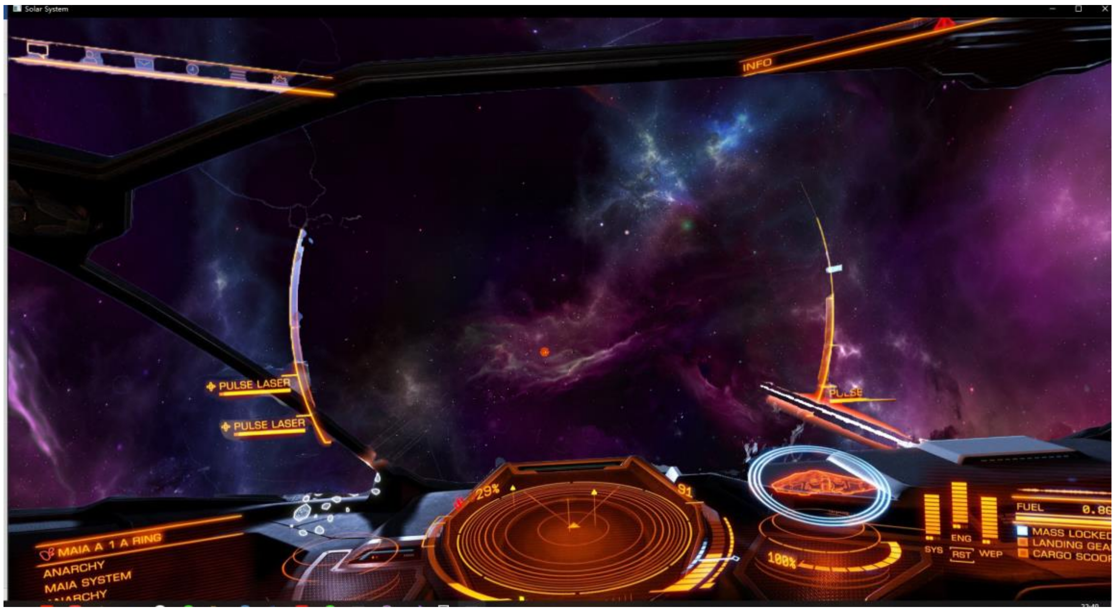
Computer Graphic Project Report

Project description:

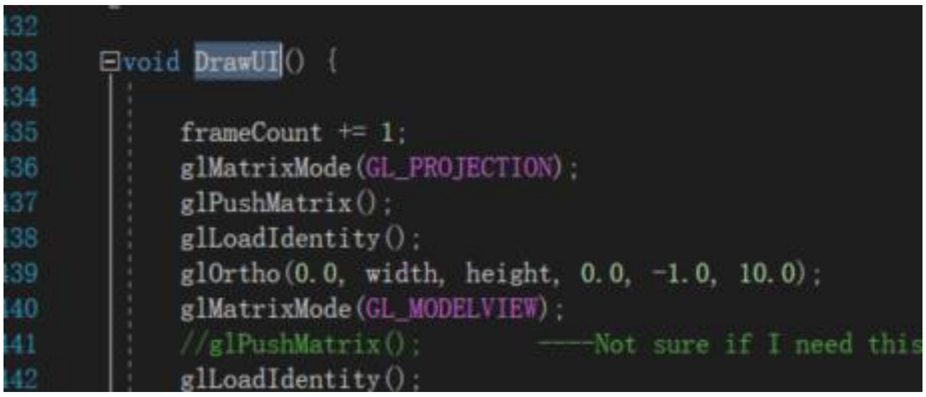
I make this project more like an interactable game. Player can control a space ship in the universe, there are some enemy UFO inside Solar System, and player can shoot laser to destroy UFO.

Project content:

1, On screen HUD changing the glOrtho to make the HUD always float on the top of screen.



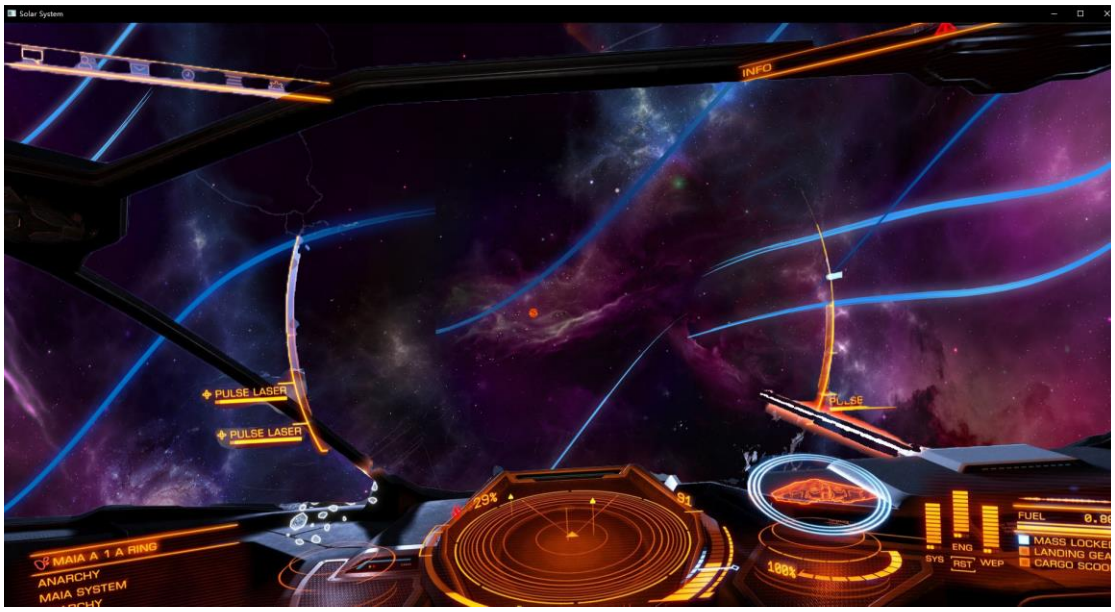
Code explanation:



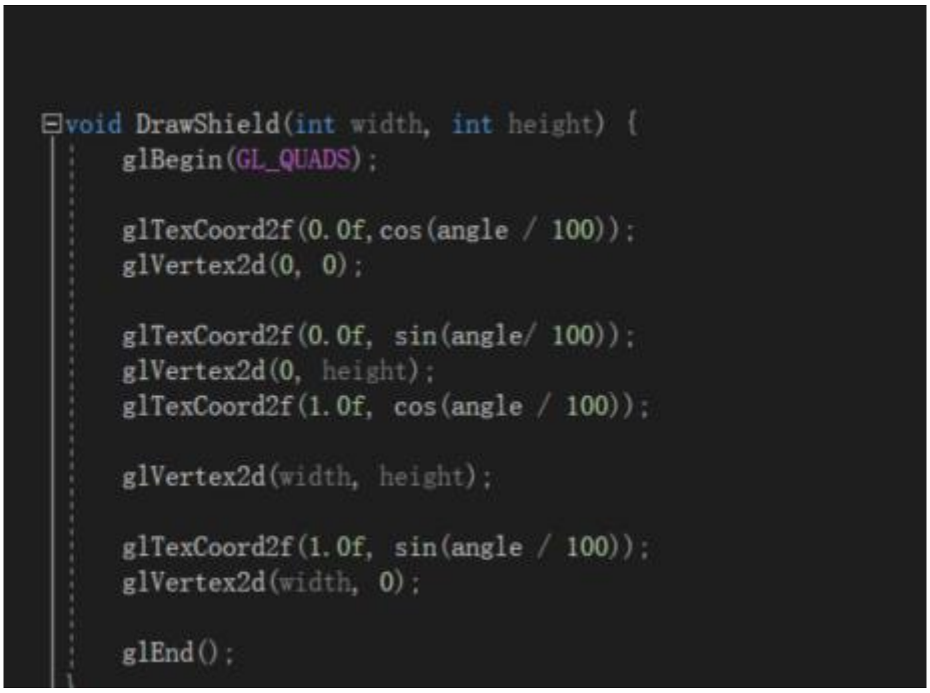
```
void DrawUI() {

    frameCount += 1;
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0.0, width, height, 0.0, -1.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    //glPushMatrix();        ----Not sure if I need this
    glLoadIdentity();
```

change the ortho of scene then draw the UI, it would make HUD always floating on the top of screen.

2, Texture animation, by adjusting the texture coordinate of HUD shield texture, making a shield animation.



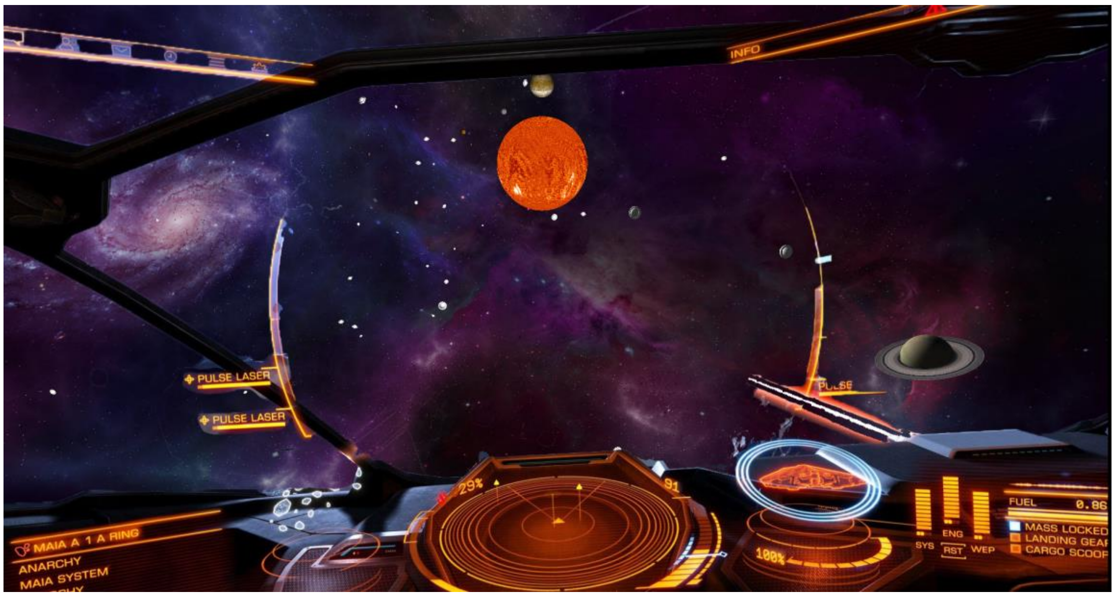
Code explanation:



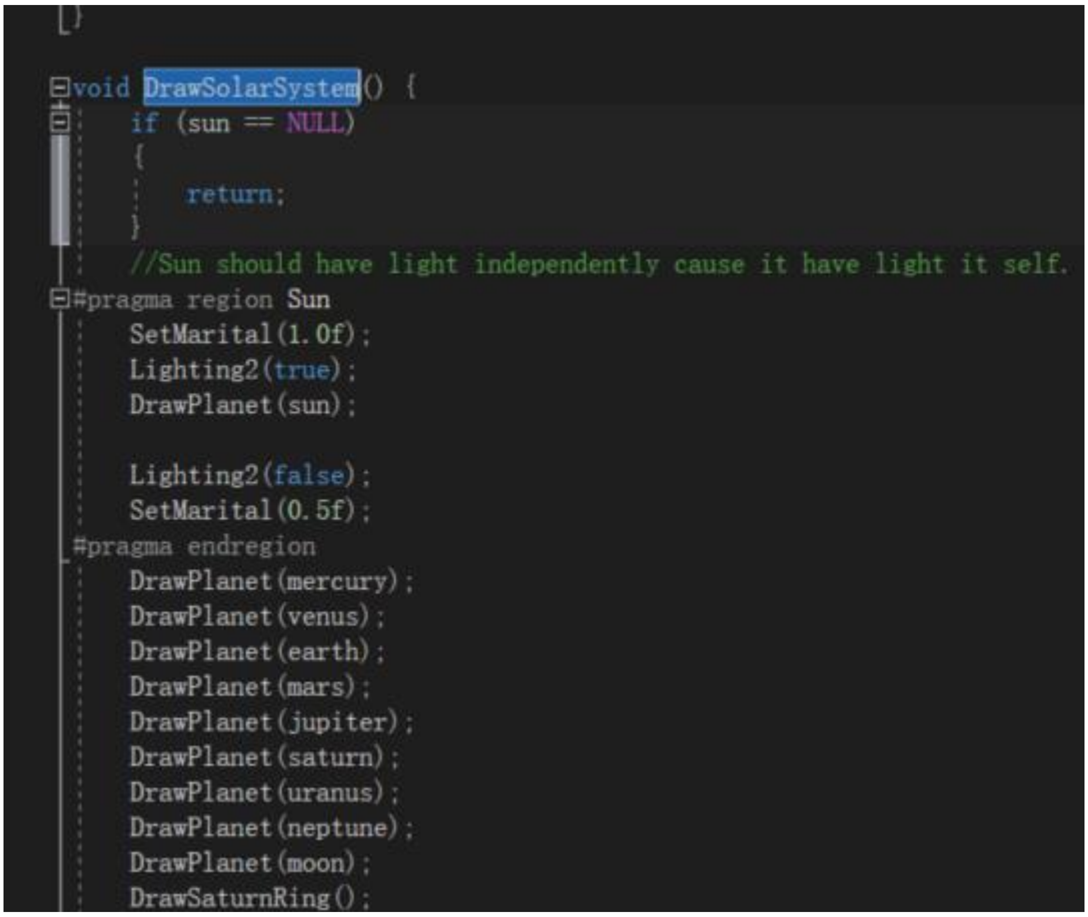
```
void DrawShield(int width, int height) {
    glBegin(GL_QUADS);

    glTexCoord2f(0.0f, cos(angle / 100));
    glVertex2d(0, 0);

    glTexCoord2f(0.0f, sin(angle/ 100));
    glVertex2d(0, height);
    glTexCoord2f(1.0f, cos(angle / 100));

    glVertex2d(width, height);

    glTexCoord2f(1.0f, sin(angle / 100));
    glVertex2d(width, 0);

    glEnd();
```

Changing the texture coordinate can have this effect.

3, Object animation, each Solar System Planets are proportional to the real Solar System (Scaled the size and position for good looking).



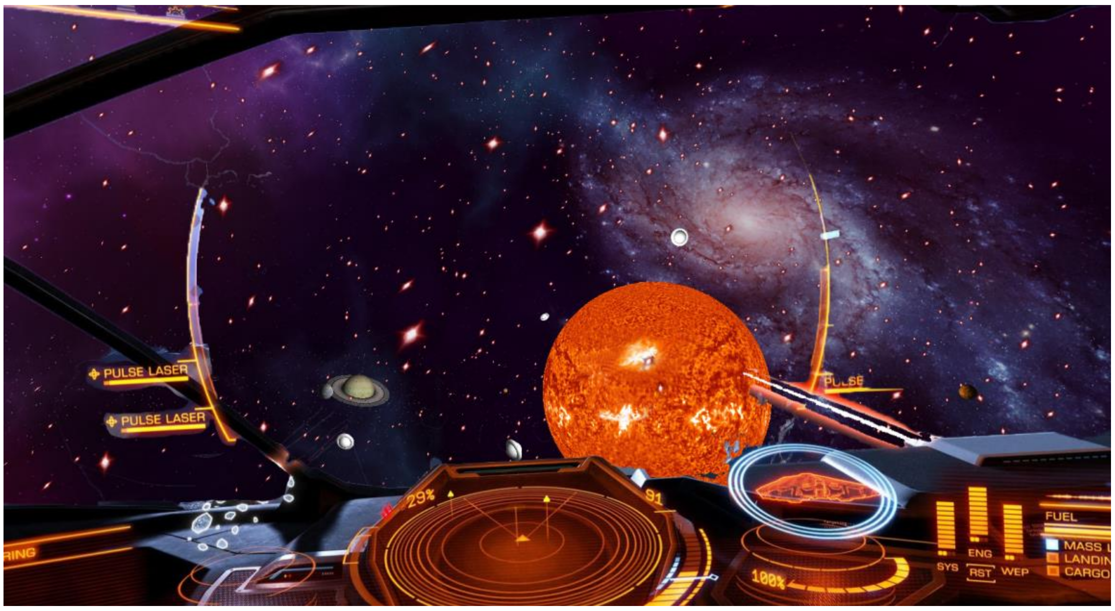
Code explanation:

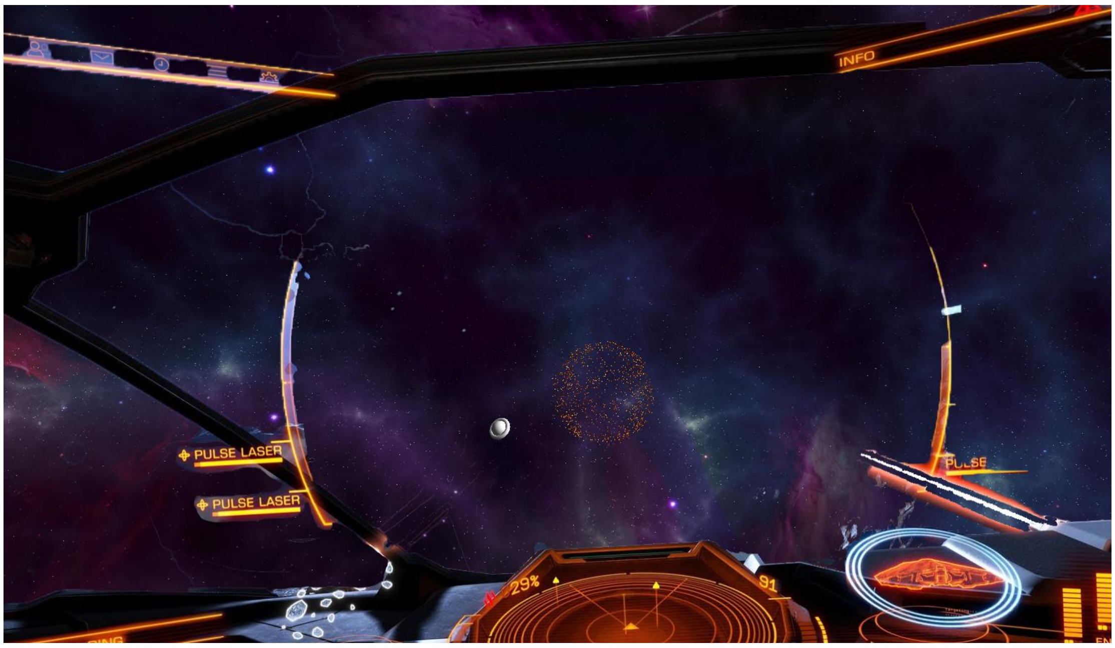


```
}

void DrawSolarSystem() {
    if (sun == NULL)
    {
        return;
    }
    //Sun should have light independently cause it have light it self.
#pragma region Sun
    SetMarital(1.0f);
    Lighting2(true);
    DrawPlanet(sun);

    Lighting2(false);
    SetMarital(0.5f);
#pragma endregion
    DrawPlanet(mercury);
    DrawPlanet(venus);
    DrawPlanet(earth);
    DrawPlanet(mars);
    DrawPlanet(jupiter);
    DrawPlanet(saturn);
    DrawPlanet(uranus);
    DrawPlanet(neptune);
    DrawPlanet(moon);
    DrawSaturnRing();
```

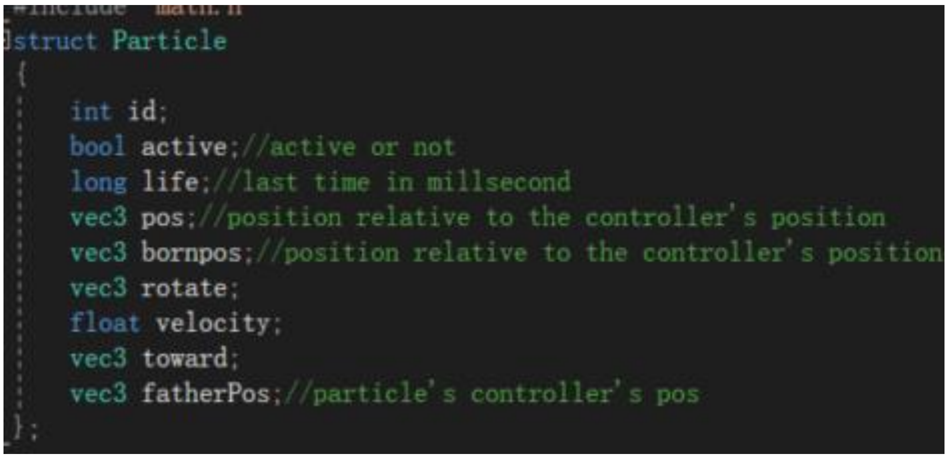
Each planet is abstract to class Planet, then use the function in Draw.h which is the drawer in the project to draw the planet.

4, Particle system, by loading the transparent textures, and building my own particle controller, I make a fire rain inside the Solar System, and an explosion effect when player hit the enemy UFO. The fire rain content 3000 fire particles, and randomly spread in the Solar System, for resource saving each particle is a triangle, so my particle system will not make a big difference of performance of the scene when it switches on or off. I rebuild the particle system after interim report, now I can have multiple effect by changing the toward and pos of particles. For example, the explosion.



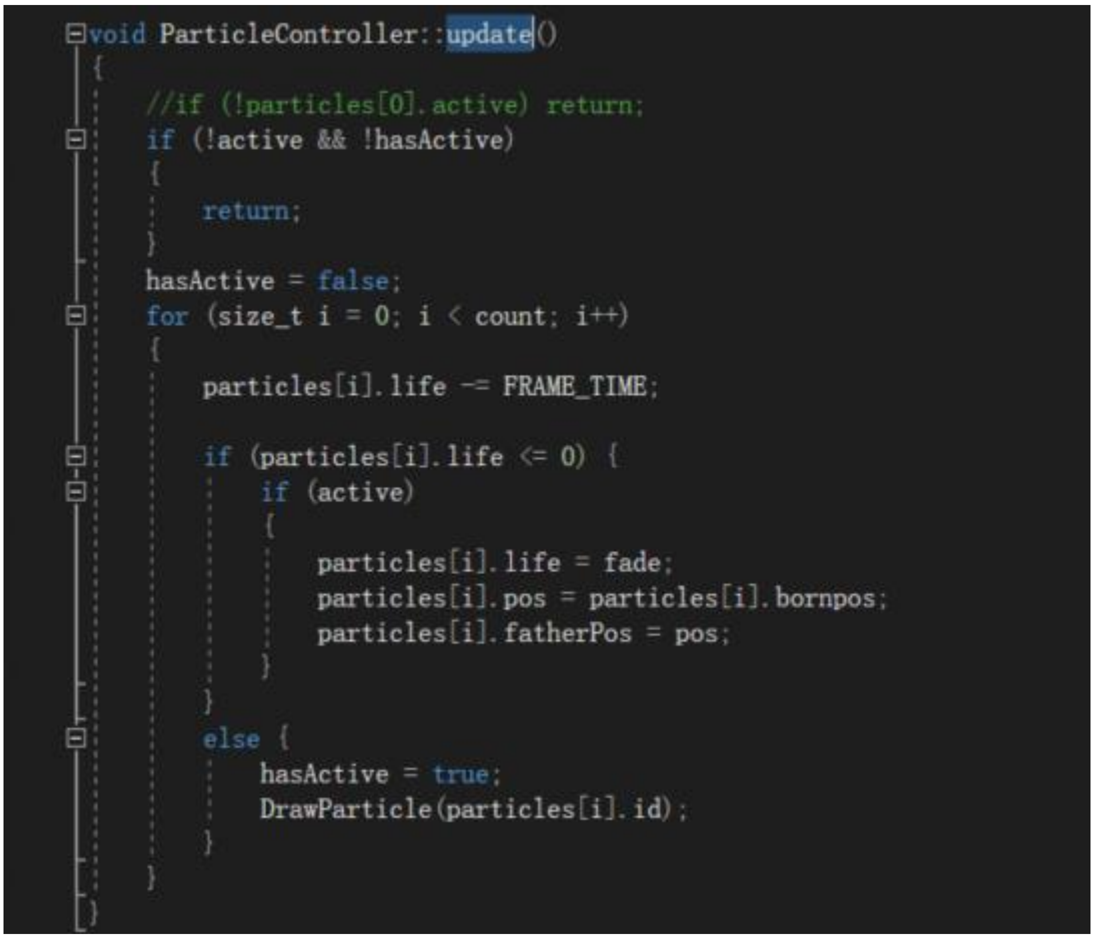


Code explanation:



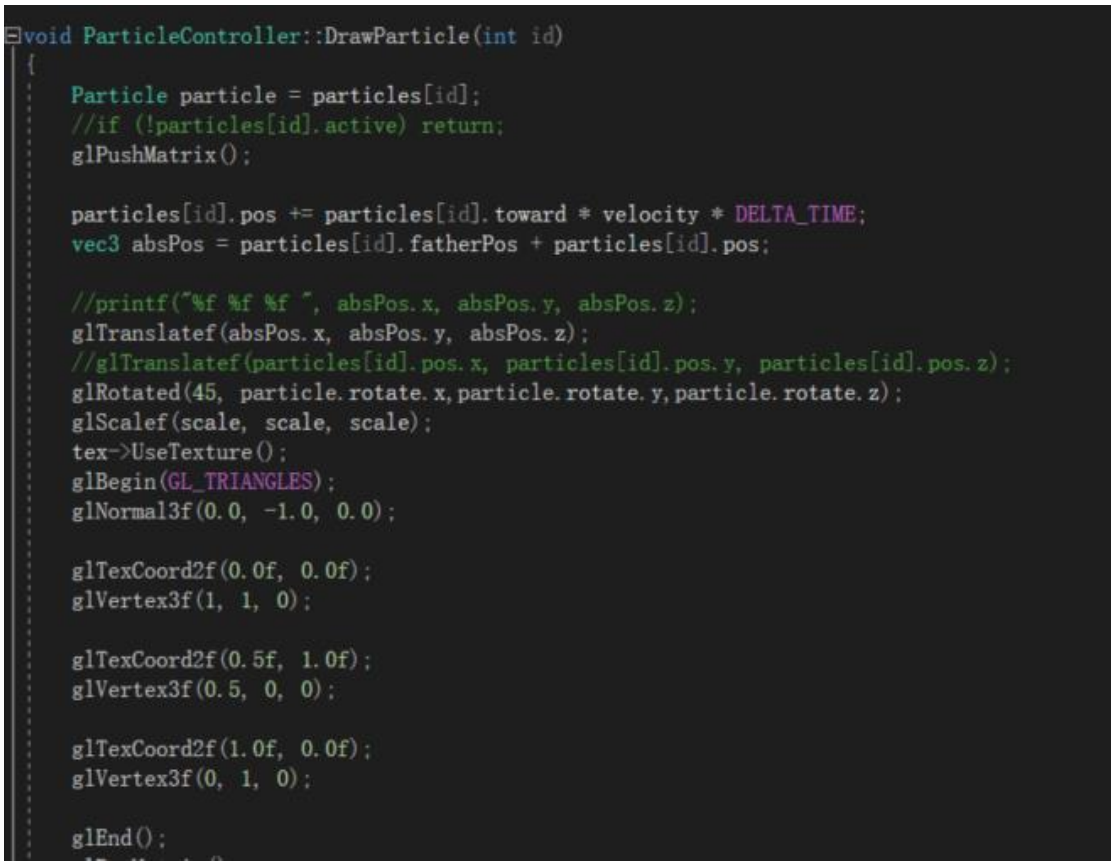
```
#include "math.h"
struct Particle
{
    int id;
    bool active;//active or not
    long life;//last time in millsecond
    vec3 pos;//position relative to the controller's position
    vec3 bornpos;//position relative to the controller's position
    vec3 rotate;
    float velocity;
    vec3 toward;
    vec3 fatherPos;//particle's controller's pos
};
```

Each particle has its own data.



```
void ParticleController::update()
{
    //if (!particles[0].active) return;
    if (!active && !hasActive)
    {
        return;
    }
    hasActive = false;
    for (size_t i = 0; i < count; i++)
    {
        particles[i].life -= FRAME_TIME;

        if (particles[i].life <= 0) {
            if (active)
            {
                particles[i].life = fade;
                particles[i].pos = particles[i].bornpos;
                particles[i].fatherPos = pos;
            }
        }
        else {
            hasActive = true;
            DrawParticle(particles[i].id);
        }
    }
}
```

Update each particle by the data of it.



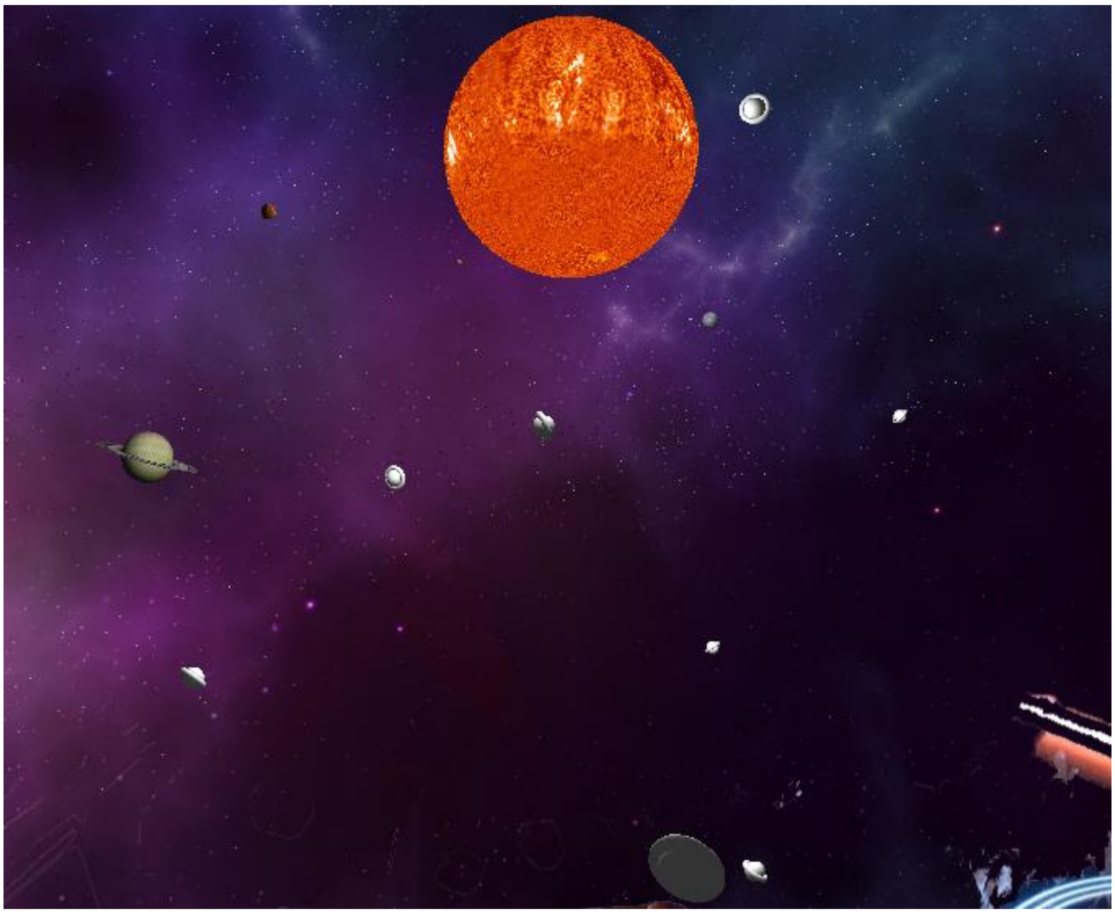
```
void ParticleController::DrawParticle(int id)
{
    Particle particle = particles[id];
    //if (!particles[id].active) return;
    glPushMatrix();

    particles[id].pos += particles[id].toward * velocity * DELTA_TIME;
    vec3 absPos = particles[id].fatherPos + particles[id].pos;

    //printf("%f %f %f ", absPos.x, absPos.y, absPos.z);
    glTranslatef(absPos.x, absPos.y, absPos.z);
    //glTranslatef(particles[id].pos.x, particles[id].pos.y, particles[id].pos.z);
    glRotated(45, particle.rotate.x,particle.rotate.y,particle.rotate.z);
    glScalef(scale, scale, scale);
    tex->UseTexture();
    glBegin(GL_TRIANGLES);
    glNormal3f(0.0, -1.0, 0.0);

    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(1, 1, 0);

    glTexCoord2f(0.5f, 1.0f);
    glVertex3f(0.5, 0, 0);

    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(0, 1, 0);

    glEnd();
```

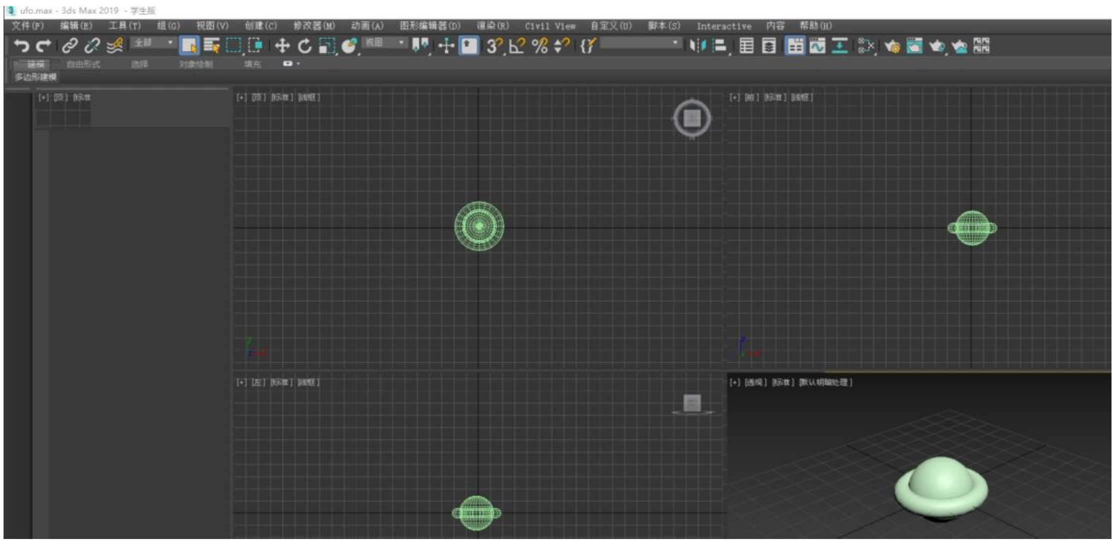
Draw each particle, using triangle for resource saving.

5, Model Loading and building

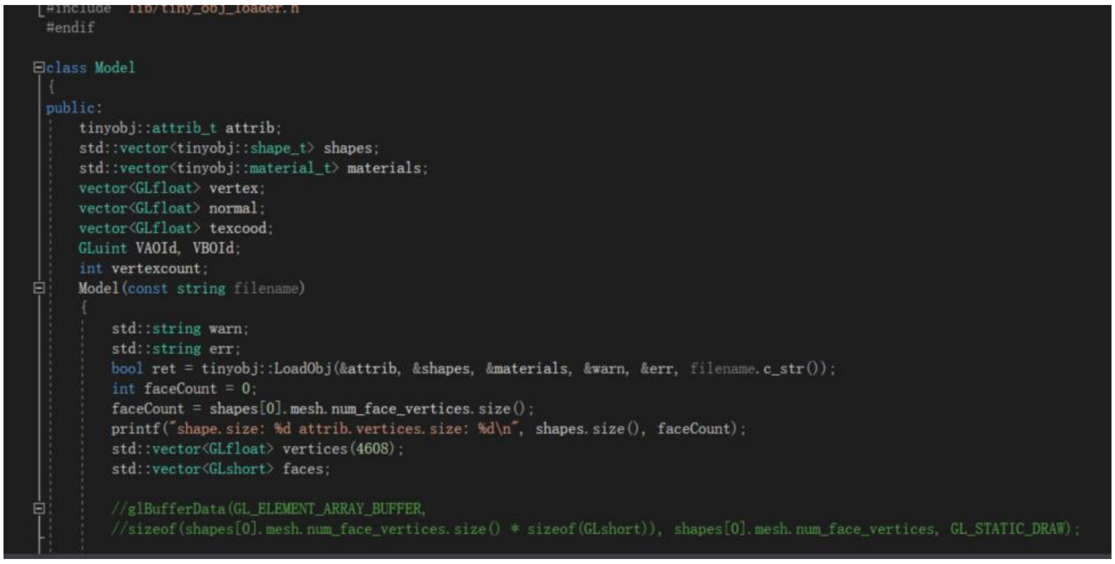
I import an obj loader library call tinyobjloader, this obj loader can load the obj from file system, then I encapsulate this library to class Model. Then I realize that the if I use glbegin to draw the model the program would be very slow even I just draw 3 or 4 UFO model, then I learn how to use OPENGL buffering and use gldrawarray to draw the model into my scene, now it can load up to 50 UFO models with not sensible effect.

And the UFO model is self-built by 3DSMax.



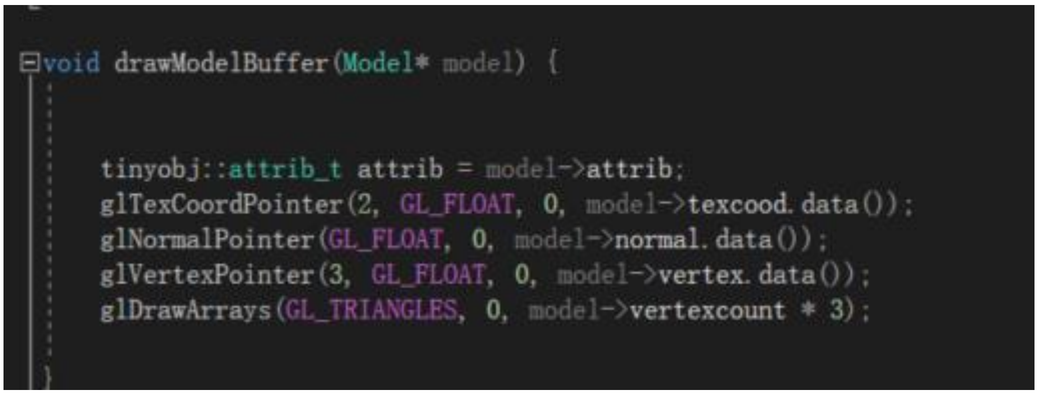


Code explanation:



```
#include "lib/tiny_obj_loader.h"
#endif

class Model
{
public:
    tinyobj::attrib_t attrib;
    std::vector<tinyobj::shape_t> shapes;
    std::vector<tinyobj::material_t> materials;
    vector<GLfloat> vertex;
    vector<GLfloat> normal;
    vector<GLfloat> texcood;
    GLuint VAOId, VBOId;
    int vertexcount;
    Model(const string filename)
    {
        std::string warn;
        std::string err;
        bool ret = tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &err, filename.c_str());
        int faceCount = 0;
        faceCount = shapes[0].mesh.num_face_vertices.size();
        printf("shape.size: %d attrib.vertices.size: %d\n", shapes.size(), faceCount);
        std::vector<GLfloat> vertices(4608);
        std::vector<GLshort> faces;

        //glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        //sizeof(shapes[0].mesh.num_face_vertices.size() * sizeof(GLshort)), shapes[0].mesh.num_face_vertices, GL_STATIC_DRAW);
```

A class Model for saving the model data for future usage.

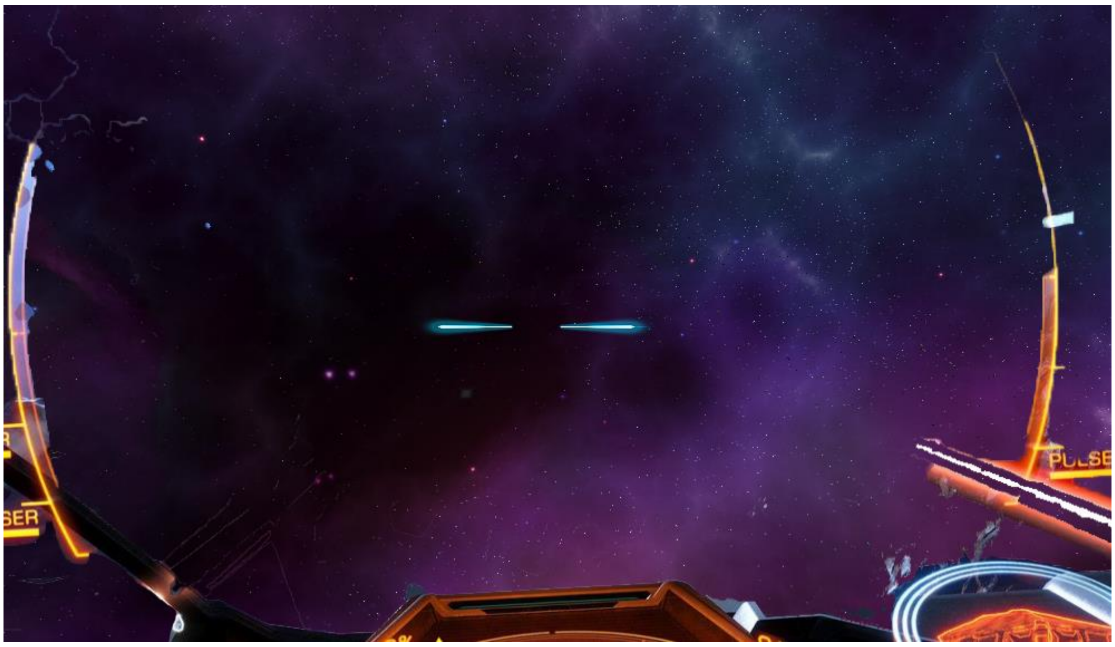


```
void drawModelBuffer(Model* model) {


    tinyobj::attrib_t attrib = model->attrib;
    glTexCoordPointer(2, GL_FLOAT, 0, model->texcood.data());
    glNormalPointer(GL_FLOAT, 0, model->normal.data());
    glVertexPointer(3, GL_FLOAT, 0, model->vertex.data());
    glDrawArrays(GL_TRIANGLES, 0, model->vertexcount * 3);

}
```

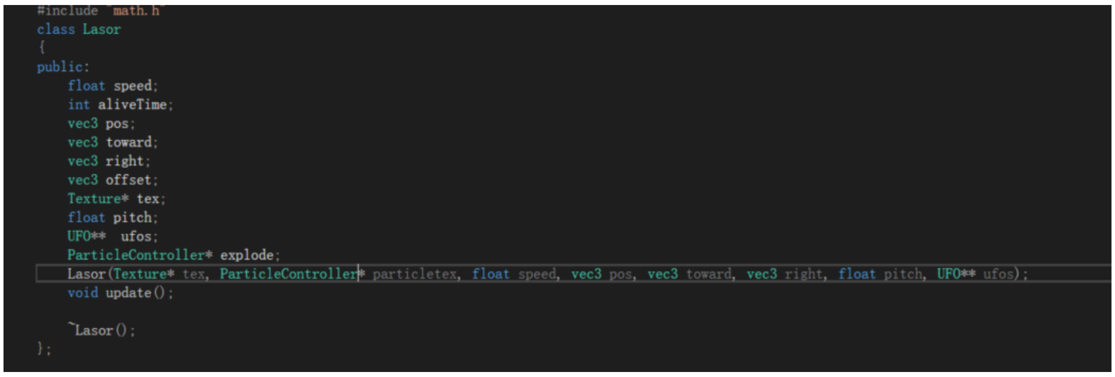
Using buffering in OPENGL for resource saving.

6, Laser

Player can shoot laser, when laser is hit the enemy UFO it will kill the UFO and there is an explosion effect at the position where UFO die.

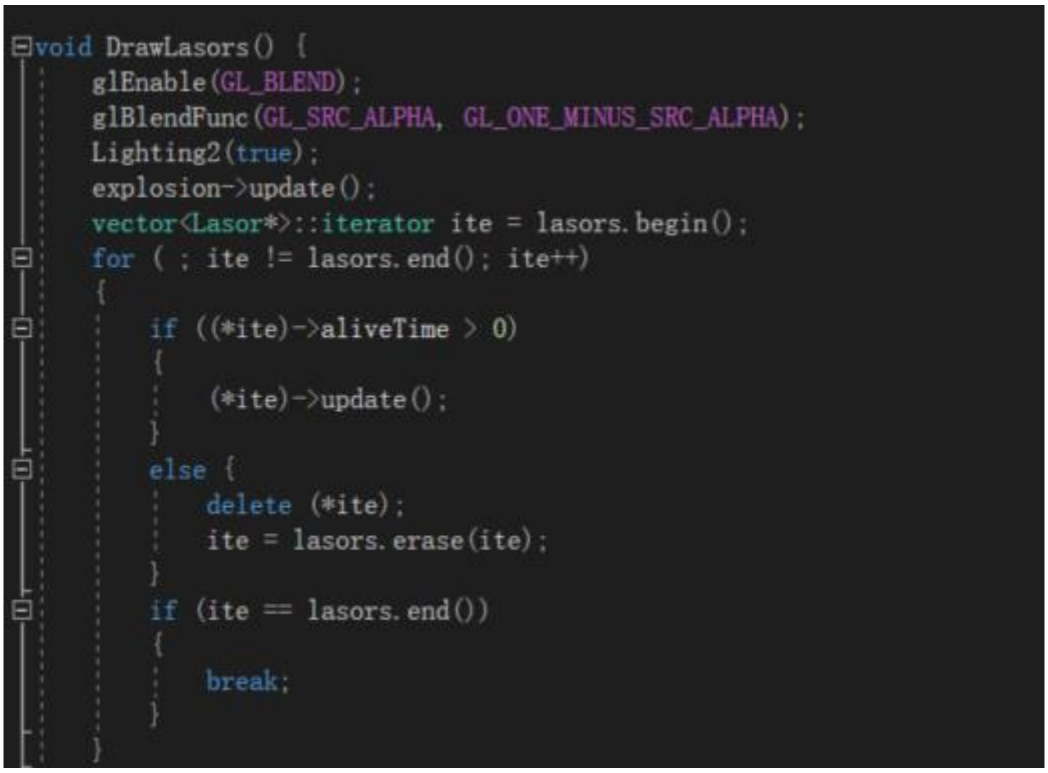


Code explanation:



```
#include "math.h"
class Lasor
{
public:
    float speed;
    int aliveTime;
    vec3 pos;
    vec3 toward;
    vec3 right;
    vec3 offset;
    Texture* tex;
    float pitch;
    UFO**  ufos;
    ParticleController* explode;
    Lasor(Texture* tex, ParticleController* particletex, float speed, vec3 pos, vec3 toward, vec3 right, float pitch, UFO** ufos);
    void update();

    ~Lasor();
};
```

Each laser is a class, I calculate the toward and position of laser according to camera position and facing.

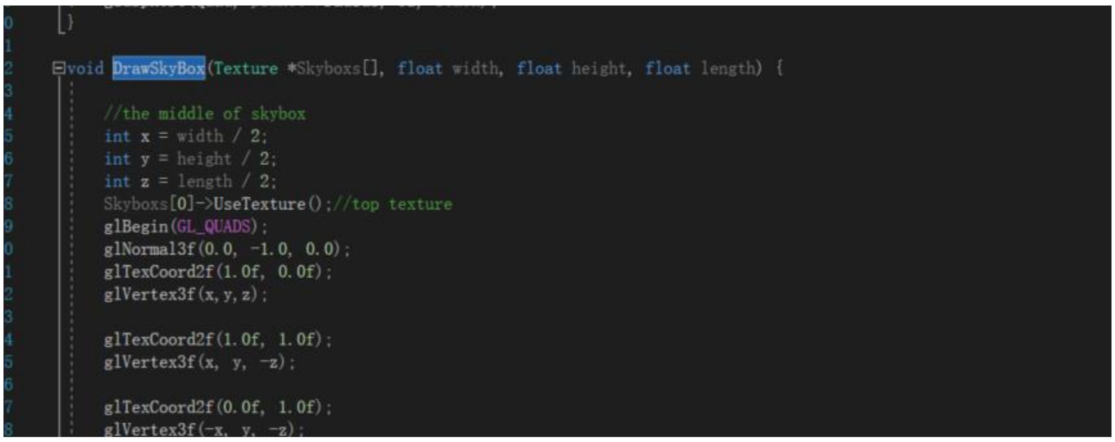


```cpp
void DrawLasors() {
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    Lighting2(true);
    explosion->update();
    vector<Lasor*>::iterator ite = lasors.begin();
    for ( ; ite != lasors.end(); ite++)
    {
        if ((*ite)->aliveTime > 0)
        {
            (*ite)->update();
        }
        else {
            delete (*ite);
            ite = lasors.erase(ite);
        }
        if (ite == lasors.end())
        {
            break;
        }
    }
```

There may have multiple laser exist in scene, so I must draw each one of them. When the laser is dead, I delete it from the vector for saving resource.

7, Skybox

A big cube around whole scene, I found the texture and make it connected smoothly.

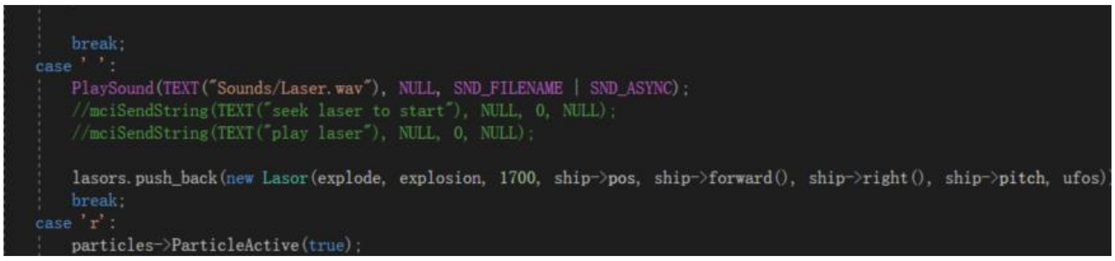


```cpp
}

void DrawSkyBox(Texture *Skyboxs[], float width, float height, float length) {

    //the middle of skybox
    int x = width / 2;
    int y = height / 2;
    int z = length / 2;
    Skyboxs[0]->UseTexture();//top texture
    glBegin(GL_QUADS);
    glNormal3f(0.0, -1.0, 0.0);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(x,y,z);

    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(x, y, -z);

    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(-x, y, -z);
```

Simple function just draws a big cube.

8, Sound

This part is out of range of OPENGL project, I add this to project just for fun.



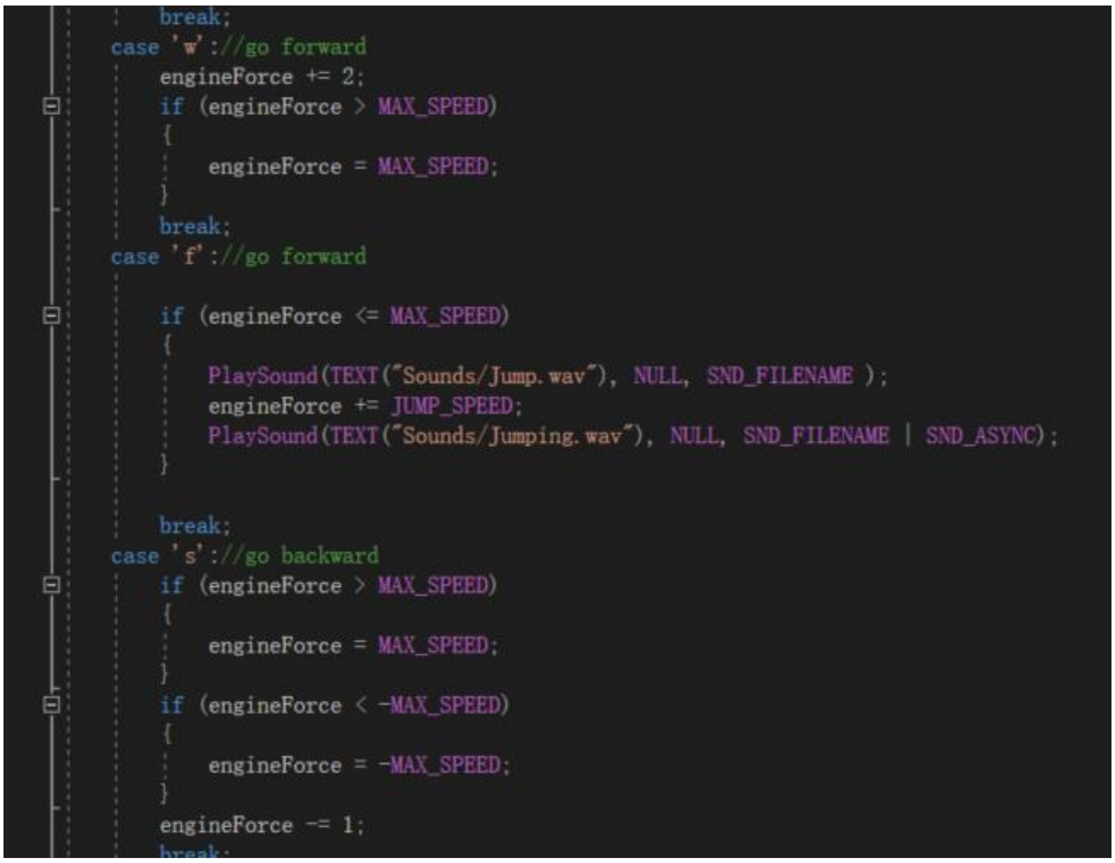
```
        break;
    case ' ':
        PlaySound(TEXT("Sounds/Laser.wav"), NULL, SND_FILENAME | SND_ASYNC);
        //mciSendString(TEXT("seek laser to start"), NULL, 0, NULL);
        //mciSendString(TEXT("play laser"), NULL, 0, NULL);

        lasors.push_back(new Lasor(explode, explosion, 1700, ship->pos, ship->forward(), ship->right(), ship->pitch, ufos)
        break;
    case 'r':
        particles->ParticleActive(true);
```

Using Microsoft library for play the sound.

9, Camera Control

Since player is controlling a space ship, I change the camera movement by increasing or decreasing the engine force, so the space ship would have its own velocity and acceleration. I also make a jump function into it, press ‘f’ and space ship would gain a large acceleration act like a space jump.

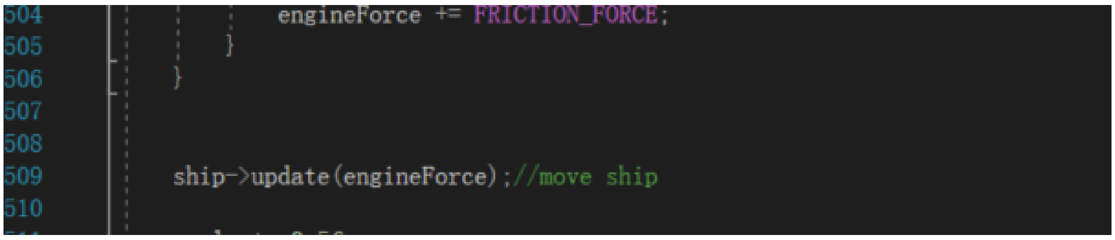
Code explanation:



```
        break;
    case 'w'://go forward
        engineForce += 2;
        if (engineForce > MAX_SPEED)
        {
            engineForce = MAX_SPEED;
        }
        break;
    case 'f'://go forward

        if (engineForce <= MAX_SPEED)
        {
            PlaySound(TEXT("Sounds/Jump.wav"), NULL, SND_FILENAME );
            engineForce += JUMP_SPEED;
            PlaySound(TEXT("Sounds/Jumping.wav"), NULL, SND_FILENAME | SND_ASYNC);
        }

        break;
    case 's'://go backward
        if (engineForce > MAX_SPEED)
        {
            engineForce = MAX_SPEED;
        }
        if (engineForce < -MAX_SPEED)
        {
            engineForce = -MAX_SPEED;
        }
        engineForce -= 1;
        break;
```

I give the camera a simulate force instead of simply moving by fixed value.

The controlling of it is better and more realistic than that.

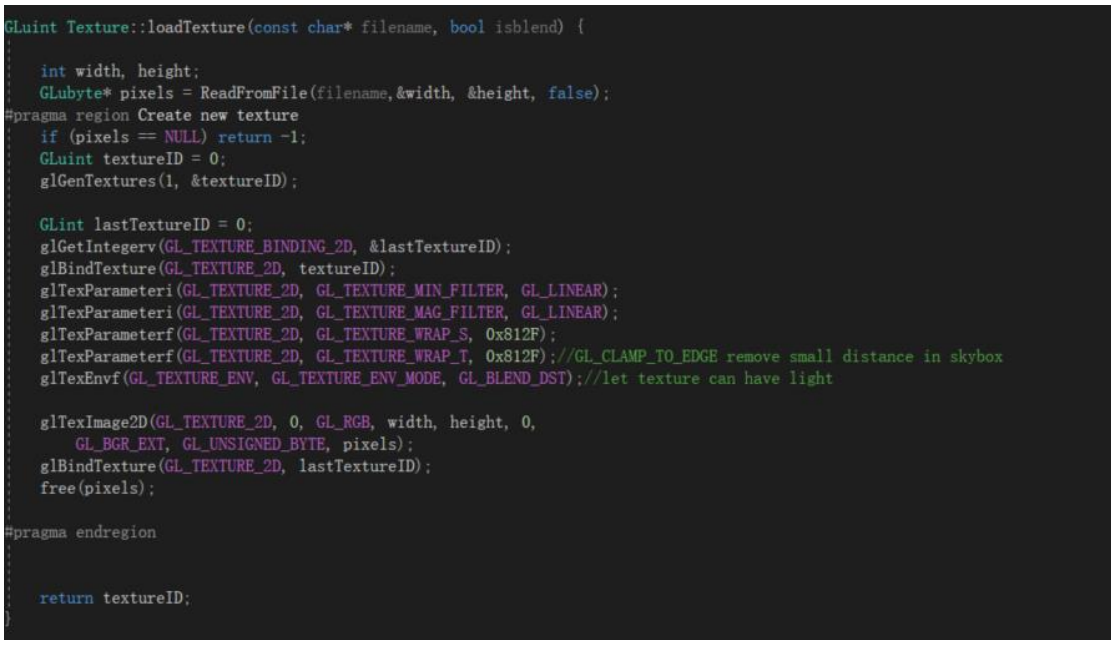


```
                engineForce += FRICTION_FORCE;
            }
        }


        ship->update(engineForce);//move ship
```

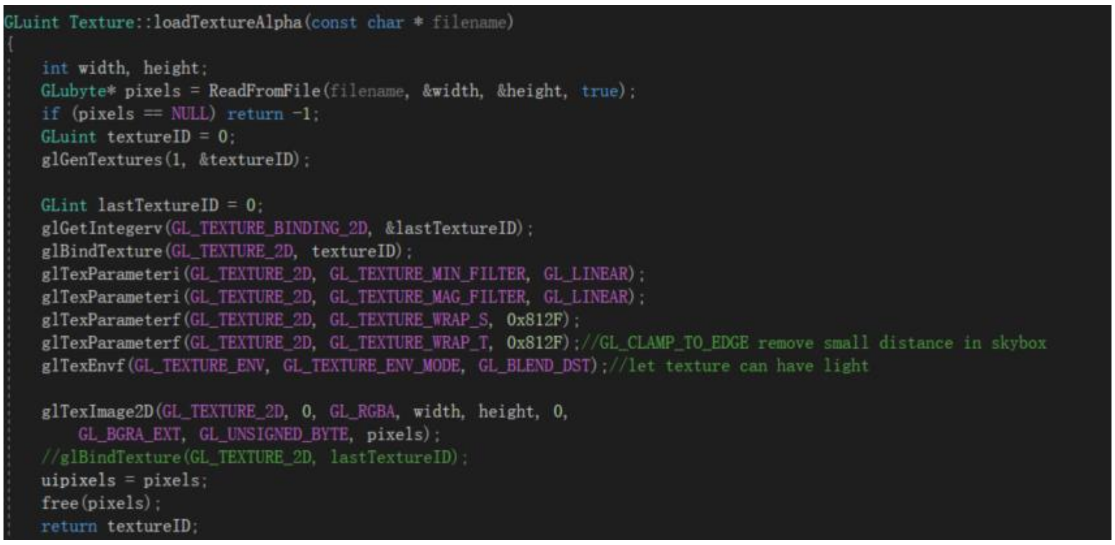
Update each frame by the force

10, Texture loader

Using opencv library for picture loading then bind it to opengl texture, can load both pic that have or don’t have alpha channel.



```
GLuint Texture::loadTexture(const char* filename, bool isblend) {

    int width, height;
    GLubyte* pixels = ReadFromFile(filename,&width, &height, false);
#pragma region Create new texture
    if (pixels == NULL) return -1;
    GLuint textureID = 0;
    glGenTextures(1, &textureID);

    GLint lastTextureID = 0;
    glGetIntegerv(GL_TEXTURE_BINDING_2D, &lastTextureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, 0x812F);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, 0x812F);//GL_CLAMP_TO_EDGE remove small distance in skybox
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND_DST);//let texture can have light

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
        GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);
    glBindTexture(GL_TEXTURE_2D, lastTextureID);
    free(pixels);

#pragma endregion


    return textureID;
}
```



```
GLuint Texture::loadTextureAlpha(const char * filename)
{
    int width, height;
    GLubyte* pixels = ReadFromFile(filename, &width, &height, true);
    if (pixels == NULL) return -1;
    GLuint textureID = 0;
    glGenTextures(1, &textureID);

    GLint lastTextureID = 0;
    glGetIntegerv(GL_TEXTURE_BINDING_2D, &lastTextureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, 0x812F);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, 0x812F);//GL_CLAMP_TO_EDGE remove small distance in skybox
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND_DST);//let texture can have light

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
        GL_BGRA_EXT, GL_UNSIGNED_BYTE, pixels);
    //glBindTexture(GL_TEXTURE_2D, lastTextureID);
    uipixels = pixels;
    free(pixels);
    return textureID;
```

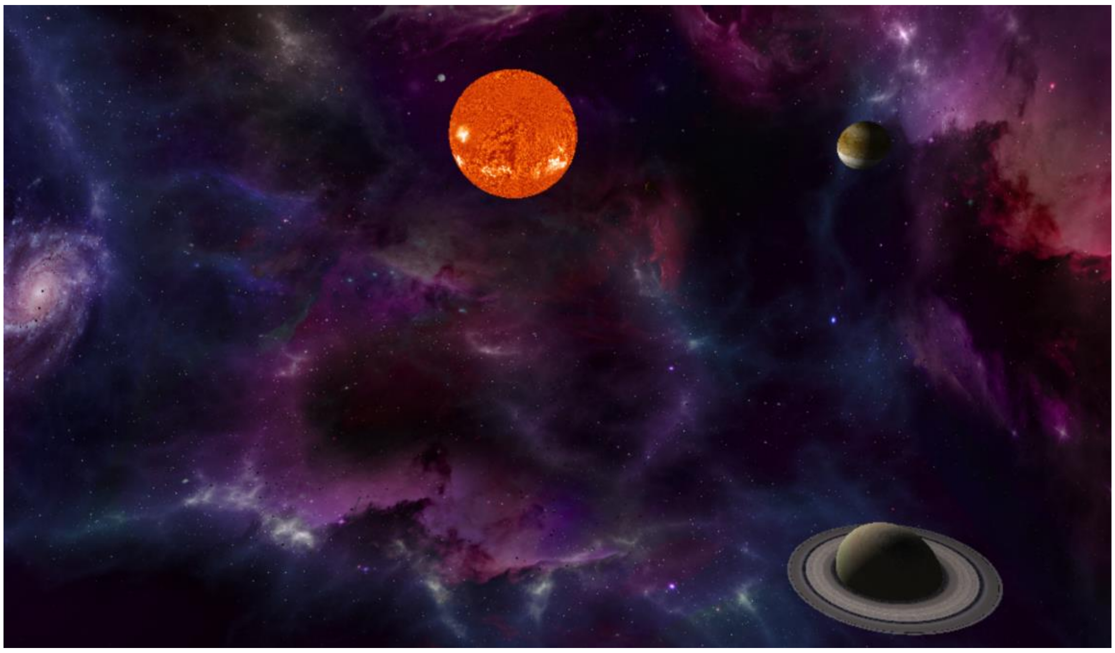
Two function one for not alpha channel picture, one for the picture that have alpha channel.

11, Solar System

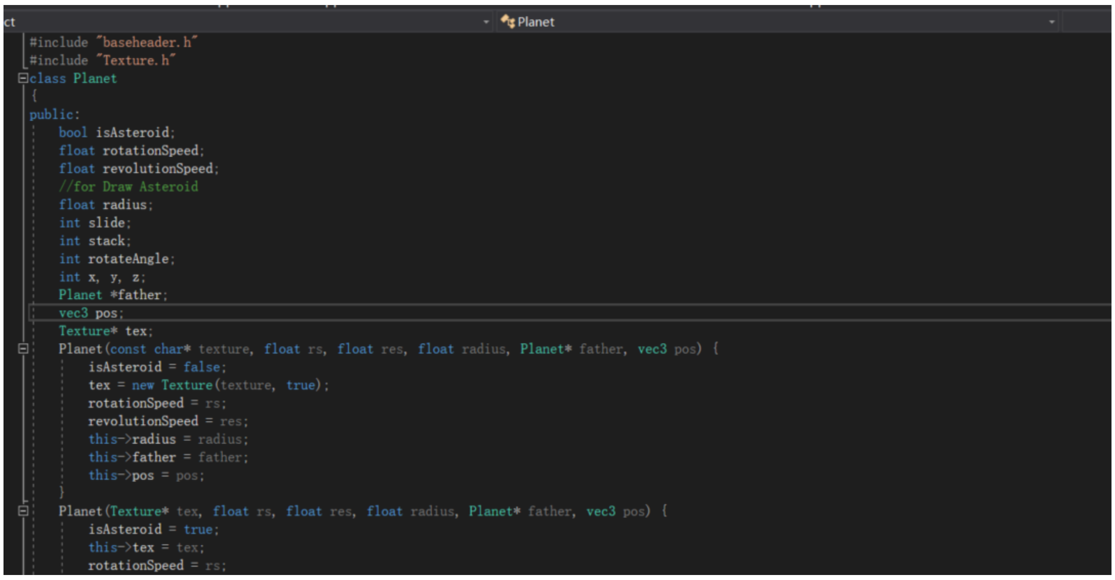
Which already explained in interim report, there is not much changing of this part.

Solar system:

Whole scene:

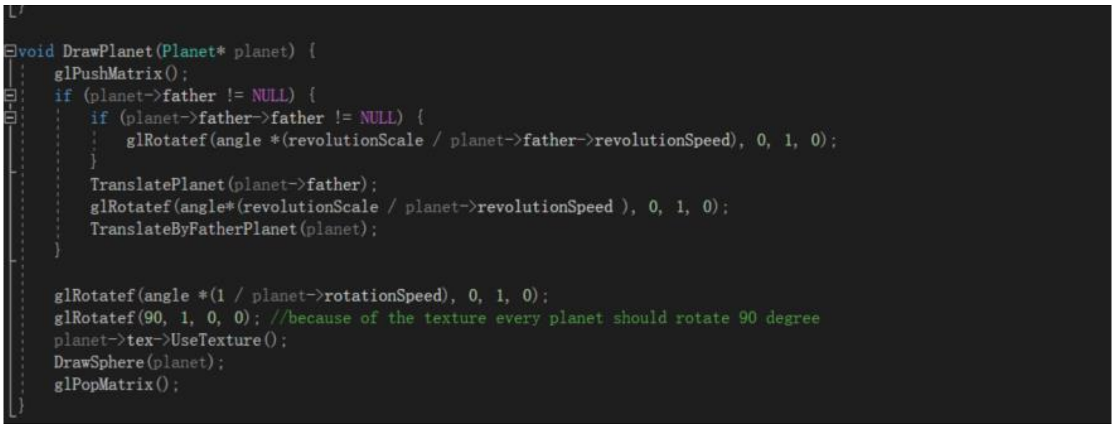


Code explanation:



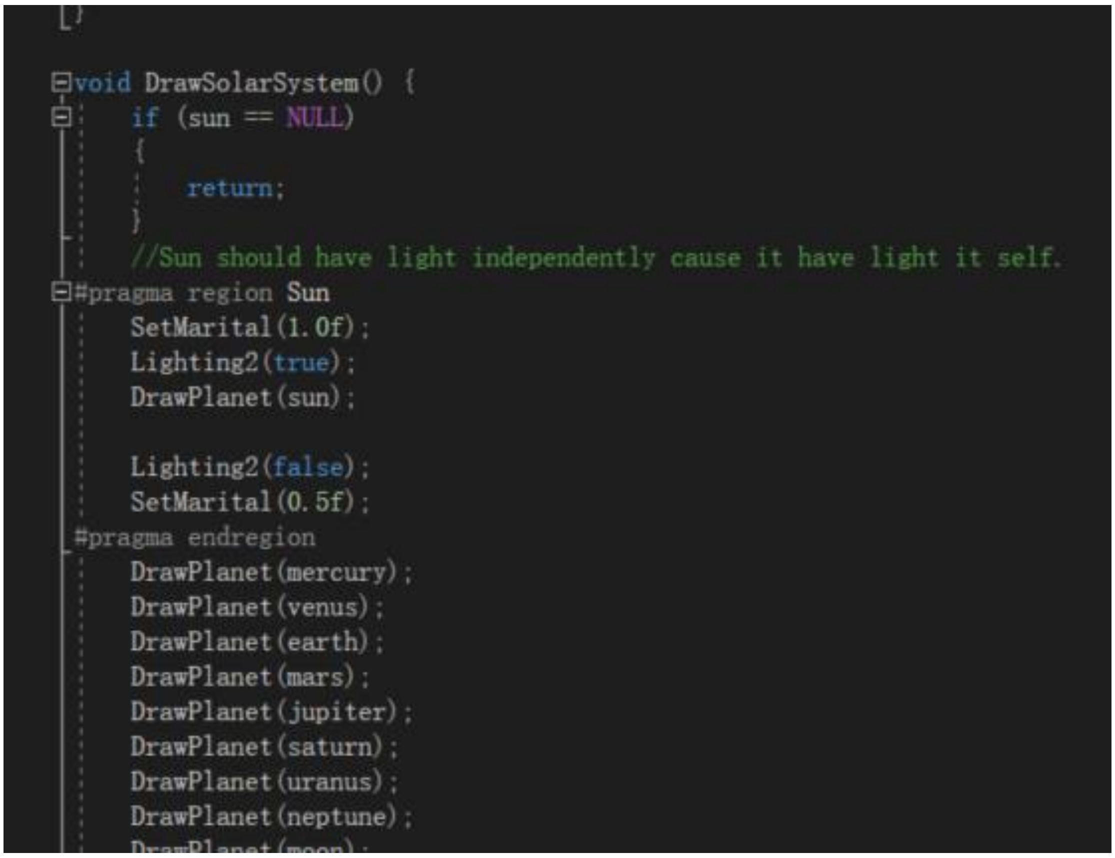
```
#include "baseheader.h"
#include "Texture.h"
class Planet
{
public:
    bool isAsteroid;
    float rotationSpeed;
    float revolutionSpeed;
    //for Draw Asteroid
    float radius;
    int slide;
    int stack;
    int rotateAngle;
    int x, y, z;
    Planet *father;
    vec3 pos;
    Texture* tex;
    Planet(const char* texture, float rs, float res, float radius, Planet* father, vec3 pos) {
        isAsteroid = false;
        tex = new Texture(texture, true);
        rotationSpeed = rs;
        revolutionSpeed = res;
        this->radius = radius;
        this->father = father;
        this->pos = pos;
    }
    Planet(Texture* tex, float rs, float res, float radius, Planet* father, vec3 pos) {
        isAsteroid = true;
        this->tex = tex;
        rotationSpeed = rs;
```

each planet is a class, has its own data.



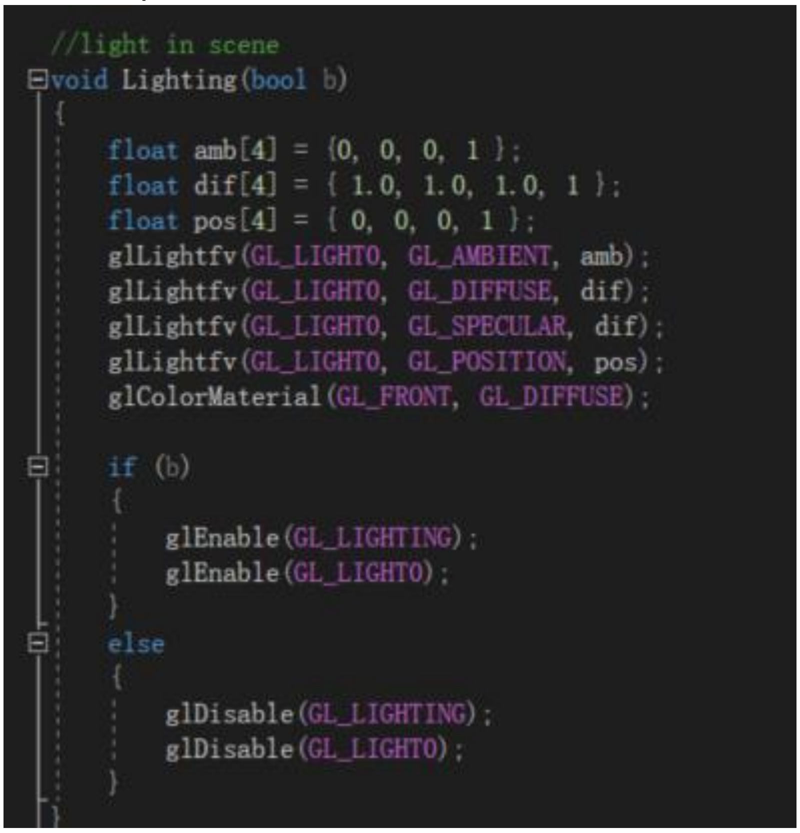
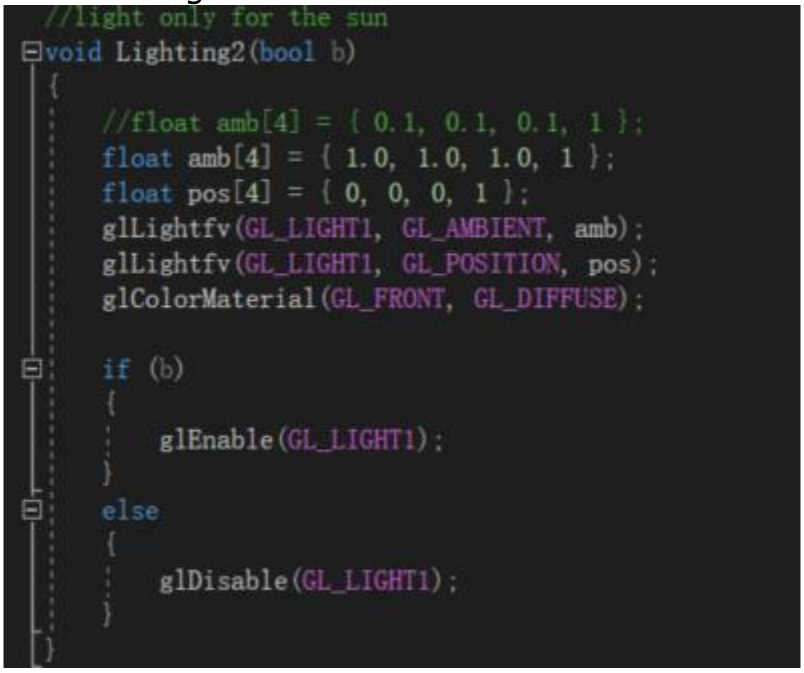
```
void DrawPlanet(Planet* planet) {
    glPushMatrix();
    if (planet->father != NULL) {
        if (planet->father->father != NULL) {
            glRotatef(angle *(revolutionScale / planet->father->revolutionSpeed), 0, 1, 0);
        }
        TranslatePlanet(planet->father);
        glRotatef(angle*(revolutionScale / planet->revolutionSpeed ), 0, 1, 0);
        TranslateByFatherPlanet(planet);
    }

    glRotatef(angle *(1 / planet->rotationSpeed), 0, 1, 0);
    glRotatef(90, 1, 0, 0); //because of the texture every planet should rotate 90 degree
    planet->tex->UseTexture();
    DrawSphere(planet);
    glPopMatrix();
}
```

Draw planet can draw each planet respectively of its own data.



```
void DrawSolarSystem() {
    if (sun == NULL)
    {
        return;
    }
    //Sun should have light independently cause it have light it self.
#pragma region Sun
    SetMarital(1.0f);
    Lighting2(true);
    DrawPlanet(sun);

    Lighting2(false);
    SetMarital(0.5f);
#pragma endregion
    DrawPlanet(mercury);
    DrawPlanet(venus);
    DrawPlanet(earth);
    DrawPlanet(mars);
    DrawPlanet(jupiter);
    DrawPlanet(saturn);
    DrawPlanet(uranus);
    DrawPlanet(neptune);
    DrawPlanet(moon);
```

And use DrawSolarSystem to draw the whole solar system.

11, Light
Use two light one for sun, the other for others, I need a extra light for rendering the Sun since Sun has its own light.
Code explanation:

```
//light in scene
void Lighting(bool b)
{
    float amb[4] = {0, 0, 0, 1 };
    float dif[4] = { 1.0, 1.0, 1.0, 1 };
    float pos[4] = { 0, 0, 0, 1 };
    glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, dif);
    glLightfv(GL_LIGHT0, GL_SPECULAR, dif);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glColorMaterial(GL_FRONT, GL_DIFFUSE);

    if (b)
    {
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
    }
    else
    {
        glDisable(GL_LIGHTING);
        glDisable(GL_LIGHT0);
    }
}
```
First one in the center of whole scene, but it cannot render Sun since the light is inside the model of the Sun.

```
//light only for the sun
void Lighting2(bool b)
{
    //float amb[4] = { 0.1, 0.1, 0.1, 1 };
    float amb[4] = { 1.0, 1.0, 1.0, 1 };
    float pos[4] = { 0, 0, 0, 1 };
    glLightfv(GL_LIGHT1, GL_AMBIENT, amb);
    glLightfv(GL_LIGHT1, GL_POSITION, pos);
    glColorMaterial(GL_FRONT, GL_DIFFUSE);

    if (b)
    {
        glEnable(GL_LIGHT1);
    }
    else
    {
        glDisable(GL_LIGHT1);
    }
}
```
Give Sun the max value of ambient light.

Project controlling:

“wasd” and mouse move for changing the view and move forward backward inside the scene.

Press left mouse for shoot laser.

Press ‘f’ for active jump engine (space ship would gain a large acceleration and would not stop until you in solar system or press ‘s’).

Press ‘r’ for active fire rain particle system in Solar System.

Press ‘m’ for deactive fire rain.

Press ‘q’ for active or deactive the shield.

Press ‘ ’ for shoot read laser.