Macau university of science and technology

Title：Animation of the Solar System Report

Team member ：

王虹博 , Wang Hong Bo ,18098532-I011-0018

刘俊锋 , Liu Jun Feng ,18098537-I011-0067

Course name：Fundamentals of Computer Graphics

Course number：CS104

Date ：2020/12/11

Contents

(1) Project description

1.1 Problem Description

Construct a virtual solar system from the sun, eight. Planets (Venus, Earth, Mars, Jupiter, Saturn, Uranus, And Neptune), the moon and a star-sphere

1.2 Basic functions

a. To achieve the construction of the solar system, every star can revolve around the sun in the correct position and orbit

b. Each planet can be rotated and configured in the correct size ratio

c. Created a starry sky background to make the solar system look more beautiful

d. The viewing angle can be moved by controlling the WASD key, and the position from the solar system can be changed by pressing the left and right buttons of the mouse

e. You can change the position of the mouse by dragging the mouse to change the perspective

f. Illumination is realized, and the side of the planet facing the sun is brighter

g. Use Torus to create a ring of Saturn

(2) Data description

2.1 Design outline

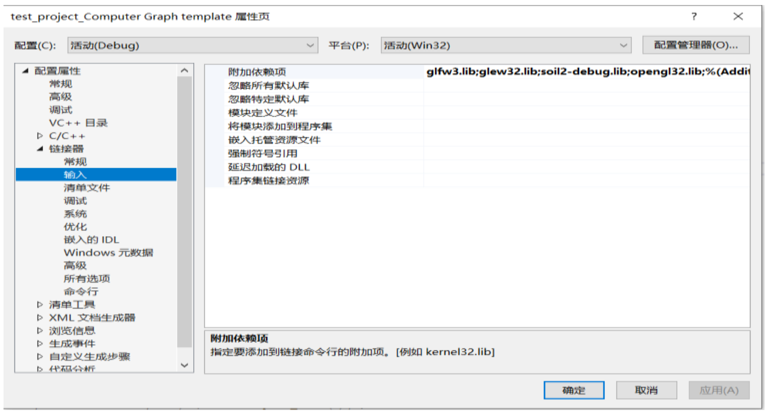
2.1.1

Step 1: Prepare theoretical knowledge, build excel, arrange the planets from the sun from far to nearer, draw table data, including planet size, planet rotation speed, planet revolution radius, planet revolution speed, distance from the sun, and pass Mathematical modeling methods have greatly narrowed the gap in proportions （$f(x) = \sqrt[4]{X}$）

	Convergent planet diameter	Convergent rotation speed	Convergent revolution speed	Convergent revolution radius
sun	34.3			
mercury	8.4	2.76	3.05	8.7
venus	10.5	3.95	3.95	10.19
earth	10.6	1	4.36	11
moon	7.68	2.28	2.28	EOF
mars	9.1	1	4.7	12.3
jupiter	19.5	0.79	8.11	16.7
saturn	18.6	0.81	10.2	19.4
uranus	15	0.92	13.24	23.2
neptune	14.9	0.9	15.66	25.9

2.1.2

Step 2: Build the project file and import the library. Configure OpenGL, GLEW, GLFW, GLM, and SOIL2, and build main.cpp to import the library file, and move the texture pictures needed by the solar system to the specified folder


test_project_Computer Graph template 属性页

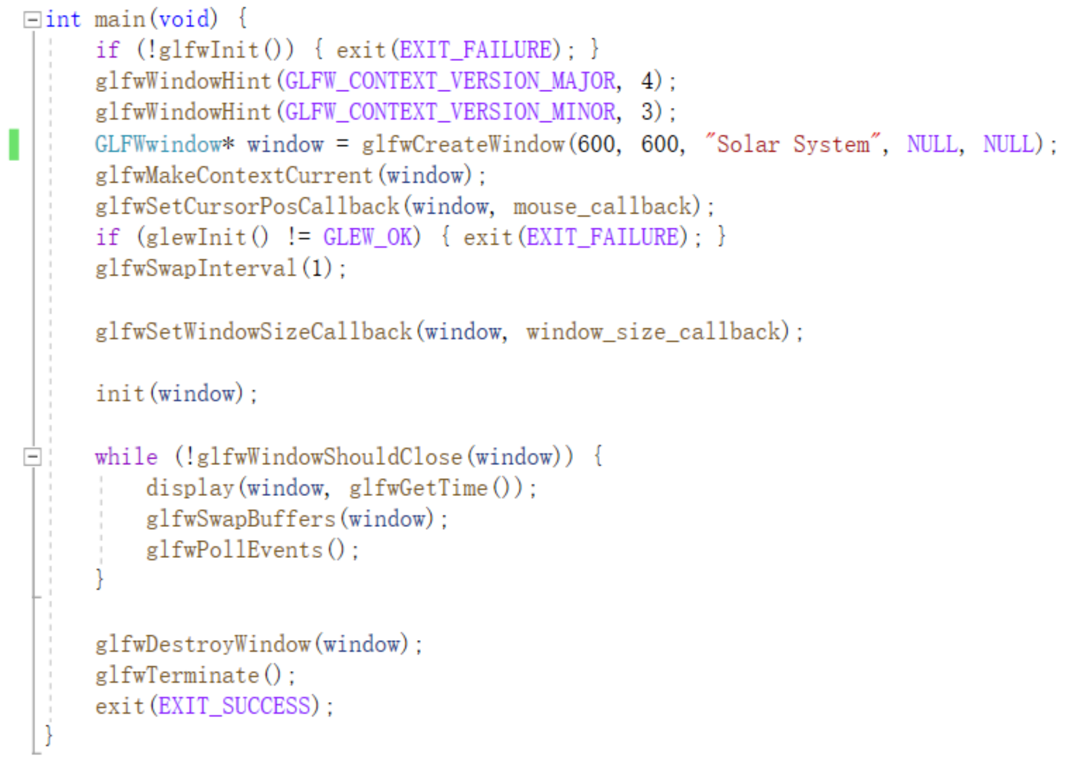
配置(C): 活动(Debug)　平台(P): 活动(Win32)　配置管理器(O)...

附加依赖项	glfw3.lib;glew32.lib;soil2-debug.lib;opengl32.lib;%(Addi
忽略所有默认库	
忽略特定默认库	
模块定义文件	
将模块添加到程序集	
嵌入托管资源文件	
强制符号引用	
延迟加载的 DLL	
程序集链接资源	

附加依赖项
指定要添加到链接命令行的附加项。[例如 kernel32.lib]

2.1.3

Step 3: Build the window and configure the window size. Add the basic shader file, introduce Utils.h and Utils.cpp, construct the basic main function body, configure the window size and window exit function

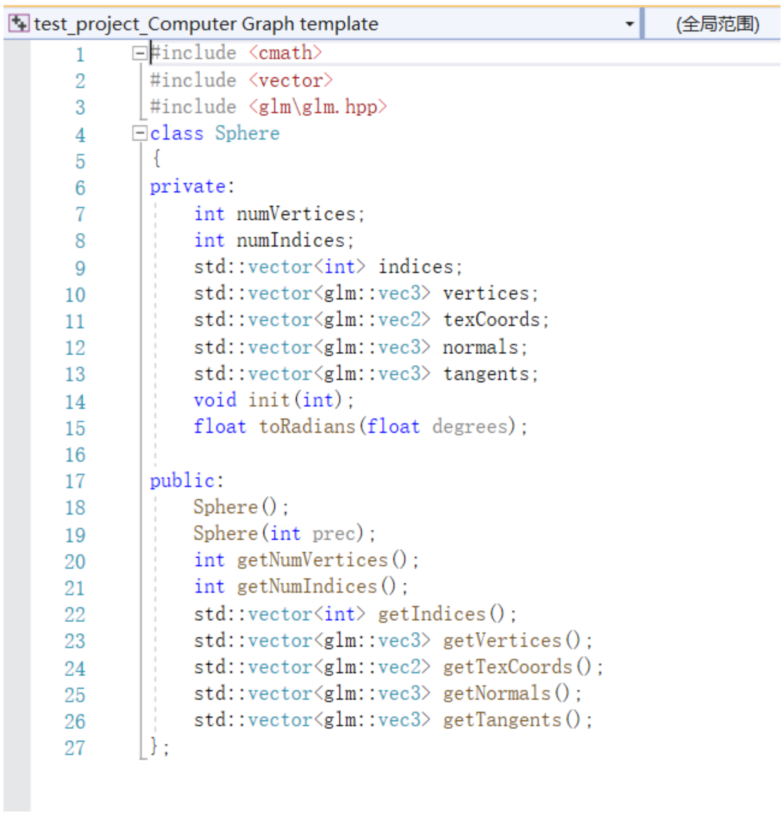


```
int main(void) {
    if (!glfwInit()) { exit(EXIT_FAILURE); }
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    GLFWwindow* window = glfwCreateWindow(600, 600, "Solar System", NULL, NULL);
    glfwMakeContextCurrent(window);
    glfwSetCursorPosCallback(window, mouse_callback);
    if (glewInit() != GLEW_OK) { exit(EXIT_FAILURE); }
    glfwSwapInterval(1);

    glfwSetWindowSizeCallback(window, window_size_callback);

    init(window);

    while (!glfwWindowShouldClose(window)) {
        display(window, glfwGetTime());
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glfwDestroyWindow(window);
    glfwTerminate();
    exit(EXIT_SUCCESS);
}
```

2.1.4

Step 4: Build and render the model of the ball. Use Sphere.h and Shere.cpp to construct the basic sphere "vertShader.glsl", and "fragShader.glsl" to render the sphere and generate the model.

```
glUseProgram(renderingProgram);

mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
nLoc = glGetUniformLocation(renderingProgram, "norm_matrix");
```



```
#include <cmath>
#include <vector>
#include <glm\glm.hpp>
class Sphere
{
private:
    int numVertices;
    int numIndices;
    std::vector<int> indices;
    std::vector<glm::vec3> vertices;
    std::vector<glm::vec2> texCoords;
    std::vector<glm::vec3> normals;
    std::vector<glm::vec3> tangents;
    void init(int);
    float toRadians(float degrees);

public:
    Sphere();
    Sphere(int prec);
    int getNumVertices();
    int getNumIndices();
    std::vector<int> getIndices();
    std::vector<glm::vec3> getVertices();
    std::vector<glm::vec2> getTexCoords();
    std::vector<glm::vec3> getNormals();
    std::vector<glm::vec3> getTangents();
};
```

2.1.5

Step 5: Use the stack to build the solar system model. Introduce the stack header file, use the push() function to push vMat onto the stack, use the top() function to push the top data onto the stack again, and use the glm::translate() function to set the sun to 0.0f, 0.0f , 0.0f (center of rotation), use glm::rotate function to change the rotation speed of the sun, use glm::scale to change the size of the sun, use glBindTexture() to introduce the texture of the sun, use pop() to throw the data of the sun itself Out of the stack, it can correctly construct the position of other planets around the sun.

```
mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f));
    mvStack.push(mvStack.top());
    mvStack.top() *= rotate(glm::mat4(1.0f), (float)currentTime, glm::vec3(0.0, 1.0, 0.0));
    mvStack.top() *= scale(glm::mat4(1.0f), glm::vec3(2.0, 2.0, 2.0));

    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()));

    mvStack.top() = glm::transpose(glm::inverse(mvStack.top()));
    glUniformMatrix4fv(nLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(2);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, sunTexture);

    //glActiveTexture(GL_TEXTURE0);

    glEnable(GL_DEPTH_TEST);
    glFrontFace(GL_CCW);
    glDepthFunc(GL_LEQUAL);

    glDrawArrays(GL_TRIANGLES, 0, mySphere.getNumIndices());
    mvStack.pop();
```

2.1.6

Step 6: The construction of planets. Use the top() function to locate the variable at the top of the stack, use the push() function to push the function subject at the top of the stack onto the stack again, use the glm::translate() function to set the planet’s orbital radius, and use glm::rotate The function changes the rotation speed of the planet, uses glm::scale to change the size of the planet, uses glBindTexture() to introduce the texture of the respective planet, and uses the pop() function to throw the data of the planet itself out of the stack to make it the correct construction position. (Venus as an example)

```
mvStack.push(mvStack.top());
mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float)(
venusRevolutionAroundSpeed * currentTime)) * venusRevolutionAroundRadiu
s, 0.0f, cos((float)(venusRevolutionAroundSpeed * currentTime)) * venus
RevolutionAroundRadius));
mvStack.push(mvStack.top());
mvStack.top() *= rotate(glm::mat4(1.0f), (float)(venusSelfAroundSpeed *
 currentTime), glm::vec3(0.0, 1.0, 0.0));
mvStack.top() *= scale(glm::mat4(1.0f), glm::vec3(venusSize, venusSize,
 venusSize));
```

```
glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
mvStack.top() = glm::transpose(glm::inverse(mvStack.top()));
glUniformMatrix4fv(nLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()));
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
glEnableVertexAttribArray(0);

glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(2);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, venusTexture);

glActiveTexture(GL_TEXTURE0);

glEnable(GL_DEPTH_TEST);
glFrontFace(GL_CCW);
glDepthFunc(GL_LEQUAL);

glDrawArrays(GL_TRIANGLES, 0, mySphere.getNumIndices());
mvStack.pop();
mvStack.pop();
```

2.1.7

Step 7: Construction of the satellite. When constructing a satellite (moon), the number of planets orbiting should be reduced by one pop(). The purpose is to find the position of the satellite and achieve the purpose of correct orbiting. Use glm::translate() to set the satellite's orbital radius , Use glm::rotate function to change the rotation speed of the satellite, use glm::scale to change the size of the satellite, use glBindTexture() to introduce the texture of the satellite, and use pop() to throw the data of the satellite itself out of the stack to make it correct Build location.

```
mvStack.push(mvStack.top());
    mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((flo
at)(moonRevolutionAroundSpeed * currentTime)) * moonRevolutionAroundRad
ius, 0.0f, cos((float)(moonRevolutionAroundSpeed * currentTime)) * moon
RevolutionAroundRadius));
    mvStack.top() *= rotate(glm::mat4(1.0f), (float)((moonSelfAroundSpe
ed * currentTime)), glm::vec3(0.0, 1.0, 0.0));
    mvStack.top() *= scale(glm::mat4(1.0f), glm::vec3(moonSize, moonSiz
e, moonSize));
```

```
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()
));
    mvStack.top() = glm::transpose(glm::inverse(mvStack.top()));
    glUniformMatrix4fv(nLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top())
);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
    glEnableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(2);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, moonTexture);

    glActiveTexture(GL_TEXTURE0);

    glEnable(GL_DEPTH_TEST);
    glFrontFace(GL_CCW);
    glDepthFunc(GL_LEQUAL);

    //glEnd();

    glDrawArrays(GL_TRIANGLES, 0, mySphere.getNumIndices());
    mvStack.pop();
    mvStack.pop();
    mvStack.pop();
```

2.1.8

Step 8: Lighting. there are some vec3 variables :currentLight , lightPosV and initialLight to set the position of light source. And then we use globalAmbient, lightAmbient, lightDiffuse and lightSpecular to set the white light. We set the material of planets to silver, it is convenience to observe.

There is a function called installLights(), it is used to store light locations and material properties. In display(), we calculate each planet’s inverse matrix of mvMat ,it is also a normal vector.

```
mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
nLoc = glGetUniformLocation(renderingProgram, "norm_matrix");

vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -
cameraZ));
mvStack.push(vMat);

currentLightPos = glm::vec3(initialLightLoc.x, initialLightLoc.y, initi
alLightLoc.z);

glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
installLights(vMat);
```

fragmentShader :

```
    vec3 ambient = ((globalAmbient * material.ambient) + (light.ambient
*material.ambient)).xyz;
    vec3 diffuse = light.diffuse.xyz * material.diffuse.xyz *
max(cosTheta,0.0);
    vec3 specular = light.specular.xyz * material.specular.xyz *
pow(max(cosPhi,0.0), material.shininess*3.0);
    fragColor = color + vec4((diffuse), 1.0);
```

ambient: Calculate the ADS component of ambient light
diffuse: Calculate diffuse reflection
specular: Calculate specular reflection
fragColor: Calculate the color to prevent the texture from being overwritten

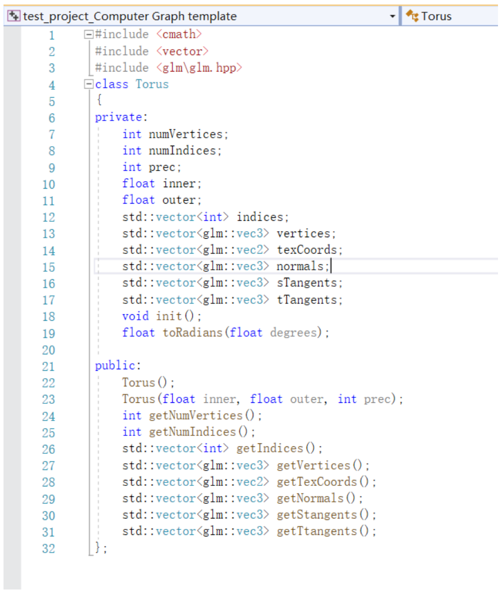
vertShader :

```
void main(void)
{   varyingVertPos = (mv_matrix * vec4(vertPos,1.0)).xyz;
    varyingLightDir = light.position - varyingVertPos;
    varyingNormal = (norm_matrix * vec4(vertNormal,1.0)).xyz;

    //varyingHalfVector =normalize(normalize(varyingLightDir) + normali
ze(-varyingVertPos)).xyz;

    varyingHalfVector=normalize(normalize(varyingLightDir) + (-
varyingVertPos)).xyz;

    gl_Position = proj_matrix * mv_matrix * vec4(vertPos,1.0);
    //varyingColor = vec4(vertPos,1.0)*0.5+vec4(0.5,0.5,0.5,0.5);gyn
    tc = tex_coord;
}
```

varyingVertPos: Output vertex position
varyingLightDir: Light direction
varyingNormal: Normal vector to rasterizer for interpolation
varyingHalfVector: Calculate angle bisector vector
gl_Position: Send location to fragshader
tc: Texture

2.1.9

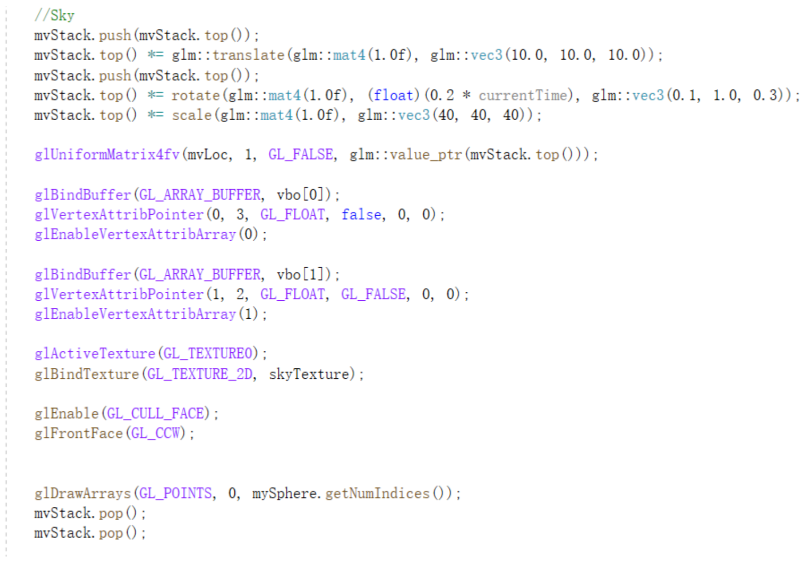
Step 9: Use Tours.h and Tours.cpp files to build a donut (Tours). The method of building a donut around Saturn is basically the same as the method of building a satellite



```
#include <cmath>
#include <vector>
#include <glm\glm.hpp>
class Torus
{
private:
    int numVertices;
    int numIndices;
    int prec;
    float inner;
    float outer;
    std::vector<int> indices;
    std::vector<glm::vec3> vertices;
    std::vector<glm::vec2> texCoords;
    std::vector<glm::vec3> normals;
    std::vector<glm::vec3> sTangents;
    std::vector<glm::vec3> tTangents;
    void init();
    float toRadians(float degrees);

public:
    Torus();
    Torus(float inner, float outer, int prec);
    int getNumVertices();
    int getNumIndices();
    std::vector<int> getIndices();
    std::vector<glm::vec3> getVertices();
    std::vector<glm::vec2> getTexCoords();
    std::vector<glm::vec3> getNormals();
    std::vector<glm::vec3> getStangents();
    std::vector<glm::vec3> getTtangents();
};
```

2.1.10

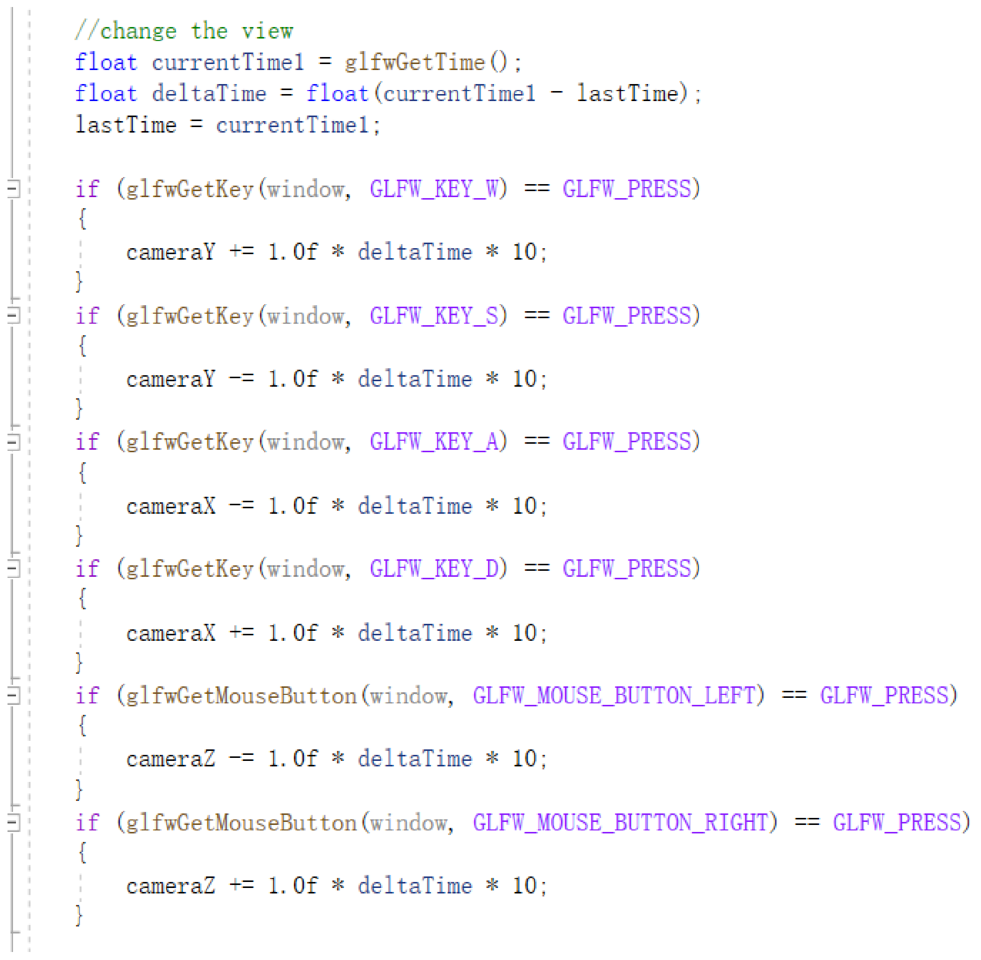
Step 10: Build a starry sky. Change the GL_TRIANGLES in glDrawArrays(GL_TRIANGLES, 0, mySphere.getNumIndices()); to GL_POINTS, and add a randomly drawn sky picture, you can set the radius of the surrounding center to 0,0,0, and scale () is set to a larger value.



```
//Sky
mvStack.push(mvStack.top());
mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(10.0, 10.0, 10.0));
mvStack.push(mvStack.top());
mvStack.top() *= rotate(glm::mat4(1.0f), (float)(0.2 * currentTime), glm::vec3(0.1, 1.0, 0.3));
mvStack.top() *= scale(glm::mat4(1.0f), glm::vec3(40, 40, 40));

glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()));

glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
glEnableVertexAttribArray(0);

glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, skyTexture);

glEnable(GL_CULL_FACE);
glFrontFace(GL_CCW);


glDrawArrays(GL_POINTS, 0, mySphere.getNumIndices());
mvStack.pop();
mvStack.pop();
```

2.1.11

Step 11: Add a function to display(). The purpose is to control the position of the camera by controlling the WASD key, and to change the distance from the solar system through the left and right mouse buttons.



```
//change the view
float currentTime1 = glfwGetTime();
float deltaTime = float(currentTime1 - lastTime);
lastTime = currentTime1;

if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
{
    cameraY += 1.0f * deltaTime * 10;
}
if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
{
    cameraY -= 1.0f * deltaTime * 10;
}
if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
{
    cameraX -= 1.0f * deltaTime * 10;
}
if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
{
    cameraX += 1.0f * deltaTime * 10;
}
if (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_LEFT) == GLFW_PRESS)
{
    cameraZ -= 1.0f * deltaTime * 10;
}
if (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_RIGHT) == GLFW_PRESS)
{
    cameraZ += 1.0f * deltaTime * 10;
}
```

2.1.12

Step 12: Write a void mouse_callback() function. The purpose is to change the perspective by dragging the mouse

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    GLfloat xoffset = xpos - lastX;
    GLfloat yoffset = lastY - ypos; // Reversed since y-coordinates go from bottom to left
    lastX = xpos;
    lastY = ypos;

    GLfloat sensitivity = 0.08; // Change this value to your liking
    xoffset *= sensitivity;
    yoffset *= sensitivity;

    yaw += xoffset;
    pitch += yoffset;

    glm::vec3 front;
    front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    front.y = sin(glm::radians(pitch));
    front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
    cameraFront = glm::normalize(front);
}
```

(3) Instructions for use

3.1 Use steps:

After running the program, you can use WASD to control the viewing angle. Click the left mouse button to get closer to the solar system and click the right mouse button to make the distance to the solar system farther.

4)Feeling and Summary

4.1 Advantages:

(1) Through mathematical means, the solar system is converged by a function, which has a strong mathematical reference value

(2) The distance between the observer and the solar system can be changed to produce more viewing angles to observe it

(3) The light is soft, there is no abrupt shadow line, and the light is correct, and the light will appear on the side facing the sun

(3) The starry sky is rendered by GPU to make it unlike the sky box and sky dome. Insert pictures with low background pixels

4.2 Disadvantages:

(1) The starry sky is not well considered. By using GL_POINTS to force the selection of the picture, a black and white picture needs to be drawn, and the position of the stars is indirectly changed by changing the white position of the picture

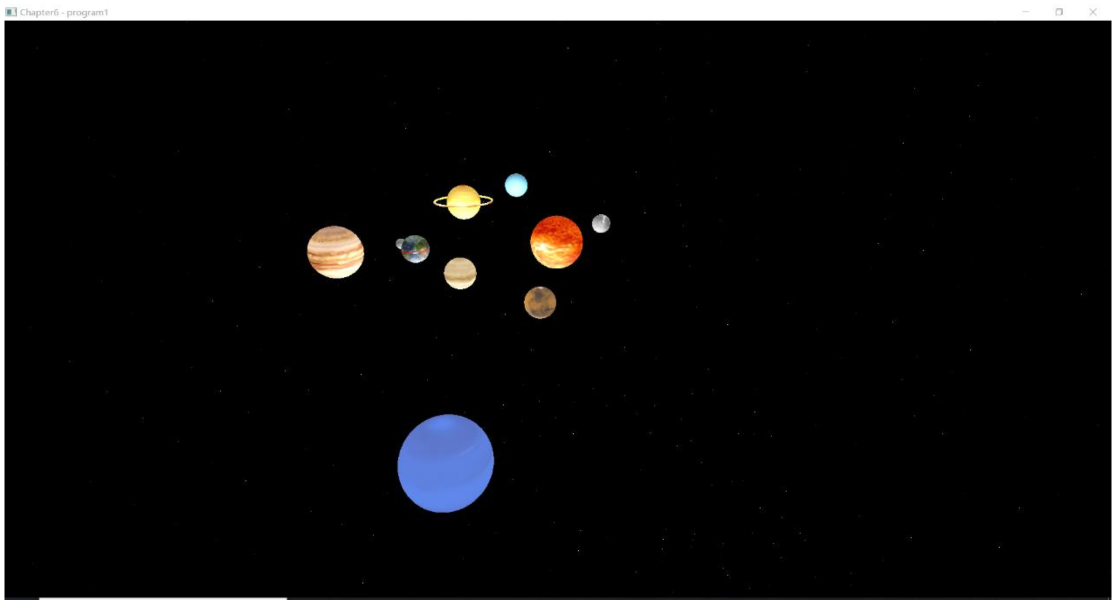
(2) Part of the code is not clearly understood, and it is not clear whether it can be removed. It can only be known by the results of running the program

(3) Saturn’s ring problem. It’s not clear how to reduce its thickness. You can only change

the value of the inner diameter and outer diameter to make it look very thin.

(5) Output result

5.1 Close range effect:



5.2 distance output effect: