Assignment 2 : Creating your own OpenGL program

NAME: 邱龙田

STUDENT NUMBER: 2018533107

EMAIL: qiult@shanghaitech.edu.cn

1 INTRODUCTION

In this assignment I implement vertex shader and fragment shader.Then, I implement Phong multiple lights modals and texture by these two shaders. In the mean time, I visualize all the lights by a small cube。Afterwards, I fulfill my_texture.cpp to achieve the function of loading .img file and process it so I can use it as texture in opengl.Then I convert the data from the .obj files loaded by tiny_obj_loader to float array so that we may bind it to VAO and VBO. To get tangent space, I calculate one coordinate axis of tangent space with the coordinates of vertices， normal， texcoords stored in the array we create before then store it inside the float array.

2 IMPLEMENTATION DETAILS

Inside Vertex shader,I import the points, normal ， texcoords 和 tangent(calculated in the main.cpp) passed by mian.cpp and calculate the TBN matrix by normal and tangent.Then calculate the view position and fragment position transformed by TBN matrix and pass it to fragment shader。

```
mat3 normalMatrix = transpose(inverse(mat3(model)));
vec3 T = normalize(normalMatrix * aTangent);
vec3 N = normalize(normalMatrix * aNormal);
T = normalize(T - dot(T, N) * N);
vec3 B = cross(N, T);

mat3 TBN = transpose(mat3(T, B, N));
vs_out.TangentViewPos  = TBN * viewPos;
vs_out.TangentFragPos  = TBN * vs_out.FragPos;
vs_out.TBN = TBN;
```

Inside fragment shader, I calculate the impact of multiple lights and parallel to the pixel.By the way, when we calculate the light position we need to transform it by TBN first. And I transform the normal vector by the normal map texture.

```
vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir)

{
    vec3 lightDir = normalize(-(fs_in.TBN * light.direction));
    // 漫反射着色
    float diff = max(dot(normal, lightDir), 0.0);
    // 镜面光着色
```

To get one coordinate axis of tangent space, we need the coordinate of a face's points and texcoords. Then return the result in the form of vec3.

```
// calculate tangent space
glm::vec3 calculate_tangent(glm::vec3 p1, glm::vec3 p2, glm::vec3 p3, glm::vec2 uv1, glm::vec2 uv2, glm::vec2 uv3) {
    glm::vec3 edge1 = p2 - p1;
    glm::vec3 edge2 = p3 - p1;
    glm::vec2 deltaUV1 = uv2 - uv1;
    glm::vec2 deltaUV2 = uv3 - uv1;
    glm::vec3 proTangent;

    GLfloat proF = 1.0f / (deltaUV1.x * deltaUV2.y - deltaUV2.x * deltaUV1.y);

    proTangent.x = proF * (deltaUV2.y * edge1.x - deltaUV1.y * edge2.x);
    proTangent.y = proF * (deltaUV2.y * edge1.y - deltaUV1.y * edge2.y);
    proTangent.z = proF * (deltaUV2.y * edge1.z - deltaUV1.y * edge2.z);
    proTangent = glm::normalize(proTangent);
    return proTangent;
}
```

To pass the parameters we need for the calculating tangent, we combine the coordinates of points and texcoords read from the tiny_obj_loader(we achieve that by modifying the make_face function)

```
std::vector<glm::vec3> proPoints;
std::vector<glm::vec2> proUVs;
int k = i * 3;
for (int j = 0; j < 3; j++)
{
    points.emplace_back(v[f[k + j] * 3], v[f[k + j] * 3 + 1], v[f[k + j] * 3 + 2]);
    proPoints.emplace_back(v[f[k + j] * 3], v[f[k + j] * 3 + 1], v[f[k + j] * 3 + 2]);
    normals.emplace_back(vn[f[k + j] * 3], vn[f[k + j] * 3 + 1], vn[f[k + j] * 3 + 2]);
    uvs.emplace_back(vt[f[k + j] * 2], vt[f[k + j] * 2 + 1]);
    proUVs.emplace_back(vt[f[k + j] * 2], vt[f[k + j] * 2 + 1]);
}
// use the data from one face to calculate the tangent space
glm::vec3 proTangent = calculate_tangent( p1: proPoints[0],  p2: proPoints[1],  p3: proPoints[2],  uv1: proUVs[0],  uv2: proUVs[1],
```

Then we need to combine the data in four separate vector into one float array so we could bind it to the memory.

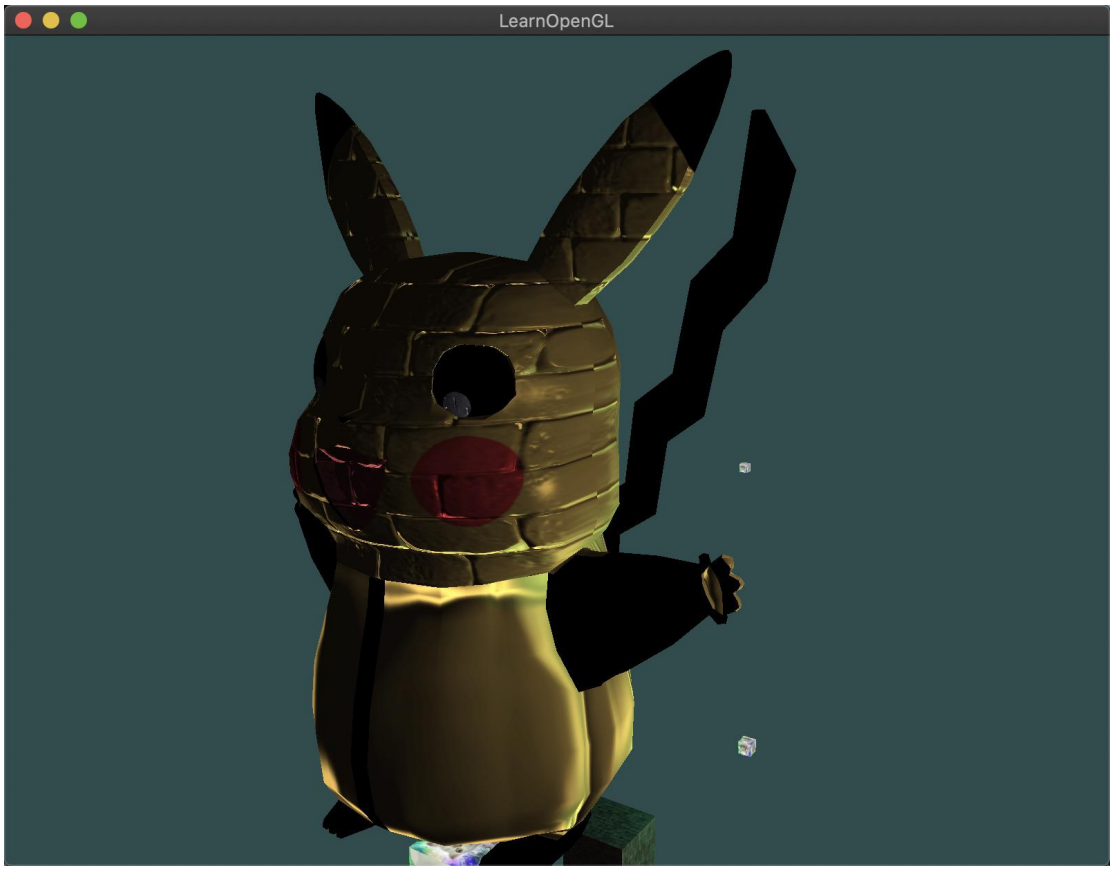
```
float vertices[out_vertices.size() * 11];
for (int j = 0;j < out_vertices.size();j++) {
    vertices[j*11] = out_vertices[j].x;
    vertices[j*11 + 1] = out_vertices[j].y;
    vertices[j*11 + 2] = out_vertices[j].z;
    vertices[j*11 + 3] = out_normals[j].x;
    vertices[j*11 + 4] = out_normals[j].y;
    vertices[j*11 + 5] = out_normals[j].z;
    vertices[j*11 + 6] = out_uvs[j].x;
    vertices[j*11 + 7] = out_uvs[j].y;
    vertices[j*11 + 8] = out_tangent[j].x;
    vertices[j*11 + 9] = out_tangent[j].y;
    vertices[j*11 + 10] = out_tangent[j].z;
}
```

Finally, in man.cpp. After all the data we need is acquired. I bind the float array consist of points, normals, texcoords and tangents with VAOs and VBOs which are stored inside vector obj_VAO_I and obj_VBO_I. Then we init the lights by pass the information of lights into shader which is provided by template. In the mean time, we load the textures we need and activate

required two textures before we draw the picture. Then we can draw the pictures!

3 RESULT

Picture of pikaqiu



Picture of cube with normal map