Tech document

This document will explain how I finish this project. In this project, what I have done are as shown below.

1. Roller coaster
2. Skybox
3. Teacups
4. Firework
5. Waving flag

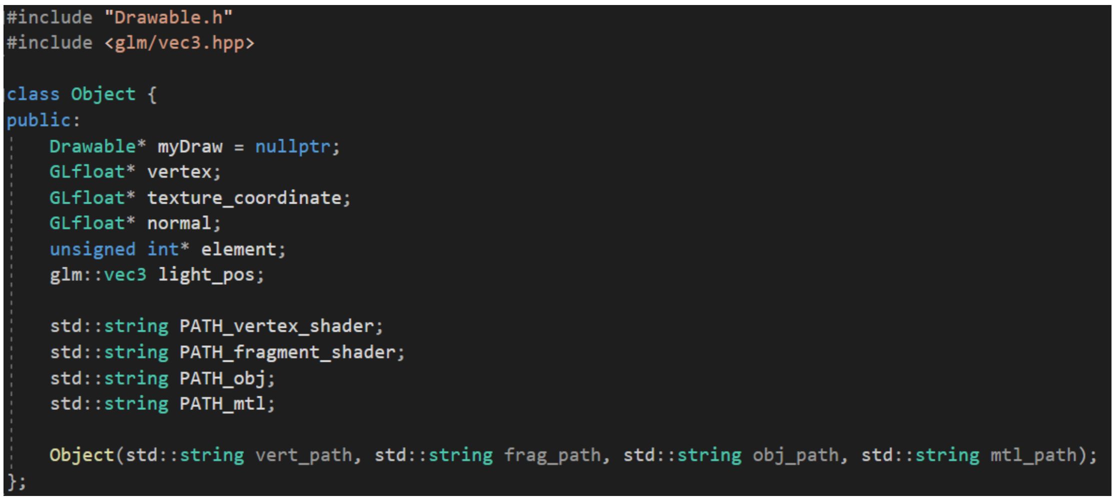
The first two the same as my former ones without any changes, so I will skip the explanation of those parts.

1. Teacups

1-1. Load .obj file

The teacups were loaded from the 3D model in .obj format. Therefore, I would like to explain how I load the .obj file first.

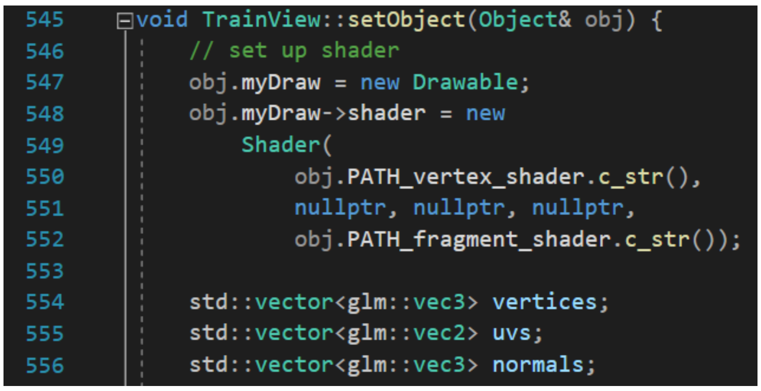
I have a class named Object that stores information a loaded 3D model will need, including the path of .obj file and shaders.



```
#include "Drawable.h"
#include <glm/vec3.hpp>

class Object {
public:
    Drawable* myDraw = nullptr;
    GLfloat* vertex;
    GLfloat* texture_coordinate;
    GLfloat* normal;
    unsigned int* element;
    glm::vec3 light_pos;

    std::string PATH_vertex_shader;
    std::string PATH_fragment_shader;
    std::string PATH_obj;
    std::string PATH_mtl;

    Object(std::string vert_path, std::string frag_path, std::string obj_path, std::string mtl_path);
};
```

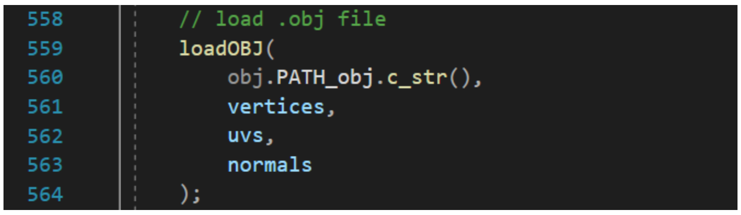
After getting the paths, the object will be set using setObject() function. This function will first set up the shader of the object, then parse the .obj file with loadObj function, and finally set up all vao, vbo, and ebo.

(set up shader)



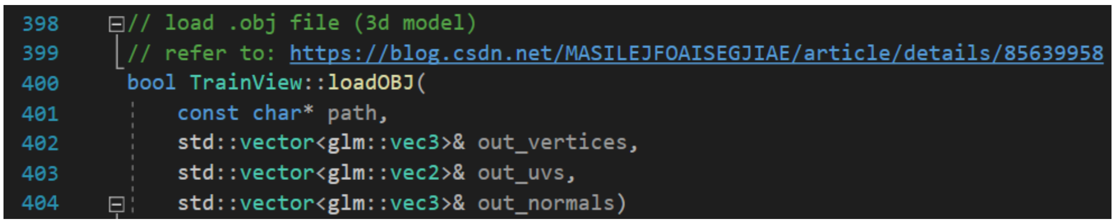
```
void TrainView::setObject(Object& obj) {
    // set up shader
    obj.myDraw = new Drawable;
    obj.myDraw->shader = new
        Shader(
            obj.PATH_vertex_shader.c_str(),
            nullptr, nullptr, nullptr,
            obj.PATH_fragment_shader.c_str());

    std::vector<glm::vec3> vertices;
    std::vector<glm::vec2> uvs;
    std::vector<glm::vec3> normals;
```

(Load .obj file with loadObj() function and store information in the given vector)



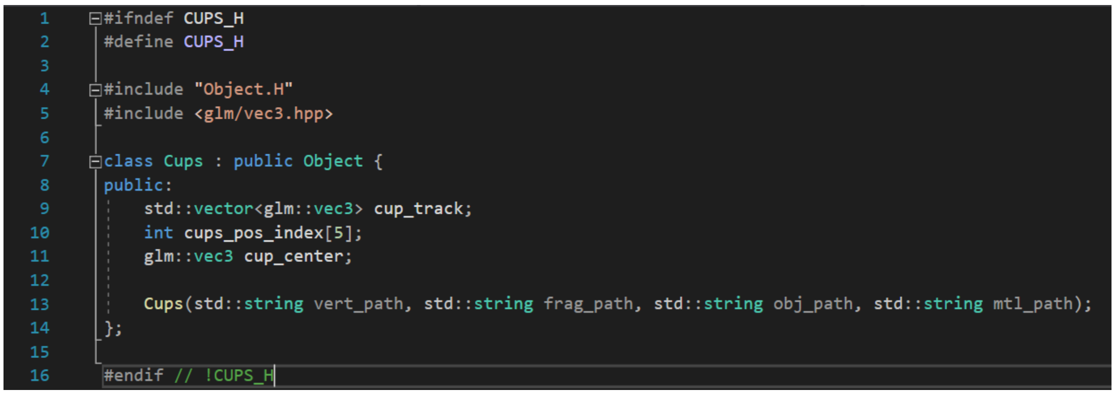
```cpp
// load .obj file
loadOBJ(
    obj.PATH_obj.c_str(),
    vertices,
    uvs,
    normals
);
```

(loadOBJ function will parse the .obj file)



```cpp
// load .obj file (3d model)
// refer to: https://blog.csdn.net/MASILEJFOAISEGJIAE/article/details/85639958
bool TrainView::loadOBJ(
    const char* path,
    std::vector<glm::vec3>& out_vertices,
    std::vector<glm::vec2>& out_uvs,
    std::vector<glm::vec3>& out_normals)
```

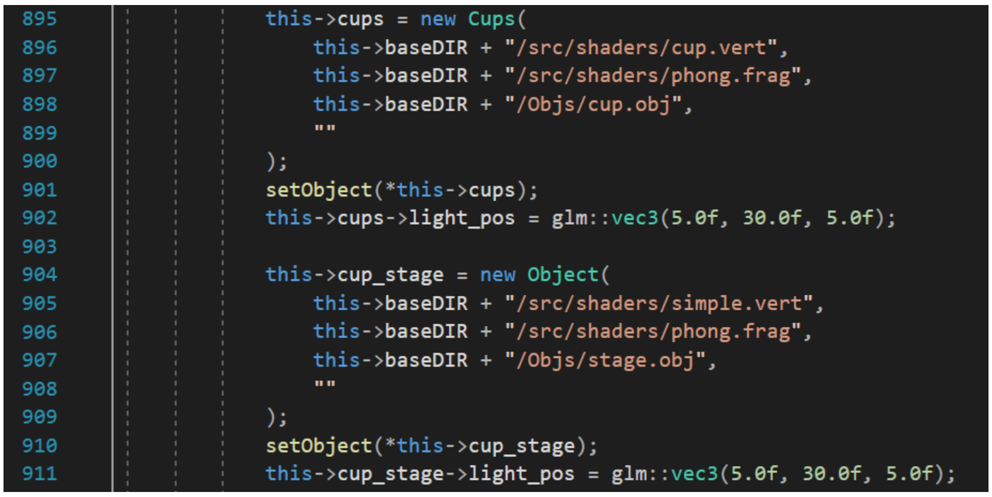
1-2. Cups class



```cpp
#ifndef CUPS_H
#define CUPS_H

#include "Object.H"
#include <glm/vec3.hpp>

class Cups : public Object {
public:
    std::vector<glm::vec3> cup_track;
    int cups_pos_index[5];
    glm::vec3 cup_center;

    Cups(std::string vert_path, std::string frag_path, std::string obj_path, std::string mtl_path);
};

#endif // !CUPS_H
```

The teacups will revolve around the center and rotate at the same time, so I create a class name Cups that is derived from Object class to store the information of the center and the track which the cups will move along.

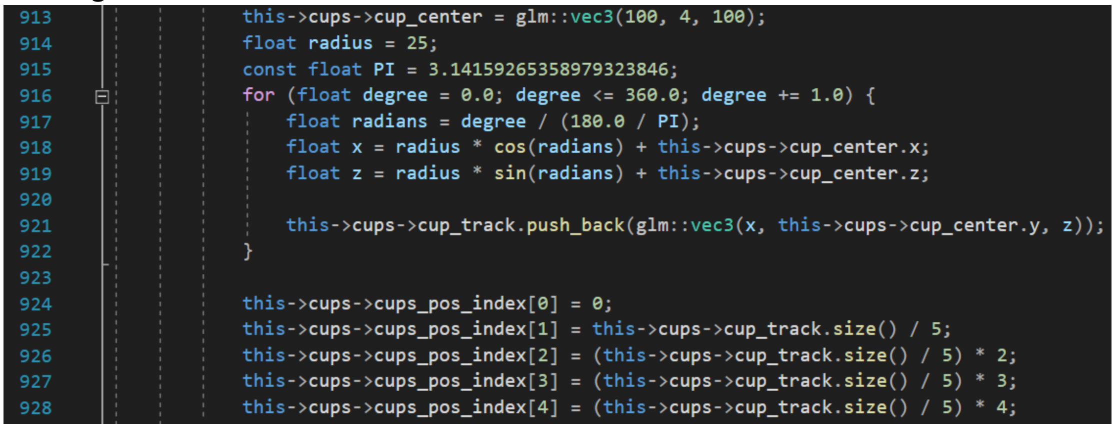
I will set the cup and the fence of the teacups first.



```cpp
this->cups = new Cups(
    this->baseDIR + "/src/shaders/cup.vert",
    this->baseDIR + "/src/shaders/phong.frag",
    this->baseDIR + "/Objs/cup.obj",
    ""
);
setObject(*this->cups);
this->cups->light_pos = glm::vec3(5.0f, 30.0f, 5.0f);

this->cup_stage = new Object(
    this->baseDIR + "/src/shaders/simple.vert",
    this->baseDIR + "/src/shaders/phong.frag",
    this->baseDIR + "/Objs/stage.obj",
    ""
);
setObject(*this->cup_stage);
this->cup_stage->light_pos = glm::vec3(5.0f, 30.0f, 5.0f);
```

After setting up the cup and fence 3D model, I made a round track using the formula of circle and assign the initial position of the five cups on the track.

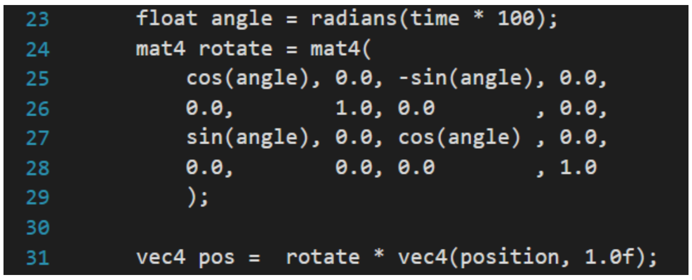
They will move along the track with time passing. This creates the effect of orbiting.



```
            this->cups->cup_center = glm::vec3(100, 4, 100);
            float radius = 25;
            const float PI = 3.14159265358979323846;
            for (float degree = 0.0; degree <= 360.0; degree += 1.0) {
                float radians = degree / (180.0 / PI);
                float x = radius * cos(radians) + this->cups->cup_center.x;
                float z = radius * sin(radians) + this->cups->cup_center.z;

                this->cups->cup_track.push_back(glm::vec3(x, this->cups->cup_center.y, z));
            }

            this->cups->cups_pos_index[0] = 0;
            this->cups->cups_pos_index[1] = this->cups->cup_track.size() / 5;
            this->cups->cups_pos_index[2] = (this->cups->cup_track.size() / 5) * 2;
            this->cups->cups_pos_index[3] = (this->cups->cup_track.size() / 5) * 3;
            this->cups->cups_pos_index[4] = (this->cups->cup_track.size() / 5) * 4;
```

1-3. cup vertex shader

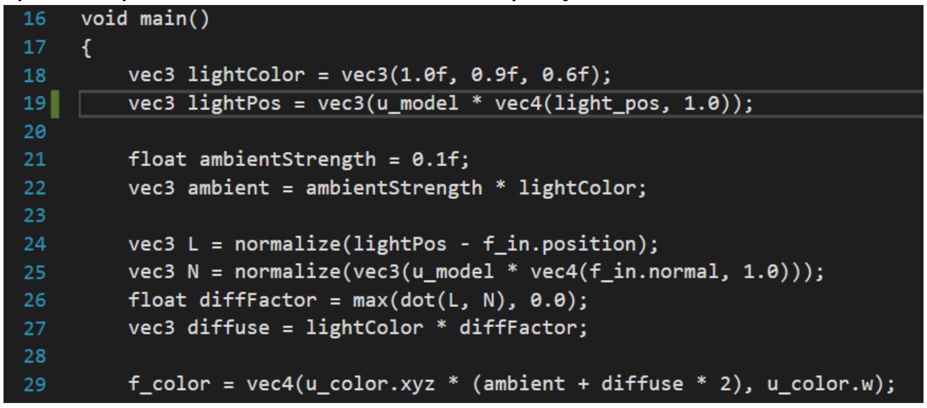
In the vertex shader of the cup, I use time as the angle and create a rotation matrix to transform the position of cup.



```
    float angle = radians(time * 100);
    mat4 rotate = mat4(
        cos(angle), 0.0, -sin(angle), 0.0,
        0.0,        1.0, 0.0        , 0.0,
        sin(angle), 0.0, cos(angle) , 0.0,
        0.0,        0.0, 0.0        , 1.0
        );

    vec4 pos =  rotate * vec4(position, 1.0f);
```

1-4. Phong shading fragment shader

I implemented Phong shading in this fragment shader, but I did not use the specular part because I did not want my objects to reflect.



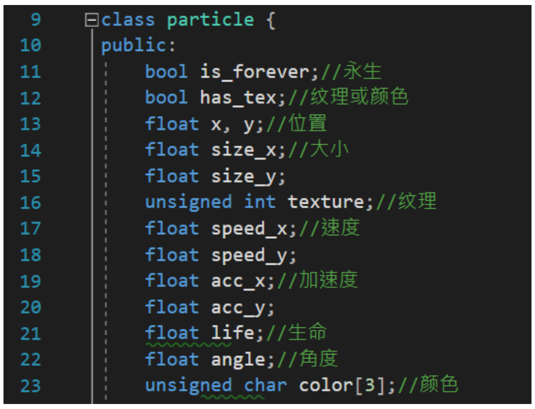
```
void main()
{
    vec3 lightColor = vec3(1.0f, 0.9f, 0.6f);
    vec3 lightPos = vec3(u_model * vec4(light_pos, 1.0));

    float ambientStrength = 0.1f;
    vec3 ambient = ambientStrength * lightColor;

    vec3 L = normalize(lightPos - f_in.position);
    vec3 N = normalize(vec3(u_model * vec4(f_in.normal, 1.0)));
    float diffFactor = max(dot(L, N), 0.0);
    vec3 diffuse = lightColor * diffFactor;

    f_color = vec4(u_color.xyz * (ambient + diffuse * 2), u_color.w);
```

2. Firework

I simulated the firework with a particle system. The particle system include emitter and particles.

2-1. particle

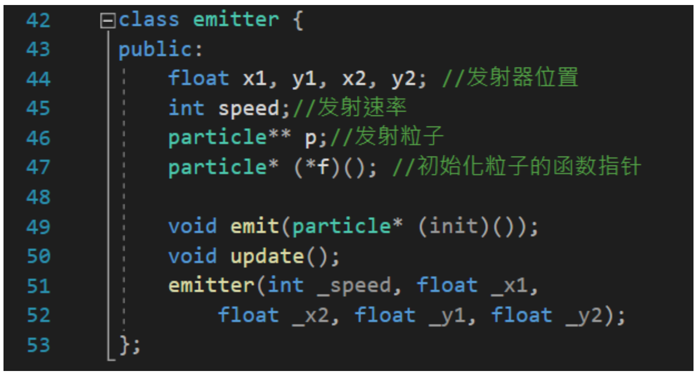
The particle class stores the attributes of the particle.



```
class particle {
public:
    bool is_forever;//永生
    bool has_tex;//纹理或颜色
    float x, y;//位置
    float size_x;//大小
    float size_y;
    unsigned int texture;//纹理
    float speed_x;//速度
    float speed_y;
    float acc_x;//加速度
    float acc_y;
    float life;//生命
    float angle;//角度
    unsigned char color[3];//颜色
```

2-2. emitter

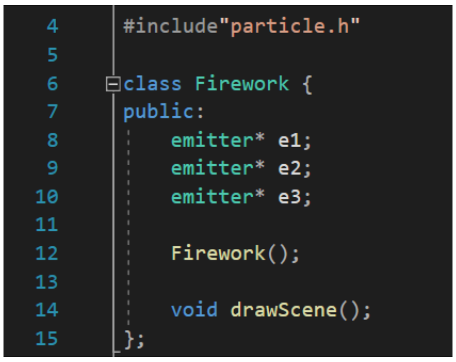
The emitter has a pool of particles and will update the state and each attribute of the particles.



```
class emitter {
public:
    float x1, y1, x2, y2; //发射器位置
    int speed;//发射速率
    particle** p;//发射粒子
    particle* (*f)(); //初始化粒子的函数指针

    void emit(particle* (init)());
    void update();
    emitter(int _speed, float _x1,
        float _x2, float _y1, float _y2);
};
```

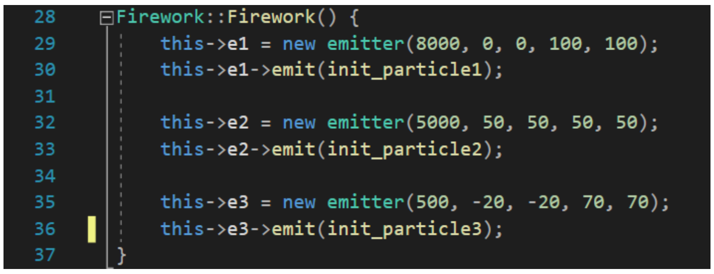
2-3. firework

My firework is formed of three emitters (three fireworks).



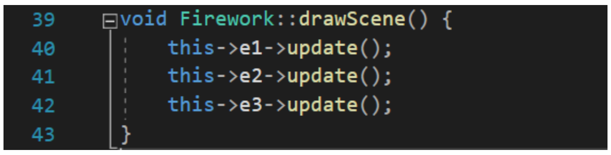
```
#include"particle.h"

class Firework {
public:
    emitter* e1;
    emitter* e2;
    emitter* e3;

    Firework();

    void drawScene();
};
```

When the firework constructing, it will set up the center of the emitter and the speed of the particles.

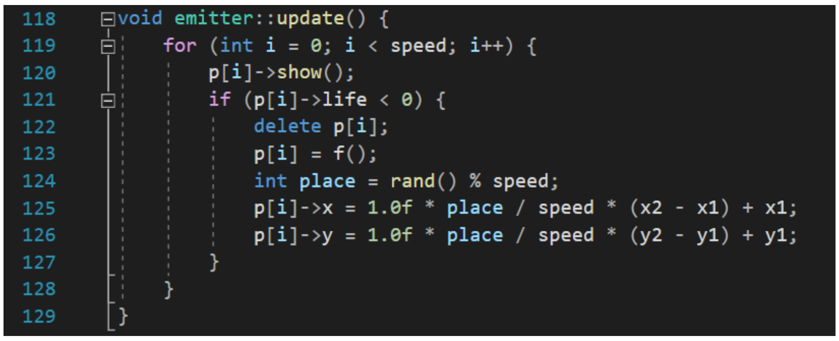


```
Firework::Firework() {
    this->e1 = new emitter(8000, 0, 0, 100, 100);
    this->e1->emit(init_particle1);

    this->e2 = new emitter(5000, 50, 50, 50, 50);
    this->e2->emit(init_particle2);

    this->e3 = new emitter(500, -20, -20, 70, 70);
    this->e3->emit(init_particle3);
}
```

When drawScene() is called, all of the emitter will update the attributes of the particle and draw all of the particles.



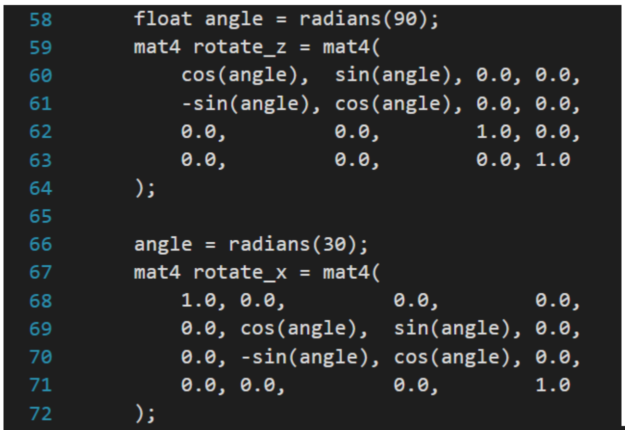
```
void Firework::drawScene() {
    this->e1->update();
    this->e2->update();
    this->e3->update();
}
```



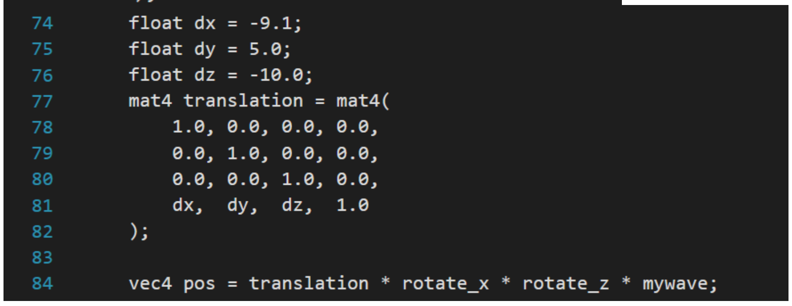
```
void emitter::update() {
    for (int i = 0; i < speed; i++) {
        p[i]->show();
        if (p[i]->life < 0) {
            delete p[i];
            p[i] = f();
            int place = rand() % speed;
            p[i]->x = 1.0f * place / speed * (x2 - x1) + x1;
            p[i]->y = 1.0f * place / speed * (y2 - y1) + y1;
        }
    }
}
```

3. Waving flag

I simulated a waving flag with sine wave. What I did just borrowed the shaders from the last project, rotated the wave along z and x axes, and add texture to it.



```
    float angle = radians(90);
    mat4 rotate_z = mat4(
        cos(angle),  sin(angle), 0.0, 0.0,
        -sin(angle), cos(angle), 0.0, 0.0,
        0.0,         0.0,        1.0, 0.0,
        0.0,         0.0,        0.0, 1.0
    );

    angle = radians(30);
    mat4 rotate_x = mat4(
        1.0, 0.0,         0.0,        0.0,
        0.0, cos(angle),  sin(angle), 0.0,
        0.0, -sin(angle), cos(angle), 0.0,
        0.0, 0.0,         0.0,        1.0
    );
```



```
    float dx = -9.1;
    float dy = 5.0;
    float dz = -10.0;
    mat4 translation = mat4(
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        dx,  dy,  dz,  1.0
    );

    vec4 pos = translation * rotate_x * rotate_z * mywave;
```

Reference

现代 OpenGL 教程（五）：obj 文件和 obj 模型加载（imgui+OpenGL3.3）
https://blog.csdn.net/MASILEJFOAISEGJIAE/article/details/85639958

Basic Lighting
https://learnopengl.com/Lighting/Basic-Lighting

[OpenGL] 简单二维粒子系统——烟花，喷水，落叶
https://blog.csdn.net/ZJU_fish1996/article/details/51968085