# 109550175 許登豪 HW2 report

1. 實作方式 ：

a. Line 19 – Line 20

這是兩個函式的宣告 ，

其中 ModelVAO 傳入一個 object , return 得到他的 VAO;

DrawModel 則是傳入一個字串代表要 render 的 object , 相應的 model , view , perspective matrix , 以及物體的材質。(函式太長就沒有完整截圖了 ，後面還有三個引數 ，分別是兩個物體的原材質 ，新的貓材質 ，以及新的木箱材質)

```
unsigned int ModelVAO(Object* model);
void DrawModel(const char* target, glm::mat4 M, glm::mat4 V, glm::mat4 P
```

b. Line 31 – Line 33

這是整支程式需要的全域變數 ，

is_changed 代表的是物體的材質是否要改變(color effect)

is_ripple 代表的是物體是否要做 wave motion(deformation effect)

wave 則是代表 is_ripple = 1 時 ，抖動的幅度 ，而抖動幅度至少要是 0 , 代表完全不抖動。

```
bool is_changed = false;
int is_ripple = 0;
int wave = 1;
```

c. Line 150 – Line 153

利用助教寫的 LoadTexture 函數 ， 除了 load 原材質外再多 load 兩個做 color effect 需要的材質

```
unsigned int catTexture, boxTexture , otherTexture , anotherTexture;
LoadTexture(catTexture, "obj/Cat_diffuse.jpg");
LoadTexture(boxTexture, "obj/CardboardBox1_Albedo.tga");
LoadTexture(otherTexture, "obj/Cat_changed.jpg");
LoadTexture(anotherTexture, "obj/Box_changed.jpg");
```

d. Line 165 – Line 166

在 render loop 之前先 enable 材質 0 號跟 1 號 ， 並且設定相對應的 texture 變數 in fragment shader

```
glUniform1i(glGetUniformLocation(shaderProgram ,"Texture1"), 0);
glUniform1i(glGetUniformLocation(shaderProgram, "Texture2"), 1);
```

e. Line192 – Line 255

按下按鍵相對應的動作被寫在這個函數裡面。

按下 A 鍵可以更改材質狀態 ， 按下 B 鍵可以更改抖動狀態

```
void keyCallback(GLFWwindow* window, int key, int scancode, int action,
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
    if (key == GLFW_KEY_A && action == GLFW_PRESS)
        if (is_changed == false) {
            is_changed = true;
        }
        else{
            is_changed = false;
        }
    if (key == GLFW_KEY_B && action == GLFW_PRESS)
        if (is_ripple == 0) {
            is_ripple = 1;
        }
        else {
            is_ripple = 0;
        }
```

按下上下鍵可以更改抖動的頻率，也限制了這兩個鍵要在 object 是抖動狀態下才會起到作用

```
if (key == GLFW_KEY_UP && action == GLFW_PRESS)
    if (is_ripple == 0) {
        return;
    }
    else {
        wave += 1;
    }
if (key == GLFW_KEY_DOWN && action == GLFW_PRESS)
    if (is_ripple == 0) {
        return;
    }
    else {
        if (wave - 1 >= 0)
            wave -= 1;
    }
```

f. Line 34 – Line 61

VAO , VBO 實作細節，設定 0 , 1 , 2 號 attribute 分別 bind 到第 0 , 1 , 2 個在 VBO 陣列裡的陣列，且 position , normal 都是 3 維的向量或座標 , text coordinate 是 2 維(對應在一張材質圖上的某處)，所以相對應參數如下。

```
unsigned int ModelVAO(Object* model) {
    unsigned int VAO, VBO[3];
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(3, VBO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (model->positions.size()), &(model->positions[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 3, 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (model->normals.size()), &(model->normals[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 3, 0);
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (model->texcoords.size()), &(model->texcoords[0]), GL_STATIC_DRAW);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 2, 0);
    glEnableVertexAttribArray(2);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindVertexArray(0);

    return VAO;
}
```

g. Line 62 – Line 109

DrawModel 實做細節 ， 傳 Model , View , Perspective , Time , 是否 ripple(前面講的 wave motion) , ripple 需要的相關參數進去 vertex shader 裡面。abs(sin(ratio))是用來做 texture 交替變化而使用的方式(因為 sin 是週期性的 ， 注意 ratio 跟時間 glfwGetTime()有關 ， 所以這個特效是一個會隨時間做材質插值變化的特效) , 最後傳入 ratio 進 fragment shader 裡面 ， 此參數代表新的 texture(對於貓咪來講 ， 是 other_texture ; 對於箱子來講 ， 是 another_texture)的比例 ， 而相對應原材質的比例則是 1 – ratio

```
void DrawModel(const char* target, glm::mat4 M, glm::mat4 V, glm::mat4 P, unsigned int texture , unsigned int other_texture ,
    unsigned int Modelloc, Viewloc, Perspectiveloc , Ratioloc , Timeloc , Rippleloc , xsloc , zsloc;
    Modelloc = glGetUniformLocation(shaderProgram, "Model");
    Viewloc = glGetUniformLocation(shaderProgram, "View");
    Perspectiveloc = glGetUniformLocation(shaderProgram, "Perspective");
    glUniformMatrix4fv(Modelloc, 1, GL_FALSE, glm::value_ptr(M));
    glUniformMatrix4fv(Viewloc, 1, GL_FALSE, glm::value_ptr(V));
    glUniformMatrix4fv(Perspectiveloc, 1, GL_FALSE, glm::value_ptr(P));

    float ratio = (float)glfwGetTime();
    Timeloc = glGetUniformLocation(shaderProgram, "time");
    glUniform1f(Timeloc, ratio);
    Rippleloc = glGetUniformLocation(shaderProgram, "ripple");
    glUniform1i(Rippleloc, is_ripple);
    xsloc = glGetUniformLocation(shaderProgram, "xs");
ratio = abs(sin(ratio));
if (!is_changed) {
    ratio = 0;
}
Ratioloc = glGetUniformLocation(shaderProgram, "ratio");
glUniform1f(Ratioloc, ratio);


if (strcmp(target, "cat") == 0)
{
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, other_texture);
    glBindVertexArray(ModelVAO(catModel));
    glDrawArrays(GL_TRIANGLES, 0, catModel->positions.size());
    glBindTexture(GL_TEXTURE_2D, aother_texture);
    glBindVertexArray(ModelVAO(boxModel));
    glDrawArrays(GL_TRIANGLES, 0, boxModel->positions.size());
    glBindVertexArray(0);
}
```

h. Vertex shader

先 initialize 三個傳入的 attribute，以及需要的 uniform 變數，並且再 initialize 一個 vertex shader 需要往下傳給 fragment shader 裡的 out 變數 texCoord。main 裡面寫的是，如果 ripple == 0，設定 gl_Position 的方式就是普通的 MVP 矩陣 apply 到原來的點(會轉到 homogeneous coordinate)，否則就是用 sin 的連乘積製作抖動效果。然後也記得要 assign 值給輸出 texCoord。

```
#version 330 core
// TO DO:
// Implement vertex shader
// note: remember to set gl_Position

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

out vec2 texCoord;

uniform mat4 Model ;
uniform mat4 View ;
uniform mat4 Perspective ;

uniform float time , xs , zs ;
uniform int ripple ;

void main()
{
    if(ripple == 0){
        gl_Position = Perspective * View * Model * vec4(aPos, 1.0);
    }
    else{
        float s = 1.0 + 0.1*sin(xs*time)*sin(zs*time) ;
        gl_Position = Perspective * View * Model *vec4(aPos.x , aPos.y*s , aPos.z , 1);
    }

    texCoord = aTexCoord;
}
```

i. Fragment shader

以 texCoord(in)接住從 vertex shader 傳下來的 texture coordinate , initialize 需要的 uniform 變數 ， 再 initialize 一個 fragment shader 需要輸出顏色的 out 變數 fragColor。main 裡面寫的是 ， 用 mix 這個 glsl 內鍵函數做出材質插值的特效。

```
#version 330 core
// TO DO:
// Implement fragment shader
in vec2 texCoord;
out vec4 fragColor ;
uniform sampler2D Texture1;
uniform sampler2D Texture2;
uniform float ratio ;
void main()
{
    fragColor = mix(texture(Texture1, texCoord), texture(Texture2, texCoord), ratio);
}
```

## 2. 遇到的問題 :

一開始把很多 vertex attribute bind 在一個很胖的大陣列裡 , 這樣寫起來 vertex attribute 的設定會很沒有一致性也很累 , 最後跟同學討論後 , 才想到原來可以有一個 vertex attribute bind 一個相對應的 vertex information 陣列的寫法。

還有一個問題是在寫 fragment shader 時不小心把某一個材質也叫作 texture , 和 glsl 裡面的 texture 函數撞名導致 render 時材質沒有成功貼上去。

最後花了一些心思想了一下 deformation 該怎麼做 , 因為感覺很多效果(例如根據 normal 等做變化)都是靠 geometry shader 做出來的 , 最後參考了上課投影片做出了不錯的 wave motion 效果。

## 3. Bonus detail :

按下 A:切換(在 activate 與否之間切換)貓咪跟箱子的材質特效是否 activate , 預設是不 activate。特效效果是一個隨時間變化的材質變動(原材質 , 新材質插值 , 比例隨時間變)。

按下 B:切換(同上)貓咪跟箱子的抖動特效是否 activate , 預設是不 activate。特效效果可以藉由按上下鍵調整抖動的程度 , 預設是 wave = 1 的抖動 , 且只能在抖動為 activated 的時候調整。