行定位賦值。

```
uniform.iLocMVP = glGetUniformLocation(p, "mvp");
uniform.iLocKa = glGetUniformLocation(p, "Ka");
uniform.iLocKd = glGetUniformLocation(p, "Kd");
uniform.iLocKs = glGetUniformLocation(p, "Ks");
uniform.iLocmv = glGetUniformLocation(p, "mv");
uniform.iLocView = glGetUniformLocation(p, "view_matrix");
uniform.iLocCamera = glGetUniformLocation(p, "cameraPos");
uniform.iLocShininess = glGetUniformLocation(p, "shininess");

uniform.iLocI_d = glGetUniformLocation(p, "I_d");
uniform.iLocI_p = glGetUniformLocation(p, "I_p");
uniform.iLocI_s = glGetUniformLocation(p, "I_s");

uniform.iLocPos_d = glGetUniformLocation(p, "lightPos_d");
uniform.iLocPos_p = glGetUniformLocation(p, "lightPos_p");
uniform.iLocPos_s = glGetUniformLocation(p, "lightPos_s");

uniform.iLocper_vertex = glGetUniformLocation(p, "per_vertex");
uniform.iLocLight_id = glGetUniformLocation(p, "cur_light_id");
uniform.iLocSpot_cutoff = glGetUniformLocation(p, "spot_cutoff");
```

vertex shader

1. Parameters

   先前利用 uniform 方法傳遞至 shader 的值可以在 vertex shader 定義後使用。

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec3 aNormal;

out vec3 vertex_color;
out vec3 vertex_normal;
out vec3 FragPos;
uniform mat4 mvp;

uniform mat4 mv;
uniform vec3 Ka;
uniform vec3 Kd;
uniform vec3 Ks;

uniform float shininess;
uniform vec3 Ia = vec3(0.15f,0.15f,0.15f);

//diffuse
uniform vec3 I_d; // directional
uniform vec3 I_p; // point
uniform vec3 I_s; //spot

//light position
uniform vec3 lightPos_d;
uniform vec3 lightPos_p;
uniform vec3 lightPos_s;
uniform int cur_light_id; //represent type of lights

uniform float spot_cutoff;
uniform mat4 view_matrix;
uniform vec3 cameraPos;
uniform int per_vertex;
```

2. 不同 lighting 的實作，根據 Phong Reflection Model 計算 directional lighting 的 color，並利用 Modified Phong Reflection Model 計算 point light 和 spot light 的 color。每個 vector 所代表的意義如註解。

甲、Directional light

```
vec3 directional_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 L = normalize(lightPos_d); //normalized light source direction
    vec3 R = reflect(-L, N); //normalized light source reflection vector
    vec3 V = normalize(cameraPos-FragPos); //normalized viewpoint direction vector

    vec3 ambient = Ia * Ka;
    vec3 diffuse = I_d * Kd * max(dot(N,L),0);
    vec3 specular = Ks * pow(max(dot(R,V),0), shininess);

    vec3 color = ambient + diffuse + specular;
    return color;
}
```

乙、Point light

```
vec3 point_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 V = normalize(cameraPos-FragPos);//normalized viewpoint direction vector
    vec3 L = normalize(lightPos_p - FragPos);
    vec3 H = normalize(L+V); //halfway vector


    vec3 ambient = Ia * Ka;
    vec3 diffuse = I_p * Kd * max(dot(N,L),0);
    vec3 specular = Ks * pow(max(dot(N,H),0), shininess);

    //calculate attenuation with respect to the distance between the light source and the object
    float d = length(lightPos_p - FragPos);
    float f_att = min(1/(0.01 + 0.8 * d + 0.1 * d * d),1);

    vec3 color = ambient + f_att * diffuse + f_att * specular;
    return color;
}
```

丙、Spot light

```
vec3 spot_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 V = normalize(cameraPos-FragPos);//normalized viewpoint direction vector
    vec3 L = normalize(lightPos_s - FragPos);
    vec3 H = normalize(L+V);  //halfway vector


    //calculate attenuation with respect to the distance between the light source and the object
    float d = length(lightPos_s - FragPos);
    float f_att = min(1/(0.05 + 0.3 * d + 0.6 * d * d),1);

    //spotlight effect
    vec4 spot_direction = vec4(0.0f,0.0f,-1.0f,1.0f);
    vec3 direction = normalize((transpose(inverse(view_matrix)) * spot_direction).xyz);
    float arc = dot(-L, direction);
    if(spot_cutoff<=degrees(acos(arc)))
    {
        vec3 ambient = Ia * Ka;
        vec3 color = ambient;
        return color;
    }
    else
    {
        float spot_effect =  pow(max(arc,0),50);
        vec3 ambient = Ia * Ka;
        vec3 diffuse = I_s * Kd * max(dot(N,L),0);
        vec3 specular = Ks * pow(max(dot(N,H),0), shininess);
        vec3 color = ambient + f_att * spot_effect * diffuse + f_att * spot_effect * specular;
        return color;
    }
}
```

3. 在 main()函數中，若 per-vertex 的值為 0，則根據傳遞進來的 light id，進行 color 的計算。

```
void main()
{
    // [TODO]
    gl_Position = mvp*vec4(aPos.x, aPos.y, aPos.z, 1.0);
    //vertex_color = aColor;
    //vertex_normal = aNormal;
    vec4 fragPos = mv*vec4(aPos.x, aPos.y, aPos.z, 1.0);
    FragPos = fragPos.xyz;
    vertex_normal = mat3(transpose(inverse(mv)))*aNormal;

    if(per_vertex == 0)
    {
        vec3 color = vec3(0,0,0);
        if(cur_light_id == 0)
        {
            color += directional_light();
        }
        else if(cur_light_id == 1)
        {
            color += point_light();
        }
        else if(cur_light_id == 2)
        {
            color += spot_light();
        }

        vertex_color = color;
    }

}
```

fragment shader

1. Fragment shader 中不同 lighting 的 color 計算方法基本上是一樣的，如下圖所示。

```
#version 330 core

out vec4 FragColor;
in vec3 vertex_color;
in vec3 vertex_normal;
in vec3 FragPos;


uniform vec3 Ka;
uniform vec3 Kd;
uniform vec3 Ks;

uniform float shininess;
uniform vec3 Ia = vec3(0.15f,0.15f,0.15f);

//diffuse
uniform vec3 I_d; // directional
uniform vec3 I_p; // point
uniform vec3 I_s; //spot

//light position
uniform vec3 lightPos_d;
uniform vec3 lightPos_p;
uniform vec3 lightPos_s;
uniform int cur_light_id; //represent type of lights

uniform float spot_cutoff;
uniform mat4 view_matrix;
uniform vec3 cameraPos;
uniform int per_vertex;
```

```
vec3 directional_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 L = normalize(lightPos_d); //normalized light source direction
    vec3 R = reflect(-L, N); //normalized light source reflection vector
    vec3 V = normalize(cameraPos-FragPos); //normalized viewpoint direction vector

    vec3 ambient = Ia * Ka;
    vec3 diffuse = I_d * Kd * max(dot(N,L),0);
    vec3 specular = Ks * pow(max(dot(R,V),0), shininess);

    vec3 color = ambient + diffuse + specular;
    return color;
}

vec3 point_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 V = normalize(cameraPos-FragPos);//normalized viewpoint direction vector
    vec3 L = normalize(lightPos_p - FragPos);
    vec3 H = normalize(L+V); //halfway vector


    vec3 ambient = Ia * Ka;
    vec3 diffuse = I_p * Kd * max(dot(N,L),0);
    vec3 specular = Ks * pow(max(dot(N,H),0), shininess);

    //calculate attenuation with respect to the distance between the light source and the object
    float d = length(lightPos_p - FragPos);
    float f_att = min(1/(0.01 + 0.8 * d + 0.1 * d * d),1);

    vec3 color = ambient + f_att * diffuse + f_att * specular;
    return color;
}
```

```
vec3 spot_light()
{
    vec3 N = normalize(vertex_normal); //normalized normal vector
    vec3 V = normalize(cameraPos-FragPos);//normalized viewpoint direction vector
    vec3 L = normalize(lightPos_s - FragPos);
    vec3 H = normalize(L+V);  //halfway vector


    //calculate attenuation with respect to the distance between the light source and the object
    float d = length(lightPos_s - FragPos);
    float f_att = min(1/(0.05 + 0.3 * d + 0.6 * d * d),1);

    //spotlight effect
    vec4 spot_direction = vec4(0.0f,0.0f,-1.0f,1.0f);
    vec3 direction = normalize((transpose(inverse(view_matrix)) * spot_direction).xyz);
    float arc = dot(-L, direction);
    if(spot_cutoff<=degrees(acos(arc)))
    {
        vec3 ambient = Ia * Ka;
        vec3 color = ambient;
        return color;
    }
    else
    {
        float spot_effect =  pow(max(arc,0),50);
        vec3 ambient = Ia * Ka;
        vec3 diffuse = I_s * Kd * max(dot(N,L),0);
        vec3 specular = Ks * pow(max(dot(N,H),0), shininess);
        vec3 color = ambient + f_att * spot_effect * diffuse + f_att * spot_effect * specular;
        return color;
    }

}
```

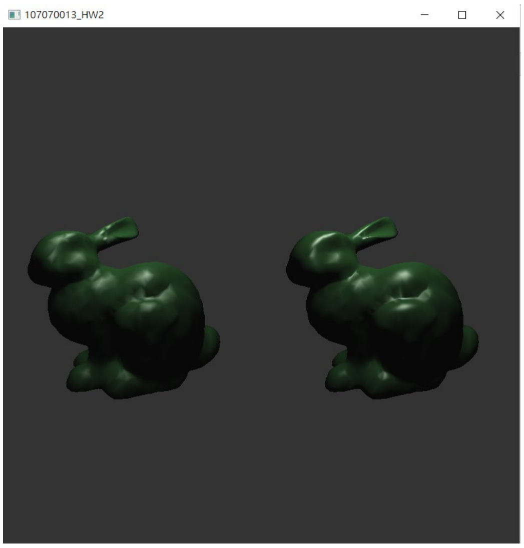
2. 不同的地方在於 main()函數，當傳遞的 per-vertex 為 1 時，FragColor 才會需要依據不同的 lighting 種類進行計算並更改。若傳遞的 per-vertex 為 0 時，FragColor 值會和 vertex color 的值相同。

```
void main() {
    // [TODO]
    //FragColor = vec4(vertex_normal, 1.0f);
    if(per_vertex == 1)
    {
        vec3 color = vec3(0,0,0);
        if(cur_light_id == 0)
        {
            //color += point_light();
            color += directional_light();
        }
        else if(cur_light_id == 1)
        {
            color += point_light();
        }
        else if(cur_light_id == 2)
        {
            color += spot_light();
        }
        FragColor = vec4 (color ,1.0);
    }
    else
    {
        FragColor = vec4 (vertex_color ,1.0);
    }
}
```

下圖為三種 lighting 初始值的 demo 圖

1. Directional light

2. Point light



3. Spot light