HW3 Report

報告內容主要針對 HW2 以後更新的部分撰寫。

main.cpp

1. Parameters

   先設定變數以及使用助教設定好的變數進行後續程式撰寫。在 PongMaterial 結構裡，會用到 isEye 來辨別該材質是否為眼睛的材質。

```
typedef struct
{
    Vector3 Ka;
    Vector3 Kd;
    Vector3 Ks;

    GLuint diffuseTexture;

    // eye texture coordinate
    GLuint isEye;
    vector<Offset> offsets;

} PhongMaterial;
```

   在 model 的結構裡，會使用到 hasEye 來判別該模型是否具有獨立眼睛 材質的部分，max_eye_offset 為最大數量的不同眼睛型態個數，
   cur_eye_offset_idx 為目前處在哪個眼睛轉換的狀態。

```
struct model
{
    Vector3 position = Vector3(0, 0, 0);
    Vector3 scale = Vector3(1, 1, 1);
    Vector3 rotation = Vector3(0, 0, 0);    // Euler form

    vector<Shape> shapes;

    bool hasEye;
    GLint max_eye_offset = 7;
    GLint cur_eye_offset_idx = 0;
};
vector<model> models;
```

   在上次作業中設定好的 uniform 架構中加入 iLocIsEye 及 Offset 的變數用於之後傳入 fragment shader 所使用。

```
    GLuint iLocper_vertex;
    GLuint iLocLight_id;
    GLuint iLocSpot_cutoff;

    GLint iLocIsEye;
    //GLint iLocEye_id;
    GLfloat iLocOffset_x;
    GLfloat iLocOffset_y;

};
Uniform uniform;
```

設定 offset_x 及 offset_y 作為 eye texture transform 位置的紀錄。Mag 則用於紀錄目前的 magnification filtering mode 為 nearest 或 linear sampling，預設為 nearest sampling。mini 用於紀錄目前的 minification texture filtering mode 為 nearest_mipmap_linear 或 linear_mipmap_linear sampling，預設為 nearest_mipmap_linear sampling。

```
int cur_idx = 0; // represent which model should be rendered now
int cur_light_id = 0; //0:directional,1:point,2:spot
int per_vertex = 0; //per-vertex shading
float shininess = 64.0f;
float spot_cutoff = 30;
float offset_x = 0;
float offset_y = 0;

bool mag = 1; //magnification texture filtering mode(1:nearest, 0:linear)
bool mini = 1; //minification texture filtering mode(1:nearest, 0:linear_mipmap_linear)
vector<string> model_list{ "../TextureModels/Fushigidane.obj", "../TextureModels/Mew.obj","
```

設定 iLocTex 全域變數紀錄本地端的 texture，用於後續傳入 fragment shader 所用。

```
// uniforms location
GLuint iLocP;
GLuint iLocV;
GLuint iLocM;

GLuint iLocTex;
//GLuint iLocTexEye;
```

2. 在 RenderScene()中，利用傳入的參數來辨別 per-vertex lighting 或 per-pixel lighting。先以 per-vertex lighting 為例，在 for 迴圈中設定 per_vertex 狀態為 1，並將此參數利用 uniform 傳入 shader 中。接著將當前 material 是否為全部都是眼睛的 isEye 參數、紀錄眼睛位置 offset 的 offset_x、offset_y 參數裡用 uniform 函數傳到 shader 中。接著利用 glActiveTexture()函數設定目前的 active texture unit 為 texture unit 0，並利用 glBindTexture()綁定模型的 texture。

```
if(!per_vertex_or_per_pixel)
{
    for (int i = 0; i < models[cur_idx].shapes.size(); i++)
    {
        per_vertex = 1;
        glUniform1i(uniform.iLocper_vertex, per_vertex);
        glUniform1ui(uniform.iLocIsEye, models[cur_idx].shapes[i].material.isEye);
        glUniform1f(uniform.iLocOffset_x, offset_x);
        glUniform1f(uniform.iLocOffset_y, offset_y);
        glBindVertexArray(models[cur_idx].shapes[i].vao);

        // [TODO] Bind texture and modify texture filtering & wrapping mode
        // Hint: glActiveTexture, glBindTexture, glTexParameteri
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, models[cur_idx].shapes[i].material.diffuseTexture);
```

在 texture parameter 的部分，利用先前設定的變數來控制 magnification texture 和 minification texture filter 的參數。

```
        if (mag)
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        }
        else
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        }

        if (mini)
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
        }
        else
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        }
        glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
        glUniform3fv(uniform.iLocKa, 1, &(models[cur_idx].shapes[i].material.Ka[0]));
        glUniform3fv(uniform.iLocKd, 1, &(models[cur_idx].shapes[i].material.Kd[0]));
        glUniform3fv(uniform.iLocKs, 1, &(models[cur_idx].shapes[i].material.Ks[0]));
    }
}
```

除了傳入的 per_vertex 參數不同之外，Per-pixel lighting 的做法和 per-vertex lighting 的做法一樣。

```
if (per_vertex_or_per_pixel)
{
    for (int i = 0; i < models[cur_idx].shapes.size(); i++)
    {
        per_vertex = 0;
        glUniform1i(uniform.iLocper_vertex, per_vertex);
        glUniform1ui(uniform.iLocIsEye, models[cur_idx].shapes[i].material1.isEye);
        glUniform1f(uniform.iLocOffset_x, offset_x);
        glUniform1f(uniform.iLocOffset_y, offset_y);
        glBindVertexArray(models[cur_idx].shapes[i].vao);

        // [TODO] Bind texture and modify texture filtering & wrapping mode
        // Hint: glActiveTexture, glBindTexture, glTexParameteri
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, models[cur_idx].shapes[i].material1.diffuseTexture);
```

```
        if (mag)
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        }
        else
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        }

        if (mini)
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
        }
        else
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        }
        glDrawArrays(GL_TRIANGLES, 0, models[cur_idx].shapes[i].vertex_count);
        glUniform3fv(uniform.iLocKa, 1, &(models[cur_idx].shapes[i].material1.Ka[0]));
        glUniform3fv(uniform.iLocKd, 1, &(models[cur_idx].shapes[i].material1.Kd[0]));
        glUniform3fv(uniform.iLocKs, 1, &(models[cur_idx].shapes[i].material1.Ks[0]));
    }
}
```

3. 在 KeyCallback()函數中，利用按鍵 G 可以切換當前 magnification texture 的 filtering mode，利用按鍵 B 可以切換當前 minification texture 的 filtering mode。

```
case GLFW_KEY_G:
    mag = !mag;
    break;
case GLFW_KEY_B:
    mini = !mini;
    break;
```

右鍵為控制模型的 eye texture transformation，根據助教的 demo 執行檔，按右鍵會同時影響所有具有 eye texture transformation 的模型，即使是在沒有 eye texture transformation 的模型中按右鍵也會影響到，所以先利用第一個 for 迴圈更新模型的 cur_eye_offset_idx，為方便，更新所有模型的 cur_eye_offset_idx。接著，根據作業 texture model 的規格，texture coordinate 會介在 0 到 1 之間，因此根據眼睛 transformation 狀態的個數去計算每個眼睛狀態的位置為何，眼睛狀態的採樣點為左上角。最後根據右鍵的操作及目前眼睛 transformation 狀態去計算右鍵操作後的眼睛狀態為何。

```
case GLFW_KEY_RIGHT:
    for (int i = 0; i < models.size();i++)
    {
        models[i].cur_eye_offset_idx = (models[i].cur_eye_offset_idx + 1) % models[i].max_eye_offset;
    }
    cout << "cur_id" << models[cur_idx].cur_eye_offset_idx << endl;
    if (models[cur_idx].cur_eye_offset_idx > 0 && models[cur_idx].cur_eye_offset_idx < 4)
    {
        offset_x = 0;
        offset_y -= 0.25;
    }
    if (models[cur_idx].cur_eye_offset_idx > 4 && models[cur_idx].cur_eye_offset_idx < 7)
    {
        offset_x = 0.50;
        offset_y -= 0.25;
    }
    if (models[cur_idx].cur_eye_offset_idx == 0)
    {
        offset_x = 0;
        offset_y = 0;
    }
    if (models[cur_idx].cur_eye_offset_idx == 4)
    {
        offset_x = 0.50;
        offset_y = 0;
    }
    break;
```

左鍵的做法和右鍵基本上一樣，僅有在按鍵操作中，位置的 offset 有小小計算上的不同。

```
case GLFW_KEY_LEFT:
    for (int i = 0; i < models.size(); i++)
    {
        models[i].cur_eye_offset_idx = (models[i].cur_eye_offset_idx - 1 + models[i].max_eye_offset) % models[i].max_eye_offset;
    }
    cout << "cur_id" << models[cur_idx].cur_eye_offset_idx << endl;
    if (models[cur_idx].cur_eye_offset_idx >= 0 && models[cur_idx].cur_eye_offset_idx < 4)
    {
        offset_x = 0;
        offset_y += 0.25;
    }
    if (models[cur_idx].cur_eye_offset_idx > 4 && models[cur_idx].cur_eye_offset_idx < 7)
    {
        offset_x = 0.50;
        offset_y += 0.25;
    }
    if (models[cur_idx].cur_eye_offset_idx == 4)
    {
        offset_x = 0.5;
        offset_y = 0;
    }
    if (models[cur_idx].cur_eye_offset_idx == 6)
    {
        offset_x = 0.50;
        offset_y = -0.5;
    }
    break;
```

4. 在 LoadTextureImage()中，利用 glGenTexture()輸入需要生成的 texture 數量，並將該 texture 傳遞至此函數裡，接著利用 glBindTexture()進行圖片 texture 的綁定，利用 glTexIamge2D()可以生成前面綁定的 texture，最後利用 glGerateMipmap()生成該 texture 的 mipmap。

```
GLuint LoadTextureImage(string image_path)
{
    int channel, width, height;
    int require_channel = 4;
    stbi_set_flip_vertically_on_load(true);
    stbi_uc *data = stbi_load(image_path.c_str(), &width, &height, &channel, require_channel);
    if (data != NULL)
    {
        GLuint tex = 0;

        // [TODO] Bind the image to texture
        // Hint: glGenTextures, glBindTexture, glTexImage2D, glGenerateMipmap
        glGenTextures(1, &tex);
        glBindTexture(GL_TEXTURE_2D, tex);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        // free the image from memory after binding to texture
        stbi_image_free(data);
        return tex;
    }
    else
    {
        cout << "LoadTextureImage: Cannot load image from " << image_path << endl;
        return -1;
    }
}
```

在 LoadTexturedModels()裡會呼叫 LoadTextureImage()將模型中的 image texture 轉換成 texture mipmap，並將它存放在 diffuseTexture 中。在讀取每個 material 的同時，會利用判斷檔名中是否含有 Eye 這個子字串，若有，則將該 material 的 isEye 設為 true，沒有則設為 false。

```
for (int i = 0; i < materials.size(); i++)
{
    PhongMaterial material;
    material.Ka = Vector3(materials[i].ambient[0], materials[i].ambient[1], materials[i].ambient[2]);
    material.Kd = Vector3(materials[i].diffuse[0], materials[i].diffuse[1], materials[i].diffuse[2]);
    material.Ks = Vector3(materials[i].specular[0], materials[i].specular[1], materials[i].specular[2]);

    //cout << string(materials[i].diffuse_texname) << endl;
    string::size_type idx;
    idx = string(materials[i].diffuse_texname).find("Eye");
    if (idx == string::npos)
    {
        //cout << "not found" << endl;
        material.isEye = false;
    }
    else
    {
        //cout << "found" << endl;
        material.isEye = true;
        tmp_model.hasEye = true;
    }

    material.diffuseTexture = LoadTextureImage(base_dir + string(materials[i].diffuse_texname));
    if (material.diffuseTexture == -1)
    {
        cout << "LoadTexturedModels: Fail to load model's material " << i << endl;
        system("pause");

    }

    allMaterial.push_back(material);
    //cout << "material diffuse" << material.diffuseTexture << endl;
}
```

本地端的 texture 參數則在 setUniformVariables()將變數位置和對應的值傳遞給 shader。

```
// [TODO] Get uniform location of texture
iLocTex = glGetUniformLocation(program, "tex");
glUniform1i(iLocTex, 0);
//iLocTexEye = glGetUniformLocation(program, "tex_eye");
```

shader.vs.glsl

```
void main()
{
    // [TODO]
    texCoord = aTexCoord;
    gl_Position = um4p * um4v * um4m * vec4(aPos, 1.0);

    vec4 fragPos = um4m * vec4(aPos,1.0);
    FragPos = fragPos.xyz;
    vertex_normal = mat3(transpose(inverse(um4m))) * aNormal;
    texCoord = aTexCoord;
    if(per_vertex == 0)
    {
        vec3 color = vec3(0,0,0);
        if(cur_light_id == 0)
        {
            color += directional_light();
        }
        else if(cur_light_id == 1)
        {
            color += point_light();
        }
        else if(cur_light_id == 2)
        {
            color += spot_light();
        }

        vertex_color = color;
    }
}
```

shader.fs.glsl

1. 宣告先前利用 uniform 函數所傳入的變數。

```
#version 330

in vec2 texCoord;
in vec3 vertex_color;
in vec3 vertex_normal;
in vec3 FragPos;

out vec4 fragColor;

// [TODO] passing texture from main.cpp
// Hint: sampler2D
uniform sampler2D tex;
```

```
uniform bool is_eye;
uniform float offset_x;
uniform float offset_y;
```

利用 is_eye 判斷目前的 texture 是否為眼睛的 texture，如果是的話，根據傳入的 offset 進行 eye texture transformation。其餘的 texture 則維持在原本的位置上。

```
void main() {
    //fragColor = vec4(texCoord.xy, 0, 1);
    if(per_vertex == 1)
    {
        vec3 color = vec3(0,0,0);
        if(cur_light_id == 0)
        {
            color += directional_light();
        }
        else if(cur_light_id == 1)
        {
            color += point_light();
        }
        else if(cur_light_id == 2)
        {
            color += spot_light();
        }
        // [TODO] sampleing from texture
        // Hint: texture
        vec4 glColor = vec4(color, 1.0);

        if(is_eye)
        {
            fragColor = vec4(texture(tex, vec2(texCoord.x+offset_x,texCoord.y+offset_y)).rgb,1.0) * glColor;
        }
        else
        {
            fragColor = vec4(texture(tex, vec2(texCoord.x,texCoord.y)).rgb,1.0) * glColor;
        }
    }
```

```
    else
    {
        vec4 glColor = vec4(vertex_color, 1.0);
        if(is_eye)
        {
            fragColor = vec4(texture(tex, vec2(texCoord.x+offset_x,texCoord.y+offset_y)).rgb,1.0) * glColor;
        }
        else
        {
            fragColor = vec4(texture(tex, vec2(texCoord.x,texCoord.y)).rgb,1.0) * glColor;
        }
    }

}
```

結果展示

State1

State2

State3

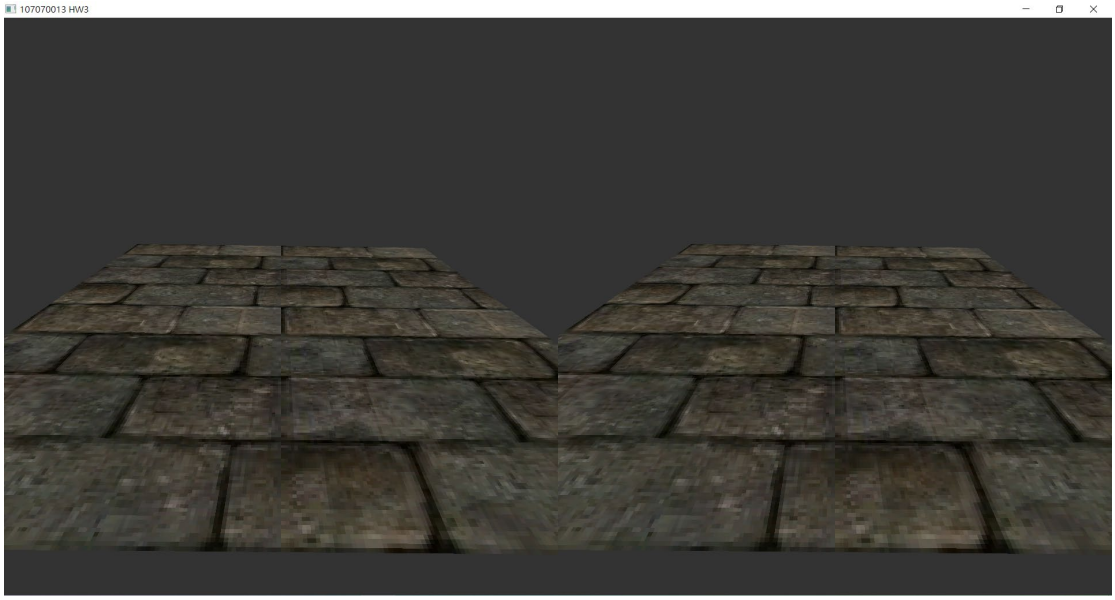
State4

State5

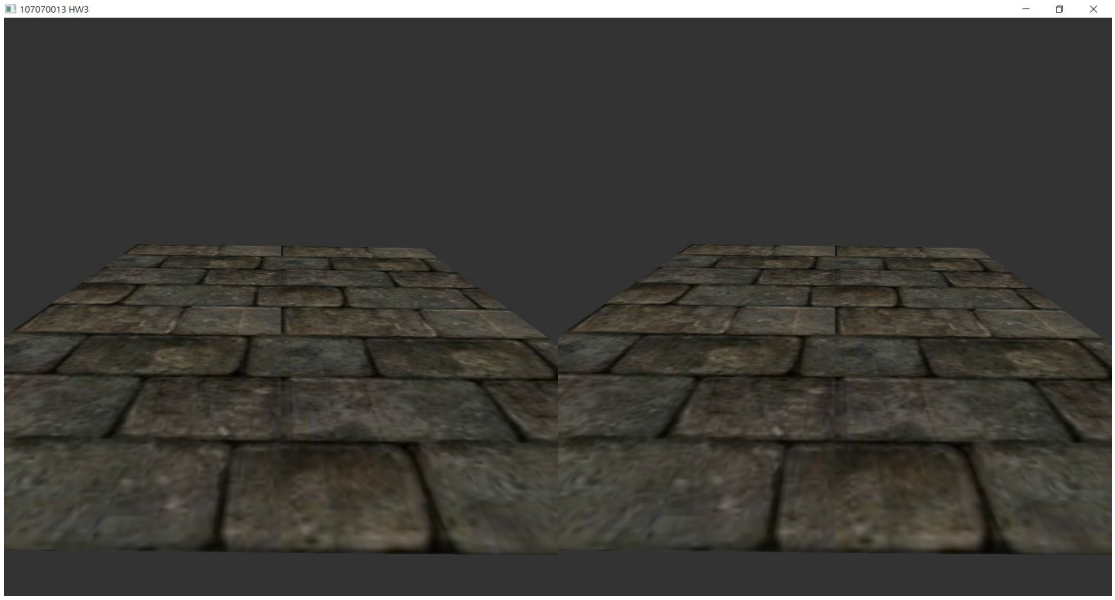
State6

State7

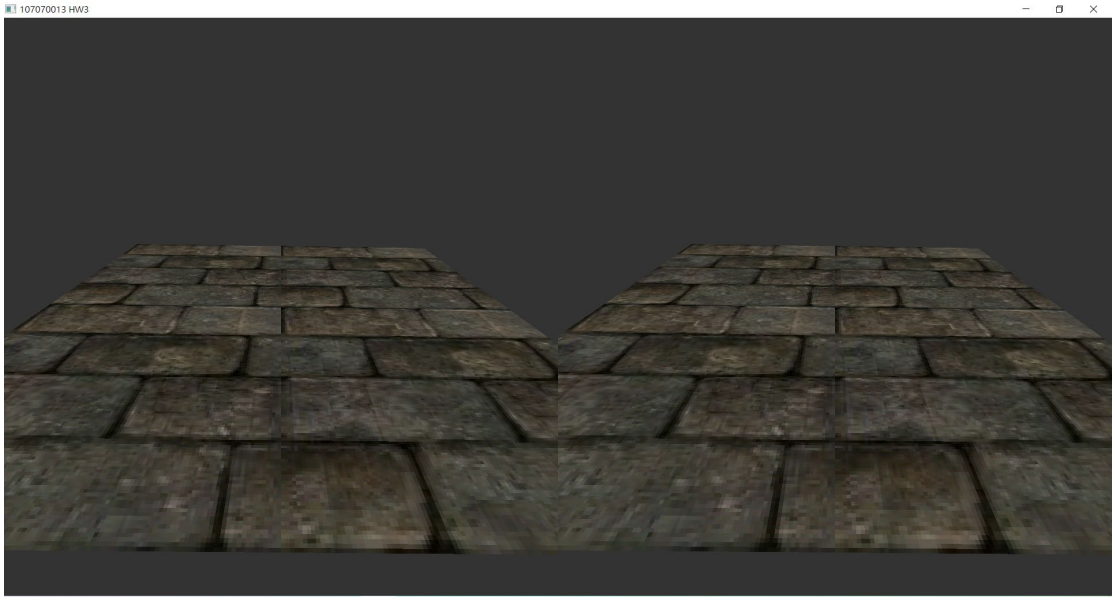
G: magnification nearest sampling



G: magnification linear sampling



B: minification nearest_mipmap_linear sampling



B: minification linear_mipmap_linear sampling