ICGHW2

一、程式實作:

基本函式的實作:

TODO1 createshader:

```
unsigned int createShader(const string &filename, const string &type) {
    unsigned int shader;
    if(type == "vert") {
        shader = glCreateShader(GL_VERTEX_SHADER);
    } else if(type == "frag") {
        shader = glCreateShader(GL_FRAGMENT_SHADER);
    } else {
        std::cout << "Unknown shader type" << std::endl;
        return 0;
    }

    FILE *file = fopen(filename.c_str(), "r");
    if (!file) {
        std::cerr << "Could not open shader file: " << filename << std::endl;
        return 0;
    }

    fseek(file, 0, SEEK_END);
    long fileSize = ftell(file);
    rewind(file);

    char *shaderCode = new char[fileSize + 1];
    shaderCode[fileSize] = '\0';
    fread(shaderCode, sizeof(char), fileSize, file);
    fclose(file);
```

```
578
579         glShaderSource(shader, 1, &shaderCode, nullptr);
580         glCompileShader(shader);
581         delete[] shaderCode;
582
583         int success;
584         glGetShaderiv(shader, GL_COMPILE_STATUS, &success);
585         if (!success) {
586             int infoLogLength;
587             glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &infoLogLength);
588
589             char *infoLog = new char[infoLogLength];
590             glGetShaderInfoLog(shader, infoLogLength, nullptr, infoLog);
591
592             std::cerr << "Shader compilation failed for " << filename << ":\n" << infoLog << std::endl;
593             delete[] infoLog;
594
595             glDeleteShader(shader);
596             return 0;
597         }
598
599         return shader;
600     }
```

createShader 函數的功能是讀取指定的著色器文件，創建並編譯相應的著色器對象（頂點或片段著色器），檢查編譯是否成功，並返回編譯好的著色器對象引用；如果編譯失敗，則打印錯誤信息並返回 0。

TODO2 createProgram:

```cpp
unsigned int createProgram(unsigned int vertexShader, unsigned int fragmentShader) {
    unsigned int program = glCreateProgram();

    glAttachShader(program, vertexShader);
    glAttachShader(program, fragmentShader);

    glLinkProgram(program);

    int success=0;
    glGetProgramiv(program, GL_LINK_STATUS, &success);

    if (!success) {
        int maxLength = 0;
        glGetProgramiv(program, GL_INFO_LOG_LENGTH, &maxLength);

        char* infoLog = (char*)malloc(sizeof(char) * maxLength);
        glGetProgramInfoLog(program, maxLength, &maxLength, infoLog);

        std::cerr << "Shader program linking failed: " << infoLog << std::endl;
        free(infoLog);

        glDeleteProgram(program);
        glDeleteShader(vertexShader);
        glDeleteShader(fragmentShader);

        return 0;
    }

    glDetachShader(program, vertexShader);
    glDetachShader(program, fragmentShader);

    return program;
}
```

createProgram 函數的功能是創建一個著色器程序，將編譯好的頂點和片段著色器附加到該程序，並連接它們。如果連接成功，返回完整的著色器程序對象；若連接失敗，則打印錯誤信息，刪除程序和著色器並返回 0。

TODO3 modelVAO:

```cpp
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(float) * model.positions.size(), model.positions.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    if (!model.normals.empty()) {
        glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model.normals.size(), model.normals.data(), GL_STATIC_DRAW);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), 0);
        glEnableVertexAttribArray(1);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
    }

    if (!model.texcoords.empty()) {
        glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model.texcoords.size(), model.texcoords.data(), GL_STATIC_DRAW);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), 0);
        glEnableVertexAttribArray(2);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
    }

    glBindVertexArray(0);
    return VAO;
}
```

modelVAO 函數的功能是為模型創建 VAO 和多個 VBO，用於管理和傳輸模型的頂點數據。它先生成並綁定 VAO，然後為頂點位置創建 VBO，將位置數據加載到 GPU 並設置頂點屬性指針。接著，如果模型包含法向量和貼圖座標，它會分

別為這些數據創建 VBO，加載數據並設置相應的頂點屬性指針。最後，解除 VAO 綁定並返回 VAO 的引用，以便後續渲染時使用。

TODO4 loadTexture:

```
unsigned int loadTexture(const string &filename) {
    glEnable(GL_TEXTURE_2D);

    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    stbi_set_flip_vertically_on_load(true);

    int width, height, nrChannels;
    unsigned char *data = stbi_load(filename.c_str(), &width, &height, &nrChannels, 0);
    if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    } else {
        cout << "Failed to load texture" << std::endl;
        return 0;
    }

    stbi_image_free(data);

    glBindTexture(GL_TEXTURE_2D, 0);

    return textureID;

}
```

loadTexture 函數的功能是從指定的文件加載貼圖圖像並生成 OpenGL 貼圖對象。它首先啟用 2D 貼圖，生成貼圖 ID，並設置貼圖的包裝和縮放參數。然後使用 stbi_load 函數加載圖像數據，將其傳送到 GPU 作為 2D 貼圖。若加載成功，返回貼圖對象 ID；若失敗，則輸出錯誤信息並返回 0。

main 流程

先 init

```
    // TODO#1: Finish function createShader
    // TODO#2: Finish function createProgram
    // TODO#3: Finish function modelVAO
    // TODO#4: Finish function loadTexture
    // You can find the above functions right below the main function

    // Initialize Object, Shader, Texture, VAO, VBO
    init();
```

1. 創建並使用 Shader:

```
// Shader
vertexShader = createShader(dirShader + "vertexShader.vert", "vert");
fragmentShader = createShader(dirShader + "fragmentShader.frag", "frag");
shaderProgram = createProgram(vertexShader, fragmentShader);
glUseProgram(shaderProgram);
```

調用 createShader 函數分別創建頂點和片段著色器，並使用 createProgram 將它們連接成著色器程序。最後，用 glUseProgram 啟用該著色器程序，以便後續渲染使用。

2. 載入 Texture:

```
// Texture
airplaneTexture = loadTexture(dirTexture + "airplane.jpg");
earthTexture = loadTexture(dirTexture + "earth.jpg");
```

根據設置好的路徑，創建飛機、地球和立方體的模型對象，並指定各自的 .obj 文件作為資源。

3. 創建 object 設置他們的 VAO 和 VBO：

```
// Object
airplaneObject = new Object(dirAsset + "airplane.obj");
earthObject = new Object(dirAsset + "earth.obj");
cubeObject = new Object(dirAsset + "cube.obj");
```

```
// VAO, VBO
airplaneVAO = modelVAO(*airplaneObject);
earthVAO = modelVAO(*earthObject);
cubeVAO = modelVAO(*cubeObject);
```

使用 modelVAO()設定三個 VAO 分別用在飛機，地球和 bonus 的 cube

4. 獲取 Uniform 變數的位置：

```
/* TODO#5: Data connection - Retrieve uniform variable locations
 *    1. Retrieve locations for model, view, and projection matrices.
 *    2. Retrieve locations for squeezeFactor, rainbowColor, and other parameters.
 * Hint:
 *    glGetUniformLocation
 */

int modelLoc = glGetUniformLocation(shaderProgram, "model");
int viewLoc = glGetUniformLocation(shaderProgram, "view");
int projectionLoc = glGetUniformLocation(shaderProgram, "projection");

int squeezeFactorLoc = glGetUniformLocation(shaderProgram, "squeezeFactor");
int rainbowColorLoc = glGetUniformLocation(shaderProgram, "rainbowColor");

if (modelLoc == -1) {
    std::cerr << "Warning: Could not find 'model' uniform location!" << std::endl;
}
if (viewLoc == -1) {
    std::cerr << "Warning: Could not find 'view' uniform location!" << std::endl;
}
if (projectionLoc == -1) {
    std::cerr << "Warning: Could not find 'projection' uniform location!" << std::endl;
}
if (squeezeFactorLoc == -1) {
    std::cerr << "Warning: Could not find 'squeezeFactor' uniform location!" << std::endl;
}
if (rainbowColorLoc == -1) {
    std::cerr << "Warning: Could not find 'rainbowColor' uniform location!" << std::endl;
}
```

使用 glGetUniformLocation 函數獲取 model、view、projection、squeezeFactor 和 rainbowColor 的位置，並檢查每個變量是否成功找到。如果變量位置為 -1，表示未找到，則輸出警告信息。

5. 渲染飛機直升機和地球:

```
        /* TODO#6-1: Render Airplane
         *    1. Set up airplane model matrix.
         *    2. Send model, view, and projection matrices to the program.
         *    3. Send squeezeFactor, rainbowColor, or other parameters to the program.
         *    4. Apply the texture, and render the airplane.
         * Hint:
         *    rotate, translate, scale
         *    glUniformMatrix4fv, glUniform1f, glUniform3fv
         *    glActiveTexture, glBindTexture, glBindVertexArray, glDrawArrays
         */
        if (useplane){
            ////////////////////////////////////////////////airplane//////////////////////////////////////////////////

            airplaneModel = glm::rotate(airplaneModel, glm::radians((float)rotateAxisDegree), glm::vec3(0.0f, 1.0f, 0.0f));
            airplaneModel = glm::rotate(airplaneModel, glm::radians(rotateAirplaneDegree), glm::vec3(-1.0f, 0.0f, 0.0f));
            airplaneModel = glm::translate(airplaneModel, glm::vec3(0.0f, airplaneHeight, 0.0f));  // 設定高度


            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(airplaneModel));
            glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
            glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

            glUniform1f(squeezeFactorLoc,0.0f);
            glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(useRainbowColor ? rainbowColor : glm::vec3(1.0f, 1.0f, 1.0f)));


            glActiveTexture(GL_TEXTURE0);
            glBindTexture(GL_TEXTURE_2D, airplaneTexture);

            glBindVertexArray(airplaneVAO);
            glDrawArrays(GL_TRIANGLES, 0, airplaneObject->positions.size() / 3);

            glBindVertexArray(0);
            ////////////////////////////////////////////////airplane//////////////////////////////////////////////////
        }
```

```
    else{
        ////////////////////////////////////////////////////cube//////////////////////////////////////////////////////

        glm::mat4 model_body(1.0f);
        cubeModel = glm::rotate(cubeModel, glm::radians((float)rotateAxisDegree), glm::vec3(0.0f, 1.0f, 0.0f));
        cubeModel = glm::rotate(cubeModel, glm::radians(rotateAirplaneDegree), glm::vec3(-1.0f, 0.0f, 0.0f));
        cubeModel = glm::translate(cubeModel, glm::vec3(0.0f, airplaneHeight, 0.0f));
        cubeModel = glm::scale(cubeModel, glm::vec3(10.0f));
        glm::mat4 eyes1 = glm::translate(cubeModel, glm::vec3(-0.25f, 0.2f, -0.5f));
        eyes1 = glm::scale(eyes1, glm::vec3(0.2f, 0.2f, 0.2f));
        glm::mat4 eyes2 = glm::translate(cubeModel, glm::vec3(0.25f, 0.2f, -0.5f));
        eyes2 = glm::scale(eyes2, glm::vec3(0.2f, 0.2f, 0.2f));
        glm::mat4 mouth1 = glm::translate(cubeModel, glm::vec3(-0.05f, -0.15f, -0.5f));
        mouth1 = glm::scale(mouth1, glm::vec3(0.17f, 0.17f, 0.17f));
        glm::mat4 water = glm::translate(cubeModel, glm::vec3(-0.08f, -0.3f, -0.49f));
        water = glm::scale(water, glm::vec3(0.09f, 0.3f, 0.17f));
        glm::mat4 model_connector = glm::translate(cubeModel, glm::vec3(0.0f, 0.5f, 0.0f));
        model_connector = glm::rotate(model_connector,glm::radians(helicopter_rotation_angle), glm::vec3(0.0f, 1.0f, 0.0f));
        model_connector = glm::scale(model_connector, glm::vec3(0.5f, 0.5f, 0.5f));
        glm::mat4 model_blade1 = glm::translate(model_connector, glm::vec3(2.0f, 0.4f, 0.0f));
        model_blade1 = glm::scale(model_blade1, glm::vec3(5.0f, 0.1f, 0.8f));
        glm::mat4 model_blade3 = glm::translate(model_connector, glm::vec3(0.0f, 0.4f, 2.0f));
        model_blade3=glm::rotate(model_blade3,glm::radians(90.f),glm::vec3(0.0f, 1.0f, 0.0f));
        model_blade3 = glm::scale(model_blade3, glm::vec3(5.0f, 0.1f, 0.8f));
        glm::mat4 model_blade2 = glm::translate(model_connector, glm::vec3(-2.0f, 0.4f, 0.0f));
        model_blade2=glm::rotate(model_blade2,glm::radians(180.f),glm::vec3(0.0f, 1.0f, 0.0f));
        model_blade2 = glm::scale(model_blade2, glm::vec3(5.0f, 0.1f, 0.8f));
        glm::mat4 model_blade4 = glm::translate(model_connector, glm::vec3(0.0f, 0.4f, -2.0f));
        model_blade4=glm::rotate(model_blade4,glm::radians(270.f),glm::vec3(0.0f, 1.0f, 0.0f));
        model_blade4 = glm::scale(model_blade4, glm::vec3(5.0f, 0.1f, 0.8f));
```

```
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(cubeModel));
        glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

        unsigned int colorTexture;
        glGenTextures(1, &colorTexture);
        glBindTexture(GL_TEXTURE_2D, colorTexture);

        unsigned char redColor[] = {122, 220, 137};
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor);

        glUniform1f(squeezeFactorLoc, 0.0f);
        glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
        glBindVertexArray(0);

        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(eyes1));
        glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

        unsigned int colorTexture1;
        glGenTextures(1, &colorTexture1);
        glBindTexture(GL_TEXTURE_2D, colorTexture1);

        unsigned char redColor1[] = {49, 167, 66};
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor1);
```

```
    glUniform1f(squeezeFactorLoc, 0.0f);
    glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

    glBindVertexArray(cubeVAO);
    glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
    glBindVertexArray(0);

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(eyes2));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

    unsigned int colorTexture2;
    glGenTextures(1, &colorTexture2);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor1);

    glUniform1f(squeezeFactorLoc, 0.0f);
    glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

    glBindVertexArray(cubeVAO);
    glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
    glBindVertexArray(0);

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(mouth1));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

    unsigned int colorTexture3;
    glGenTextures(1, &colorTexture3);
    glBindTexture(GL_TEXTURE_2D, colorTexture3);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor1);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(water));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

unsigned int colorTexture4;
glGenTextures(1, &colorTexture4);
glBindTexture(GL_TEXTURE_2D, colorTexture4);

unsigned char redColor3[] = {84, 245, 255};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor3);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_connector));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));
```

```
unsigned int colorTexture5;
glGenTextures(1, &colorTexture5);
glBindTexture(GL_TEXTURE_2D, colorTexture5);

unsigned char redColor4[] = {239, 255, 84};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor4);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_blade1));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

unsigned int colorTexture6;
glGenTextures(1, &colorTexture6);
glBindTexture(GL_TEXTURE_2D, colorTexture6);

unsigned char redColor5[] = {169, 34, 5};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor5);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_blade3));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

unsigned int colorTexture7;
glGenTextures(1, &colorTexture7);
glBindTexture(GL_TEXTURE_2D, colorTexture7);

unsigned char redColor6[] = {74, 94, 235};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor6);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));

glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_blade2));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

unsigned int colorTexture8;
glGenTextures(1, &colorTexture8);
glBindTexture(GL_TEXTURE_2D, colorTexture8);

unsigned char redColor7[] = {168, 249, 111};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor7);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));
```

```
glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_blade4));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

unsigned int colorTexture9;
glGenTextures(1, &colorTexture9);
glBindTexture(GL_TEXTURE_2D, colorTexture9);

unsigned char redColor8[] = {69, 69, 69};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, redColor8);

glUniform1f(squeezeFactorLoc, 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));
glBindVertexArray(cubeVAO);
glDrawArrays(GL_TRIANGLES, 0, cubeObject->positions.size() / 3);
glBindVertexArray(0);
////////////////////////////////////////////////////cube//////////////////////////////////////////////////
```

```
/* TODO#6-2: Render Earth
 *    1. Set up earth model matrix.
 *    2. Send model, view, and projection matrices to the program.
 *    3. Send squeezeFactor, rainbowColor, or other parameters to the program.
 *    4. Apply the texture, and render the earth.
 * Hint:
 *    rotate, translate, scale
 *    glUniformMatrix4fv, glUniform1f, glUniform3fv
 *    glActiveTexture, glBindTexture, glBindVertexArray, glDrawArrays
 */
////////////////////////////////////////////////////erath//////////////////////////////////////////////////
earthModel = glm::rotate(earthModel, glm::radians(rotateEarthDegree), glm::vec3(0.0f, 1.0f, 0.0f));
earthModel = glm::scale(earthModel, glm::vec3(10.0f));

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(earthModel));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));

glUniform1f(squeezeFactorLoc, useSqueeze ? squeezeFactor : 0.0f);
glUniform3fv(rainbowColorLoc, 1, glm::value_ptr(glm::vec3(1.0f, 1.0f, 1.0f)));


glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, earthTexture);

glBindVertexArray(earthVAO);
glDrawArrays(GL_TRIANGLES, 0, earthObject->positions.size() / 3);

glBindVertexArray(0);
////////////////////////////////////////////////////erath//////////////////////////////////////////////////
```

設定飛機地球和直升機的 model matrix 包括平移旋轉縮放等，用 glBindTexture()把 Texture 綁到對應的單元，而直升機則是生成純色的 Texture 再綁定，透過 glUniform()將,model、view、perspective matrix 傳遞給 shader，再將 squeezeFactor 和 rainbowColor 參數傳遞給 shader，控制變形和顏色效果之後，綁定飛機的貼圖並啟用 VAO，最後調用 glDrawArrays 將模型渲染出來。

如果 useplane 為 false 則顯示直升機，如果沒必要 squeeze 就回傳 0，根據有沒有 userainbowcolor 來決定回傳 rainbowcolor 或者是 glm::vec3(1.0f, 1.0f, 1.0f)(不變色)

6. 更新參數:

```
/* TODO#7: Update "rotateEarthDegree", "rotateAirplaneDegree", "rotateAxisDegree",
 *         "squeezeFactor", "rainbowColor"
 */

if (useRainbowColor) {
    rainbowDegree += rainbowSpeed * dt;

    if (rainbowDegree >= 360.0f) {
        rainbowDegree -= 360.0f;
    }
    float H = rainbowDegree;
    float C = 1.0f;
    float X = C * (1 - fabs(fmod(H / 60.0f, 2) - 1));
    float m = 0.0f;

    float r, g, b;

    if (0 <= H && H < 60) {
        r = C; g = X; b = 0;
    } else if (60 <= H && H < 120) {
        r = X; g = C; b = 0;
    } else if (120 <= H && H < 180) {
        r = 0; g = C; b = X;
    } else if (180 <= H && H < 240) {
        r = 0; g = X; b = C;
    } else if (240 <= H && H < 300) {
        r = X; g = 0; b = C;
    } else {
        r = C; g = 0; b = X;
    }
    rainbowColor = glm::vec3(r, g, b);
} else {
    rainbowColor = glm::vec3(1.0f, 1.0f, 1.0f);
}
```

```
if (useSqueeze) {
    squeezeFactor += glm::radians(float(squeezeSpeed)) * dt;
} else {
    squeezeFactor = 0.0f;
}
rotateEarthDegree += rotateEarthSpeed * dt;
rotateAirplaneDegree += rotateAirplaneSpeed * dt;
helicopter_rotation_angle ++;
if (rotateEarthDegree >= 360.0f) {
    rotateEarthDegree -= 360.0f;
}


if (rotateAirplaneDegree >= 360.0f) {
    rotateAirplaneDegree -= 360.0f;
}
if (helicopter_rotation_angle >= 360.0f) {
    helicopter_rotation_angle -= 360.0f;
}
```

包括 HSV 轉 RGB 的彩虹顏色，直升機螺旋槳的旋轉，飛機繞地球的速度，地球自轉的速度，squeezFactor 的頻率。

7. Key callback 實作:

```
/* TODO#8: Key callback
 *    1. Press 'd' to increase the "rotateAxisDegree" by 1.
 *    2. Press 'a' to decrease the "rotateAxisDegree" by 1.
 *    3. Press 's' to squeeze the earth.
 *    4. Press 'r' to make the color of the airplane rainbow.
 * Hint:
 *      GLFW_KEY_D, GLFW_KEY_A, GLFW_KEY_S, GLFW_KEY_R
 *      GLFW_PRESS, GLFW_REPEAT
 */
void keyCallback(GLFWwindow *window, int key, int scancode, int action, int mods) {
    if (key == GLFW_KEY_D && (action == GLFW_PRESS || action == GLFW_REPEAT)) {
        rotateAxisDegree += 1;
        if (rotateAxisDegree >= 360) {
            rotateAxisDegree -= 360;
        }
    }

    if (key == GLFW_KEY_A && (action == GLFW_PRESS || action == GLFW_REPEAT)) {
        rotateAxisDegree -= 1;
        if (rotateAxisDegree < 0) {
            rotateAxisDegree += 360;
        }
    }

    if (key == GLFW_KEY_S && action == GLFW_PRESS) {
        useSqueeze = !useSqueeze;
    }

    if (key == GLFW_KEY_R && action == GLFW_PRESS) {
        useRainbowColor = !useRainbowColor;
    }

    if (key == GLFW_KEY_H && action == GLFW_PRESS) {
        useplane = !useplane;
    }
}
```

keyCallback 函數的功能是處理鍵盤輸入，以改變場景中的一些變量狀態。按下或持續按住 D 鍵會讓 rotateAxisDegree 增加 1 度，A 鍵則讓它減少 1 度；這兩者都會確保角度在 0 到 360 度之間循環。按下 S 鍵會切換 useSqueeze 狀態，用來控制是否啟用擠壓效果；R 鍵會切換 useRainbowColor 狀態，控制彩虹顏色效果；H 鍵會切換 useplane 狀態，用來切換飛機的顯示模式。

二、問題及解決:

1. 一開始在實作 squeezfactor 時地球變化超快，慢慢地 debug 後才發現 sin 函式中需要的是弧度，所以我從角度轉成弧度就沒問題了。

2. 在做 bonus 的時候因為方塊如果不綁定 texture 的話，他會預設綁定上一個的 texture，就算綁定空的也不好上色，所以最後我自己創一個純色的 texture 給他綁定，來解決問題。