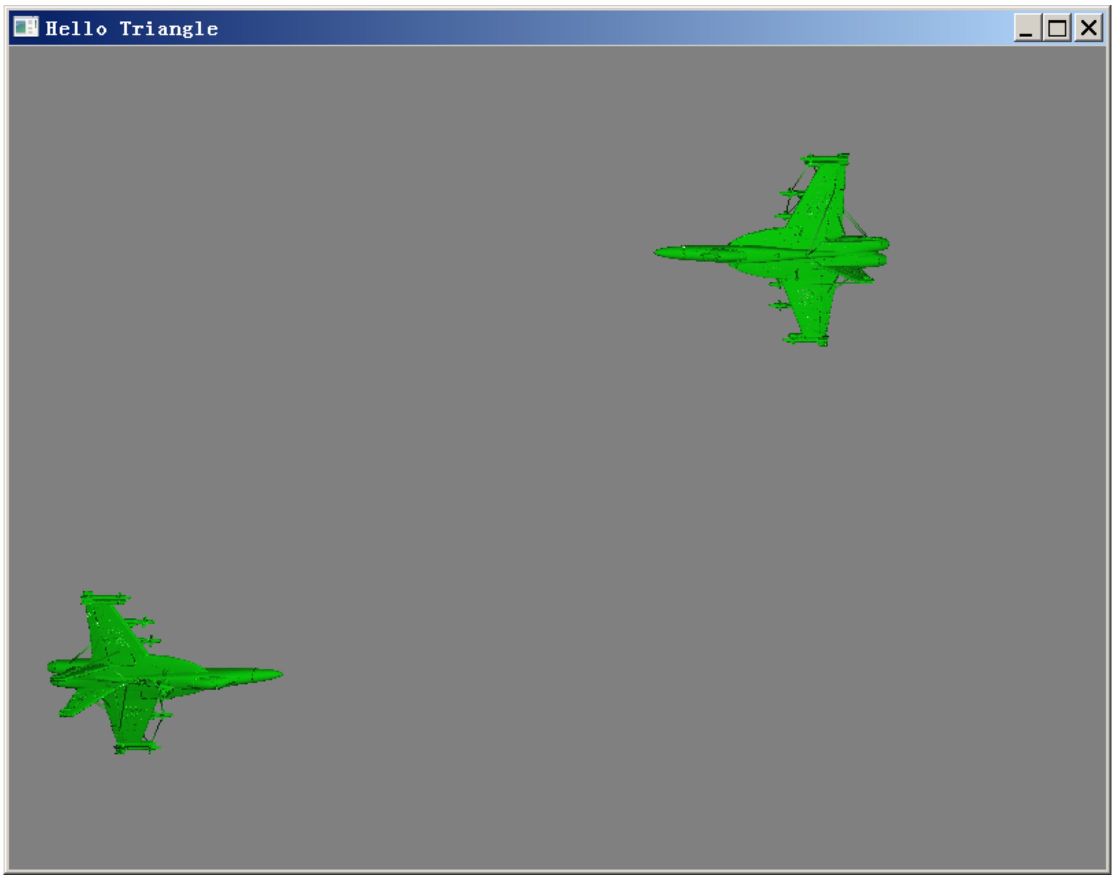
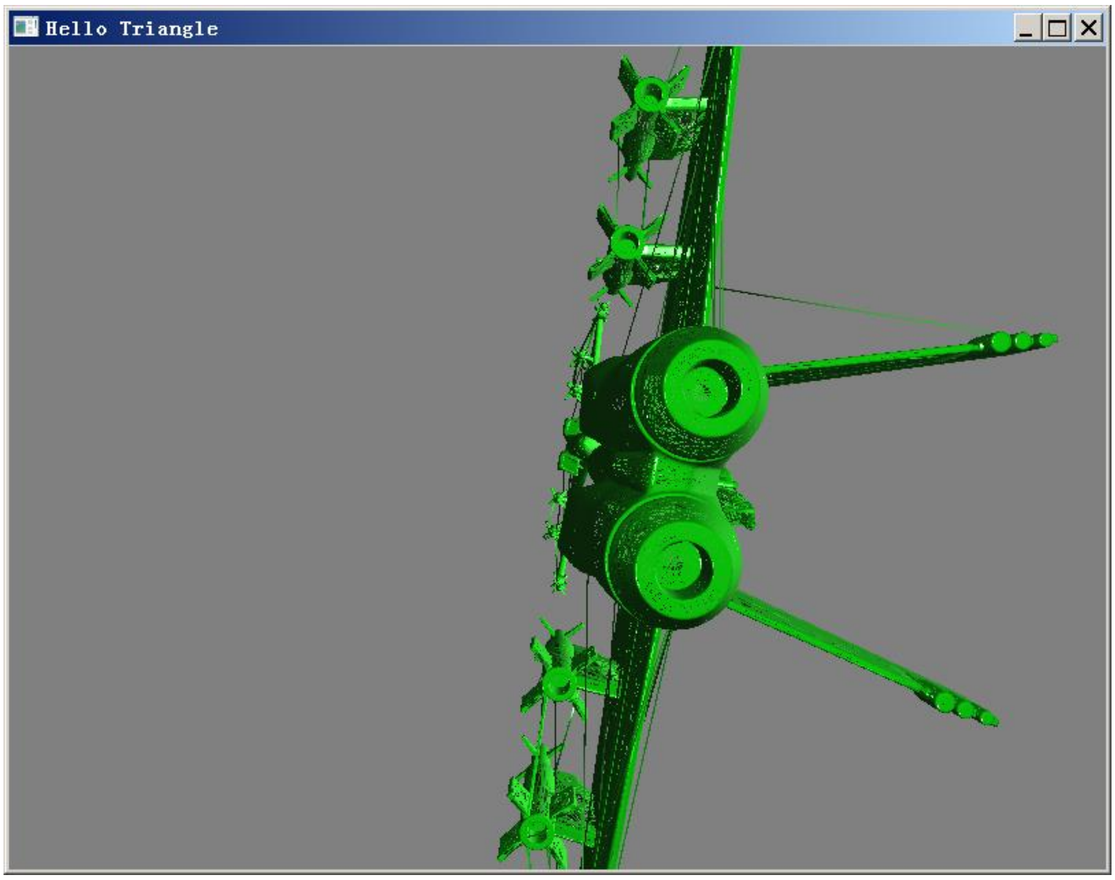
Report

Youtube Link: https://youtu.be/RiClqEjdHm0

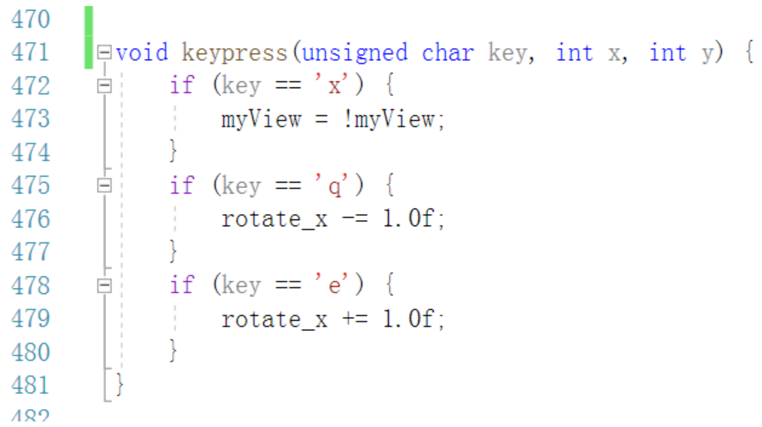
Two 3D mesh model files are used in my design, one is the aircraft mesh model file Jet USA_ V1.obj, one is the machine off grid model file MachinegunTurret.obj. These two network model files are in obj format. I used Assist to import these two 3D grid model files. The aircraft model was imported twice, that is, two aircraft. I put one aircraft on the left side of the scene and one aircraft on the right side of the scene to play a game of two aircraft attacking each other.



I put the imported 3D mesh model of the machine gun on the left plane. The right plane has no machine gun. The size of the machine gun model is much smaller than that of the aircraft model, which can only be seen by zooming in.

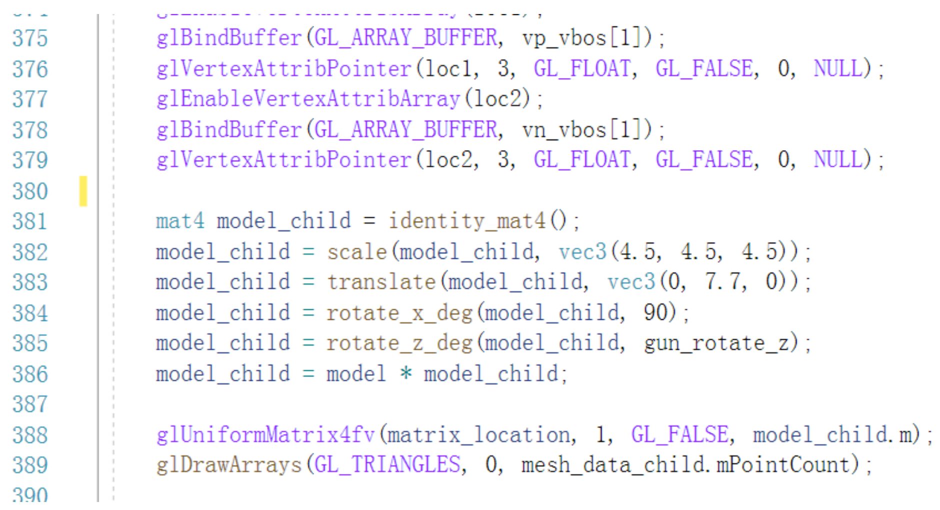


I used the keyboard for game interaction, mainly to control the rotation speed of the aircraft. When I press the key q, the rotation speed of the aircraft will slow down, and when I press the key e, the rotation speed of the aircraft will speed up.



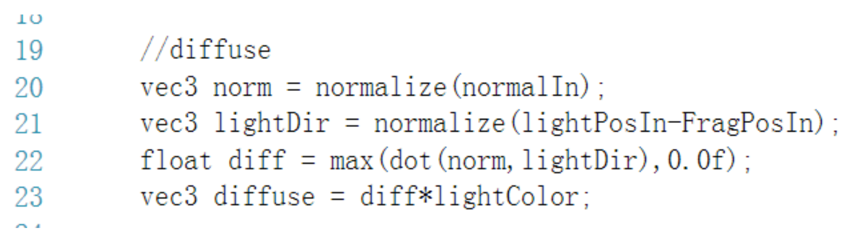
```
void keypress(unsigned char key, int x, int y) {
    if (key == 'x') {
        myView = !myView;
    }
    if (key == 'q') {
        rotate_x -= 1.0f;
    }
    if (key == 'e') {
        rotate_x += 1.0f;
    }
}
```

The aircraft model and machine gun model are hierarchical structures. When the aircraft model is transformed, the machine gun model can transform with the aircraft model, and the machine gun model can rotate at the same time. The specific implementation process is as follows: the aircraft model is the root of the Hierarchy, and the machine gun model is the child model of Hierarchy. First do the transformation of the aircraft, then do the transformation of the machine gun model, and finally combine the transformations of the two models.



```
glBindBuffer(GL_ARRAY_BUFFER, vp_vbos[1]);
glVertexAttribPointer(loc1, 3, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(loc2);
glBindBuffer(GL_ARRAY_BUFFER, vn_vbos[1]);
glVertexAttribPointer(loc2, 3, GL_FLOAT, GL_FALSE, 0, NULL);

mat4 model_child = identity_mat4();
model_child = scale(model_child, vec3(4.5, 4.5, 4.5));
model_child = translate(model_child, vec3(0, 7.7, 0));
model_child = rotate_x_deg(model_child, 90);
model_child = rotate_z_deg(model_child, gun_rotate_z);
model_child = model * model_child;

glUniformMatrix4fv(matrix_location, 1, GL_FALSE, model_child.m);
glDrawArrays(GL_TRIANGLES, 0, mesh_data_child.mPointCount);
```

The lighting calculation is put in the segment shader program. First, calculate the diffuse. The normal vector used in the calculation is passed from the vertex shader. Use the normal vector, the position vector of the light source, and the position vector of the segment to calculate the diffuse. First, normalize the normal vector, then normalize the difference between the position vector of the light source and the position vector of the segment, and finally do the dot multiplication. If the angle between the two vectors is greater than 90 degrees, The dot multiplication results in a negative number, which causes the diffuse reflection to become negative. Use the max function to avoid this situation. If it is a negative number, it will return 0.0f to ensure that the diffuse reflection will not become a negative number. Multiply the dot multiplication result by the color of the light, and you will get the diffuse light.



```
    //diffuse
    vec3 norm = normalize(normalIn);
    vec3 lightDir = normalize(lightPosIn-FragPosIn);
    float diff = max(dot(norm,lightDir),0.0f);
    vec3 diffuse = diff*lightColor;
```

Next, calculate specular light. Like diffuse light, specular light also depends on the direction vector of light and the normal vector of the object, but it also depends on the viewing direction, such as the direction from which the player looks at the segment. Specular illumination depends on the reflective characteristics of the surface. If we think of the object surface as a mirror, the place where the mirror light is strongest is where we see the reflected light on the surface. The reflection vector is calculated by reversing the direction of the incident light according to the normal vector. Then we calculate the angle difference between the reflection vector and the observation direction. The smaller the angle between them, the greater the effect of the mirror light. The result is that when we look at the reflection direction of the incident light on the surface, we will see a little highlight. The observation vector is an additional variable we need to calculate the mirror illumination. We can use the world space position of the observer and the position of the clip to calculate it. Then we calculate the light intensity of the mirror surface, multiply it by the color of the light source, and combine it with the ambient light and the diffuse light to form the final lighting effect.

To get the world space coordinates of the observer, we directly use the camera's position vector to add a uniform vec3 viewPos to the clip shader; Use this uniform to transmit the position vector of the camera.

Now that we have all the required variables, we can calculate the highlight intensity. First, we define a specular intensity variable float specularStrength=0.5; Next, we calculate the line of sight direction vector and the corresponding reflection vector along the normal axis:

```
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
```

Note that we negate the lightDir vector. The reflect function requires that the first vector is the vector pointing from the light source to the position of the clip, but lightDir is currently the opposite, pointing from the clip to the light source (determined by the order of subtraction when we calculated the lightDir vector earlier). To ensure that we get the correct reflection vector, we get the opposite direction by reversing the lightDir vector. The second parameter is required to be a normal vector, so we provide a normalized norm vector. Finally, we calculate the specular component. First, we calculate the dot product of the line of sight direction and the reflection direction (and ensure that it is not negative), and then take it to the 32nd power. This 32 is the shininess of the highlight. The higher the reflectivity of an object, the stronger the ability to reflect light, and the less it scatters, the smaller the highlight will be:

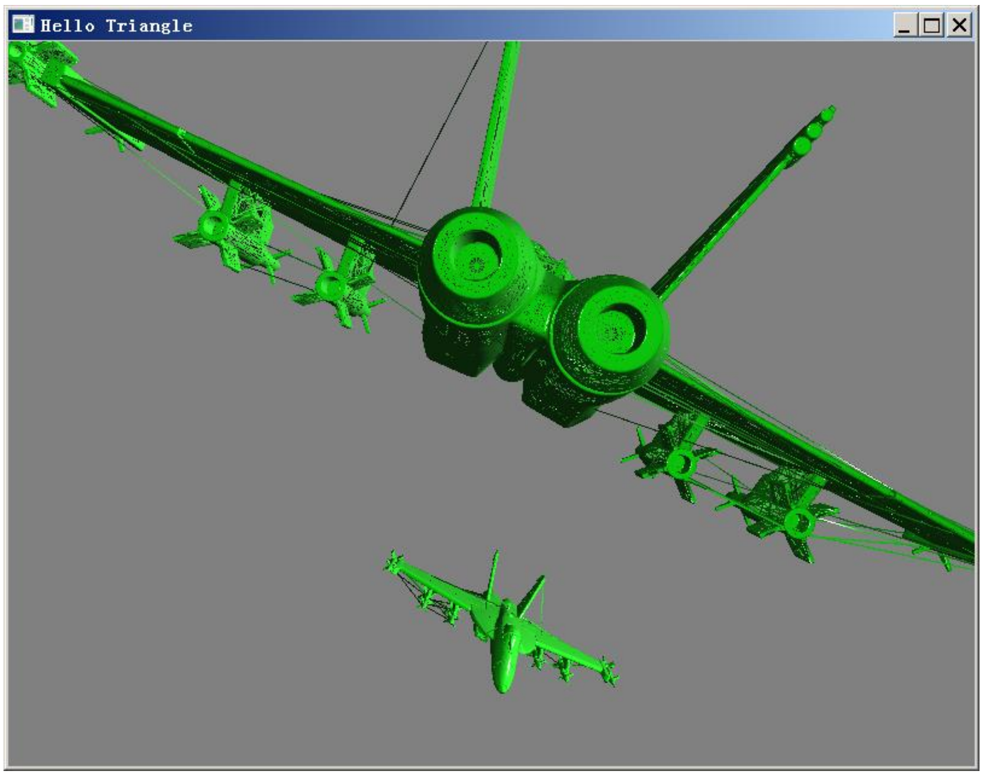
```
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```

The complete lighting shader procedure is as follows:

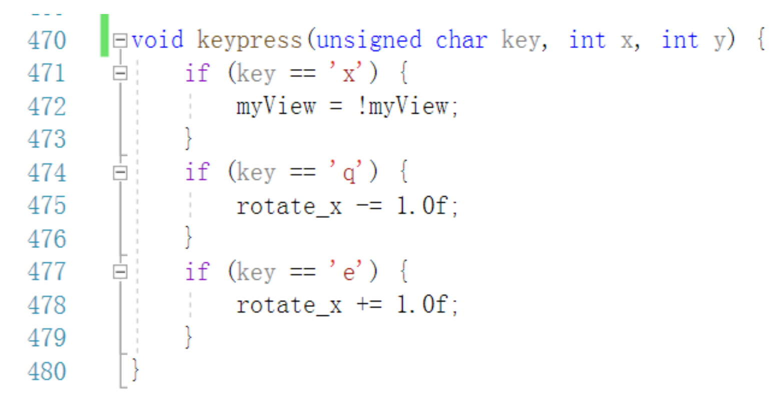
```
#version 330
out vec4 color;


uniform vec3 viewPos;


in vec3 LightIntensity;
in vec3 FragPosIn;
in vec3 normalIn;
in vec3 lightPosIn;
```

```
void main(){

    vec3 lightColor = vec3 (1.0f,1.0f,1.0f);

    vec3 objectColor = vec3 (0.2f,0.7f,0.2f);


    //ambient

    float ambientStrength = 0.2f;

    vec3 ambient = ambientStrength * lightColor;


    //diffuse

    vec3 norm = normalize(normalIn);

    vec3 lightDir = normalize(lightPosIn-FragPosIn);

    float diff = max(dot(norm,lightDir),0.0f);

    vec3 diffuse = diff*lightColor;


    //specular

    int power = 64;

    float specularStrength = 7f;

    vec3 viewDir = normalize(- FragPosIn);

    vec3 reflectDir = reflect(-lightDir,norm);

    float spec = pow(max(dot(viewDir,reflectDir),0.0),power);

    vec3 specular = specularStrength * spec * lightColor;
    //mix

    vec3 result = (ambient+diffuse+specular)*objectColor;

    color = vec4(LightIntensity,1.0f) + vec4(result,1.0f);

}
```

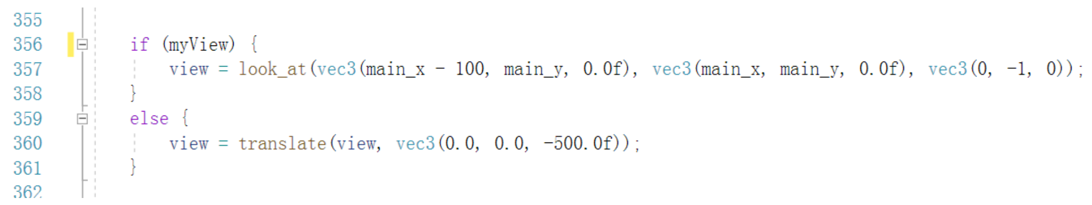
Apply specular mirror light to our airplane model. Look at the small airplane in the following figure. The effect is good.



My game has two camera views, one is the top down view, and the other is the first person view. You can use the key x to switch between the two views.

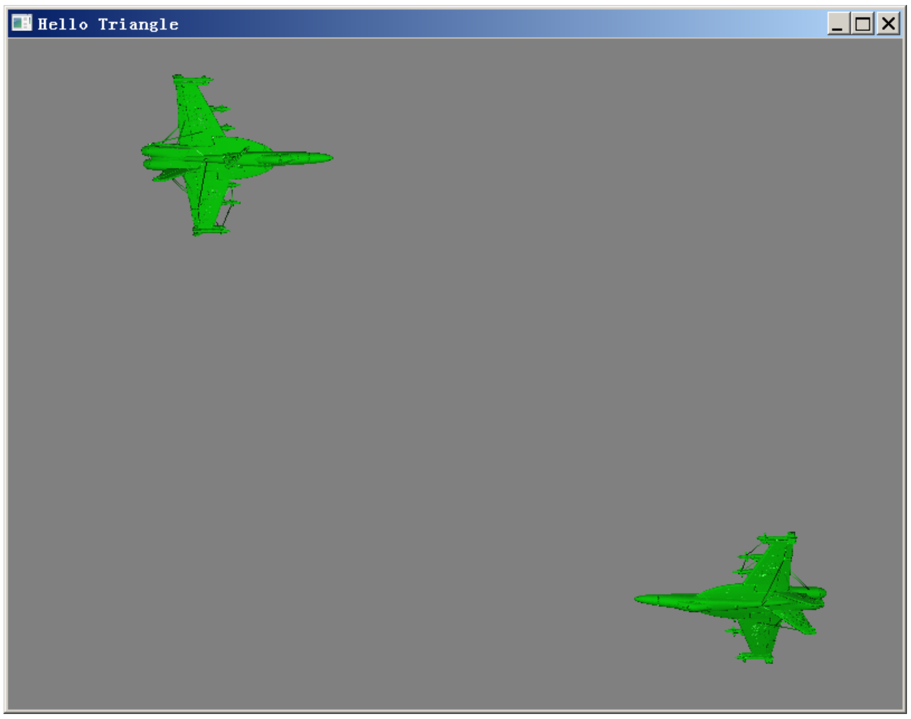


```
void keypress(unsigned char key, int x, int y) {
    if (key == 'x') {
        myView = !myView;
    }
    if (key == 'q') {
        rotate_x -= 1.0f;
    }
    if (key == 'e') {
        rotate_x += 1.0f;
    }
}
```

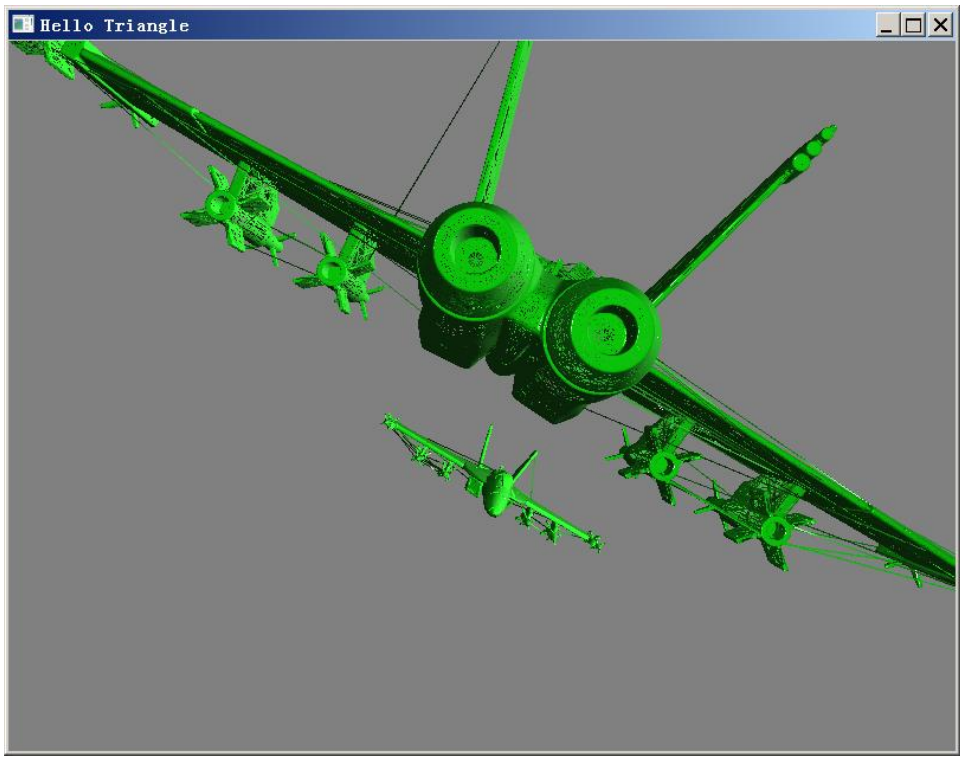


```
    if (myView) {
        view = look_at(vec3(main_x - 100, main_y, 0.0f), vec3(main_x, main_y, 0.0f), vec3(0, -1, 0));
    }
    else {
        view = translate(view, vec3(0.0, 0.0, -500.0f));
    }
```

This is top down view:



This is first person view:


Hello Triangle