ICG HW2

Main.cpp:

TODO#1-1: createShader

```
unsigned int createShader(const string& filename, const string& type)
{
    string shaderSource;
    ifstream shaderFile(filename);
    if (shaderFile.is_open()) {
        string line;
        while (getline(shaderFile, line)) {
            shaderSource += line + "\n";
        }
        shaderFile.close();
    }
    else {
        cerr << "Failed to open shader file: " << filename << endl;
        return 0;
    }

    unsigned int shaderID = glCreateShader(type == "vert" ? GL_VERTEX_SHADER : GL_FRAGMENT_SHADER);

    const char* source = shaderSource.c_str();
    glShaderSource(shaderID, 1, &source, nullptr);
    glCompileShader(shaderID);
```

1.打開 shader 檔案，並把 shader 內容存在 shaderSource。

2.依據 shader 的種類，用 glCreateShader 建立 shader，再把 shaderSource 轉換成 c-string 後輸入 glShaderSource，設定好 shader 的 source，最後用 glCompilerShader 編譯。

```
    int success;
    glGetShaderiv(shaderID, GL_COMPILE_STATUS, &success);
    if (!success) {
        char infoLog[512];
        glGetShaderInfoLog(shaderID, 512, nullptr, infoLog);
        cerr << "Shader compilation error: " << infoLog << endl;
        return 0;
    }

    return shaderID;
}
```

3.用 glGetShaderiv 檢查 shader 是否編譯成功，若編譯失敗會印出錯誤訊息，成功則回傳 shaderID。

TODO#1-2: createProgram

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

    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);

    int success=0;
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success) {
        char infoLog[512];
        glGetProgramInfoLog(shaderProgram, 512, nullptr, infoLog);
        cerr << "Shader program linking error: " << infoLog << endl;
        return 0;
    }

    glDetachShader(shaderProgram, vertexShader);
    glDetachShader(shaderProgram, fragmentShader);


    return shaderProgram;

}
```

1. 用 glCreateProgram 建立一個 program。
2. 用 glAttachShader 把 vertex shader 跟 fragment shader 附著到 program 上，並用 glLinkProgram 把兩個 shader 連接在一起，形成一個完整的 shader program。
3. 用 glGetProgramiv 檢查 shader program 是否連接成功。
4. 確定連接成功後，因為 vertex shader 跟 fragment shader 不會再用到，所以用 glDetachShader 解除。

TODO#2: Load texture

```
unsigned int loadTexture(const char* tFileName) {
    unsigned int texture;
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &texture);
    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);
    stbi_set_flip_vertically_on_load(true);

    int width, height, nrChannels;
    unsigned char* data = stbi_load(tFileName, &width, &height, &nrChannels, 0);
    if (data) {
        glActiveTexture(GL_TEXTURE0);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, 0);
        stbi_image_free(data);
    }
    else {
        cerr << "Failed to load texture: " << tFileName << endl;
        return 0;
    }

    return texture;
}
```

1. 用 glEnable 跟 glGenTextures 生成 texture，並把 texture ID 存在變數 texture。
2. 用 glBindTexture 把 GL_TEXTURE_2D 綁到 texture。
3. 用 glTexParameteri 調整參數，分別使用了 GL_TEXTURE_MIN_FILTER 跟 GL_TEXTURE_MAG_FILTER，在縮小或放大十，會使用線性內插，進而提供較平滑的視覺效果。
4. 遇到問題：執行時發現衝浪板顏色分布左右相反，企鵝顏色分布上下相反，而如果將照片上下翻轉，就會呈現正常分布，所以這裡保留原本的照片，並用 stbi_set_flip_vertically_on_load 翻轉。
5. 用 stbi_data 載入圖片資訊，取得 data
6. 檢查是否成功取得 data，若成功取得，把 texture unit 設為 GL_TEXTURE0，把圖像用 glTexImage2D 數據傳給 OpenGL，用 glGenerateMipmap 生成 Mipmap，使縮小時能提供更好的效果，最後用 glBindTexture 把 bind 解開。
7. 用 sbti_image_free 釋放圖片 。
8. 回傳 texture 的 ID。

TODO#3: Set up VAO, VBO

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

    glGenBuffers(3, VBO);
```

1. 用 glGenVertexArrays 生成 VAO，再用 glBindVertexArray 生成 VAO bind。
2. 使用 glGenBuffers 生成三個 VBO，分別用來存儲頂點位置、法線和 texture 坐標的數據。

```
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT)*model.positions.size(), model.positions.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);


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


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


    glBindVertexArray(0);

    return VAO;

}
```

3. 接下來會用到三個 VBO 處理數據。第一個 VBO 處理 position，先用 glBindBuffer 綁定 VBO，接者用 glBufferData 把資料從 CPU 傳到 GPU，再來用 glVectoexAttribPointer 連接 vertex buffer 和 position 資料，最後用 glEnableVertexAttribArray 啟用 array。第二個跟第三個 VBO 執行步驟相同，只是第二個是處理 normals，第三個是處理 texcoods。
4. 用 glBindVertexArray 解綁 VAO，回傳 VAO。

```
unsigned int vertexShader, fragmentShader, shaderProgram;
vertexShader = createShader("vertexShader.vert", "vert");
fragmentShader = createShader("fragmentShader.frag", "frag");
shaderProgram = createProgram(vertexShader, fragmentShader);
unsigned int penguinTexture, boardTexture;
penguinTexture = loadTexture("obj/penguin_diffuse.jpg");
boardTexture = loadTexture("obj/surfboard_diffuse.jpg");
unsigned int penguinVAO, boardVAO;
penguinVAO = modelVAO(penguinModel);
boardVAO = modelVAO(boardModel);
```

用 TODO#1、TODO#2 和 TODO#3 的函式處理 Shader, Program, Texture 跟 VAO。

TODO#4 & TODO#5: Data connection, Render Board, Render Penguin

```
void drawModel(const string& target, unsigned int& shaderProgram, const glm::mat4& M, const glm::mat4& V, const glm::mat4& P,
    unsigned int& vao, unsigned int& texture)
{
    unsigned int mLoc, vLoc, pLoc;
    mLoc = glGetUniformLocation(shaderProgram, "M");
    vLoc = glGetUniformLocation(shaderProgram, "V");
    pLoc = glGetUniformLocation(shaderProgram, "P");
    glUniformMatrix4fv(mLoc, 1, GL_FALSE, glm::value_ptr(M));
    glUniformMatrix4fv(vLoc, 1, GL_FALSE, glm::value_ptr(V));
    glUniformMatrix4fv(pLoc, 1, GL_FALSE, glm::value_ptr(P));

    int useGrayscaleLoc = glGetUniformLocation(shaderProgram, "useGrayscale");
    int squeezeFactorLoc = glGetUniformLocation(shaderProgram, "squeezeFactor");

    if (useGrayscaleLoc != -1) {
        glUniform1i(useGrayscaleLoc, useGrayscale ? 1 : 0);
    }
```

由於兩個物件的繪製過程類似，所以寫成 drawModel。

1. 用 glGetUniformLocation 找出 M(Model), V(View), P(Perspective)的位置，再用 glUniformMatrix4fv 把 4*4 的矩陣傳到對應的位置。
2. 用 glGenUniformLocation 找到 useGrayscale 跟 squeezeFactor 的位置。如果有找到，就用 glUniform1i 把整數值傳相對的位置。

```
if (target == "board") {

    if (squeezeFactorLoc != -1) {
        glUniform1f(squeezeFactorLoc, glm::radians(0.0f));
    }

    glUniform1i(glGetUniformLocation(shaderProgram, "boardTexture"), 0);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glBindVertexArray(vao);
    glDrawArrays(GL_TRIANGLES, 0, boardModel.positions.size());

}
else if (target == "penguin") {

    if (squeezeFactorLoc != -1) {
        glUniform1f(squeezeFactorLoc, glm::radians(squeezeFactor));
    }
    glUniform1i(glGetUniformLocation(shaderProgram, "penguinTexture"), 0);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glBindVertexArray(vao);
    glDrawArrays(GL_TRIANGLES, 0, penguinModel.positions.size());


}
glBindVertexArray(0);
```

1. 如果是 board，因為 board 不會 squeeze，所以用 glUniform1f 傳浮點數值 0 到 squeezeFactor 的位置，texture 的位置後傳送 0。用 glActiveTexture 決定 texture unit，接著把用 glBindTexture 把 texture 綁到 GL_TEXTURE_2D 上，再用 glBindVertexArray 綁定頂點，最後用 glDrawArrays 根據 VAO 和 texture，從第 0 個元素，繪製到第 penguinModel.position.size 個元素。
2. 如果是 penguin，因為 penguin 會 squeeze，所以把變數 squeezeFactor 用 glUniform1f 傳到相對應的位置，其他步驟和 board 一樣，只是輸入的是 penguin 的資料。

Main function:

```
glUseProgram(shaderProgram);

glm::mat4 ModelMatrix = glm::mat4(1.0f);
glm::mat4 boardModelMatrix = glm::mat4(1.0f);
ModelMatrix = glm::translate(ModelMatrix, glm::vec3(0.0f, 0.0f, swingPos));
ModelMatrix = glm::translate(ModelMatrix, glm::vec3(0.0f, -0.5f, 0.0f));
ModelMatrix = glm::rotate(ModelMatrix, glm::radians(swingAngle), glm::vec3(0.0f, 1.0f, 0.0f));
boardModelMatrix = glm::rotate(ModelMatrix, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
boardModelMatrix = glm::scale(boardModelMatrix, glm::vec3(0.03f, 0.03f, 0.03f));

drawModel("board", shaderProgram, boardModelMatrix, view, perspective, boardVAO, boardTexture);
```

1. 用 glUseProgram 告訴 OpenGL 之後要用的 shader program
2. 把衝浪板的位置、前後移動和繞 y 軸 20~-20 度旋轉的動作存在 ModelMatrix，以便隨著衝浪板移動的企鵝使用。
3. 把對 x 軸旋轉-90 度和縮小的動作放在 boardModelMatrix 裡。
4. 把資料傳給 drawModel。

```
glm::mat4 penguinModelMatrix = glm::mat4(1.0f);
penguinModelMatrix = glm::translate(ModelMatrix, glm::vec3(0.0f, 0.5f, 0.0f));
penguinModelMatrix = glm::rotate(penguinModelMatrix, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
penguinModelMatrix = glm::scale(penguinModelMatrix, glm::vec3(0.025f, 0.025f, 0.025f));

drawModel("penguin",shaderProgram, penguinModelMatrix, view, perspective, penguinVAO, penguinTexture);
```

1. 企鵝的位置在(0,0,0)，衝浪板在(0,-0.5,0)，所以企鵝在衝浪板上 0.5 的位置。把 ModelMatrix 乘上移動、旋轉跟縮小動作，存在 penguinModelMatrix 裡，傳給 drawModel。

TODO#6: Update "swingAngle", "swingPos", "squeezeFactor"

```
swingAngle = swingAngle +20.0f * swingAngleDir * dt;
if (swingAngle >= 20.0f) {
    swingAngleDir = -1;
}
else if (swingAngle <= -20.0f) {
    swingAngleDir = 1;
}

swingPos = swingPos + 1* swingPosDir * dt;
if (swingPos >= 2) {
    swingPosDir = -1;
}
else if(swingPos<=0) {
    swingPosDir = 1;
}

if (squeezing) {
    squeezeFactor += 90.0f * dt;
}
```

1. 根據和上一個畫面的時間差更新 swingAngle 和 swingPos。
2. 如果 swingAngle 在 20 度以上或-20 度以下，就換方向。
3. 如果 swingPos 在 2 以上或 0 以下，就換方向。
4. 如果 squeezing 是 true，就根據時間差更新 squeezeFactor。

TODO#7: Key callback

```
if (key == GLFW_KEY_S && action == GLFW_PRESS)
{
    cout << "KEY S PRESSED\n";
    if (squeezing) {
        squeezing = false;
    }
    else {
        squeezing = true;
    }

}

if (key == GLFW_KEY_G && action == GLFW_PRESS) {
    cout << "KEY G PRESSED\n";
    if (useGrayscale) {
        useGrayscale = false;
    }
    else {
        useGrayscale = true;
    }

}
```

1. 按下 s 鍵，如果 squeezing 是 true 就改成 false，false 就改成 true
2. 按下 g 鍵，如果 useGrayscale 是 true 就改成 false，false 就改成 true

vertexShader.vert

```
vec3 squeezedPos = aPos;
squeezedPos.y += aPos.z * sin(squeezeFactor) / 2.0;
squeezedPos.z += aPos.y * sin(squeezeFactor) / 2.0;

worldPos = M * vec4(squeezedPos, 1.0);

gl_Position = P * V * worldPos;

mat3 normalMatrix = transpose(inverse(mat3(M)));
normal = normalize(normalMatrix * aNormal);

texCoord = aTexCoord;
```

1. 跟 squeezeFactor 調整，並且把 squeeze 後的位置存到 squeezePos
2. 用 M 跟 squeezePos 更新 worldPos
3. 用 P, V, worldPos 計算 glPos
4. 用 M 跟 aNormal 找出 global normal
5. texCoord 設為 aTexCoord

fragmentShader.frag

```
vec4 color = texture2D(ourTexture, texCoord);
if (useGrayscale) {
    float grayscaleValue = dot(color.rgb, vec3(0.299, 0.587, 0.114));

    FragColor = vec4(vec3(grayscaleValue), color.a);
} else {
    FragColor = color;
}
```

1. 用 texture2D 取得 texture 在 texCord 位置的顏色。
2. 如果 useGraysclae 是 true，FragColor 就設成灰階；如果是 False，FragColor 就設為 color。

Creativity:

vertexShader.vertex:

```
uniform float squeezeFactor;
uniform float offest;
uniform float time;
uniform bool tremble;
```

新增 time 跟 tremble 兩個變數。Time 是當下的時間，tremble 是要不要發抖。

```
if(tremble){
    float amplitude = 1;
    float frequency = 10.0;
    float randValue = random(squeezedPos.xz, time);
    vec3 deform = squeezedPos + amplitude * sin(frequency * time) * normalize(aNormal) * (randValue * 2.0 - 1.0);

    vec4 finalPosition = M * vec4(deform, 1.0);
    gl_Position = P * V * finalPosition ;

}
```

```
float random(vec2 uv, float seed) {
    return fract(sin(dot(uv, vec2(12.9898, 78.233)) + seed) * 43758.5453);
}
```

如果 tremble 是 true，利用 x,z 位置跟時間取得界在[0,1)的隨機變數，再用取得的隨機變數(乘以 2 減 1 使變數界在[-1,1))成以振幅、sin 跟 normal 改變點的位置，最後把位置的變化存到 gl_Position。

Main.cpp

```
if (key == GLFW_KEY_T && action == GLFW_PRESS) {
    cout << "KEY T PRESSED\n";
    if (tremble) {
        tremble = false;
    }
    else {
        tremble = true;
    }

}
```

按 T 控制要不要發抖。

```
glUniform1i(glGetUniformLocation(shaderProgram, "tremble"), tremble?1:0);
glUniform1f(glGetUniformLocation(shaderProgram, "time"), currentTime);
```

在 drawModel 的地方，如果對象是企鵝，就用 glGetUniformLocation 找變數的位置，並用 glUniform1i 跟 glUniform1f 傳變數到是當的位置。