HW2 – Pokémon

0816039 李品慈

- **Explain in detail how to use GLSL by taking screenshots.**

```
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
```

首先，main 會呼叫一個 DrawSphere()，來初始化 ball 內的 position 和 texcoord。

```
u = theta/360.0f;
v = (phi+90.0f) / 180.0f;
temp.setTexcoord(u, v);

ball.push_back(temp);
```

```
u = theta / 360.0f;
v = (phi + 90.0f + z_step) / 180.0f;
temp.setTexcoord(u, v);

ball.push_back(temp);
```

因為 position 已經 implement 完成，這裡我們只需要算出 u, v 再將其填入 ball 即可。

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

然後，我們會先呼叫 shaderInit 來初始化我們的 vetex 和 fragment shader。如同助教提到，這裡只要利用已經寫好的 function 來創建這兩個 shader 和 program 即可。

```
void textureInit() {
    glEnable(GL_TEXTURE_2D);
    // bind Pikachu texture with GL_TEXTURE0
    LoadTexture(Pikachu_texture, "Pikachu.png",0);
    // bind Pokeball texture with GL_TEXTURE1
    LoadTexture(ball_texture, "Pokeball.png",1);
}
```

下一個會呼叫到的是 textureInit，這個 function 已經寫好不需要 implement，但可以注意到這邊呼叫了 LoadTexture 這個 function 並傳入了三

個相對應的變數，用意是要將兩個 png 檔讀入兩個 texture 變數。

```
// 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 differe

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

    glActiveTexture(GL_TEXTURE0 + i);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

LoadTexture 的主要功能便是將 attribute 中的 tFileName 存入前面的 texture attribute，而最後一項的 i 則是兩個 texture 的 index。

這裡我們先呼叫 glActiveTexture 來告訴程式我們要啟用 texture，並用 glGenTextures 為 texture 創建一個 array 將其存入，最後將其 bind 給一個 texturing target。

後面接著的四行 glTexParameteri 則是設定 texture 的行為模式。

```
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);
```

在初始化結束後我們便能將 texture 傳入剛剛創建好的 array 中。

```
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

    // pikachu
    glGenVertexArrays(1, &vaoPikachu);
    glGenBuffers(1, &vboPikachu);

    glBindVertexArray(vaoPikachu);
    glBindBuffer(GL_ARRAY_BUFFER, vboPikachu);
```

再來我們會呼叫到 bindBuffer，這個 function 主要的功能是創建兩個 model 各自的 vao, vbo buffer，將 position 和 texcoord 資料和 vbo buffer bind 在一起，並且告訴 vao 要怎麼取用 vbo 裡面的資料，之後要用到這兩個 model 的資料時都會從 buffer 中取用。
上面圖片的上半部是創建 Pikachu 的 vao, vbo buffer，下半部則是將這兩個 buffer bind 給兩個分別的 array buffer。

```
combined = merge(model->positions, model->texcoords);
```

這裡我調用了一個自己寫的小 function，主要功能是將 Pikachu 兩項資料合成為一個，方便之後撰寫 link vao 的 pointer。

```
float* vertice;
vertice = combined.data();
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * combined.size(), vertice, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, position)));
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, texcoord)));
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

接下來便是要將 data 跟剛剛創建的 vao, vbo link 起來，首先我們先呼叫 glBufferData，將 vertice 的 data 傳入 ARRAY_BUFFER(vbo)。再來我們會呼叫 glEnableVertexAttribArray 來告訴程式有兩項 attribute，並且 enable 兩個 attribute array，後面 glVertexAttribPointer 則是告訴 vao 這兩個 attribute 在 buffer 中是如何排序的。最後結束時呼叫 glBindBuffer 和 glBindVertexArray 來 unbind 剛剛更動的兩個 buffer，避免之後不小心更動內容。

```
VertexAttribute* vertices;
vertices = ball.data();
glBufferData(GL_ARRAY_BUFFER, sizeof(VertexAttribute) * ball.size(), vertices, GL_STATIC_DRAW);
```

Ball 與上述 Pikachu 的內容大致相同，不過因為最初設定的 ball 是一個 VertexAttribute 的 vector，所以這邊會將 float 改為 VertexAttribute。

```cpp
vector<float> merge(vector<float> a, vector<
{
    vector<float> result;

    auto v1 = a.begin();
    auto v2 = b.begin();

    while (v1 != a.end() && v2 != b.end())
    {
        result.push_back(*v1);
        ++v1;
        result.push_back(*v1);
        ++v1;
        result.push_back(*v1);
        ++v1;
        result.push_back(*v2);
        ++v2;
        result.push_back(*v2);
        ++v2;
    }
```

自己建的 merge function 和一般認知的 merge function 不同的小地方是在存入新 vector 時會將 v1(position)一次存入三個 float(x, y, z)，v2(texcoord)存入兩個 float(x, y)，其餘皆相同。

接著程式會呼叫 Display，分別又呼叫 init, light, drawModel。

```cpp
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

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

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

drawModel 的主要功能就是將 Pikachu 和 ball 兩個 model 經過 transformation 後傳入 shader，處理 vertex 跟 texture 後再印出來。

首先我們先創造兩個 matrix(projection, modelview)，並取得 projection matrix 和 modelview matrix 在 shader 中的位置。

```
// using shader
glUseProgram(program);

GLint texLoc = glGetUniformLocation(program, "Pikachu_texture");
GLint tex2Loc = glGetUniformLocation(program, "ball_texture");
glUniform1i(texLoc, 0);
glUniform1i(tex2Loc, 1);
```

在啟動 shader 後，一樣取得 texture 在 shader 中的位置，並用 glUniorm 傳給 shader。

```
// draw the vertex stored in buffer
// pikachu
glPushMatrix();
    glRotatef(revolve_angle, 0.0f, 1.0f, 0.0f); // revolve y axis
    glScalef(5, 5, 5);
    glRotatef(25, 0, 1, 0);
    glBindVertexArray(vaoPikachu);
    glBindTexture(GL_TEXTURE_2D, Pikachu_texture);
    // get current modelview & projection matrix & pass to vertex shader
    glGetFloatv(GL_PROJECTION_MATRIX, pmtx);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmtx);
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, mmtx);
    glDrawArrays(GL_TRIANGLES, 0, sizeof(float) * combined.size());
glPopMatrix();

glBindVertexArray(0);
```

接下來便可以開始繪製 model，上面圖片是繪製 Pikachu 的過程，先將 model 經過一連串的 transformation 後，將更動後的 modelview 和 projection matrix 傳給 shader，再利用 glDrawArrays 根據上面 bind 的 vao 繪製 model，最後將 bind 的 vao unbind 掉。

第一行的的 glRotate 為按下鍵盤 s 鍵後需要的旋轉，revolve_angle 初始為 0，在 idle function 中會根據 frame 逐步增加。

```
//ball
glPushMatrix();
    glRotatef(revolve_angle, 0.0f, 1.0f, 0.0f); // revolve y axis
    glRotatef(45, 0, 1, 0);
    glTranslatef(3, 0, -3);
    glRotatef(270, 1, 0, 0); // turn to face the user
    glBindVertexArray(vaoBall);
    glBindTexture(GL_TEXTURE_2D, ball_texture);
    // get current modelview & projection matrix & pass to vertex shader
    glGetFloatv(GL_PROJECTION_MATRIX, pmtx);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmtx);
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, pmtx);
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, mmtx);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, sizeof(VertexAttribute) * ball.size());
glPopMatrix();

glBindVertexArray(0);
glUseProgram(0);
```

Ball 跟 Pikachu 大致相同，但因為 texture 座標問題，球面會朝向上方，我在這裡增加了一個 glRotate 將其轉回正面。

```
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':
        /* DO SOMETHING HERE TO START/STOP ALL MODEL ROTATION */
        flag_s = !flag_s;
        break;
    }
}
```

本次作業中按下 s 需要讓兩個 model 相對 y 軸旋轉，在這裡我加入一個 bool 的 flag_s 來讓 idle function 根據 flag_s 的值判斷是否需要增加 revolve_angle。

```
void idle() {
    // FPS control
    clock_t CostTime = End - Start;
    float PerFrameTime = 1000.0 / MAX_FPS;
    if (CostTime < PerFrameTime) {
        Sleep(ceil(PerFrameTime) - CostTime);
    }
    if (flag_s) {
        revolve_angle += 1;
        if (revolve_angle > 360.0) revolve_angle -= 360.0;
    }
    glutPostRedisplay();
}
```

如上述所說，我在 idle function 中加入一個 if statement，如果 flag_s 為 true 即可讓兩個 model 旋轉，再次按下 s 鍵可停止。

- **Describe the problems you met and how you solved them.**

1. Transformation 與 Matrix

   因為本次作業加入了 shader，所以在資料的相互傳送變得更複雜，原先我按照 HW1 的方式在 drawArray 前加入 transformation，但 render 出的畫面都沒有改變。

   而解決方法是在 drawArray 前再重新取得 Matrix 並傳入 shader，因為 shader 會根據最近一次傳入的 Matrix 來繪製畫面，若沒有重新傳入 Matrix，前面的 tramsformation 便無法產生效果。

2. 無法 render 畫面

   在最一開始 debug 時，發現一直都無法印出畫面，後來才發現是 bindBuffer 中的 size 錯誤，本來應該傳入 vector 整體的大小，誤將 pointer 的大小傳入。