浙江大學
ZHEJIANG UNIVERSITY

计算机图形学

Solar System

3220103422 刘韬

3220103422@zju.edu.cn

2024 年 10 月 29 日

目录

1 需求分析

首先分析需求

1. 2 suns,2+ planets,1+ satellite
2. Planets orbit around the sun
3. Satellites orbit around its planet
4. Trajectories are not co-planar
5. Navigation in the system (3D viewing)

要求 5 个星球，其中 2 个为太阳，2 个为行星，1 个为卫星。行星绕太阳公转，卫星绕行星公转，轨迹不在同一平面上。要求能够在系统中导航（3D 视图）。接着逐个分析如何实现。

2 suns,2+ planets,1+ satellite 确定颜色，大小，绘制实心球即可。

orbit around 这里要求绕某个中心旋转，实现这种动画通过每一帧旋转一定角度即可。通过角度变量随时间变化实现。

Trajectories are not co-planar 要求轨迹不在同一平面上，就是要求在动的同时做一个旋转，绕另一个轴旋转一定角度即可。

Navigation in the system 这里的要求是调整视角，可以通过调整 camera 的位置和朝向实现。

2 代码实现

2.1 定义变量

在上面的分析中，我们将具体的需求分解成了离散的元素，据此我们可以确定一些需要的变量。

常量 星球的大小，颜色，轨道半径，旋转速度，轨道角度

变量 摄影机的位置，朝向，鼠标灵敏度，是否旋转

2.2 绘制图形

需要绘制轨道、星球，分别用如下实现：

```
void star(const float* color, const float* sphere)
{
    glColor3f(color[0], color[1], color[2]);
    glutSolidSphere(sphere[0], sphere[1], sphere[2]);
}

void orbit(const int radius)
{
    #ifdef ORBIT
    glBegin(GL_LINE_LOOP);
    glColor3f(colorOfOrbit[0],colorOfOrbit[1],colorOfOrbit[2]);
    for(int i = 0; i < 360; i++)
    {
        float theta = i * M_PI / 180;
        glVertex3f(radius * cos(theta), 0.0, radius * sin(theta));
    }
    glEnd();
    #endif
    return;
}
```

确定颜色后使用 `glColor3f` 设置颜色，使用 `glutSolidSphere` 绘制实心球。轨道的绘制使用点绘制圆形。添加了是否需要绘制轨道的宏定义，可以在编译时选择是否绘制轨道。

2.3 组合图形

上面我们已经有了基本的图形元素，现在通过 `display` 来将这些图形排列与显示。在函数中，设置是否转动，摄影位置以及角度，然后绘制星球和轨道。

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // 清除颜色缓冲区和深度缓冲区
    if(rotate) angle += 3; // angle 更新才能开始转动，这里的 3 也是代表基本转动速度
    glLoadIdentity();
    gluLookAt(eyePosition[0], eyePosition[1], eyePosition[2], 0.0, 0.0, 0.0, 0.0,
    ↪  1.0, 0.0); // 设置摄像机位置
    // 设置摄像机角度，围绕 x 轴和 y 轴旋转
    glRotated(camera_angle_v, 1.0, 0.0, 0.0);
    glRotated(camera_angle_h, 0.0, 1.0, 0.0);
    // solar system 1
```

```
    glPushMatrix();
    {
        glRotatef(angle * speedOfSun, 0.0, 1.0, 0.0); // 公转
        orbit(orbitOfSun); // 太阳轨道
        glTranslatef(orbitOfSun, 0.0, 0.0); // 太阳位置
        star(colorOfSun, sphereOfSun); // 太阳
        // planet 1
        glPushMatrix();
        {
            glRotatef(angleOfPlanet1, 0.0, 0.0, 1.0); // 轨道调整
            glRotatef(angle * speedOfPlanet1, 0.0, 1.0, 0.0); // 行星 1 公转
            orbit(orbitOfPlanet1); // 行星 1 轨道
            glTranslatef(orbitOfPlanet1, 0.0, 0.0); // 行星 1 位置
            star(colorOfPlanet1, sphereOfPlanet1); // 行星 1
        }
        glPopMatrix();
    }
    glPopMatrix();
    // solar system 2
    ...

}
```

上面展示了如何设置视角以及更新旋转，然后使用了 glPushMatrix 和 glPopMatrix 来保存和恢复状态，实现相对运动，其余的星球和卫星的绘制类似。

2.4 视图设置

在三维的图像显示中，视图的设置保证了我们能够看到我们想要的图像。调整视口、投影矩阵和模型视图矩阵，以适应新的窗口大小。

```
void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION); // 设置投影矩阵
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 0.01f, 2000.0f);
    glMatrixMode(GL_MODELVIEW); // 设置模型视图矩阵
    glLoadIdentity();
}
```

2.5 控制设置

为了实现在系统中导航，我们需要设置一些控制，这里使用键盘和鼠标来控制视角和旋转，接受键盘和鼠标事件，然后设置相应的变量。

```
void onKeyboard(unsigned char key, int x, int y) {
    switch (key){
        case 'r':
            rotate = !rotate;
            break;
        case 27:
            exit(0);
            break;
        case 'w':
            eyePosition[1] += 10;
            break;
        case 's':
            eyePosition[1] -= 10;
            break;
        case 'a':
            eyePosition[0] -= 10;
            break;
        case 'd':
            eyePosition[0] += 10;
            break;
        default:
            break;
    }
}

void onSpecialKey(int key, int x, int y) {
    switch (key){
        case GLUT_KEY_UP:
            eyePosition[2] -= 10;
            break;
        case GLUT_KEY_DOWN:
            eyePosition[2] += 10;
            break;
        default:
            break;
    }
}
```

键盘主要用于控制是否旋转，退出，上下左右移动，控制摄影机的位置。

```
void mouseButton(int button, int state, int x, int y){
    if(button == GLUT_LEFT_BUTTON){
        if(state == GLUT_DOWN){
            Dragging = true;
            drag_x_origin = x;
            drag_y_origin = y;
        }
        else{
            Dragging = false;
        }
    }
}

void mouseMove(int x, int y){
    if(Dragging){
        camera_angle_h += (x - drag_x_origin) * mouse_sensitivity;
        camera_angle_v += (y - drag_y_origin) * mouse_sensitivity;
        drag_x_origin = x;
        drag_y_origin = y;
    }
    return;
}
```

鼠标主要用于控制视角，左键按下时记录当前位置，移动时计算移动距离，更新视角。

2.6 编译运行

编译时需要链接 OpenGL 库，在 darwin 和 linux 下，链接的库不同，在程序和编译指令中都做了适配

```
ifeq ($(UNAME_S), Linux)
^^ILDFLAGS := -lGL -lGLU -lglut
else ifeq ($(UNAME_S), Darwin)
^^ILDFLAGS := -L/System/Library/Frameworks -framework GLUT -framework OpenGL
else
    $(error Unsupported platform)
endif
```

```
#if defined (__APPLE__)
    #define GL_SILENCE_DEPRECATION
    #include <GLUT/glut.h>
```

```
#elif defined(_WIN32) || defined(_WIN64) || defined(__linux__)
    #include <GL/glut.h>
#endif
```

编译命令为 `make all`，运行命令为 `make run`。另外提供了是否绘制轨道的选项，在编译时使用 `make all ORBIT=0` 即可。

3 实现效果

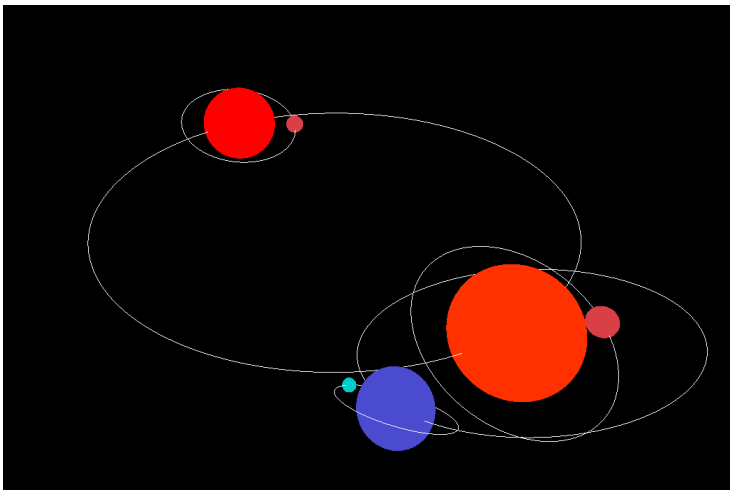

图 1: 太阳系模拟有轨道

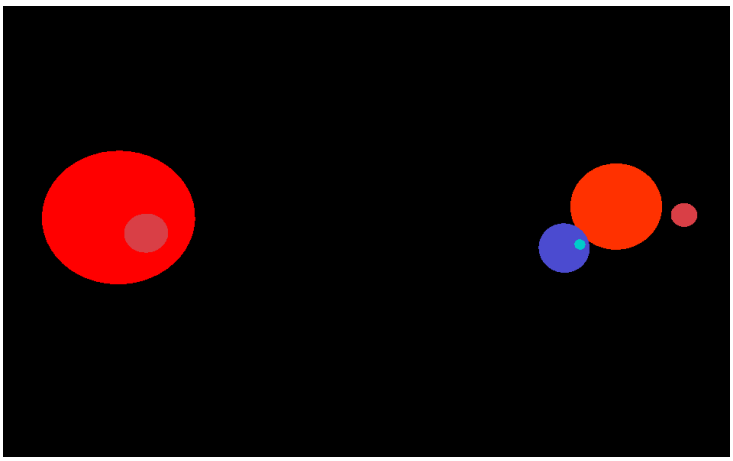

图 2: 太阳系模拟无轨道