0816028 周孟謙

ShaderInit:

一開始先用 glCreateShader 創造 vertex 和 fragment shader，再用他們去 creates a program object。

```
void shaderInit() {
    // TODO : create the shader & program
    // Hint :
    //      # create the shader file named "filename.vert" & "filename.frag" under the "/hw2/dll/Shaders/"
    //      # use the function defined in shaer.h to create shader
    GLuint vert = createShader("Shaders/hw1.vert", "vertex");
    GLuint frag = createShader("Shaders/hw1.frag", "fragment");
    program = createProgram(vert, frag);
}
```

BindBuffer:

創一個 vector<VertexAttribute> va，把皮卡丘的 position 和 textcoords 給放進去，使其之後能用 glBufferData()複製到 vbo_pi 裡面。接下來就是 generate vao 和 vbo buffer 並 bind。其中 vao pointer 對 vbo 的指法為前三個是 target 的 x, y, z，開頭為(void*)0 是因為在整個 buffer 的最前面。後面兩個為 texcoord 的 x, y，開頭為(void*)sizeof(Veertex)是因為前面的 xyz 大小為一個 Vertex，而 stride 皆為 sizeof(Vertex) + sizeof(GLfloat) * 3 是因為每個 VertexAttribute 皆有宣告 Vertex position 和 GLfloat texcoord[3]。Pokeball 的原理跟 pikachu 的一模一樣，只是一開始資料已經被塞好進 vector<VertexAttribute>了。

```
void bindBuffer(Object* model) {
    // TODO : use VAO & VBO to buffer the vertex data which will be passed to shader
    // Hint :
    //      # use "VertexAttribute" defined in Vertex.h to store the infomation of vertex needed in shader
    //      # the Pikachu model data is stored in function paramete "model" (or using global variable "Pikachu")
    //      # the pokeball vertex is stored in global variabl "ball"
    //      # see Object class detail in "Object.h" to pick up needed data to buffer
    // Generate a new buffer object

    vector<VertexAttribute> va;
    VertexAttribute temp;
    int max = Pikachu->positions.size();
    for (int it = 0, it2 = 0; it < max; it += 3, it2 += 2)
    {
        temp.setPosition(Pikachu->positions[it], Pikachu->positions[it + 1], Pikachu->positions[it + 2]);

        temp.setTexcoord(Pikachu->texcoords[it2], Pikachu->texcoords[it2+1]);

        va.push_back(temp);
    }

    glGenVertexArrays(1, &vao_pi);
    glBindVertexArray(vao_pi);
    glGenBuffers(1, &vbo_pi);

    glBindBuffer(GL_ARRAY_BUFFER, vbo_pi);
    glBufferData(GL_ARRAY_BUFFER, va.size() * sizeof(VertexAttribute), &va[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex) + sizeof(GLfloat) * 3, (void*)0);

    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex) + sizeof(GLfloat) * 3, (void*)sizeof(Vertex));

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
```

```
    glGenVertexArrays(1, &vao_po);
    glBindVertexArray(vao_po);
    glGenBuffers(1, &vbo_po);

    glBindBuffer(GL_ARRAY_BUFFER, vbo_po);
    glBufferData(GL_ARRAY_BUFFER, ball.size() * sizeof(VertexAttribute), &ball[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex) + sizeof(GLfloat) * 3, (void*)0);

    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex) + sizeof(GLfloat) * 3, (void*)sizeof(Vertex));

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

DrawSphere:

利用 theta 從 0 到 360 把 u 從 0 到 1 掃一遍，再利用 phi 從-90 到 90 把 v 從 0 到 1 掃一遍，把 textcoord 也塞進 ball 的陣列。而 xy_step 和 z_step 遇到的鋸齒狀問題會在後面講解。

```
void DrawSphere(float radius, float slice, float stack) {
    // TODO : calculate the texture coordinate of sphere
    // Hint :
    //      # read the code below to know how to iteratively compute the vertex position of sphere by sphere coordinate
    //      # find the mathematical way to iteratively calculate the u , v textur coordinate of a vertex on sphere
    float theta, phi, xy_step = 360 / slice, z_step = 180 / stack;
    Vertex vert;
    float u, v;
    for (phi = -90; phi <= 90; phi += z_step) {
        VertexAttribute temp;
        for (theta = 0; theta <= 360; theta += xy_step) {
            vert.x = radius * sin(theta * M_PI / 180) * cos(phi * M_PI / 180);
            vert.y = radius * cos(theta * M_PI / 180) * cos(phi * M_PI / 180);
            vert.z = radius * sin(phi * M_PI / 180);
            temp.setPosition(vert);

            u = theta / 360;
            v = (phi + 90) / 180;
            temp.setTexcoord(u, v);

            ball.push_back(temp);

            vert.x = radius * sin(theta * M_PI / 180) * cos((phi + z_step) * M_PI / 180);
            vert.y = radius * cos(theta * M_PI / 180) * cos((phi + z_step) * M_PI / 180);
            vert.z = radius * sin((phi + z_step) * M_PI / 180);
            temp.setPosition(vert);

            u = (theta + xy_step) / 360;
            v = (phi + 90 + z_step) / 180;
            temp.setTexcoord(u, v);

            ball.push_back(temp);
        }
    }
}
```

DrawModel:

ro 為按鍵 s 的旋轉角度(1 degree / per frame)。接下來是先畫出皮卡丘，把旋轉、移動、放大依照 pdf 給的數值弄好，然後創建並連結 modelview & projection matrix 後(glGetFloatv)，用 glGetUniformLocation 來 get the "location" of

uniform variable in shader，再把牠們放進 shader(glUniformMatrix4fv)，Texture 則是用 glUniform1i 放進 shader。之後便 bind vao_pi，使 glDrawArrays(GL_TRIANGLES, 0, Pikachu->positions.size() / 3)可以 draw the vertex stored in buffer，使用 GL_TRIANGLES 是因為 GL_TRIANGLE_STRIP 會不斷用之前兩點加上新的一點來畫三角形，造成皮卡丘的突出部分之間會有絲，Pikachu->positions.size() / 3 則是因為 position 每個單位不是存一點，而是一點的 x or y or z。

```
void drawModel(Object* model) {
    // TODO : draw all the models on correct position & send modelview and projection matrix to shader
    // Hint :
    //      # move the model by glRotate,glTranslate , glScale , glPush , glPop ... and so on
    //      # pass modelview matrix to shader after transformation
    //      # use program , VAO , texture , "glDrawArrays" ... and so on to render model
    if (key_s)
    {
        ro += 1;
        ro %= 360;
    }

    glPushMatrix();
    glRotatef(ro+25, 0.0f, 1.0f, 0.0f);
    glTranslatef(0.0f, 0.0f, 0.0f);
    glScalef(5.0f, 5.0f, 5.0f);

    GLfloat pmtx[16];
    GLfloat mmtx[16];

    // get current modelview & projection matrix used in OpenGL rendering
    glGetFloatv(GL_PROJECTION_MATRIX, pmtx);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmtx);

    // get the "location" (It's similar to pointer) of uniform variable in shader
    GLint pmatLoc = glGetUniformLocation(program, "Projection");
    GLint mmatLoc = glGetUniformLocation(program, "ModelView");

    // using shader
    glUseProgram(program);

    glUniform1i(glGetUniformLocation(program, "Texture") , 0);

    glBindVertexArray(vao_pi);
```

球的部分跟皮卡丘幾乎相同，但需要注意的是球因為之前 DrawSphere 掃 u 和 v 的時候沒有注意方向，所以需要另外加上 glRotatef(270, 1, 0, 0)轉到正面。還有 glDrawArrays 是使用 GL_TRIANGLE_STRIP，原因正好跟皮卡丘相反，若是使用 GL_TRIANGLES 則反而會造成球面會有洞。

```
    // pass the projection matrix into vertex shader
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
    // pass the modelview matrix into vertex shader
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, mmtx);

    // draw the vertex stored in buffer
    glDrawArrays(GL_TRIANGLES, 0, Pikachu->positions.size() / 3);
    glUseProgram(0);
    glPopMatrix();
    //glutSwapBuffers();

    glPushMatrix();
    glRotatef(ro+45, 0.0f, 1.0f, 0.0f);
    glTranslatef(3.0f, 0.0f, -3.0f);
    glRotatef(270, 1, 0, 0);

    GLfloat pmtx2[16];
    GLfloat mmtx2[16];

    // get current modelview & projection matrix used in OpenGL rendering
    glGetFloatv(GL_PROJECTION_MATRIX, pmtx2);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmtx2);

    // get the "location" (It's similar to pointer) of uniform variable in shader
    GLint pmatLoc2 = glGetUniformLocation(program, "Projection");
    GLint mmatLoc2 = glGetUniformLocation(program, "ModelView");
    // using shader
    glUseProgram(program);

    glUniform1i(glGetUniformLocation(program, "Texture"), 1);

    glBindVertexArray(vao_po);
```

```
    // pass the projection matrix into vertex shader
    glUniformMatrix4fv(pmatLoc2, 1, GL_FALSE, pmtx2);
    // pass the modelview matrix into vertex shader
    glUniformMatrix4fv(mmatLoc2, 1, GL_FALSE, mmtx2);

    // draw the vertex stored in buffer
    glDrawArrays(GL_TRIANGLE_STRIP, 0, ball.size());
    glUseProgram(0);

    glPopMatrix();
}
```

LoadTexture:
用 glActiveTexture 去 selects texture unit。然後 glGenTextures 去 Takes as input one textures we want to generate and stores them in a unsigned int array(texture)，再 bind 它。 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_NEAREST); glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);則是為了 Texture wrapping 和 Texture filtering。最後用 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);來使用圖像數據生成 texture。

```
// i indicate the texture unit number
void LoadTexture(unsigned int& texture, const char* tFileName, int i) {
    // TODO : generating the texture with texture unit
    // Hint :
    //      # glActiveTexture() , glGenTextures() and so on ...
    //      # GL_TEXTUREi = (GL_TEXTURE0 + i)
    //      # make sure one texture unit only binds one texture
    //      # It's different with VAO,VBO that texture don't need to unbind. (Just active different texture unit)
    glActiveTexture(GL_TEXTURE0+i);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);

    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_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


    int width, height, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load(tFileName, &width, &height, &nrChannels, 0);

    if (data)
    {
    // TODO : use image data to generate the texture here
    // Hint :
    //      # glTexImage2D()
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    }
    else
    {
        cout << "Failed to load texture" << endl;
    }
    stbi_image_free(data);
}
```

Keyboard:
設定鍵盤按鍵 s，每按一下會開始/暫停旋轉。

```
bool key_s = false;
int ro = 0;

void keyboard(unsigned char key, int x, int y) {
    // TODO : keyboard function , press key  's'  to start/stop all models rotation around y axis.
    // Hint :
    //      # The concept is as same as keyboard function in HW1
    switch (key) {
    case 's':
        if (key_s)
        {
            key_s = false;
        }
        else
        {
            key_s = true;
        }
        break;
    }

}
```

之前說過 xy_step 和 z_step 如果太大的話球會跑出鋸齒狀，因此在這邊 DrawSphere 加大了 slice 和 stack 的數字，使得 xy_step 和 z_step 變小。

```
int main(int argc, char** argv) {
    // set up OpenGL & window
    glutInit(&argc, argv);
    glutInitWindowSize(700, 700);
    glutInitWindowPosition(0, 0);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("HW2");

    // create & initial related data
    DrawSphere(3, 720, 360);
    glewInit();
    shaderInit();
    textureInit();
    bindBuffer(Pikachu);

    // bind OpenGL event function
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}
```

Vertex shader:
把 vertex 的 position 和 texcoord 讀進來，把 apos 從 vec3 變成 vec4 再乘上 Projection 和 ModelView 傳給 gl_Position。以及把 texcoord 傳給 fragment shader。

hw1.vert | hw1.frag | main.cpp

```
#version 330

layout(location = 0) in vec3 apos;
layout (location = 1) in vec2 atexcoord;

uniform mat4 Projection;
uniform mat4 ModelView;

out vec2 texcoord; //to fragment shader

void main() {
    gl_Position = Projection * ModelView * vec4(apos, 1.0);
    texcoord = atexcoord;
}
```

Fragment shader:
輸入 texture 和 textcoord，再用 texture2D 來找出 texture 再 textcoord 位置的顏色後輸出。

hw1.vert | hw1.frag | main.cpp

```
#version 330

uniform sampler2D Texture;

in vec2 texcoord;
out vec4 outColor;

void main() {
    outColor = texture2D(Texture, texcoord);
}
```