110612117 張仲瑜 HW3 report

整體流程簡介：

1. 建立並編譯 shaders，再將兩者連接為著色器

   這次總共有 6 種不同的 shading

2. 載入模型紋理並綁定(助教已完成)

3. 設定 VAO 與 VBO 供頂點綁定(助教已完成)

4. 讀取 uniform 變數位置

5. 開始根據初始化變數製圖並更新 uniform 參數

本次說明主要著重在 shader 撰寫方式

```
32  unsigned int BlinnPhongProgram, GouraudProgram, FlatProgram, ToonProgram, BorderProgram, DissolveProgram,
33               CurrentProgram;
34  float dissolveFactor = -25;
76      unsigned int BlinnPhongVertexShader, GouraudVertexShader, FlatVertexShader, ToonVertexShader,
77                   BorderVertexShader, DissolveVertexShader;
78      unsigned int BlinnPhongFragmentShader, GouraudFragmentShader, FlatFragmentShader, ToonFragmentShader,
79                   BorderFragmentShader, DissolveFragmentShader;
80      unsigned int FlatGeometryShader;
81
82      BlinnPhongVertexShader = createShader("shaders/Blinn-Phong.vert", "vert");
83      BlinnPhongFragmentShader = createShader("shaders/Blinn-Phong.frag", "frag");
84      BlinnPhongProgram = createProgram(BlinnPhongVertexShader, 0, BlinnPhongFragmentShader);
85
86      GouraudVertexShader = createShader("shaders/Gouraud.vert", "vert");
87      GouraudFragmentShader = createShader("shaders/Gouraud.frag", "frag");
88      GouraudProgram = createProgram(GouraudVertexShader, 0, GouraudFragmentShader);
89
90      FlatVertexShader = createShader("shaders/Flat.vert", "vert");
91      FlatGeometryShader = createShader("shaders/Flat.geom", "geom");
92      FlatFragmentShader = createShader("shaders/Flat.frag", "frag");
93      FlatProgram = createProgram(FlatVertexShader, FlatGeometryShader, FlatFragmentShader);
94
95      ToonVertexShader = createShader("shaders/Toon.vert", "vert");
96      ToonFragmentShader = createShader("shaders/Toon.frag", "frag");
97      ToonProgram = createProgram(ToonVertexShader, 0, ToonFragmentShader);
98
99      BorderVertexShader = createShader("shaders/Border.vert", "vert");
100     BorderFragmentShader = createShader("shaders/Border.frag", "frag");
101     BorderProgram = createProgram(BorderVertexShader, 0, BorderFragmentShader);
102
103     DissolveVertexShader = createShader("shaders/Dissolve.vert", "vert");
104     DissolveFragmentShader = createShader("shaders/Dissolve.frag", "frag");
105     DissolveProgram = createProgram(DissolveVertexShader, 0, DissolveFragmentShader);
106
107     CurrentProgram = BlinnPhongProgram;
108     glUseProgram(CurrentProgram);
```

(圖零). 建立六種 shading 需要的 shader/program

Line 32

宣告 6 種 shader program

Line 33

宣告 dissolve shader program 需要用到的消失

起始點

Line 76~80

各 vertex/geometry/fragment shader 宣告

Line 82~105

依照路徑建立各 shader 後

以 vertex/geometry/fragment shader 的參數順序

組合成 program

因為只有 flat shading 有 geometry shader

其餘 shading 在該參數需填 0

```
glUniformMatrix4fv(glGetUniformLocation(CurrentProgram, "M"), 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(glGetUniformLocation(CurrentProgram, "V"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(CurrentProgram, "P"), 1, GL_FALSE, glm::value_ptr(perspective));
glUniform3fv(glGetUniformLocation(CurrentProgram, "material.ambient"), 1, glm::value_ptr(material.ambient));
glUniform3fv(glGetUniformLocation(CurrentProgram, "material.diffuse"), 1, glm::value_ptr(material.diffuse));
glUniform3fv(glGetUniformLocation(CurrentProgram, "material.specular"), 1, glm::value_ptr(material.specular));
glUniform1f(glGetUniformLocation(CurrentProgram, "material.shininess"), material.shininess);
glUniform1f(glGetUniformLocation(CurrentProgram, "dissolveFactor"), dissolveFactor);
glUniform3fv(glGetUniformLocation(CurrentProgram, "light.ambient"), 1, glm::value_ptr(light.ambient));
glUniform3fv(glGetUniformLocation(CurrentProgram, "light.diffuse"), 1, glm::value_ptr(light.diffuse));
glUniform3fv(glGetUniformLocation(CurrentProgram, "light.specular"), 1, glm::value_ptr(light.specular));
glUniform3fv(glGetUniformLocation(CurrentProgram, "light.position"), 1, glm::value_ptr(light.position));
glUniform3fv(glGetUniformLocation(CurrentProgram, "cameraPos"), 1, glm::value_ptr(cameraPos));
```

```
void loadMaterialLight() {
    material.ambient = glm::vec3(1.0, 1.0, 1.0);
    material.diffuse = glm::vec3(1.0, 1.0, 1.0);
    material.specular = glm::vec3(0.7, 0.7, 0.7);
    material.shininess = 10.5;

    light.ambient = glm::vec3(0.2, 0.2, 0.2);
    light.diffuse = glm::vec3(0.8, 0.8, 0.8);
    light.specular = glm::vec3(0.5, 0.5, 0.5);
    light.position = glm::vec3(10, 10, 10);
}
```

(圖一、二).

將需要用到的 uniform 變數傳入 shader 中

Line150~152

其中 view、model、perspective matrix 傳入 vertex

shader 中的 M、V、P uniform 變數中

Line153~162

觀察助教提供的程式碼架構可以發現

物體與光源為類別的一種，裡面存有其對應的係數

我們可以在 fragment shader 中建立對應的 uniform 結構

如此便可以將整個類別物件搬過去取用他們的參數

```cpp
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (key == GLFW_KEY_SPACE && action == GLFW_PRESS) {
        rotating = !rotating;
        cout << "KEY SPACE PRESSED\n";
    }
    else if (key == GLFW_KEY_1 && action == GLFW_PRESS) {
        CurrentProgram = BlinnPhongProgram;
        glUseProgram(CurrentProgram);
        cout << "KEY 1 PRESSED\n";
    }
    else if (key == GLFW_KEY_2 && action == GLFW_PRESS) {
        CurrentProgram = GouraudProgram;
        glUseProgram(CurrentProgram);
        cout << "KEY 2 PRESSED\n";
    }
    else if (key == GLFW_KEY_3 && action == GLFW_PRESS) {
        CurrentProgram = FlatProgram;
        glUseProgram(CurrentProgram);
        cout << "KEY 3 PRESSED\n";
    }
    else if (key == GLFW_KEY_4 && action == GLFW_PRESS) {
        CurrentProgram = ToonProgram;
        glUseProgram(CurrentProgram);
        cout << "KEY 4 PRESSED\n";
    }
    else if (key == GLFW_KEY_5 && action == GLFW_PRESS) {
        CurrentProgram = BorderProgram;
        glUseProgram(CurrentProgram);
        cout << "KEY 5 PRESSED\n";
    }
    else if (key == GLFW_KEY_6 && action == GLFW_PRESS) {
        CurrentProgram = DissolveProgram;
        dissolveFactor = -25;
        glUseProgram(CurrentProgram);
        cout << "KEY 6 PRESSED\n";
    }
}
```

(圖三). 按下對應按鍵後切換執行中的 shading program

dissolve effect 在按下時須重製 dissolve 起始點

接下來介紹 6 種 shading 的 shader 內容

```
1  #version 330 core
2
3  // TODO:
4  // implement Blinn-Phong shading
5
6  layout (location = 0) in vec3 aPos;
7  layout (location = 1) in vec3 aNormal;
8  layout (location = 2) in vec2 aTexCoord;
9
10 uniform mat4 M;
11 uniform mat4 V;
12 uniform mat4 P;
13
14 out vec2 texCoord;
15 out vec4 worldPos;
16 out vec3 normal;
17
18 void main()
19 {
20     gl_Position = P * V * M * vec4(aPos, 1.0);
21     texCoord = aTexCoord;
22     worldPos = M * vec4(aPos, 1.0);
23     mat4 normal_transform = transpose(inverse(M));
24     normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
25 }
```

(圖四). Blinn–Phong shading vertex shader

Line 6~8

建立 VertexAttribute 的索引與儲存內容作為輸入

一個 vertex 有頂點位置、法線與紋理座標三個數據

Line 10~12

宣告三個 uniform matrix 儲存 main 傳入的座標轉換矩陣

Line 14~16

宣告輸出紋理座標、世界座標、法向量

Line 20

將原始座標乘上 MVP 轉換存在頂點裝置座標的內建變數中

Line 21~24

將紋理座標、世界座標、法向量計算後傳至 fragment

shader 供後續計算光線使用

```
6   struct Material{
7       vec3 ambient;
8       vec3 diffuse;
9       vec3 specular;
10      float shininess;
11  };
12
13  struct Light{
14      vec3 ambient;
15      vec3 diffuse;
16      vec3 specular;
17      vec3 position;
18  };
19
20  in vec2 texCoord;
21  in vec4 worldPos;
22  in vec3 normal;
23
24  uniform sampler2D deerTexture;
25  uniform Material material;
26  uniform Light light;
27  uniform vec3 cameraPos;
28
29  out vec4 fragColor;
30
31  void main()
```

(圖五). Blinn-Phong shading fragment shader

Line 6~18 建立物件與光源類別以便存取各樣係數

Line 20~22 承接 vertex shader 的輸出

Line 24~27 建立接收 main 傳入參數的 uniform 變數

Line 29 宣告最終要輸出的顏色變數

```
33        vec4 objectColor = texture2D(deerTexture, texCoord);
34        vec3 L = normalize(light.position - worldPos.xyz);
35        vec3 V = normalize(cameraPos - worldPos.xyz);
36        vec3 H = normalize(L + V);
37        vec3 N = normalize(normal);
38
39        vec4 Ka = vec4(material.ambient, 1.0f);
40        vec4 Kd = vec4(material.diffuse, 1.0f);
41        vec4 Ks = vec4(material.specular, 1.0f);
42
43        vec4 La = vec4(light.ambient, 1.0f);
44        vec4 Ld = vec4(light.diffuse, 1.0f);
45        vec4 Ls = vec4(light.specular, 1.0f);
46
47        vec4 OutPutAmbient = La * Ka * objectColor;
48        vec4 OutPutDiffuse = Ld * Kd * objectColor * max(dot(L, N), 0.0);
49        vec4 OutPutSpecular = Ls * Ks  * pow(max(dot(N, H), 0.0), material.shininess);
50
51        fragColor = OutPutAmbient + OutPutDiffuse + OutPutSpecular;
```

(圖六). Blinn-Phong shading fragment shader (cont.)

Line 33 從鹿的紋理取得對應座標的紋理顏色

Line 34~37 計算與正規化 Blinn-Phong 所需的光源、視角、半角、法 向量

Line 39~45 存取各項係數

Line 47~49

根據公式將紋理顏色乘上係數以及其對應向量以計算

環境光、漫反射光、鏡反射光產生的顏色

(參考助教回覆這邊採取跟 demo 一樣的寫法，specular 不乘原本的 object color)

Line 51

三樣加總即為最終顏色

```
1   #version 330 core
2
3   // TODO:
4   // Implement Gouraud shading
5   layout (location = 0) in vec3 aPos;
6   layout (location = 1) in vec3 aNormal;
7   layout (location = 2) in vec2 aTexCoord;
8
9   struct Material{
10      vec3 ambient;
11      vec3 diffuse;
12      vec3 specular;
13      float shininess;
14  };
15
16  struct Light{
17      vec3 ambient;
18      vec3 diffuse;
19      vec3 specular;
20      vec3 position;
21  };
22
23  uniform mat4 M;
24  uniform mat4 V;
25  uniform mat4 P;
26  uniform sampler2D deerTexture;
27  uniform Material material;
28  uniform Light light;
29  uniform vec3 cameraPos;
30
```

(圖七). Gouraud shading vertex shader

與 Blinn–Phong 的差別在於

Blinn–Phong 是法向量內差後

針對每個 Potential Pixel 計算光線與顏色

Gouraud 則是針對頂點算好顏色後

再利用頂點顏色對三角形做內差

因此在實作上就是將

紋理顏色乘上內差後光線加總的部分

保留在 fragment shader

其餘搬到 vertex shader

```
#version 330 core

// TODO:
// Implement Gouraud shading
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoord;

struct Material{
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

struct Light{
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    vec3 position;
};

uniform mat4 M;
uniform mat4 V;
uniform mat4 P;
uniform sampler2D deerTexture;
uniform Material material;
uniform Light light;
uniform vec3 cameraPos;
```

```
out vec2 texCoord;
out vec4 ambient;
out vec4 diffuse;
out vec4 specular;

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

    vec4 objectColor = texture2D(deerTexture, texCoord);
    vec3 L = normalize(light.position - worldPos.xyz);
    vec3 V = normalize(cameraPos - worldPos.xyz);
    vec3 R = normalize(reflect(-L, normal));
    vec3 N = normalize(normal);

    vec4 Ka = vec4(material.ambient, 1.0f);
    vec4 Kd = vec4(material.diffuse, 1.0f);
    vec4 Ks = vec4(material.specular, 1.0f);

    vec4 La = vec4(light.ambient, 1.0f);
    vec4 Ld = vec4(light.diffuse, 1.0f);
    vec4 Ls = vec4(light.specular, 1.0f);

     ambient = La * Ka;
     diffuse = Ld * Kd * max(dot(L, N), 0.0);
     specular = Ls * Ks * pow(max(dot(V, R), 0.0), material.shininess);

 }
```

(圖八、九). Gouraud shading vertex shader

因為實作大致都與 Blinn–Phong 出現過的相同

只是先計算完顏色才根據頂點針對三角形做內差

這裡的不同處還有使用 Phong 而非 Blinn–Phong

可以看到反射光 R 向量

計算方法為入射光 L 在法線對側的映射向量

```
#version 330 core

// TODO:
// Implement Gouraud shading

in vec2 texCoord;
in vec4 ambient;
in vec4 diffuse;
in vec4 specular;

uniform sampler2D deerTexture;

out vec4 fragColor;

void main()
{
    vec4 objectColor = texture2D(deerTexture, texCoord);
    fragColor = ambient * objectColor + diffuse * objectColor + specular;
}
```

(圖十). Gouraud shading fragment shader

取得內差後的三種光線來源後

乘上紋理顏色並加總即為最後顏色

```
#version 330 core

// TODO:
// Implement Flat shading

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoord;

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

out vec2 texCoord;
out vec4 worldPos;
out vec3 normal;

void main()
{
    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);
}
```

(圖十一). flat shading vertex shader

與前兩者的差別在於

Flat shading 每個三角形只有一個法向量

就呈現那一種顏色

計算複雜度低上許多

但精細度也相對下降

vertex shader 部分與 Phong 相同

不再贅述，主要針對 geometry shader 進行說明

```
#version 330 core

// TODO:
// Implement Flat shading
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;
in vec2 texCoord[];
in vec4 worldPos[];
in vec3 normal[];

out vec2 FragTexCoord;
out vec4 FragWorldPos;
out vec3 FragNormal;


void main(void)
{
    vec3 flatNormal = normal[0] + normal[1] + normal[2] / 3;
    for(int i = 0; i < 3; i++){
        gl_Position = gl_in[i].gl_Position;
        FragTexCoord = texCoord[i];
        FragWorldPos = worldPos[i];
        FragNormal = flatNormal;
        EmitVertex();
    }
    EndPrimitive();
}
```

(圖十二). flat shading geometry shader

Line 5~6

定義輸入與輸出

輸入為三角形

輸出為三角形帶、三個頂點為一組串起來

Line 7~9

讀入三個頂點的紋理座標、世界座標與法向量

Line 18

三角形的法向量為頂點法向量的平均

Line 20~23

利用迴圈依序將三個頂點的位置、世界座標、平均後的法

向量存起來，以便繼續傳給後面的 fragment shader

Line 24

每完成一個頂點就發射一次

Line 25

完成個三個頂點即結束一個圖元

```
#version 330 core

// TODO:
// Implement Flat shading

struct Material{
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

struct Light{
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    vec3 position;
};

in vec2 FragTexCoord;
in vec4 FragWorldPos;
in vec3 FragNormal;

uniform sampler2D deerTexture;
uniform Material material;
uniform Light light;
uniform vec3 cameraPos;

out vec4 fragColor;
```

```
void main()
{
    vec4 objectColor = texture2D(deerTexture, FragTexCoord);
    vec3 L = normalize(light.position - FragWorldPos.xyz);
    vec3 V = normalize(cameraPos - FragWorldPos.xyz);
    vec3 R = normalize(reflect(-L, FragNormal));
    vec3 N = normalize(FragNormal);

    vec4 Ka = vec4(material.ambient, 1.0f);
    vec4 Kd = vec4(material.diffuse, 1.0f);
    vec4 Ks = vec4(material.specular, 1.0f);

    vec4 La = vec4(light.ambient, 1.0f);
    vec4 Ld = vec4(light.diffuse, 1.0f);
    vec4 Ls = vec4(light.specular, 1.0f);

    vec4 OutPutAmbient = La * Ka * objectColor;
    vec4 OutPutDiffuse = Ld * Kd * objectColor * max(dot(L, N), 0.0);
    vec4 OutPutSpecular = Ls * Ks * pow(max(dot(V, R), 0.0), material.shininess);

    fragColor = OutPutAmbient + OutPutDiffuse + OutPutSpecular;
}
```

(圖十三、四). flat shading fragment shader

大致與 Blinn–Phong 的 fragment shader 相同

只是改省用標準的 Phong 以及三角形統一的法向量

此處就不再贅述

```
#version 330 core

// TODO:
// Implement Toon shading
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoord;

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

out vec2 texCoord;
out vec4 worldPos;
out vec3 normal;

void main()
{
    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);
}
```

(圖十四). Toon shading vertex shader

與 Blinn–Phong 的 vertex shader 完全相同

主要差異在 fragment shader

```
37        float darkColorTreshhold = 0.5,
38              lightColorTreshhold = 0.12;
39        vec4 darkColor = vec4(0.173, 0.0, 0.0, 1.0),
40             lightColor = vec4(1.0, 0.859 , 0.753, 1.0);
49        vec4 OutPutSpecular = Ls * Ks * pow(max(dot(V, R), 0.0), material.shininess);
50        if(dot(N, L) < darkColorTreshhold){
51            fragColor = darkColor;
52        }
53        else if(OutPutSpecular.x > lightColorTreshhold ||
54                OutPutSpecular.y > lightColorTreshhold ||
55                OutPutSpecular.z > lightColorTreshhold){
56            fragColor = lightColor;
57        }
58        else{
59            fragColor = vec4(0.514, 0.314, 0.247, 1.0);
60        }
```

(圖十五、六). Toon shading fragment shader (critical)

Line 37~40

設定高低閾值與對應顏色

總共有三種深淺咖啡色

Line 50~60

如果 N dot L 小於深色閾值即將顏色設為深色

如果 specular 強度大於淺色閾值則將顏色設為淡色

否則就設成中間色

參數皆以仿造 demo 來調整

```
        if(abs(dot(V, N)) < edgeColorTreshhold){
            fragColor = edgeColor;
        }
        else{
            fragColor = objectColor;
        }
```

(圖十七). Border Effect fragment shader (critical)

Vertex shader 與 Blinn-Phong、 Toon 的完全相同(略)

Border effect 的核心即為在 fragment shader 中

如果 V dot N 的絕對值小於一定值，即將其設為 edge color(白色)

這裡鹿本體仿照 demo 不乘上額外參數打光

```
#version 330 core

// Advanced:
// Implement Dissolve effect

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoord;

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

out vec2 texCoord;
out vec4 worldPos;
out vec3 normal;
out float horizontalPosition;
void main()
{
    gl_Position = P * V * M * vec4(aPos, 1.0);
    texCoord = aTexCoord;
    horizontalPosition = aPos.x;
    worldPos = M * vec4(aPos, 1.0);
    mat4 normal_transform = transpose(inverse(M));
    normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
}
```

(圖十八). Dissolve Effect vertex shader

Line 17、22

額外取出物體的水平座標送到 fragment shader

等等才能用它來完成水平消失

```
20  in vec2 texCoord;
21  in vec4 worldPos;
22  in vec3 normal;
23  in float horizontalPosition;
24
25  uniform sampler2D deerTexture;
26  uniform float dissolveFactor;
27  uniform Material material;
28  uniform Light light;
29  uniform vec3 cameraPos;
30
31  out vec4 fragColor;
32
33  void main()
34  {
35      vec4 objectColor = texture2D(deerTexture, texCoord);
36      vec3 color = objectColor.rgb;
37      if(horizontalPosition < dissolveFactor){
38          discard;
39      }
40      else{
41          fragColor = objectColor;
42      }
43
44  }
45
```

(圖十九). Dissolve Effect fragment shader (critical)

Line 26

需額外讀取當前的 dissolveFactor (水平消失基準線)

Line 37~42

如果當前的水平座標小於水平消失基準線

則直接丟棄

否則就正常輸出

這裡一樣仿照 demo 不對鹿的本體做任何打光

```
172         // Status update
173         currentTime = glfwGetTime();
174         dt = currentTime - lastTime;
175         lastTime = currentTime;
176         if (rotating)   angle += glm::radians(45.0f) * dt;
177         if (angle > glm::radians(360.0f))   angle -= glm::radians(360.0f);
178
179         if (dissolveFactor < 25) {
180             dissolveFactor += 10.0 * dt;
181         }
182         //cout << dissolveFactor << endl;
183         glfwSwapBuffers(window);
184         glfwPollEvents();
185     }
```

(圖二十). 在 main 裡更新 dissolveFactor

Line 179~181

dissolveFactor 隨時間增加

來達成由左往右消失的水平基準線

過程遇到的問題/心得

在設計 dissolve effect 時遇到不少的問題

一開始嘗試以 gl_Position 作為判定的依據

但效果卻不如預期

後來幾經測試才發現 aPos 能做出與 demo 相同的效果。

可以發現這次的六種 shading 架構其實差不多

因此只要設計好第一個之後

再針對各自的特點去做微幅修改就可以順利完成

我花費在第一個 shading 的時間也是最多的

也深刻體會到好的架構的重要性

能夠節省許多精力完成不同但相似的設計

算是一舉多得