Final report

Shijun Zhang 2021/1/3

Youtube link: https://youtu.be/ynh1QRVjltQ

Game design:

Sorry i change a little game design, before i want to do a serious game that can teach people how to make Chinese food, but later i thought it is very hard to make the kitchen scene,because i can't draw the model or draw texture. So i change my mind, i want to make a space style, combine the space physical knowledge and food knowledge so i design this game which main content is : Player need to control the space ship and find 4 material: Sauce, Pork, Carrot and broccoli. Then player control the ship return to earth and finish the assignment. If player doesn't get all of the 4 materials, he won't win and he need to continue collecting the material until he get 4 materials. And the game hints shows in the console.

Technical and result:

1. Display a 3D polygon mesh

Use Assimp package to read model .obj file

```
#include<vector>
#include"Mesh.h"
#include<assimp/Importer.hpp>
#include"stb_image.h"
#include<assimp/scene.h>
#include<assimp/postprocess.h>
#include<iostream>
#include"Shader.h"
class Model
{
public:
    std::vector<Texture> textures_loaded;
    Model(std::string path);
    std::vector<Mesh> meshes;
    std::string directory;
    void Draw(Shader* shader);
private:
    void loadModel(std::string path);
    void processNode(aiNode* node,const aiScene* scene);
    Mesh processMesh(aiMesh* mesh, const aiScene* scene);
    unsigned int TextureFromFile(const char *path, const std::string &directory);
    std::vector<Texture> loadMaterialTextures(aiMaterial *mat, aiTextureType type, std::string typeName);
};
```

Wrote a model class to read and analysis .obj file.

```
#include "Shader.h"
#include<glm/glm.hpp>
#include<string>
#include<vector>
#include <GL/glew.h>
struct Vertex {
    glm::vec3 Position;
    glm::vec3 Normal;
    glm::vec2 TexCoords;
};

struct Texture {
    unsigned int id;
    std::string type;
    std::string path;
};

class Mesh
{
public:
    Mesh(float vertices[]);
    Mesh(std::vector<Vertex> vertices,std::vector<unsigned int> indices,std::vector<Texture> textures);
    std::vector<Vertex> vertices;
    std::vector<unsigned int> indices;
    std::vector<Texture> textures;
    void Draw(Shader* shader);

private:
    unsigned int VAO, VBO, EBO;
    void SetupMesh();
};
```

Transform the vertex resource to VBO->VAO and get the indices and write in EBO.

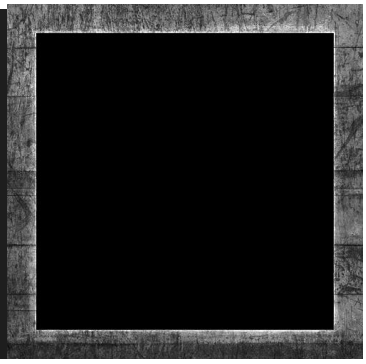
```
void Mesh::SetupMesh()
{
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);

    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER,VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex)*vertices.size(),&vertices[0] , GL_STATIC_DRAW);

    glGenBuffers(1, &EBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned int)*indices.size(), &indices[0], GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)(3*sizeof(GL_FLOAT)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)(6 * sizeof(GL_FLOAT)));

    glBindVertexArray(0);//解绑操作
}
```

Function to load image, i write the function in Main.cpp

```
//Load Image
unsigned int LoadImageToGPU(const char* filename,GLint internalFormate,GLenum format,int texSlot) {
    unsigned int TexBuffer;
    glGenTextures(1, &TexBuffer);//产生一个VAO并且把ID赋给上一行的VAO
    glActiveTexture(GL_TEXTURE0+texSlot);//开启textureBuffer中的几号位置
    glBindTexture(GL_TEXTURE_2D, TexBuffer);//把纹理buffer绑到操作位置上
    int width, height, nrChannel;
    stbi_set_flip_vertically_on_load(true);//翻转纹理贴图
    unsigned char *data = stbi_load(filename, &width, &height, &nrChannel, 0);
    if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, internalFormate, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        cout << "Load image failed." << endl;
    }
    stbi_image_free(data);
    return TexBuffer;
}
```

There are 9 objects in my game so i initialize 9 materials, include specular texture and diffuse texture.Diffuse textures are the Color map of the object itself, Specular texture is a Texture of metal frame.

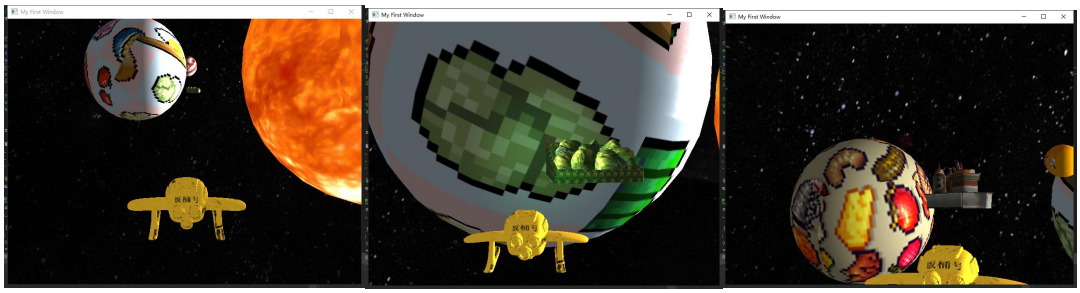
```
#pragma region Init Material
Material* MyMaterial = new Material(MyShader,
                                    LoadImageToGPU("ShipTex.jpg", GL_RGB, GL_RGB, 1),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess

Material* MyMaterial1 = new Material(MyShader1,
                                    LoadImageToGPU("food.png", GL_RGB, GL_RGB, 3),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess

Material* MyMaterial2 = new Material(MyShader2,
                                    LoadImageToGPU("earth.jpg", GL_RGB, GL_RGB, 6),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess
Material* MyMaterial3 = new Material(MyShader3,
                                    LoadImageToGPU("Sun.jpg", GL_RGB, GL_RGB, 7),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess

Material* MyMaterial4 = new Material(MyShader4,
                                    LoadImageToGPU("Moon.png", GL_RGBA, GL_RGBA, 9),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess

Material* MyMaterial5 = new Material(MyShader5,
                                    LoadImageToGPU("StarBackgound.jpg", GL_RGB, GL_RGB, 11),//diffuse, 材质漫反射贴图
                                    LoadImageToGPU("container2_specular.png", GL_RGBA, GL_RGBA, 2),//specular, 镜面反射贴图
                                    glm::vec3(1.0f, 1.0f, 1.0f),//ambient
                                    32.0f);//shininess
```



Specular texture

At last, i write a draw function to draw the meshes and the function can Automatic identification of diffuse texture and specular texture and transform them to the uniform variable in the shader.

```
void Mesh::Draw(Shader * shader)
{
    for (unsigned int i = 0; i < textures.size(); i++){
        if (textures[i].type == "texture_diffuse") {
            glActiveTexture(GL_TEXTURE0);
            glBindTexture(GL_TEXTURE_2D, textures[i].id);
            shader->SetUniform1i("material.diffuse", 0);
        }
        else if (textures[i].type == "texture_specluar") {
            glActiveTexture(GL_TEXTURE1);
            glBindTexture(GL_TEXTURE_2D, textures[i].id);
            shader->SetUniform1i("material.specular", 1);
        }
    }
    glBindVertexArray(VAO);
    //glDrawArrays(GL_TRIANGLES, 0, 36);
    glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
    glActiveTexture(GL_TEXTURE0);
}
```



Space ship Vegetable Sauce

Interactive manipulation of part of the 3D scene (i.e. transforms) using keyboard/mouse or some other device

In the first person mode, the mouse slides to control the angle of the camera.

```
void mouse_callback(GLFWwindow* window, double xPos, double yPos) {//Use mouse to change view angle
    if (firstMouse == true) {
        LastX = xPos;
        LastY = yPos;
        firstMouse = false;
    }
    float deltaX, deltaY;
    deltaX = xPos - LastX;
    deltaY = yPos - LastY;
    LastX = xPos;
    LastY = yPos;
    MyCamera.ProcessMouseMovement(-deltaX, -deltaY);
}

#pragma endregion
```

```
void Camera::ProcessMouseMovement(float deltaX, float deltaY) {
    Pitch += deltaY*0.001;
    Yaw += deltaX*0.001;
    UpdateCameraVectors();
}
```

```
void Camera::UpdateCameraVectors() {
    Forward.x = glm::cos(Pitch)*glm::sin(Yaw);
    Forward.y = glm::sin(Pitch);
    Forward.z = glm::cos(Pitch)*glm::cos(Yaw);
    Right = glm::normalize(glm::cross(Forward, WorldUp));
    Up = glm::normalize(glm::cross(Right, Forward));
}
```

The space ship will also adjust the rotation angle according to the angle of the camera.forward direction.

```
if (MyCamera.Forward.z <= 0) { modelMat = glm::rotate(modelMat, -1.0f*MyCamera.Forward.x, glm::vec3(0.0f, 1.0f, 0.0f)); }
else if (MyCamera.Forward.z > 0) { modelMat = glm::rotate(modelMat, 1.0f*MyCamera.Forward.x + glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f)); }
modelMat = glm::rotate(modelMat, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

The Keyboard input:

```
void processInput(GLFWwindow* window) {//Input control
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
        glfwSetWindowShouldClose(window, true);
    }
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        MyCamera.speedZ=1.5f;
        modelposition.z += MyCamera.speedZ;
    }
    else if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
        MyCamera.speedZ = -1.5f;
        modelposition.z += MyCamera.speedZ;
    }
    else if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
        MyCamera.speedX = -1.5f;
        modelposition.x += MyCamera.speedX;
    }
    else if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
        MyCamera.speedX = 1.5f;
        modelposition.x += MyCamera.speedX;
    }
    else if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS) {
        MyCamera.speedY = 1.5f;
        modelposition.y += MyCamera.speedY;
    }
    else if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS) {
        MyCamera.speedY = -1.5f;
        modelposition.y += MyCamera.speedY;
    }
    else
    {
        MyCamera.speedZ = 0;
        MyCamera.speedX = 0;
        MyCamera.speedY = 0;
    }
}
```

```
void processInput1(GLFWwindow* window) {//Input control
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
        glfwSetWindowShouldClose(window, true);
    }
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        modelposition.z += -0.1f;
    }
    else if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
        modelposition.z += 0.1f;
    }
    else if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
        modelposition.x += -0.1f;
    }
    else if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
        modelposition.x += 0.1f;
    }
    else if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS) {
        modelposition.y += 0.1f;
    }
    else if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS) {
        modelposition.y += -0.1f;
    }
    else
    {
        MyCamera.speedZ = 0;
        MyCamera.speedX = 0;
        MyCamera.speedY = 0;
    }
}
```

W A S D SPACE Z Control the camera to move up and down, and then the motion matrix of the spacecraft is consistent with the camera, So as to achieve the effect of the spaceship following the camera.

In the Top down view,The keyboard directly controls the motion matrix of the spacecraft model to achieve the effect that the spacecraft does not move and the camera does not move.

```
Camera::Camera(glm::vec3 position, float pitch, float yaw, glm::vec3 worldup) {
    Position = position;
    WorldUp = worldup;
    Pitch = pitch;
    Yaw = yaw;
    Forward.x = glm::cos(Pitch)*glm::sin(yaw);
    Forward.y = glm::sin(Pitch);
    Forward.z = glm::cos(Pitch)*glm::cos(Yaw);
    Right = glm::normalize(glm::cross(Forward, WorldUp));
    Up = glm::normalize(glm::cross(Right,Forward));
}

glm::mat4 Camera::GetViewMatrix(glm::vec3 Position) {
    return glm::lookAt(Position, Position + Forward, WorldUp);
}

void Camera::UpdateCameraVectors() {
    Forward.x = glm::cos(Pitch)*glm::sin(Yaw);
    Forward.y = glm::sin(Pitch);
    Forward.z = glm::cos(Pitch)*glm::cos(Yaw);
    Right = glm::normalize(glm::cross(Forward, WorldUp));
    Up = glm::normalize(glm::cross(Right, Forward));
}

void Camera::ProcessMouseMovement(float deltaX, float deltaY) {
    Pitch += deltaY*0.001;
    Yaw += deltaX*0.001;
    UpdateCameraVectors();
}

void Camera::UpdateCameraPosition() {
    Position += Forward * speedZ + Right * speedX + Up * speedY;

}
```

This is the Camera class and the functions.

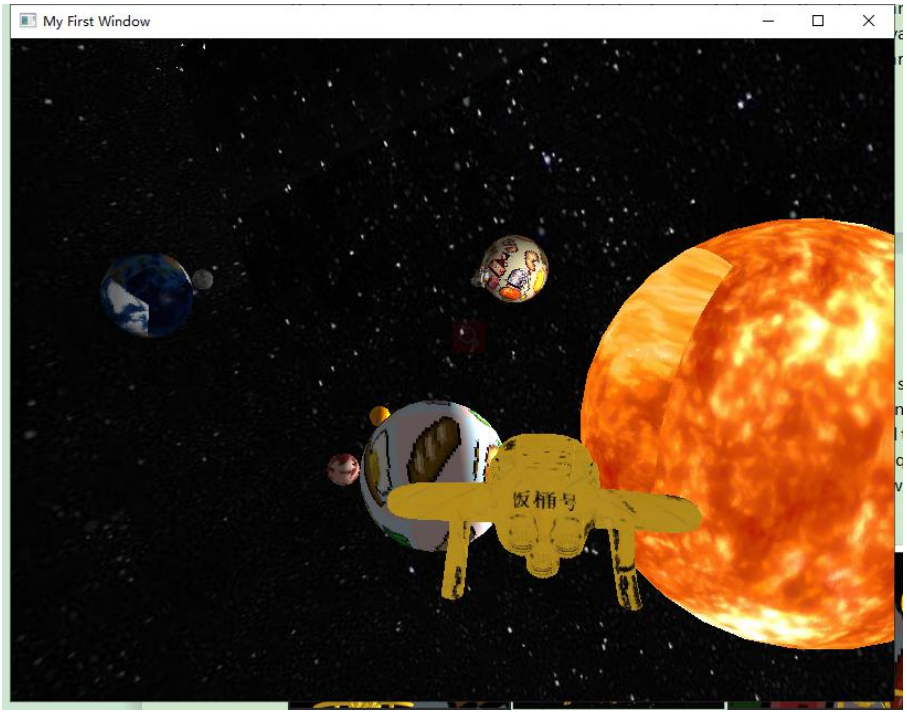
hierarchical structure undergoing transformations;

The hierarchical structure is the planets' structure, include planet's rotation and revolution, the game include some Physical formulas to realize the star motivation. The broccoli ,pork and carrot moving around the white star and the white star goes around the sun.

The Sauce moving around the brown star, and the brown moving around the sun.

The moon moving around the earth and the earth also moving around the sun.

All the objects in the game have rotation with different angle velocity, and all the object’s speed of revolution is different. It's astrophysical, objects with small quality has large linear speed. So this game can also teach player some physical knowledge.



Revolution is realized by transform matrix, all the transform matrix is calculated based on their center object. Rotation is realized by rotate matrix.

```
    //Sun
gion Shader3 set and use, Draw model3
  MyShader3->use();
  modelMat3 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -0.0f, -50.0f));
  modelMat3 = glm::rotate(modelMat3, 0.07f*(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
  modelMat3 = glm::scale(modelMat3, glm::vec3(1200.0f, 1200.0f, 1200.0f));
```

```
  //White star
egion Shader1 set and use, Draw model1
  MyShader1->use();
  modelMat1 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -10.0f, 0.0f));
  modelMat1 = glm::translate(modelMat1, glm::vec3((500 * sin(-0.05f*(float)glfwGetTime())), 0.0f, -50 + 500 * cos(-0.05f*(float)glfwGetTime())));
  modelMat1 = glm::rotate(modelMat1, -0.6f*(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
  modelMat1 = glm::scale(modelMat1, glm::vec3(400.0f, 400.0f, 400.0f));
```

```
  //earth
egion Shader2 set and use, Draw model2
  MyShader2->use();
  modelMat2 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -10.0f, 0.0f));
  modelMat2 = glm::translate(modelMat2, glm::vec3((1200 * sin(0.1f*(float)glfwGetTime())), 0.0f, -50 + 1200 * cos(0.1f*(float)glfwGetTime())));
  modelMat2 = glm::scale(modelMat2, glm::vec3(360.0f, 360.0f, 360.0f));
  modelMat2 = glm::rotate(modelMat2, 0.6f*(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
```

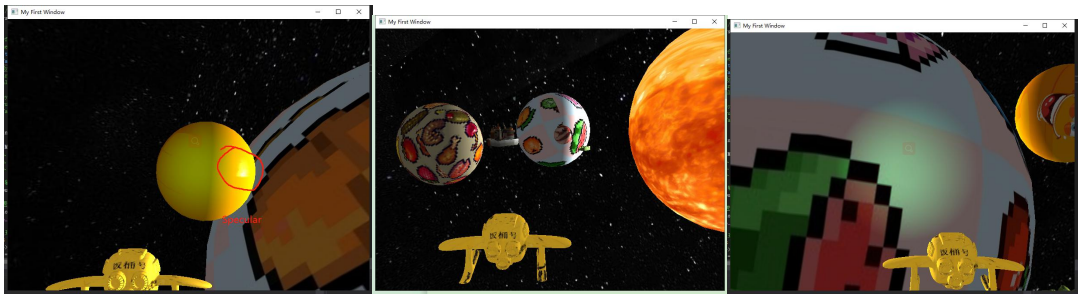
White star and earth move around the sun

```
//Moon
gion Shader4 set and use, Draw model4
  MyShader4->use();
  modelMat4 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -10.0f, 0.0f));
  modelMat4 = glm::translate(modelMat4, glm::vec3((120 * sin(1.5f*(float)glfwGetTime())) + (1200 * sin(0.1f*(float)glfwGetTime())),
                          0.0f, -50 + 120 * cos(1.5f*(float)glfwGetTime()) + 1200 * cos(0.1f*(float)glfwGetTime())));

  modelMat4 = glm::scale(modelMat4, glm::vec3(12.0f, 12.0f, 12.0f));
  modelMat4 = glm::rotate(modelMat4, -0.8f*(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
```

Moon moves around the earth.

Use glfwGetTime() to get system's time and then the matrix value can change with the time goes.

Lit and shaded; including diffuse and specular objects



The Sun is a point light, it is the main light source in the game.

```
//point lighr
LightPoint MyLight0(glm::vec3(0.0f, -10.0f, -50.0f), //position
                            glm::vec3(glm::radians(45.0f), glm::radians(45.0f), 0),//angle
                            glm::vec3(1000.0f, 1000.0f,1000.0f));//color
```

```
#include "LightPoint.h"

LightPoint::LightPoint(glm::vec3 _position, glm::vec3 _angles, glm::vec3 _color)
{
    position = _position;
    angles = _angles;
    color = _color;
    constant = 1.0f;
    linear =0.005f;
    quadratic = 0.002f;
}
```

In shader

```
//点光源
vec3 CalcLightPoint(LightPoint light,vec3 uNormal,vec3 dirToCamera){
    //attenuation
    float dist=length(light.Pos-FragPos);
    float attenuation=1.0f/(light.constant+light.linear*dist+light.quadratic*(dist*dist));//点光源衰减 attenuation
    //diffuse
    float diffIntensity=max(dot(normalize(light.Pos-FragPos),uNormal),0)*attenuation;
    vec3 diffColor=diffIntensity*light.Color*texture(material.diffuse,TexCoord).rgb;
    //specular
    vec3 R=normalize(reflect(-normalize(light.Pos-FragPos),uNormal));
    float specIntensity=pow(max(dot(R,dirToCamera),0),material.shininess)*attenuation;
    vec3 specColor=specIntensity*light.Color*texture(material.specular,TexCoord).rgb;

    vec3 result=diffColor+specColor;
    return result;
}
```

To make the game more interesting , i also added a spot light in front the space ship and it will always in front of the space ship. Now the light show is very beautiful.

```
//spot light
LightSpot MyLightSpot(glm::vec3(0.0f, 5.0f, 0.0f), //position
                            glm::vec3(glm::radians(90.0f), 0, 0),
                            100.0f*glm::normalize(glm::vec3(55.0, 192.0f, 103.0f)));//color
```

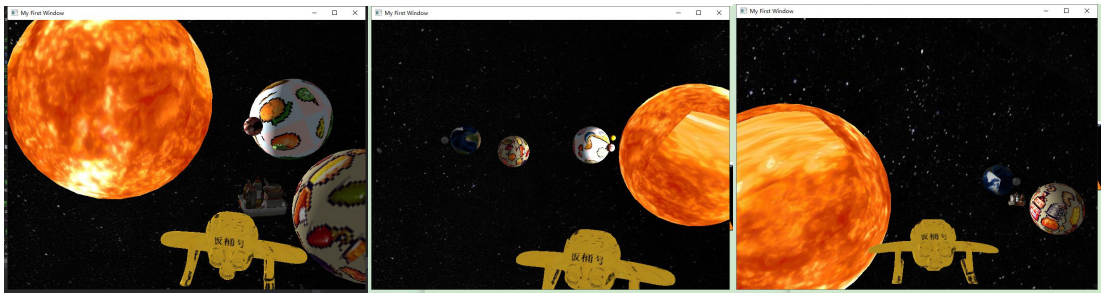
```cpp
#include "LightSpot.h"

LightSpot::LightSpot(glm::vec3 _position, glm::vec3 _angles, glm::vec3 _color)
{
    position = _position;
    angles = _angles;
    color = _color;
    constant = 1.0f;
    linear = 0.09f;
    quadratic = 0.032f;
    UpdateDirection();
}

void LightSpot::UpdateDirection() {
    direction = glm::vec3(0, 0, 1.0f);//pointng to Z(Forward)
    direction = glm::rotateZ(direction, angles.z);
    direction = glm::rotateX(direction, angles.x);
    direction = glm::rotateY(direction, angles.y);
    direction = -1.0f*direction;
}
```

```cpp
//聚光灯spot light
vec3 CalcLightSpot(LightSpot light,vec3 uNormal,vec3 dirToCamera){
    //attenuation
    float dist=length(light.Pos-FragPos);
    float attenuation=1.0f/(light.constant+light.linear*dist+light.quadratic*(dist*dist));//点光源衰减attenuation
    float spotRatio;
    float CosTheta=dot(normalize(FragPos-light.Pos),-light.DirToLight);
    if(CosTheta>light.cosInnerPhy){
        spotRatio=1.0f;
    }
    else if(CosTheta>light.cosOutterPhy){
        spotRatio=1.0f-(CosTheta-light.cosInnerPhy)/(light.cosOutterPhy-light.cosInnerPhy);
    }
    else{
        spotRatio=0;
    }
    //diffuse
    float diffIntensity=max(dot(normalize(light.Pos-FragPos),uNormal),0)*attenuation*spotRatio;
    vec3 diffColor=diffIntensity*light.Color*texture(material.diffuse,TexCoord).rgb;
    //specular
    vec3 R=normalize(reflect(-normalize(light.Pos-FragPos),uNormal));
    float specIntensity=pow(max(dot(R,dirToCamera),0),material.shininess)*attenuation*spotRatio;
    vec3 specColor=specIntensity*light.Color*texture(material.specular,TexCoord).rgb;

    vec3 result=diffColor+specColor;
    return result;
}
```

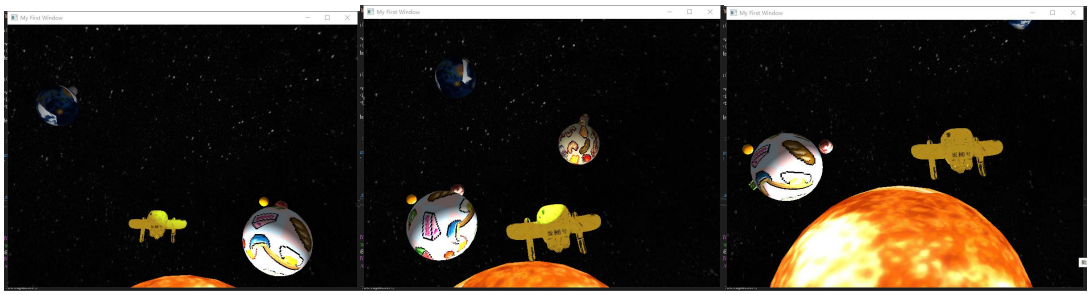
```cpp
MyLightSpot.position = glm::vec3(MyCamera.Position.x + (5 * MyCamera.Forward.x), MyCamera.Position.y - 2 + (5 * MyCamera.Forward.y),
                                 MyCamera.Position.z + (7 * MyCamera.Forward.z));
MyLightSpot.direction = -MyCamera.Forward;
```

Spot light follow the camera.

First person view: The camera is follow the space ship and the ship will rotate withe the mouse move, to ensure the ship is back-toward the camera.



Top down view: The camera is fixed and the ship won't rotate with the mouse move, WASD SPACE Z still can control the space ship move.



About the shader

Vertex shader:

```
#version 330 core

layout(location = 0) in vec3 aPos;
layout(location=1)in vec3 aNormal;
layout(location = 2) in vec2 aTexCoord;//texture uv

//uniform mat4 mytransform;
uniform mat4 modelMat;
uniform mat4 viewMat;
uniform mat4 projMat;

out vec4 vertexColor;
out vec2 TexCoord;
out vec3 FragPos;
out vec3 Normal;

void main() {
    //gl_Position = mytransform*vec4(aPos.x, aPos.y, aPos.z, 1.0);
    gl_Position = projMat*viewMat*modelMat*vec4(aPos.xyz, 1.0);

    FragPos=(modelMat*vec4(aPos,1.0f)).xyz;

    Normal =mat3(transpose(inverse(modelMat)))*aNormal;

    TexCoord=aTexCoord;
}
```

Fragment shader:

```
void main() {

    vec3 finalResult=vec3(0,0,0);
    vec3 uNormal=normalize(Normal);
    vec3 dirToCamera=normalize(CameraPos-FragPos);
    vec3 Ambient=texture(material.diffuse,TexCoord).rgb*material.ambient*0.4f*AmbientColor;

    finalResult+=CalcLightDirectional(lightdirectional,uNormal,dirToCamera);
    finalResult+=CalcLightPoint(lightpoint0,uNormal,dirToCamera);
    finalResult+=CalcLightSpot(lightspot,uNormal,dirToCamera);
    finalResult+=Ambient;
    FragColor=vec4(finalResult,1.0f);
    //FragColor=vec4(1.0f,1.0f,1.0f,1.0f);
}
```

Calculate all the light using normal direction, light direction, and view direction and the diffuse texture, specular texture

```
//点光源 pointlight
vec3 CalcLightPoint(LightPoint light,vec3 uNormal,vec3 dirToCamera){
    //attenuation
    float dist=length(light.Pos-FragPos);
    float attenuation=1.0f/(light.constant+light.linear*dist+light.quadratic*(dist*dist));//点光源衰减 attenuation
    //diffuse
    float diffIntensity=max(dot(normalize(light.Pos-FragPos),uNormal),0)*attenuation;
    vec3 diffColor=diffIntensity*light.Color*texture(material.diffuse,TexCoord).rgb;
    //specular
    vec3 R=normalize(reflect(-normalize(light.Pos-FragPos),uNormal));
    float specIntensity=pow(max(dot(R,dirToCamera),0),material.shininess)*attenuation;
    vec3 specColor=specIntensity*light.Color*texture(material.specular,TexCoord).rgb;

    vec3 result=diffColor+specColor;
    return result;
}
```

By the way the Reflection model i use is phong model, calculate the reflection light and dot with the view direction.

```
#pragma region Init Shader Program
Shader* MyShader = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader1 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader2 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader3 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader4 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader5 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader6 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader7 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader8 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader9 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
Shader* MyShader10 = new Shader("vertexSource.vert", "FragmentSource.frag");//new一个shader对象
#pragma endregion
```

I new 9 shader objects and different object use different shader.

```
#pragma region Init Shader Program
Shader* MyShader = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader1 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader2 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader3 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader4 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader5 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader6 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader7 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader8 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader9 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
Shader* MyShader10 = new Shader("vertexSource.vert", "FragmentSource.frag");//new a shader object
```

Give the uniform variable data and use shader to draw.

```
        //White star
#pragma region Shader1 set and use, Draw model1
        MyShader1->use();
        modelMat1 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -10.0f, 0.0f));
        modelMat1 = glm::translate(modelMat1, glm::vec3((500 * sin(-0.05f*(float)glfwGetTime())), 0.0f, -50 + 500 * cos(-0.05f*(float)glfwGetTime())));
        modelMat1 = glm::rotate(modelMat1, -0.6f*(float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
        modelMat1 = glm::scale(modelMat1, glm::vec3(400.0f, 400.0f, 400.0f));
        glUniformMatrix4fv(glGetUniformLocation(MyShader1->ID, "modelMat"), 1, GL_FALSE, glm::value_ptr(modelMat1));//给uniform变量赋予位移矩阵
        glUniformMatrix4fv(glGetUniformLocation(MyShader1->ID, "viewMat"), 1, GL_FALSE, glm::value_ptr(viewMat));//给uniform变量赋予视角矩阵
        glUniformMatrix4fv(glGetUniformLocation(MyShader1->ID, "projMat"), 1, GL_FALSE, glm::value_ptr(projMat));//给uniform变量赋予投影矩阵
        glUniform3f(glGetUniformLocation(MyShader1->ID, "ObjColor"), 1.0f, 1.0f, 1.0f);//材料的颜色
        glUniform3f(glGetUniformLocation(MyShader1->ID, "AmbientColor"), 1.0f, 1.0f, 1.0f);//环境光
        glUniform3f(glGetUniformLocation(MyShader1->ID, "LightPos"), MyLight.position.x, MyLight.position.y, MyLight.position.z);//光源位置
        glUniform3f(glGetUniformLocation(MyShader1->ID, "LightColor"), MyLight.color.x, MyLight.color.y, MyLight.color.z);//光源颜色
        glUniform3f(glGetUniformLocation(MyShader1->ID, "LightDirUniform"), MyLight.direction.x, MyLight.direction.y, MyLight.direction.z);//光源方向

        //point light
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightpoint0.Pos"), MyLight0.position.x, MyLight0.position.y, MyLight0.position.z);
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightpoint0.DirToLight"), MyLight0.direction.x, MyLight0.direction.y, MyLight0.direction.z);
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightpoint0.Color"), MyLight0.color.x, MyLight0.color.y, MyLight0.color.z);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightpoint0.constant"), MyLight0.constant);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightpoint0.linear"), MyLight0.linear);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightpoint0.quadratic"), MyLight0.quadratic);

        //spot light
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightspot.Pos"), MyLightSpot.position.x, MyLightSpot.position.y, MyLightSpot.position.z);
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightspot.DirToLight"), MyLightSpot.direction.x, MyLightSpot.direction.y, MyLightSpot.direction.z);
        glUniform3f(glGetUniformLocation(MyShader1->ID, "lightspot.Color"), MyLightSpot.color.x, MyLightSpot.color.y, MyLightSpot.color.z);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightspot.constant"), MyLightSpot.constant);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightspot.linear"), MyLightSpot.linear);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightspot.quadratic"), MyLightSpot.quadratic);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightspot.cosInnerPhy"), MyLightSpot.cosInnerPhy);
        glUniform1f(glGetUniformLocation(MyShader1->ID, "lightspot.cosOutterPhy"), MyLightSpot.cosOutterPhy);

        glUniform3f(glGetUniformLocation(MyShader1->ID, "CameraPos"), MyCamera.Position.x, MyCamera.Position.y, MyCamera.Position.z);//视角位置（影响镜面反射）
        MyMaterial1->shader->SetUniform3f("material.ambient", MyMaterial1->ambient);
        MyMaterial1->shader->SetUniform1i("material.diffuse", 3);
        MyMaterial1->shader->SetUniform1i("material.specular", 2);
        MyMaterial1->shader->SetUniform1f("material.shininess", MyMaterial1->shininess);
        model1.Draw(MyMaterial1->shader);
#pragma endregion
```

```
void Mesh::Draw(Shader * shader)
{
    for (unsigned int i = 0; i < textures.size(); i++){
        if (textures[i].type == "texture_diffuse") {
            glActiveTexture(GL_TEXTURE0);
            glBindTexture(GL_TEXTURE_2D, textures[i].id);
            shader->SetUniform1i("material.diffuse", 0);
        }
        else if (textures[i].type == "texture_specluar") {
            glActiveTexture(GL_TEXTURE1);
            glBindTexture(GL_TEXTURE_2D, textures[i].id);
            shader->SetUniform1i("material.specular", 1);
        }
    }
    glBindVertexArray(VAO);
    //glDrawArrays(GL_TRIANGLES, 0, 36);
    glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
    glActiveTexture(GL_TEXTURE0);
}
```

Game logic judgement: I use position judgement when the space ship's position is in the range of the pork, sauce, carrot and the vegetable, means player get the food , and if he return to earth now, he will win. How ever if he didn't get all the food and he return to earth, he won't win.

```
//游戏判定 game judgement
float distance = 20;

//捕获猪肉 get pork
if (glm::abs(MyCamera.Position.x - ((120 * sin(0.5f*(float)glfwGetTime())) + (500 * sin(-0.05f*(float)glfwGetTi
    glUniform3f(glGetUniformLocation(MyShader->ID, "AmbientColor"), 2.0f, 0.0f, 0.0f);//变红
    if (pork&&carrot&&broccoli&&sauce) { std::cout << "YOU CAN MAKE THE BBQ PORK NOW!!! RETURN TO THE EARTH!!!"
    else { std::cout << "GET PORK!!!" << endl; }
    pork=true;
}
//捕获胡萝卜 get carrot
if (glm::abs(MyCamera.Position.x - ((110 * sin(-0.3f*(float)glfwGetTime())) + (500 * sin(-0.05f*(float)glfwGetT
    glUniform3f(glGetUniformLocation(MyShader->ID, "AmbientColor"), 0.0f, 0.0f, 2.0f);//变蓝
    if (pork&&carrot&&broccoli&&sauce) { std::cout << "YOU CAN MAKE THE BBQ PORK NOW!!! RETURN TO THE EARTH!!!"
    else { std::cout << "GET CORROT!!!" << endl; }
    carrot = true;
}
//捕获西兰花 get vegetable
if (glm::abs(MyCamera.Position.x - ((110 * sin(-0.8f*(float)glfwGetTime())) + (500 * sin(-0.05f*(float)glfwGetT
    glUniform3f(glGetUniformLocation(MyShader->ID, "AmbientColor"), 0.0f, 2.0f, 0.0f);//变绿
    if (pork&&carrot&&broccoli&&sauce) { std::cout << "YOU CAN MAKE THE BBQ PORK NOW!!! RETURN TO THE EARTH!!!"
    else { std::cout << "GET BROCCOLI!!!" << endl; }
    broccoli = true;
}
//捕获酱汁 get sauce
if (glm::abs(MyCamera.Position.x - ((110 * sin(-0.2f*(float)glfwGetTime())) + (760 * sin(0.25f*(float)glfwGetTi
    glUniform3f(glGetUniformLocation(MyShader->ID, "AmbientColor"), 0.0f, 0.0f, 0.0f);//变棕色
    if (pork&&carrot&&broccoli&&sauce) { std::cout << "YOU CAN MAKE THE BBQ PORK NOW!!! RETURN TO THE EARTH!!!"
    else { std::cout << "GET SAUCE!!!" << endl; }
    sauce = true;
}
//回到地球 return to the earth
if (glm::abs(MyCamera.Position.x - (1200 * sin(0.1f*(float)glfwGetTime()))) <= 100 && glm::abs(MyCamera.Positio
    if (pork&&carrot&&broccoli&&sauce) {
        std::cout << "THANK YOU FOR YOUR BBQ PORK!!!EARTHMEN APPRECIATE YOU!!!" << endl;
    }
    else {
        std::cout << "YOU NEED CONTINUE FINDING THE BBQ PORK MATERIALS!!!" << endl;
    }
}
```