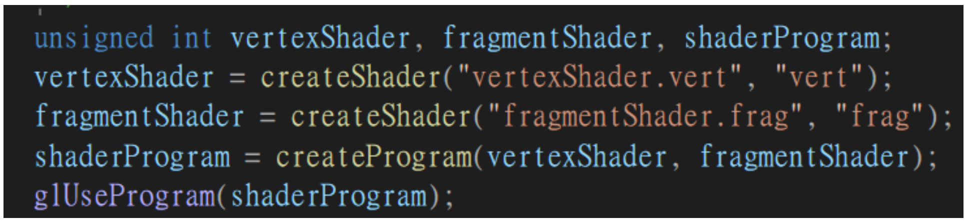
HW2 Report

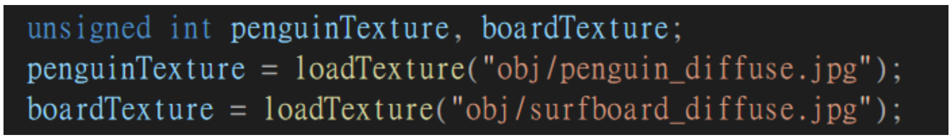
110550071 田松翰

1. Create shader and program. The detail of function refers to HW1.



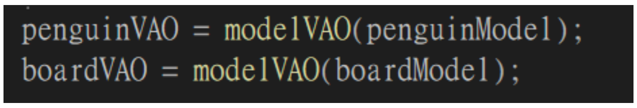
```
unsigned int vertexShader, fragmentShader, shaderProgram;
vertexShader = createShader("vertexShader.vert", "vert");
fragmentShader = createShader("fragmentShader.frag", "frag");
shaderProgram = createProgram(vertexShader, fragmentShader);
glUseProgram(shaderProgram);
```

2. Load texture. The detail of function refers to HW1.



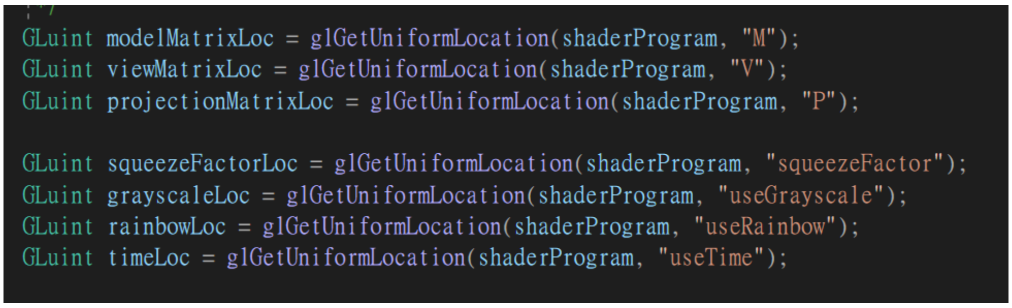
```
unsigned int penguinTexture, boardTexture;
penguinTexture = loadTexture("obj/penguin_diffuse.jpg");
boardTexture = loadTexture("obj/surfboard_diffuse.jpg");
```

3. Set up VAO. The detail of function refers to HW1.



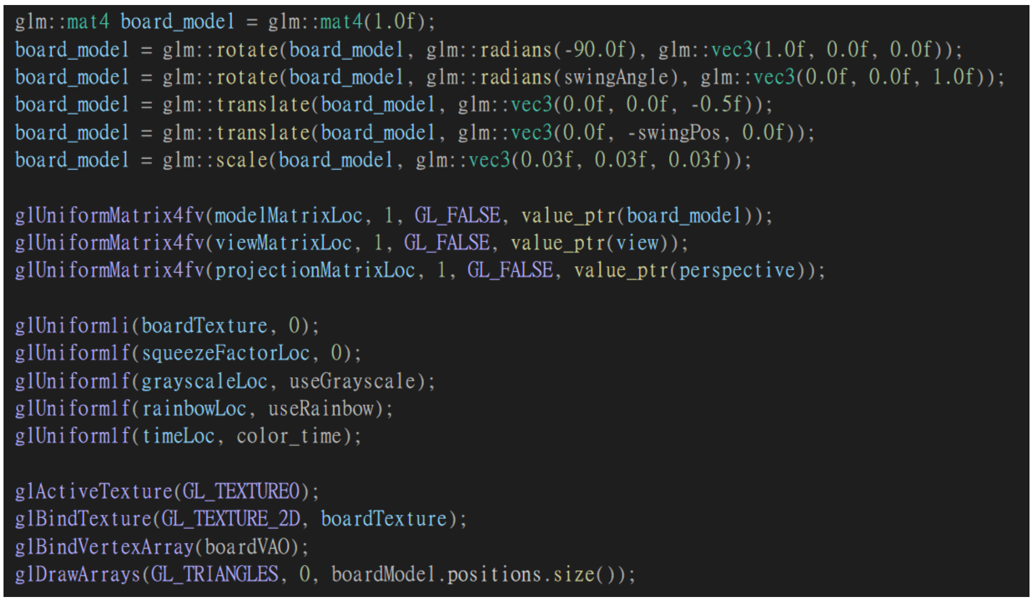
```
penguinVAO = modelVAO(penguinModel);
boardVAO = modelVAO(boardModel);
```

4. Data connection: get the uniform variables locations.



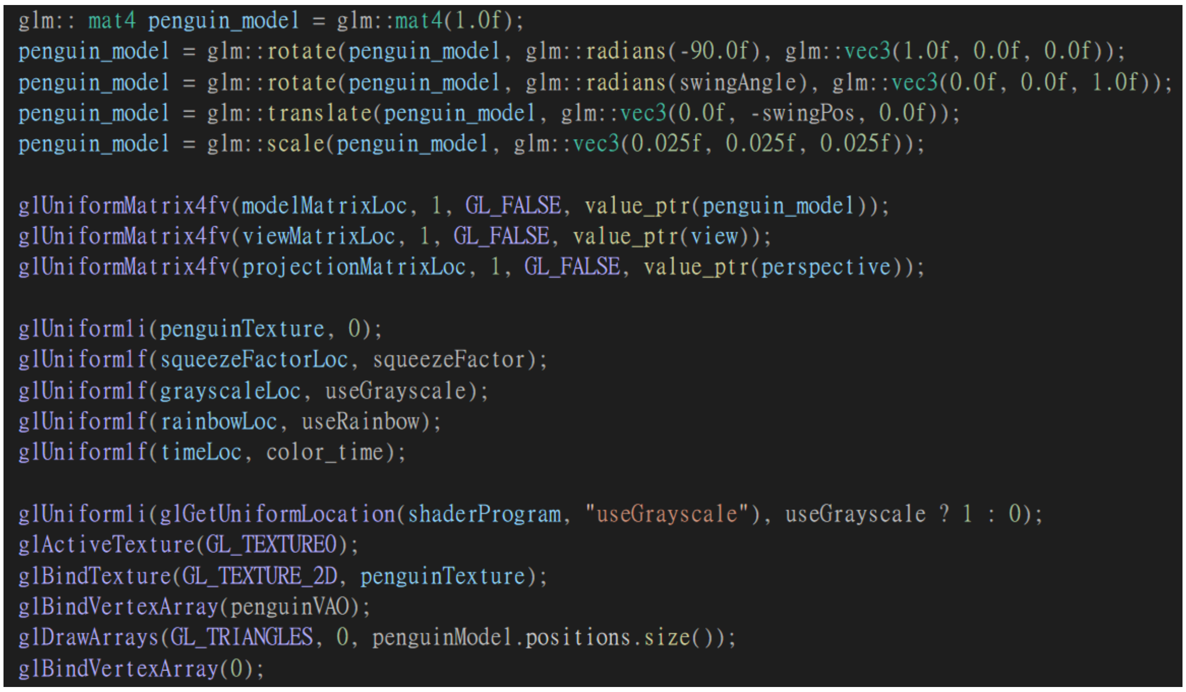
```
GLuint modelMatrixLoc = glGetUniformLocation(shaderProgram, "M");
GLuint viewMatrixLoc = glGetUniformLocation(shaderProgram, "V");
GLuint projectionMatrixLoc = glGetUniformLocation(shaderProgram, "P");

GLuint squeezeFactorLoc = glGetUniformLocation(shaderProgram, "squeezeFactor");
GLuint grayscaleLoc = glGetUniformLocation(shaderProgram, "useGrayscale");
GLuint rainbowLoc = glGetUniformLocation(shaderProgram, "useRainbow");
GLuint timeLoc = glGetUniformLocation(shaderProgram, "useTime");
```

5-1. Render board: set board model, send uniform matrices to shaders.



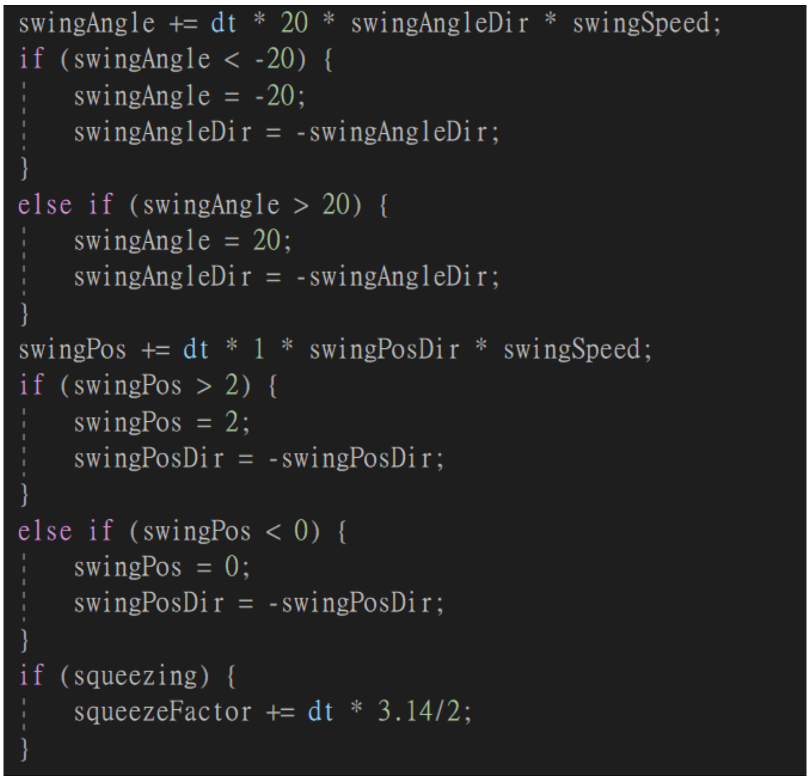
```
glm::mat4 board_model = glm::mat4(1.0f);
board_model = glm::rotate(board_model, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
board_model = glm::rotate(board_model, glm::radians(swingAngle), glm::vec3(0.0f, 0.0f, 1.0f));
board_model = glm::translate(board_model, glm::vec3(0.0f, 0.0f, -0.5f));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

glUniformMatrix4fv(modelMatrixLoc, 1, GL_FALSE, value_ptr(board_model));
glUniformMatrix4fv(viewMatrixLoc, 1, GL_FALSE, value_ptr(view));
glUniformMatrix4fv(projectionMatrixLoc, 1, GL_FALSE, value_ptr(perspective));

glUniform1i(boardTexture, 0);
glUniform1f(squeezeFactorLoc, 0);
glUniform1f(grayscaleLoc, useGrayscale);
glUniform1f(rainbowLoc, useRainbow);
glUniform1f(timeLoc, color_time);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, boardTexture);
glBindVertexArray(boardVAO);
glDrawArrays(GL_TRIANGLES, 0, boardModel.positions.size());
```

5-2. Render penguin: set penguin model, send uniform matrices to shaders.



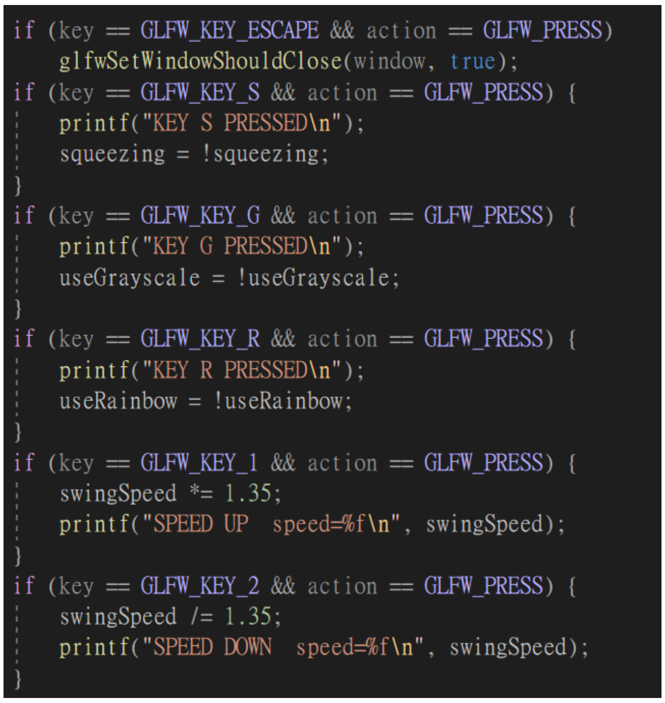
```
glm:: mat4 penguin_model = glm::mat4(1.0f);
penguin_model = glm::rotate(penguin_model, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
penguin_model = glm::rotate(penguin_model, glm::radians(swingAngle), glm::vec3(0.0f, 0.0f, 1.0f));
penguin_model = glm::translate(penguin_model, glm::vec3(0.0f, -swingPos, 0.0f));
penguin_model = glm::scale(penguin_model, glm::vec3(0.025f, 0.025f, 0.025f));

glUniformMatrix4fv(modelMatrixLoc, 1, GL_FALSE, value_ptr(penguin_model));
glUniformMatrix4fv(viewMatrixLoc, 1, GL_FALSE, value_ptr(view));
glUniformMatrix4fv(projectionMatrixLoc, 1, GL_FALSE, value_ptr(perspective));

glUniform1i(penguinTexture, 0);
glUniform1f(squeezeFactorLoc, squeezeFactor);
glUniform1f(grayscaleLoc, useGrayscale);
glUniform1f(rainbowLoc, useRainbow);
glUniform1f(timeLoc, color_time);

glUniform1i(glGetUniformLocation(shaderProgram, "useGrayscale"), useGrayscale ? 1 : 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, penguinTexture);
glBindVertexArray(penguinVAO);
glDrawArrays(GL_TRIANGLES, 0, penguinModel.positions.size());
glBindVertexArray(0);
```

6. Update variables: swingAngle for rotation, swingPos for translate, swingSpeed for bonus parts.
   Bonus: increase or decrease swing speed.



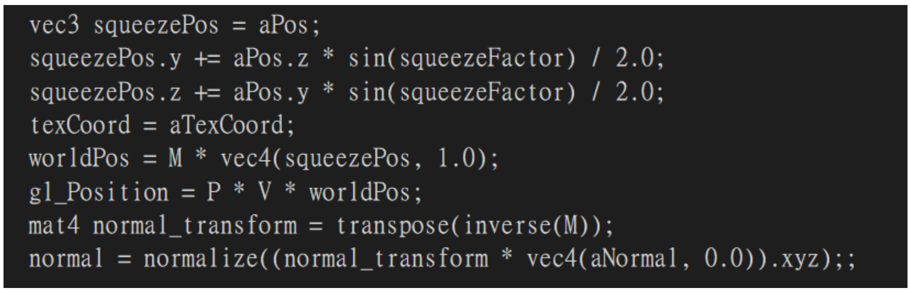
```
swingAngle += dt * 20 * swingAngleDir * swingSpeed;
if (swingAngle < -20) {
    swingAngle = -20;
    swingAngleDir = -swingAngleDir;
}
else if (swingAngle > 20) {
    swingAngle = 20;
    swingAngleDir = -swingAngleDir;
}
swingPos += dt * 1 * swingPosDir * swingSpeed;
if (swingPos > 2) {
    swingPos = 2;
    swingPosDir = -swingPosDir;
}
else if (swingPos < 0) {
    swingPos = 0;
    swingPosDir = -swingPosDir;
}
if (squeezing) {
    squeezeFactor += dt * 3.14/2;
}
```

7. Key callback: esc for escape, s for squeezing, g for gray scale.
   Bonus: r for rainbow effect, 1 for speed up, 2 for speed down.



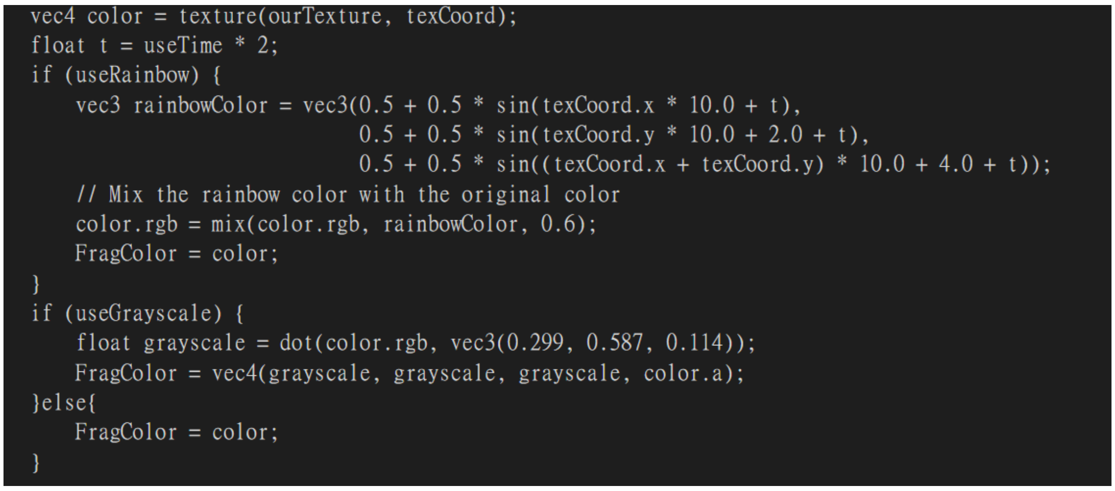
```
if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    glfwSetWindowShouldClose(window, true);
if (key == GLFW_KEY_S && action == GLFW_PRESS) {
    printf("KEY S PRESSED\n");
    squeezing = !squeezing;
}
if (key == GLFW_KEY_G && action == GLFW_PRESS) {
    printf("KEY G PRESSED\n");
    useGrayscale = !useGrayscale;
}
if (key == GLFW_KEY_R && action == GLFW_PRESS) {
    printf("KEY R PRESSED\n");
    useRainbow = !useRainbow;
}
if (key == GLFW_KEY_1 && action == GLFW_PRESS) {
    swingSpeed *= 1.35;
    printf("SPEED UP  speed=%f\n", swingSpeed);
}
if (key == GLFW_KEY_2 && action == GLFW_PRESS) {
    swingSpeed /= 1.35;
    printf("SPEED DOWN  speed=%f\n", swingSpeed);
}
```

8. Vertex shader: applying squeezing effect and return gl_Position and normal.
   If Squeezing variables in main.cpp is on, the vertex will start squeezing.
   Otherwise, the object will stay the appearance when the variables turns off.



```
vec3 squeezePos = aPos;
squeezePos.y += aPos.z * sin(squeezeFactor) / 2.0;
squeezePos.z += aPos.y * sin(squeezeFactor) / 2.0;
texCoord = aTexCoord;
worldPos = M * vec4(squeezePos, 1.0);
gl_Position = P * V * worldPos;
mat4 normal_transform = transpose(inverse(M));
normal = normalize((normal_transform * vec4(aNormal, 0.0)).xyz);;
```

9. Fragment shader: if uniform useGrayscale is true, grayscale effect will be applied.
   Rainbow effect will be detailed in bonus.

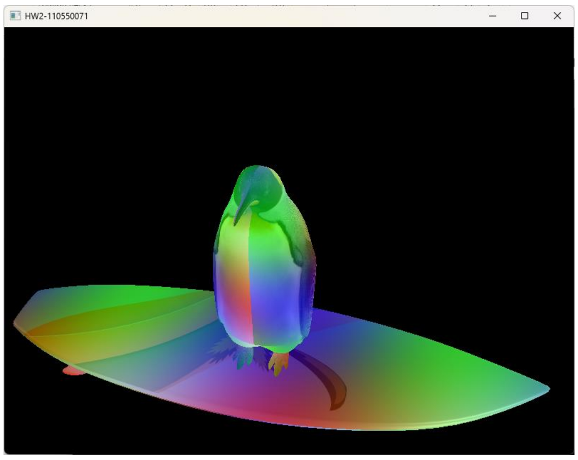


```
vec4 color = texture(ourTexture, texCoord);
float t = useTime * 2;
if (useRainbow) {
    vec3 rainbowColor = vec3(0.5 + 0.5 * sin(texCoord.x * 10.0 + t),
                             0.5 + 0.5 * sin(texCoord.y * 10.0 + 2.0 + t),
                             0.5 + 0.5 * sin((texCoord.x + texCoord.y) * 10.0 + 4.0 + t));
    // Mix the rainbow color with the original color
    color.rgb = mix(color.rgb, rainbowColor, 0.6);
    FragColor = color;
}
if (useGrayscale) {
    float grayscale = dot(color.rgb, vec3(0.299, 0.587, 0.114));
    FragColor = vec4(grayscale, grayscale, grayscale, color.a);
}else{
    FragColor = color;
}
```

10. Bonus:

1. Rainbow effect: If we press "R", rainbow color will be applied on the model. The color will be varied with the coordinate and the time.

```
if (useRainbow) {
    vec3 rainbowColor = vec3(0.5 + 0.5 * sin(texCoord.x * 10.0 + t),
                             0.5 + 0.5 * sin(texCoord.y * 10.0 + 2.0 + t),
                             0.5 + 0.5 * sin((texCoord.x + texCoord.y) * 10.0 + 4.0 + t));
    // Mix the rainbow color with the original color
    color.rgb = mix(color.rgb, rainbowColor, 0.6);
    FragColor = color;
}
```



2. Speed variation: As HW1, I designed the speed control function to control the speed. Press 1 to speed up and 2 to speed down.

```
if (key == GLFW_KEY_1 && action == GLFW_PRESS) {
    swingSpeed *= 1.35;
    printf("SPEED UP  speed=%f\n", swingSpeed);
}
if (key == GLFW_KEY_2 && action == GLFW_PRESS) {
    swingSpeed /= 1.35;
    printf("SPEED DOWN  speed=%f\n", swingSpeed);
}
```

```
 swingAngle += dt * 20 * swingAngleDir * swingSpeed;
swingPos += dt * 1 * swingPosDir * swingSpeed;
```

Problems I met:

1. I can't make any object showed on the window in the beginning.

➔ I found that I give the different name on the variables for glGetUniformLocation, that the names must be same with the vertex file. Hence, I'm clear to me for how the function works between each file.

2. The directions of the models are wrong for surfing.

➔ I misunderstood the direction after rotation, so I rotate my hands to find the correct direction step by step and I get the right result.