# Computer Graphic – HW01

309553048_多工所_陳柏丞

- **Program Environment：**
  - Xcode：12.0.1
  - Libraries：
    - OpenGL.framework
    - Libglfw.3.3.dylib
    - libGLEW.2.0.1.dylib
- **Implementation：**

Q1. Render a 3D cube with texture and let cube rotate over time：

1. Declare an array：

   `float cube_vertices[]` = {vertices*3, texture*2, normal*3}

2. Load an image：

```
unsigned int loadImageToGPU(const char* filename, GLint internalFormat, GLenum format, int textureSLot)
{
    unsigned int TexBuffer;
    glGenTextures(1, &TexBuffer);
    // 圖片槽位
    glActiveTexture(GL_TEXTURE0 + textureSLot);
    glBindTexture(GL_TEXTURE_2D, TexBuffer);

    int img_w, img_h, nrChannel;

    stbi_set_flip_vertically_on_load(true); // 上下顛倒

    unsigned char *data = stbi_load(filename, &img_w, &img_h, &nrChannel, 0);

    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, img_w, img_h, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        cout << filename << endl;
        cout << "load image failed" << endl;
    }

    stbi_image_free(data);

    return TexBuffer;
}
```

3. Load data to VAO & VBO：

```
unsigned int VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

unsigned int VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(cube_vertices), cube_vertices, GL_STATIC_DRAW);

// 位置屬性
glVertexAttribPointer(6, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(6);

// 紋理屬性
glVertexAttribPointer(8, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(8);
```

4. Design rotate & project matrices：

```
glm::mat4 modelMat(1.0f); // 初始化
glm::mat4 modelMat2(1.0f);
modelMat2 = glm::rotate(modelMat2, glm::radians(90.0f), glm::vec3(1.0, 0, 0));

glm::mat4 viewMat(1.0f);
glm::mat4 projMat(1.0f);

projMat = glm::perspective(glm::radians(45.0f), (float)width/(float)height, 0.1f, 100.0f);

while(!glfwWindowShouldClose(window))
{
    processInput(window); // 關閉視窗(esc)

    modelMat = glm::rotate(glm::mat4(1.0f), (float)glfwGetTime(), glm::vec3(1.0f, 1.0f, 0.0f));
```

5. Load data to Shader：

```
// Set Material -> Shader Program
myShader->use();

// Transform
glUniformMatrix4fv(glGetUniformLocation(myShader->ID, "modelMat"), 1, GL_FALSE, glm::value_ptr(modelMat * modelMat2));
glUniformMatrix4fv(glGetUniformLocation(myShader->ID, "viewMat"), 1, GL_FALSE, glm::value_ptr(viewMat));
glUniformMatrix4fv(glGetUniformLocation(myShader->ID, "projMat"), 1, GL_FALSE, glm::value_ptr(projMat));

// Texture
glUniform1i(glGetUniformLocation(myShader->ID, "ourTexture"), 0);
```

6. Draw：

   glDrawArrays(GL_TRIANGLES, 0, 6 * 3 * 2);

Q2. Render two **spheres** with two textures and let them rotate over time：

1. Vertices coordinate(x, y, z) & Get sphere data：

```
void createSphere(float *sphere, int slice, int stack){
    float r = 1.0f;
    float lengthInv = 1.0f / r;
    // slice：經線切分個數
    // stack：緯線切分個數

    float stack_step = 1.0f/stack;
    float slice_step = 1.0f/slice;

    int size = 8;

    int offset = 0;
    for(int u = 0; u < stack; u++){
        for(int v = 0;v < slice; v++){
            // 4個點，形成兩個三角形
            // { vertices * 3，texture * 2，normal * 3 }
            float point1[8] = {...};
            float point2[8] = {...};
            float point3[8] = {...};
            float point4[8] = {...};

            memcpy(sphere + offset, point4, size * sizeof(float));
            offset += size;
            memcpy(sphere + offset, point1, size * sizeof(float));
            offset += size;
            memcpy(sphere + offset, point3, size * sizeof(float));
            offset += size;

            memcpy(sphere + offset, point3, size * sizeof(float));
            offset += size;
            memcpy(sphere + offset, point1, size * sizeof(float));
            offset += size;
            memcpy(sphere + offset, point2, size * sizeof(float));
            offset += size;
```

```
float GetX(float r, float u, float v)
{
    float x = r * sin(PI * u) * cos(2 * PI * v);
    return x;
}

float GetY(float r, float u, float v)
{
    float y = r * sin(PI * u) * sin(2 * PI * v);
    return y;
}

float GetZ(float r, float u, float v)
{
    float z = r * cos(PI * u);
    return z;
}
```

2. Load images：same as above
3. Load data to VAO & VBO：same as above
4. Design rotate & project matrices：same as above
5. Load data to Shader：same as above
6. Draw：

   glDrawArrays(GL_TRIANGLES, 0, 3 * 2 * slice * stack);

Q3. Add lighting component to the scene：

1. Load ambient, light to Shader：

```
// Lighting
glUniform3f(glGetUniformLocation(myShader->ID, "objColor"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(myShader->ID, "ambientColor"), 0.5f, 0.5f, 0.5f);
glUniform3f(glGetUniformLocation(myShader->ID, "lightPos"), 10.0f, 10.0f, 10.0f);
glUniform3f(glGetUniformLocation(myShader->ID, "lightColor"), 1.0f, 1.0f, 1.0f);
```

2. Calculate diffuse & object color：

```
vec3 lightDir = normalize(lightPos - FragPos);
vec3 diffuse = dot(lightDir, Normal) * lightColor;
FragColor = texture(ourTexture, TexCoord) * vec4((ambientColor + diffuse) * objColor, 1.0);
```

Q4. Some keys callback functions for switching three objects：

1. Function written in while loop & change shape and texture：

   1→earth, 2→moon, 3→cube

```
void switchShape(GLFWwindow* window, unsigned int ID)
{
    if(glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)
    {
        mySphere();
        glUniform1i(glGetUniformLocation(ID, "ourTexture"), 1);
    }
    if(glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS)
    {
        mySphere();
        glUniform1i(glGetUniformLocation(ID, "ourTexture"), 2);
    }
    if(glfwGetKey(window, GLFW_KEY_3) == GLFW_PRESS)
    {
        myCube();
        glUniform1i(glGetUniformLocation(ID, "ourTexture"), 3);
    }
}
```

Q5. Use keyboard or mouse event to change camera view：

1. Data in camera：

```
class Camera
{
public:
    Camera(glm::vec3 position, glm::vec3 target, glm::vec3 worldup);
    Camera(glm::vec3 position, float pitch, float yaw, glm::vec3 worldup);
    ~Camera();

    glm::vec3 Position;
    glm::vec3 Forward;
    glm::vec3 Right;
    glm::vec3 Up;
    glm::vec3 WorldUp;
    float Pitch;
    float Yaw;
    float SenseX = 0.001f;
    float SenseY = 0.001f;
    float speedZ = 0;

    glm::mat4 GetViewMatrix();
    void ProcessMouseMovement(float deltaX, float deltaY);
    void ProcessKeyMoveX(bool type);
    void ProcessKeyMoveY(bool type);
    void UpdateCameraPos();

private:
    void UpdateCameraVectors();
};
```

2. Keyboard part & function should be written in while loop：

   Q→forward, E→back,

   W→up, A→left, S→down, D→right

```
void processInput(GLFWwindow* window)
{
    if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if(glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS)
        camera.speedZ = 1.0f;
    else if(glfwGetKey(window, GLFW_KEY_E) == GLFW_PRESS)
        camera.speedZ = -1.0f;
    else
        camera.speedZ = 0.0f;


    if(glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyMoveY(true);
    if(glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyMoveY(false);

    if(glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyMoveX(true);
    if(glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyMoveX(false);
}
```

3. Mouse part：

   mouse callback functions：

```
void mouse_callback(GLFWwindow* window, double xPos, double yPos)
{
    float deltaX, deltaY;

    if (firstMouse == true) {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    deltaX = xPos - lastX;
    deltaY = yPos - lastY;

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(deltaX, deltaY);
}

void mouse_scroll(GLFWwindow* window, double x_offset, double y_offest)
{
    camera.speedZ -= y_offest;
}
```

   mouse event：

```
glfwSetCursorPosCallback(window, mouse_callback);
glfwSetScrollCallback(window, mouse_scroll);
```

- **Problems：**

  第一次寫 openGL 花蠻多時間在架設環境＆學習語法，主要遇到兩個問題：

  1. 在球體上的 texture 座標：

     一開始以為塞入點座標的（x, y）應該就可以了，結果是圖片切成網格狀時的 2D 座標而非 3D 立體時的座標。

  2. Array in stack 大小不夠：

     這比較是 c++不熟的關係，在建立球體資料的時候會需要 6*8*360*180 個 float，故一般 array 宣告在 stack 的空間不夠用，需要利用指向 heap 的方法，也就是 malloc。

- **Demo：**