ICG HW3 report

黃芷柔 110550142

main.cpp:

```
if (shader == 1) {
    currentProgram = BlinnPhongProgram;
}
else if (shader == 2) {
    currentProgram = GouraudProgram;
}
else if (shader == 3) {
    currentProgram = FlatProgram;
}
else if (shader == 4) {
    currentProgram = ToonProgram;
}
else if (shader == 5) {
    currentProgram = BorderProgram;
}
glUseProgram(currentProgram);
```

用 createShader 跟 createProgram 做出 5 種 Program，變數 shader 會隨著鍵盤的輸入改變，根據 shader 改變要使用的 program

```
unsigned int mLoc, vLoc, pLoc, camera_Loc;
mLoc = glGetUniformLocation(currentProgram, "M");
vLoc = glGetUniformLocation(currentProgram, "V");
pLoc = glGetUniformLocation(currentProgram, "P");
glUniformMatrix4fv(mLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(vLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(pLoc, 1, GL_FALSE, glm::value_ptr(perspective));

camera_Loc = glGetUniformLocation(currentProgram, "camera");
glUniform3fv(camera_Loc, 1, &cameraPos[0]);
```

用 glGetUniformLocation 找出 M, V, P, camera 的位置，再用 glUniformMatrix4fv 或 glUniform3fv 把相對的變數傳過去。

1. Blinn-Phong Shading:

Vertex shader:

```
gl_Position = P * V * M * vec4(aPos, 1.0);
texCoord = aTexCoord;
worldPos = M * vec4(aPos, 1.0);
mat4 normal_trans = transpose(inverse(M));
normal = normalize((normal_trans * vec4(aNormal, 0.0)).xyz);
view_pos = camera;
vec3 viewDir = normalize(view_pos);
```

Fragment shader:

```
vec3 Ka = vec3(1.0, 1.0, 1.0);
vec3 Kd = vec3(1.0, 1.0, 1.0);
vec3 Ks = vec3(0.7, 0.7, 0.7);
float a = 10.5;
vec3 La = vec3(0.2, 0.2, 0.2);
vec3 Ld = vec3(0.8, 0.8, 0.8);
vec3 Ls = vec3(0.5, 0.5, 0.5);
vec3 lightPos = vec3(10, 10, 10);

vec3 obj_color = vec3(texture(ourTexture, texCoord));

vec3 ambient = La * Ka;

vec3 N = normalize(normal);
vec3 L = (normalize(lightPos - worldPos.xyz));
vec3 diffuse = Ld * Kd  * max(0.0, dot(L, N));
```

```
vec3 V = normalize(view_pos - vec3(worldPos));
vec3 H = normalize(L + V);
float spec = pow(max(0.0, dot(N,H)), a);
vec3 specular = Ls * Ks * spec;

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

在 vertex shader 計算 normal 位置，然後把他們跟 texture, view position 一起傳給 fragment shader。

在 fragment shader 計算 ambient, diffuse 跟 specular

Ambient: La*Ka

Diffuse: 先取 N 跟 L，N 是把 vertex shader 傳來的 normal normalize，L 是 light position 減掉 worldPos。取 L, N 的內積，並用 max 取內積結果大於 0 的，再乘上系數。

Specular: 先取 V 跟 H，V 是 view position 減掉 worldPos，H 是 L+V 做 normalize。取 N,H 內積，並用 max 取內積結果大於 0 的，再做 a 次方，最後乘上系數，得到 specular。

最後把 Ambient, Diffuse, Specular 加在一起，乘以 texture 的顏色，得到物體最後的顏色。

2. Gouraud Shading

Vertex shader:

```
gl_Position = P * V * M * vec4(aPos, 1.0);
texCoord = aTexCoord;
worldPos = M * vec4(aPos, 1.0);
mat4 normal_transform = transpose(inverse(M));
normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
vec3 view_pos = camera;

vec3 lightPos = vec3(10, 10, 10);
vec3 Ka = vec3(1.0, 1.0, 1.0);
vec3 Kd = vec3(1.0, 1.0, 1.0);
vec3 Ks = vec3(0.7, 0.7, 0.7);
vec3 La = vec3(0.2, 0.2, 0.2);
vec3 Ld = vec3(0.8, 0.8, 0.8);
vec3 Ls = vec3(0.5, 0.5, 0.5);
float a = 10.5;
```

```
ambient = La * Ka;

vec3 N = normalize(normal);
vec3 L = (normalize(vec4(lightPos, 1.0) - worldPos)).xyz;
diffuse = Ld * Kd* max(0.0, dot(L, N));

vec3 V = normalize(view_pos - vec3(worldPos));
vec3 R = normalize(reflect(-lightPos, normal));
specular = Ls * Ks * pow(max(0.0, dot(V, R)), a);
```

Fragment Shader:

```
vec3 obj_color = vec3(texture(ourTexture, texCoord));

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

在 vertex shader 裡計算 ambient, diffuse, specular，ambient 跟 diffuse 的算法和 Phong Shading 一樣。算 Specular 要先找 V 跟 R，V 是 view position-world position，R 是用 reflect(-lightPos, normal)取得，接著 dot(V,R)，並用 max 取大於 0 的結果，再做 a 次方，乘上系數，得到 specular
傳給 fragment shader。
在 fragment shader 把收到的 ambient, diffuse, specular 相加，乘上 texture 的顏色，得到最後的顏色。

3. Flat Shading

Vertex shader

```
gl_Position = P * V * M * vec4(aPos, 1.0);
vs_out.texCoord = aTexCoord;
vs_out.worldPos = M * vec4(aPos, 1.0);
mat4 normal_transform = transpose(inverse(M));
vs_out.normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
```

Geometry shader

```
vec3 edge1 = gs_in[1].worldPos.xyz -gs_in[0].worldPos.xyz;
vec3 edge2 = gs_in[2].worldPos.xyz - gs_in[0].worldPos.xyz;
vec3 normal = normalize(cross(edge1, edge2));

for(int i = 0; i < 3; ++i)
{
    fragTexCoord = gs_in[i].texCoord;
    fragWorldPos = gs_in[i].worldPos;
    fragNormal =  normal;
    gl_Position = gl_in[i].gl_Position;
    EmitVertex();
}
EndPrimitive();
```

Fragment shader

```
vec3 Ka = vec3(1.0, 1.0, 1.0);
vec3 Kd = vec3(1.0, 1.0, 1.0);
vec3 Ks = vec3(0.7, 0.7, 0.7);
float a = 10.5;
vec3 La = vec3(0.2, 0.2, 0.2);
vec3 Ld = vec3(0.8, 0.8, 0.8);
vec3 Ls = vec3(0.5, 0.5, 0.5);
vec3 lightPos = vec3(10, 10, 10);


vec3 obj_color = vec3(texture(ourTexture, fragTexCoord));

vec3 ambient = La * Ka; // Ambient color


vec3 N = normalize(fragNormal);
vec3 L = normalize(lightPos - fragWorldPos.xyz);
```

```
float diffuseIntensity = max(0.0, dot(L, N));
vec3 diffuse = Ld * Kd * diffuseIntensity;

vec3 V = normalize(camera-fragWorldPos.xyz);
vec3 R = reflect(-L, N);
float spec = pow(max(0.0, dot(V, R)), a);
vec3 specular = vec3(0.5, 0.5, 0.5) * spec; // Specular color


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

在 vertex shader 裡計算位置跟 normal，並且把 texture 座標一定傳給 geometry shader。

在 geometry shader，接收三角形輸入，並把 max_vertices 設為 3，把從 vertex shader 傳來的 texCord, world position,normal 定義成一個 array。用 array 中的三個點得到兩個編，再取兩邊外積得到 polygon 的 normal，進到 for 迴圈處理

三個點，傳給 fragment shader polygon 的 normal，以及點本身的 worldPos, texCord, gl_Position。

在 fragment shader，用和 Gouraud shading 一樣的方式計算 ambient, diffuse 跟 specular，家在一起成以 texture 顏色得到最後輸出顏色。

4. Toon Shading:

Vertex Shader

```
gl_Position = P * V * M * vec4(aPos, 1.0);
texCoord = aTexCoord;
worldPos = M * vec4(aPos, 1.0);
mat4 normal_trans = transpose(inverse(M));
normal = normalize((normal_trans * vec4(aNormal, 0.0)).xyz);
```

Fragment Shader:

```
vec3 obj_color = vec3(texture(ourTexture, texCoord));
vec4 color;
vec3 lightPos = vec3(10, 10, 10);
vec3 N = normalize(normal);
vec3 L = (normalize(vec4(lightPos, 1.0) - worldPos)).xyz;
float intensity = max(dot(L, N), 0.0);


if(intensity > 0.95) color = vec4(1.0, 0.8, 0.8, 1.0);
else if(intensity > 0.5) color = vec4(0.45, 0.3, 0.3, 1.0);
else if(intensity > 0.25) color = vec4(0.3, 0.2, 0.2, 1.0);
else color = vec4(0.15, 0.1, 0.1, 1.0);

FragColor = color;
```

在 vertex shader 裡計算位置跟 normal，並且把 texture 座標一定傳給 fragment shader。

在 Fragment shader，用 normalization 後的 normal, light 方向內積當作 intensity，也就是 diffuse 乘上係數前的值，intensity 界在 0~1 之間，將 intensity 分成四個等級，越大的顏色越亮。

5. Border effect

Vertex Shading:

```
gl_Position = P * V * M * vec4(aPos, 1.0);
texCoord = aTexCoord;
worldPos = M * vec4(aPos, 1.0);
mat4 normal_transform = transpose(inverse(M));
normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
view_pos = camera;
```

Fragment shading:

```
vec3 obj_color = vec3(texture(ourTexture, texCoord));
vec3 n_normal = normalize(normal);
float facing = dot(normalize(view_pos), n_normal);
vec4 color = vec4(obj_color, 1.0);
FragColor = mix(color, vec4(1.0, 1.0, 1.0, 1.0), 1.0-facing);
```

在 vertex shader 裡計算位置跟 normal，並且把 texture 座標一定傳給 fragment shader。

在 fragment shader，將 view 跟 normal 座內積，得到的值設為 facing，界在 0~1，最後用 mix 讓原本的顏色和白色混和，facing 越小，越接近邊緣，白色的比例越大。