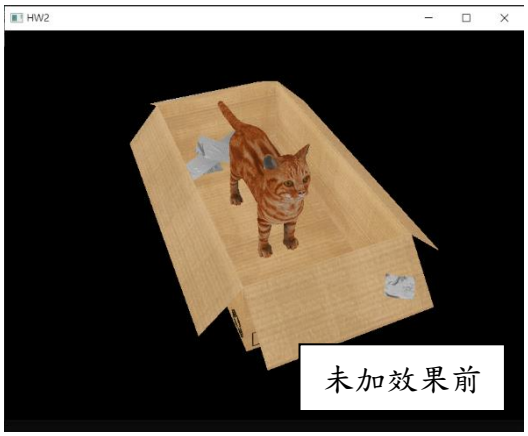
計算機圖學概論 HW2 Report

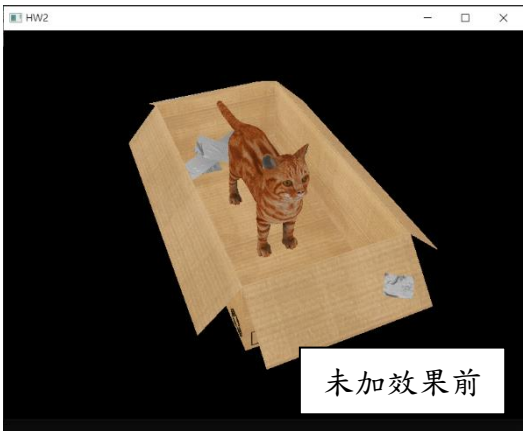
運管 11 0713216 龔祐萱

一、效果與觸發鍵

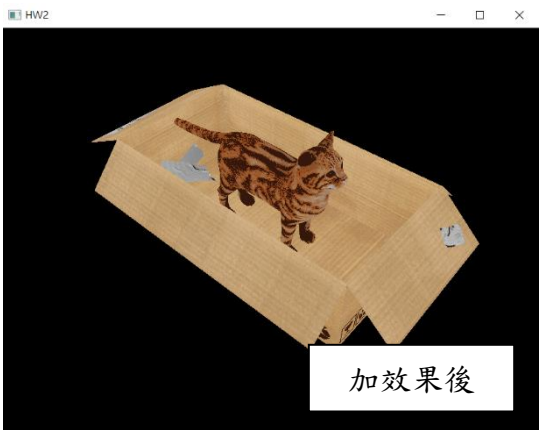
1. 數字鍵 1 -> 貓向 y 軸縮小成 0.5 倍



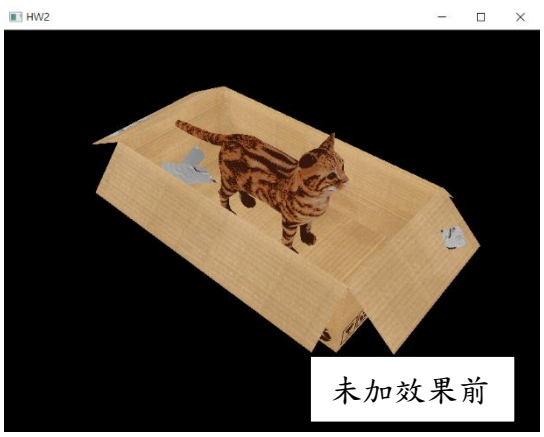
2. 數字鍵 2 -> 貓向 y 軸放大成 2 倍



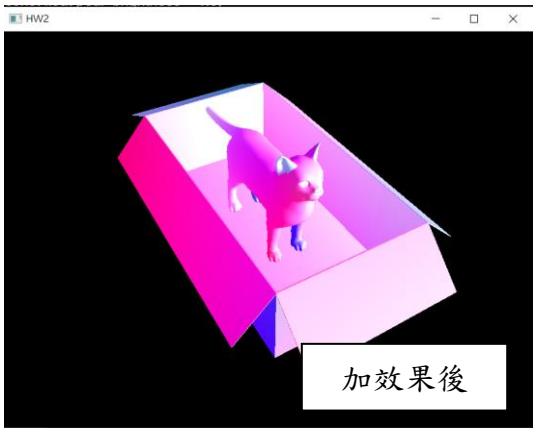
3. 數字鍵 3 -> 加深貓條紋較深的部分



4. 數字鍵 4 -> 還原貓的顏色

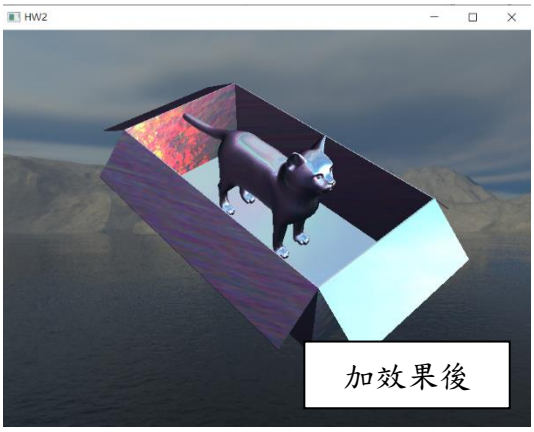


5. 數字鍵 5 -> 箱子和貓的顏色會隨時間改變，且有打光效果

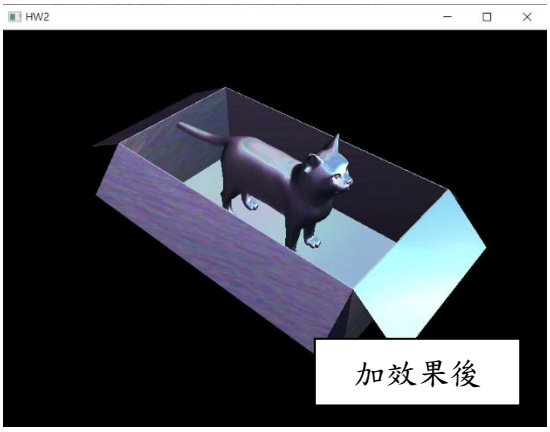


6. 數字鍵 6 -> 天空盒以及物體反射環境

7. 數字鍵 7 -> 天空盒、物體反射環境加珍珠光澤



8. 數字鍵 8 -> 關閉天空盒



二、程式實作方法

1. VAO、VBO 設置

- VBO 設置:

(1) glGenBuffers(GLsizei n, GLuint * buffers)

產生一塊受 OpenGL 管理的連續記憶體，創建 n 個 buffer object 並回傳 buffer object 的 ID。

(2) glBindBuffer(GLenum target, GLuiint buffer)

把 buffer object bind 到 GL_ARRAY_BUFFER 上。

(3) glBufferData(GLenum target, GLsizeptr size, const GLvoid * data, GLenum usage)

把物件的 position, normal, texCoord 分別輸到 VBO 相對應的位置上，並告知要出入多少資料到 target buffer 上。

- 設置 VAO:

(1) glGenVertexArrays(GLsizei n, GLuint * array)

產生一個 VAO 物件，並得到該物件的 ID。

(2) glBindVertexArray(GLuint array)

綁定 VAO。

(3) glVertexAttribPointer()

把物件的 VAO 含其 VBO 使用 pointer 組合起來。

(4)使用完後就 call glBindVertexArray(0) unbind VAO。

VAO		VBO					
		Vertex 1			Vertex 2		
Attribute pointer 0	→	X	Y	Z	X	Y	Z
Attribute pointer 1	↔	n1	n2	n3	n1	n2	n3
Attribute pointer 2	←	t1	t2		t1	t2	

VAO 和 VBO 之間的資料結構示意圖

```
// TO DO:
// Create VAO, VBO
catVAO = ModelVAO(catModel);
boxVAO = ModelVAO(boxModel);
cubeVAO = SkyboxModelVAO(cubeModel);
```

```
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;
}
```

2. Texture 設置

- 利用助教幫忙包好的 LoadTexture 把紋理圖片 bind 到 GL_TEXTURE_2D，並設成 uniform 使其在 main.cpp 的 while 中可以告知 fragment shader 目前要貼到物件上的紋理是什麼，並依照 texCoord 輸出相對應的顏色。

```
// Texture
unsigned int catTexture, boxTexture;
LoadTexture(catTexture, "obj/Cat_diffuse.jpg");
LoadTexture(boxTexture, "obj/CardboardBox1_Albedo.tga");
```

```
//box
glBindTexture(GL_TEXTURE_2D, boxTexture); 告知目前要貼的 texture 是什麼
box_model = glm::rotate(box_model, glm::radians(90.0f) * (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
box_model = glm::scale(box_model, glm::vec3(0.0625f, 0.05f, 0.05f));
DrawModel("box", box_model, getView(), getPerspective(), glm::mat4(1.0f), shaderProgram);

//Cat
glBindTexture(GL_TEXTURE_2D, catTexture);
```

```
void LoadTexture(unsigned int& texture, const char* tFileName) {

    glEnable(GL_TEXTURE_2D);
    glActiveTexture(GL_TEXTURE0);
    glGenTextures(1, &texture);
    if (texture == (size_t)"catTexture")
        glUniform1i(glGetUniformLocation(shaderProgram, "Texture"), 1); 把 texture 設成 uniform，
    else if (texture == (size_t)"boxture")
        glUniform1i(glGetUniformLocation(shaderProgram, "Texture"), 0); 並告知在哪個 program 中使用

    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    int width, height, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(tFileName, &width, &height, &nrChannels, 0);

    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
    glBindTexture(GL_TEXTURE_2D, 0);
    stbi_image_free(data);
}
```

3. Draw model

- 設定一個 DrawModel function 輸入該物件的 transformation, view, projection model，並將以上 model 放置到各物件的 VAO 中以及設成 umiform 方便 vertex shader 使用，告知這些 model 是在哪個 program 中執行並輸出。

```
DrawModel("cube", glm::mat4(1.0f), glm::mat4(1.0f), getPerspective(), glm::mat4(1.0f), skyBoxProgram);
```

```
//box
glBindTexture(GL_TEXTURE_2D, boxTexture);
box_model = glm::rotate(box_model, glm::radians(90.0f) * (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
box_model = glm::scale(box_model, glm::vec3(0.0625f, 0.05f, 0.05f));
DrawModel("box", box_model, getView(), getPerspective(), glm::mat4(1.0f), shaderProgram);
```

```
//Transformation
cat_model = glm::rotate(cat_model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
cat_model = glm::rotate(cat_model, glm::radians(90.0f) * (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
DrawModel("cat", cat_model, getView(), getPerspective(), cat_scale_model, shaderProgram);
```

```
void DrawModel(const char* target, glm::mat4 M, glm::mat4 V, glm::mat4 P, glm::mat4 S, unsigned int Program) {

    glUniformMatrix4fv(glGetUniformLocation(Program, "P"), 1, GL_FALSE, glm::value_ptr(P));
    glUniformMatrix4fv(glGetUniformLocation(Program, "V"), 1, GL_FALSE, glm::value_ptr(V));
    glUniformMatrix4fv(glGetUniformLocation(Program, "M"), 1, GL_FALSE, glm::value_ptr(M));
    glUniformMatrix4fv(glGetUniformLocation(Program, "S"), 1, GL_FALSE, glm::value_ptr(S));

    if (strcmp(target, "box") == 0)                    把各個 model 設成 uniform
    {
        glBindVertexArray(boxVAO);
        glDrawArrays(GL_TRIANGLES, 0, boxModel->positions.size());
    }
    else if (strcmp(target, "cat") == 0)
    {
        glBindVertexArray(catVAO);    和各物件的 VAO bind 在一起
        glDrawArrays(GL_TRIANGLES, 0, catModel->positions.size());
    }
    else if (strcmp(target, "cube") == 0)
    {
        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
    }
    glBindVertexArray(0);
}
```

4. Basic vertex shader

- 把物件 VAO 和 VBO 中位置、法向量、紋理座標輸到 vertex shader 中。
- 把 model matrix, view matrix 以及 projection matrix 設為 uniform 輸入 vertex shader。
- 在 vertex shader 進行物件的 transformation，並把物件投射至螢幕上，再把紋理座標等變數送至 fragment shader。

```
#version 330 core

// TO DO:
// Implement vertex shader
// note: remember to set gl_Position

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

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

out vec4 worldPos;
out vec3 normal;
out vec2 texCoord;
out vec3 Normal;
out vec3 Position;
```

```
void main(){

    gl_Position = P * V * M * S * vec4(aPosition, 1.0f);
    texCoord = aTexCoord;

    //lighting
    worldPos = M * S * vec4(aPosition, 1.0f);
    mat4 normal_transform = transpose(inverse(M*S));
    normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);
    onormal = aNormal;

    //Environmentl mapping
    Normal = mat3(transpose(inverse(M*S))) * aNormal;
    Position = vec3(M * S * vec4(aPosition, 1.0f));
}
```

每個 vertex 分別在 vertex shader 和 transformation, view, projection model 進行相乘，
並把 texCoord 傳給 fragment shader

5. Basic fragment shader

- 從 vertex shader 取得物件的 texCoord，並將物件的紋理設定成 uniform，將紋理貼至 texCoord 對應的地方，並輸出物件的顏色。

```
#version 330 core

// TO DO:
// Implement fragment shader

in vec4 worldPos;
in vec3 normal;
in vec2 texCoord;
in vec3 viewDirection;
in vec3 Normal;
in vec3 Position;

uniform sampler2D Texture;
uniform samplerCube cubeMap;
uniform int changeColor;
uniform float time;
uniform mat4 Rotate;
```

```
out vec4 FragColor;

const float pearl_brightness = 1.5;

bool inRange(float color1, float color2){
    return color1 - color2 < 0.01;
}

void main(){

    if(changeColor == 0){
        FragColor = texture(Texture, texCoord);
    }
```

把 texture 貼到 texCoord 的位置上並輸出顏色

6. Effect 1 : Deformation (貓模型的放大縮小)

- 因為採用 scale 的程度隨時間以 0.625 的程度遞增或遞減的方式實作，所以為了不讓縮小程度太小，使 scale 數值失去精準度導致放大後會在兩個數值之間震盪，讓貓起來會快速的放大縮小，因此設定 scaling 的範圍為 0.0625~8 倍之間。如果超過此區間，則以最小值或最大值作為收縮的倍數。
- 在 main.cpp 中計算 scale model，再利用 DrawModel 把 scale model 設成 uniform 並 bind 到物件的 VAO 中，最後在 vertex shader 進行物件的 scaling。

```
//Deformation
//scale range : 0.0625 ~ 8
if (cat_size < 0.0625)
    cat_size = 0.0625;
else if (cat_size > 8)
    cat_size = 8;

if ((add > cat_size) && (press[1] == true)) {
    add -= 0.0625;
    if (add < cat_size || add == cat_size || add < 0)
        press[1] == false;

    cat_scale_model = glm::scale(cat_scale_model, glm::vec3(1.0f, 1.0 * add, 1.0f));
}
else if ((add < cat_size) && (press[2] == true)) {
    add += 0.0625;
    if (add > cat_size || add == cat_size)
        press[2] = false;

    cat_scale_model = glm::scale(cat_scale_model, glm::vec3(1.0f, 1.0 * add, 1.0f));
}
else
    cat_scale_model = glm::scale(cat_scale_model, glm::vec3(1.0f, 1.0 * add, 1.0f));

//Transformation
cat_model = glm::rotate(cat_model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
cat_model = glm::rotate(cat_model, glm::radians(90.0f) * (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
DrawModel("cat", cat_model, getView(), getPerspective(), cat_scale_model, shaderProgram);
```

計算 scale model

利用 DrawModel 把 scale model 設成 uniform，並和物件的 VAO bind 在一起

7. Effect 2 : Change color (加深貓較深紋路的顏色)

- 利用按鍵改變控制效果的變數，來觸發不同改變顏色的效果。
- 先設定 FragColor 為 texture 貼到相對應位置的顏色，再使用 inRange function 去判斷每個 vertex 的顏色數值是否比 RGB 顏色(118, 52, 23)還要深，如果是的話就把輸出顏色更改為較深的 RGB 顏色(54, 23, 11)。
- 以上的顏色是將貓的 texture 輸入網站中查詢而得到的數值。
  網站: https://imagecolorpicker.com/en

```
bool inRange(float color1, float color2){
    return color1 - color2 < 0.01;
}

void main(){

    if(changeColor == 0){
        FragColor = texture(Texture, texCoord);
    }
    else if(changeColor == 1){
        FragColor = texture(Texture, texCoord);
        if(inRange(FragColor.r, 118.0/255.0) || inRange(FragColor.g, 52.0/255.0) || inRange(FragColor.b, 23.0/255.0))
            FragColor = vec4(54.0/255.0, 23.0/255.0, 11.0/255.0, 1);
    }
```

比較目前輸出的顏色是否比基準色還深

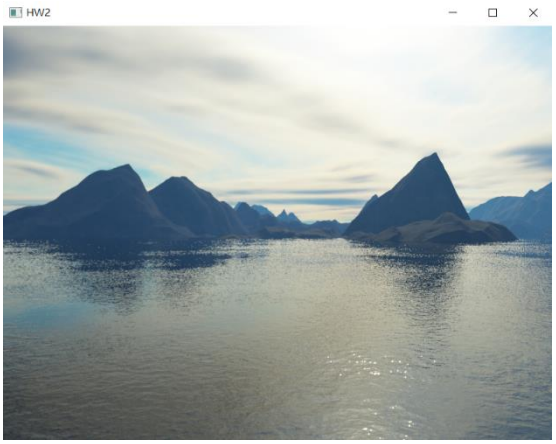
如果比基準色還深就加深它的顏色，否則保持目前輸出的顏色

三、bouns 實作方法

1.天空盒設置

- 天空盒使用 Cubemap 的方式實作,利用一個正方體將物體包裹在正方體中,並在正方體的六面分別貼上六張連續的 texture，使其看起來是一個連續的空間。

從 viewer 視角看到的天空盒



以物體為中心看到的天空盒

- 設定正方體的各頂點，分別為正方體六個面。

```
float skyboxVertices[] = {
    //back
    -1.0f, 1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, 1.0f, -1.0f,
    -1.0f, 1.0f, -1.0f,

    // left
    -1.0f, -1.0f, 1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f, 1.0f, -1.0f,
    -1.0f, 1.0f, -1.0f,
    -1.0f, 1.0f, 1.0f,
    -1.0f, -1.0f, 1.0f,
```

```
    //right
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,

    //front
    -1.0f, -1.0f, 1.0f,
    -1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, -1.0f, 1.0f,
    -1.0f, -1.0f, 1.0f,
```

```
    //top
    -1.0f, 1.0f, -1.0f,
    1.0f, 1.0f, -1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    -1.0f, 1.0f, 1.0f,
    -1.0f, 1.0f, -1.0f,

    //bottom
    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f, 1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
```

- Create 一個有 skybox.vert 和 skybox.frag 的 program 來執行天空盒的程式。

```
skyBoxVertexShader = createShader("skybox.vert", "vert");
skyBoxFragmentShader = createShader("skybox.frag", "frag");
skyBoxProgram = createProgram(skyBoxVertexShader, skyBoxFragmentShader);
```

- 設定 skybox 的 VAO 和 VBO，作法與貓和箱子模型的 VAO 和 VBO 設定大致相同，但 skybox 的 VAO 只有一個 attribute pointer 指到 VBO 存放 vertex 的位置。

```
cubeVAO = SkyboxModelVAO(cubeModel);
```

```
unsigned int SkyboxModelVAO(Object* model) {

    unsigned int VAO, VBO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(1, &VBO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 3, 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    glBindVertexArray(0);

    return VAO;
}
```

- 宣告一個型別為 string 的 vector，將圖片位置依序放到 vector 中。
- 建立 LoadCubeMap function 將 vector 中所存的圖片依照順序貼到正方體的六個面上，並且 bind 到 GL_TEXTURE_CUBE_MAP。

```
//CubeMap
unsigned int cubeMap;
texture_faces.push_back("obj/right.jpg");
texture_faces.push_back("obj/left.jpg");
texture_faces.push_back("obj/bottom.jpg");
texture_faces.push_back("obj/top.jpg");
texture_faces.push_back("obj/back.jpg");
texture_faces.push_back("obj/front.jpg");
cubeMap = LoadCubeMap(texture_faces);
```

```
unsigned int LoadCubeMap(vector<string> texture_faces) {

    unsigned int textureID;
    glEnable(GL_TEXTURE_CUBE_MAP);
    glGenTextures(1, &textureID);
    glActiveTexture(GL_TEXTURE1);

    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);

    int width, height, nrChannels;
    for (unsigned int i = 0; i < texture_faces.size(); i++)
    {
        unsigned char* data = stbi_load(texture_faces[i].c_str(), &width, &height, &nrChannels, 0);

        if (data)
        {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
            stbi_image_free(data);
        }
        else
        {
            std::cout << "Cubemap tex failed to load at path: " << texture_faces[i] << std::endl;
            stbi_image_free(data);
        }
    }

    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    glBindTexture(GL_TEXTURE_CUBE_MAP, 0);

    return textureID;
}
```

- 使用 skyBoxProgram 來執行天空盒的程式，並開啟 GL_TEXTURE1，告知 fragment shader 要貼的 texture 為 cubeMap，最後在 skyBoxProgram 中繪製出天空盒，繪製完成後關閉 skyBoxProgram。

```
glUseProgram(skyBoxProgram);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubeMap);
DrawModel("cube", glm::mat4(1.0f), glm::mat4(1.0f), getPerspective(), glm::mat4(1.0f), skyBoxProgram);
glUseProgram(0);
```

2.天空盒的 vertex shader (skybox.vert)

- 天空盒的 vertex shader 的設定與貓和箱子的 vertex shader 相似。
- 先把正方體的 vertex 位置輸入到天空盒的 vertex shader 中，並計算 texCoord 輸出給天空盒的 fragment shader，並輸出物件的位置。

```
#version 330 core

// TO DO:
// Implement vertex shader
// note: remember to set gl_Position

layout (location = 0) in vec3 aPosition;

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

out vec3 texCoord;

void main(){

    texCoord = vec3(aPosition.x, -aPosition.yz);
    vec4 pos =  P * V * M * vec4(aPosition, 1.0f);
    gl_Position = pos.xyww;

}
```

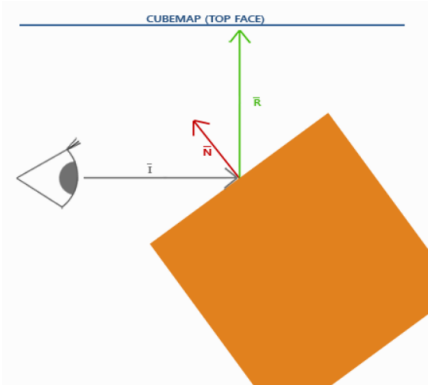
3.天空盒的 fragment shader(sky.frag)

- 輸入 cubeMap 並將 texture 貼到相對應的位置，最後由 fragment shader 將顏色輸出。

```
#version 330 core

// TO DO:
// Implement fragment shader

in vec3 texCoord;

uniform samplerCube cubeMap;
uniform float time;

out vec4 FragColor;

void main(){

        FragColor = texture(cubeMap, texCoord);
}
```

4. 物體反射環境 (environment mapping)

- 反射效果是透過觀察向量(I)、物體的法向量(N)來計算反射向量(R)，由反射向量對正方體的 texture 進行採樣，並回傳一個環境的顏色值作為物體輸出的顏色(如下圖所示)。



- 物體的 vertex shader

  計算物體的法向量以及物體的位置並傳給 fragment shader。

```
//Environmentl mapping
Normal = mat3(transpose(inverse(M*S))) * aNormal;
Position = vec3(M * S * vec4(aPosition, 1.0f));
```

- 物體的 fragment shader

  觀察向量(I)是由物體座標減掉觀察者的座標而得；反射向量(R)是使用 GLSL 中內建的 reflect function 輸入觀察向量和標準化的法向量所得，最後將天空盒的 texture 依照反射向量貼到物體上面，並輸出顏色，就可以得到 environment mapping 的效果。

```
else if(changeColor == 3){

    vec3 I = normalize(Position - (0, 5, 5));
    vec3 R =  -reflect(I, normalize(Normal));
    FragColor =  texture(cubeMap, R);
}
```

5. 物體加珍珠光澤

- 使用物體的 fragment shader 進行計算。
- 使用 view dot normal 的方式去計算物體反射光的強度，如果越強就增加該顏色分量所輸出的數值，並同時反射天空盒的 texture，達成類似珍珠的七彩光澤。

```
else if(changeColor == 4){

    const float peal_brightness = 1.5;

    vec3 I = normalize(Position - (0, 5, 5));
    vec3 R = -reflect(I, normalize(Normal));
    vec3 fvNormal = normalize(Normal);

    float view_dot_normal = max(dot(fvNormal, I), 0.0);
    float view_dot_normal_inverse = 1.0 - view_dot_normal;

    float red = texture(cubeMap, R).r * view_dot_normal + pearl_brightness  * texture(cubeMap, R + vec3(0.1, 0.0, 0.0) * view_dot_normal_inverse).r * (1.0 - view_dot_normal);
    float green = texture(cubeMap, R).g * view_dot_normal + pearl_brightness  * texture(cubeMap, R + vec3(0.0, 0.1, 0.0) * view_dot_normal_inverse).g * (1.0 - view_dot_normal);
    float blue = texture(cubeMap, R).b * view_dot_normal + pearl_brightness  * texture(cubeMap, R + vec3(0.0, 0.0, 0.1) * view_dot_normal_inverse).b * (1.0 - view_dot_normal);
    FragColor = vec4(red, green, blue, 1.0);
}
```

6. 讓物件的顏色隨時間改變

- 將 RGB 顏色分量分開計算，加入 time 去讓物件顏色隨著時間呈 sin 函數的顏色變化，最後再將 RGB 三個顏色分量合成一個 vec 由 fragment shader4 輸出到對應的位置。

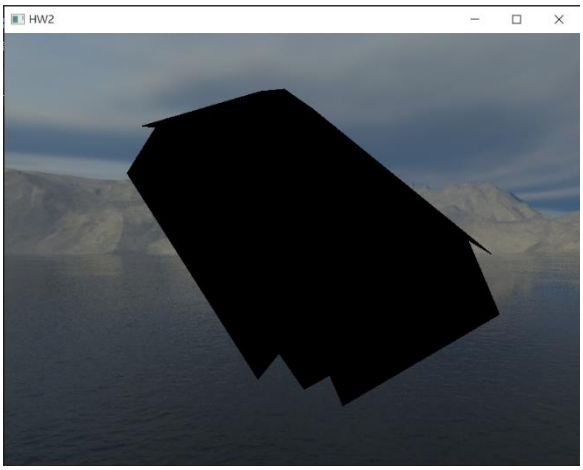
```
else if(changeColor == 2){
    vec2 uv = texCoord;
    vec3 lightPos = vec3(20, 10, 10);
    vec3 n_normal = normalize(normal);
    vec3 light = (normalize(vec4(lightPos, 1.0) - worldPos)).xyz;

    float diffuse = max(dot(light, n_normal), 0.0);

    float red = 0.5 + 0.5 * sin(time * 1.0 + uv.x * 3.0 + uv.y * 0.0);
    float green = 0.5 + 0.5 * sin(time * 0.5 + uv.x * 0.0 + uv.y * 3.0);
    float blue = 0.5 + 0.5 * sin(time * 0.75 + uv.x * 2.0 + uv.y * 2.0);

    FragColor = vec4(red, green, blue, 1.0) + vec4(diffuse, diffuse, diffuse, 1);
}
```

四、遇到的問題與解決方法

1. 將 texture 貼到相對應的物件

因為貓和箱子與天空盒所採用的貼法不同，因此剛開始將天空盒的 LoadCubeMap()設置好後，發現貓和盒子的 texture 會不見，只留下天空盒的 texture(如下圖所示)。

後來經過檢查之後發現，因為沒有將貓和盒子的 texture 以及天空盒的 texture 分別放到不同的 GL_TEXTURE 中，導致按觸發鍵後會讓原本貓和箱子的 GL_TEXTURE 變成去執行黏貼天空盒的 texture，造成兩者無法同時出現。因此我分別使用 GL_TEXTURE0 去存貓和盒子的 texture、GL_TEXTURE1 去儲存天空盒的 texture，並在各自的 program 中使用 glActiveTexture()去告知目前要使用哪個 GL_TEXTURE。



2. 讓物件隨時間慢慢放大或縮小

一開始是採用 glm::scale 去寫物件的放大以及縮小，但會出現兩個問題。

第一個問題是箱子和貓會同時放大以及縮小，但我的目標是只要讓貓進行放大和縮小的 transformation。因此我將箱子的 scale model 寫成 identity matrix 並回傳到 DrawModel()變成 uniform，而貓的 scale model 則是利用變數去控制要放大還是縮小，最後再傳到 DrawModel()變成 uniform，讓兩個物件可以分開處理。

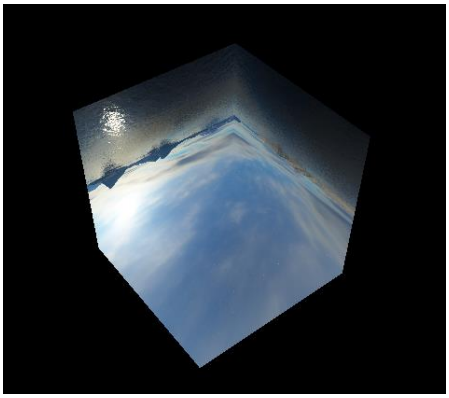
第二個問題是，如果只是寫 cat_scale_model = glm::scale(cat_scale_model, glm::vec3(1.0f, 1.0 * add, 1.0f));的話，執行時貓會瞬間變大然後再變回原本的大小。後來我使用 add 變量以及 cat_size 去改善，add 變量每次會加或減 0.0625，使物件可以成 2^n 倍去進行縮放，而 cat_size 則表示目前貓應該要放大或縮小成原本的幾倍，如果 add 和 cat_size 一樣大的話，就表示已經放大到指定的倍數了，因此就會讓 add 值不變，使每個 while loop 中的 scale 大小保持相同。

3. 天空盒 texture 的黏貼問題

天空盒 texture 的黏貼問題花了我最多的時間，我按照網路上的教學嘗試了不同的黏貼方法，但所跑出的畫面都是全黑的畫面。起初我以為是正方體沒有繪製到畫面上，因此使得天空盒的 texture 無法進行黏貼，但後來我透過打光的方式確認正方體是有被繪製在畫面上的。於是我在網路上看到有人提出黏貼至天空盒的圖片需要全部圖片都大小相同，因此我改使用網路上所找到的 texture 進行更換之後，才成功把天空盒的 texture 黏貼上去。

4. 天空盒 texture 上下顛倒的問題

將天空盒的前置作業都設定好之後，發現輸出的天空盒為上下前後顛倒的效果，因此我將天空盒的 vertex shader 中的 aPosition.yz 改為輸出反向的值，使其能正確的輸出天空盒的 texture。

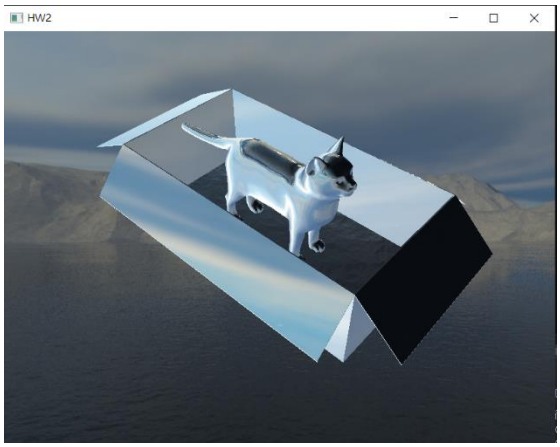


```
texCoord = vec3(aPosition.x, -aPosition.yz);
vec4 pos =  P * V * M * vec4(aPosition, 1.0);
gl_Position = pos.xyww;
```

將 texCoord 的 y, z 座標設成反向，
使其能將 texture 貼至正確的位置

5. 物體反射環境上下顛倒的問題

因為物件反射環境的向量和天空盒輸出的顏色是分開計算的，因此會出現天空盒輸出的方向正確，但物件所反射的環境內容仍為上下左右前後相反的結果，因此我將物件 fragment shader 中的反射向量®加上一個負號，使其能正確的反射環境內容到物件上。




```
else if(changeColor == 3){

    vec3 I = normalize(Position - (0, 5, 5));
    vec3 R = -reflect(I, normalize(Normal));
    FragColor =  texture(cubeMap, R);
}
```

從左圖可看到貓的上方反射的環境為海的部分，
因此在 fragment shader 中將反射向量(R)設為反向，
使物件能正確的反射環境內容。