ICG HW2 Report

0716081 葉晨

- How to use GLSL

1. Create shader and program object

```
GLuint vert = createShader("Shaders/vertexShader.vert", "vertex");
GLuint frag = createShader("Shaders/fragmentShader.frag", "fragment");
program = createProgram(vert, frag);
```

   - createShader()

     用於創建 vertex shader 及 fragment shader 的物件。

     其中第一個參數傳入檔案的位址，第二個參數傳入 shader 的類型 (e.g. vertex, fragment, ...)。

   - createProgram()

     用於創建 program object，並將剛剛創建的兩個 shader object link 在一起。

2. Create VAO

```
// VAO
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);
```

   - glGenVertexArrays()

     用於創建 VAO object。

     第一個參數代表要創建的數量，第二個參數用來指定 VAO 變數的名稱

   - glBindVertexArray()

     用來 bind 剛剛創建出來的 VAO object。

3. Create VBO & link to shader

```
// vertex position
GLuint vertex_vbo;
glGenBuffers(1, &vertex_vbo);
glBindBuffer(GL_ARRAY_BUFFER, vertex_vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), model->positions.data(), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
```

```
// vertex normal
GLuint normal_vbo;
glGenBuffers(1, &normal_vbo);
glBindBuffer(GL_ARRAY_BUFFER, normal_vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(1);
```

```
// vertex texture coordinate
GLuint tex_vbo;
glGenBuffers(1, &tex_vbo);
glBindBuffer(GL_ARRAY_BUFFER, tex_vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), model->texcoords.data(), GL_STATIC_DRAW);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(2);
```

- glGenBuffers()

  用於創建 VBO 物件。

  第一個參數傳入要創建的數量，第二個參數用來指定用來儲存 VBO 物件的變數名稱 (e.g. 本次作業中，一個 vertex 有三種屬性：position, normal, texture coordinate，所以創建三個 VBO 去儲存這些屬性)

- glBindBuffer()

  用於 bind 剛剛創建的 VBO object。

  第一個參數傳入 buffer 類型 (e.g.此處為 GL_ARRAY_BUFFER)，第二個參數傳入要 bind 的 VBO object name。

- glBufferData()

  用於將資料傳入 VBO object。

  第一個參數傳入 buffer 類型，第二個參數傳入所有資料的大小 (bytes)，第三個參數傳入資料本身，第四個參數傳入 buffer 用法 (e.g. 此處為 GL_STATIC_DRAW，代表 buffer 初始化之後不會被修改)。

  此處的三個 attribute 的 type 皆為 float，且 model 的資料內容皆使用 vector 類型儲存，因此可以利用 size()取得資料大小、利用 data()取得資料內容。

- glVertexAttributePointers()

  用於連結 buffer 和 shader input。

  第一個參數指定該 buffer 的 index

  第二個參數傳入對於每個 vertex，該屬性擁有幾個值 (e.g. 像是 position & normal 皆是三個值一組，而 vertex coordinate 則是兩個值一組)

  第三個參數傳入 buffer 內資料的類型

  第四個參數傳入是否要 normalize

  第五個參數傳入 stride (byte)

第六個參數傳入第一個需要的 attribute 距離 array 起點的 offset，若一個 buffer 裡面存有不同的 attribute，就會需要指定此參數 (e.g. 此處因為將每個 attribute 都分開在不同的 VBO 儲存，所以每個 VBO offset=0)

- glEnableVertexAttributeArray()
  用於啟用剛剛設定完的 vertex attribute array。
  參數傳入 attribute 的 index (glVertexAttributePointers()的第一個參數)

4. Pass projection matrix to shader

```
glm::mat4 pmtx = getP();
GLuint pmatLoc = glGetUniformLocation(program, "Projection");
glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, &pmtx[0][0]);
```

- getP()
  用於取得 projection matrix。

- glGetUniformLocation()
  用於取得 uniform 變數的位置。
  第一個參數指定要查詢的 program，第二參數指定要查詢的變數名稱。

- glUniformMatrix4fv()
  用於向 shader 中的 uniform 變數賦值 (uniform 變數可視為所有 shader 共用的變數)
  (e.g. 此處將 projection matrix 傳入 shader 中的 ***Projection*** 變數)

5. Pass view matrix to shader

```
glm::mat4 vmtx = getV();
GLuint vmatLoc = glGetUniformLocation(program, "View");
glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, &vmtx[0][0]);
```

- getP()
  用於取得 view matrix。

- glGetUniformLocation()
  同上

- glUniformMatrix4fv()
  用於向 shader 中的 uniform 變數賦值 (uniform 變數可視為所有

shader 共用的變數)

(e.g. 此處將 view matrix 傳入 shader 中的 ***View*** 變數)

6. Pass texture to shader

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, modeltexture);
GLint texLoc = glGetUniformLocation(program, "Texture");
glUniform1i(texLoc, 0);
```

- glActiveTexture()

  用於 activate the texture unit

- glBindTexture()

  用於 bind 想要使用的 texture

  第一個參數傳入 texture 類型，第二個參數傳入 texture 變數名稱

- glUniform1i()

  用於向 shader 中的 uniform 變數賦值 (uniform 變數可視為所有 shader 共用的變數)

  (e.g. 此處將 modeltexture 傳入 shader 中的 ***Texture*** 變數)

7. Draw

```
glBindVertexArray(VAO);
glDrawArrays(GL_QUADS, 0, 4 * model->fNum);
```

- glDrawArrays()

  利用 VAO 的資訊畫出圖樣 (因此一定要在前面 bind VAO)

8. Vertex shader

- Receive position, normal, texture coordinate from bind buffer

```
layout(location = 0) in vec3 in_position;
layout(location = 1) in vec3 in_normal;
layout(location = 2) in vec2 in_texture;
```

要注意 location 必須對應到先前在 ***glVertexAttributePointers()*** 的第一個參數所指定的 index

(e.g. 在定義 position 的 VBO 時

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

所以 location=0

)

另外，根據該 attribute 是幾個值一組，決定了 vector 的維度 (e.g. position 是 3 個值一組，所以是 vec3)

- Receive Model matrix, View matrix, and Projection matrix from uniform

```
uniform mat4 M;
uniform mat4 Projection;
uniform mat4 View;
```

在 shader 中宣告三個 uniform 變數：***M*** (modelView matrix), ***Projection*** (projection matrix), ***View*** (view matrix)，賦值的工作則交由 main.c 負責 (利用先前的 ***glUniformMatrix4fv()*** 函數)

- Pass texture coordinate and Normal to fragment shader

```
out vec2 texcoord;
out vec3 normal;
```

```
texcoord = in_texture;
normal = in_normal;
```

宣告兩個 output 變數 (傳到 fragment shader)：***texcoord*** (texture coordinate), ***normal***
並在 main 函數中將傳入的 buffer 資料傳給 output 變數

- Calculate view space by gl_Position

```
gl_Position = Projection * View * M * vec4(in_position, 1.0);
```

9. Fragment shader
   - Receive texture coordinate and Normal from vertex shader

```
in vec2 texcoord;
in vec3 normal;
```

   - Receive texture

```
uniform sampler2D Texture;
```

宣告一個 uniform 變數：***Texture***，賦值的工作則交由 main.c 負責 (利用先前的 ***glUniform1i()*** 函數)

   - Calculate and return final color to opengl

```
out vec4 frag_color;
```

```
frag_color = texture2D(Texture, texcoord);
```

利用 ***texture2D()*** 將 texture 貼上 model
第一個參數傳入 texture，第二個參數傳入 texture coordinate

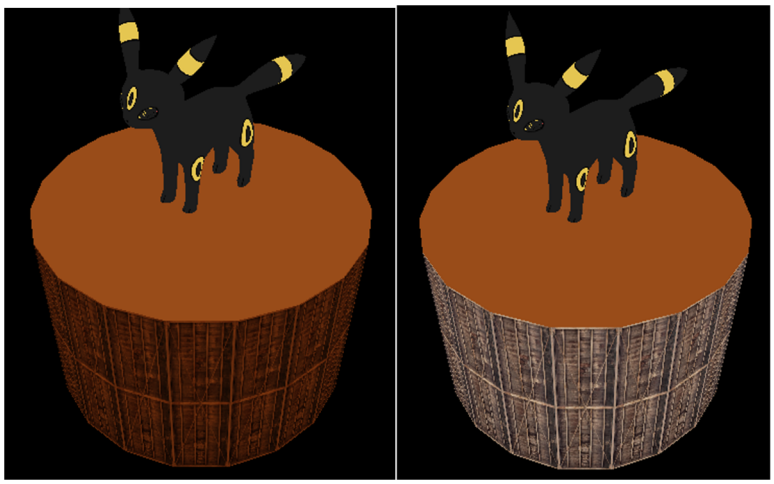
Problems & Solutions

1. Model 無法顯示

- Reason：我在 bindbufferInit()內宣告了 VAO，造成 VAO **變成函數內的 local variable**，函數結束之後就會消失

- Solution：將 VAO 宣告成**全域變數**之後便解決了此問題

2. Basis 側面貼圖的顏色遭到汙染



（左圖為我一開始畫出來的側面，右圖為正常的側面）

- Reason：我在畫 Basis 的時候是先畫頂面及底面，再畫側面，而我在畫頂面及底面的時候會用 glColor3f()指定顏色

```
glColor3f(0.6, 0.3, 0.1);
glBegin(GL_POLYGON);
glNormal3f(0, 1, 0);
for (int i = 0; i < 20; ++i)
    glVertex3f(vertex_list[i][0], vertex_list[i][1], vertex_list[i][2]);
glEnd();
```

但是當我畫到側面的時候，**該顏色跟貼圖會同時作用在側面上**，造成上方左圖所示，類似「汙染」的結果

- Solution：在畫側面之前，一樣用 **glColor3f()函數將顏色還原成預設值**(1.0,1.0,1.0 代表白色)，這樣的話就不會汙染到側面的顏色了

```
glColor3f(1.0, 1.0, 1.0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, basistexture);
for (int i = 0; i < 20; i++) {
    glBegin(GL_POLYGON);
    int j = (i + 1) % 20, h = 5;
    float nor_x = (vertex_list[i][0] + vertex_list[j][0]) / 2.0f;
    float nor_z = (vertex_list[i][2] + vertex_list[j][2]) / 2.0f;
    float div = sqrt((nor_x * nor_x) + (nor_z * nor_z));
    glNormal3f(nor_x / div, 0, nor_z / div);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(vertex_list[i][0], vertex_list[i][1], vertex_list[i][2]);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(vertex_list[j][0], vertex_list[j][1], vertex_list[j][2]);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(vertex_list[j][0], vertex_list[j][1] - h, vertex_list[j][2]);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(vertex_list[i][0], vertex_list[i][1] - h, vertex_list[i][2]);
    glEnd();
}
```

- Bonus

1. 調整 camera 位置

   - 方向鍵↑：camera 向上移動
   - 方向鍵↓：camera 向下移動
   - 【**實作**】利用 ***SpecialInput()*** 去擷取方向鍵輸入，再用變數 ***cam_h*** 控制攝影機高度

```
void SpecialInput(int key, int x, int y) {
    if (key == GLUT_KEY_UP)
        cam_h++;
    else if (key == GLUT_KEY_DOWN)
        cam_h--;
}
```

2. 調整 model 透明度

   - 0：透明度增加 (變得更透明)
   - 1：透明度減少 (變得不透明)
   - Space：還原成初始透明度
   - 【實作】利用一個 uniform 變數 ***transparency*** 將透明度 (介於 0 至 1 之間) 傳送到 fragment shader，並在 fragment shader

裡面改動 ***frag_color*** 的第四個元素(透明度)

```
GLuint transLoc = glGetUniformLocation(program, "transparency");
glUniform1f(transLoc, transparency);
```

```
void main()
{
    frag_color = texture2D(Texture, texcoord);
    frag_color.x += r;
    frag_color.y += g;
    frag_color.z += b;
    frag_color.a = transparency;
}
```

3. 調整 model 顏色

- r：增加紅色
- R (shift+r)：減少紅色
- g：增加綠色
- G (shift+g)：減少綠色
- b：增加藍色
- B (shift+b)：減少藍色
- Space：還原成初始顏色
- 【**實作**】利用三個 uniform 變數（*r*, *g*, *b*）將三個顏色要增加得值(介於-1 到 1 之間）傳送到 fragment shader，並在 fragment shader 裡面改動 ***frag_color*** 的前三個元素（分別代表紅色、綠色、藍色)

```
GLuint rLoc = glGetUniformLocation(program, "r");
glUniform1f(rLoc, r);

GLuint gLoc = glGetUniformLocation(program, "g");
glUniform1f(gLoc, g);

GLuint bLoc = glGetUniformLocation(program, "b");
glUniform1f(bLoc, b);
```

```
void main()
{
    frag_color = texture2D(Texture, texcoord);
    frag_color.x += r;
    frag_color.y += g;
    frag_color.z += b;
    frag_color.a = transparency;
}
```