

浙江大学



《计算机图形学》 实验报告

实验名称：	Solar System
姓名：	王晓宇
学号：	3220104364
电子邮箱：	3220104364@zju.edu.cn
联系电话：	19550222634
授课教师：	吴鸿智
助教：	丁华铿

2024 年 11 月 1 日

Solar System

- 1 实验内容及简要原理介绍
 - 1.1 实验内容
 - 1.2 简要原理介绍
- 2 实验框架思路与代码实现
 - 2.1 实验框架思路
 - 2.2 代码实现
- 3 实验结果与分析

Solar System

1 实验内容及简要原理介绍

Assignment #4

- Build a solar system with OpenGL
 - 2 suns, 2+ planets, 1+ satellite
 - Planets orbit around the sun
 - Satellites orbit around its planet
 - Trajectories are not co-planar
 - Navigation in the system (3D viewing)
 - E.g., W+S+A+D -> walk, mouse-move -> change viewdir
- Submit Code(Source+Exec) + Report
- Due Date: 11/03

1.1 实验内容

本次实验旨在通过OpenGL和GLUT库实现一个简单的太阳系模拟程序。程序将展示一个太阳和两个行星，以及它们的卫星，模拟它们围绕太阳的旋转和轨道运动。

1.2 简要原理介绍

OpenGL(Open Graphics Library)是一个跨语言、跨平台的图形API，用于渲染2D和3D矢量图形。GLUT(OpenGL Utility Toolkit)是OpenGL的一个工具包，提供了创建窗口、处理用户输入等辅助功能。本实验中，我们利用OpenGL的3D图形渲染能力和GLUT的窗口管理功能，创建了一个太阳系的动态模拟。

2 实验框架思路与代码实现

2.1 实验框架思路

代码整体分为几个主要部分：全局变量定义、函数声明、OpenGL绘图函数、键盘处理函数、窗口大小调整函数、鼠标移动处理函数和主函数。

- 全局变量定义

包括行星角度、移动速度、窗口大小、摄像机位置等。

- 函数声明

定义了定时器函数 `timer`、窗口大小调整函数 `reshape`、鼠标移动处理函数 `passiveMotion`、键盘处理函数 `keyboard`、OpenGL绘图函数 `drawSolarSystem` 和显示函数 `display`。

- OpenGL绘图函数

`drawSolarSystem` 函数中，使用 `glutWireTorus` 和 `glutSolidSphere` 绘制太阳、行星的轨道。

- 键盘处理函数

`keyboard` 函数处理键盘输入，实现摄像机的前后左右移动。

- 窗口大小调整函数

`reshape` 函数调整视窗大小，更新投影矩阵。

- 鼠标移动处理函数

`passiveMotion` 函数处理鼠标移动，实现摄像机的视角旋转。

2.2 代码实现

代码中使用了多个OpenGL函数来实现3D图形的绘制和变换。例如，`glRotatef` 和 `glTranslatef` 用于实现对象的旋转和平移，`glutSolidSphere` 用于绘制球体，`glutWireTorus` 用于绘制圆环表示轨道。

先介绍全局变量定义

```
1 #define pi 3.1415926
2
3 // The angle of the planet
4 float planetAngle ;
5 // The speed of the planet
```

```

6  float moveSpeed = 0.1f;
7  // The window size
8  int windowHeight = 600;
9  int windowHeight = 600;
10 // The move(left/right) direction
11 bool isMoveLeft;
12 bool isMoveRight;
13 // The camera position for lookAt
14 float eyeX = 5.0f, eyeY = 0.0f, eyeZ = 5.0f; // The camera position
15 float centerX = 0.0f, centerY = 0.0f, centerZ = 0.0f; // The center
    position
16 float upX = 0.0f, upY = 0.0f, upZ = 1.0f; // The up vector
17 // The normal vector of the (center and eye vector)
18 #define normalX ((centerY-eyey)*upZ-(centerZ-eyez)*upY)
19 #define normalY ((centerZ-eyez)*upX-(centerX-eyex)*upZ)
20 #define normalZ ((centerX-eyex)*upY-(centerY-eyey)*upX)
21 // The standard normal vector
22 float normalLength;
23 float temp_normalX;
24 float temp_normalY;
25 float temp_normalZ;

```

- 1 - 这里定义了行星的旋转角度`planetAngle`
- 2 - 行星的移动速度`moveSpeed`
- 3 - 窗口的宽度和高度`windowWidth`和`windowHeight`
- 4 - 摄像机的位置`eyeX`、`eyeY`和`eyeZ`
- 5 - 观察点的位置`centerX`、`centerY`和`centerZ`
- 6 - Up vector向量`upX`、`upY`和`upZ`
- 7 - 观察向量与Up Vector的法向量`normalX`、`normalY`和`normalZ`
- 8 - 法向量的标准长度`normalLength`
- 9 - 标准化法向量`temp_normalX`、`temp_normalY`和`temp_normalZ`

以下是每个函数的原理说明：

- **timer** 函数

定时器函数，用于更新行星的旋转角度，并触发重新绘制。

```

1 // The function of the timer for time refresh
2 void timer(int p)
3 {
4     planetAngle += 1; // Update the angle of the planet
5     glutPostRedisplay();
6     glutTimerFunc(7, timer, 0);
7 }

```

- 这里控制刷新微秒数为7，即每7ms刷新一次，刷新率为 $1000/7=142.85\text{Hz}$ 接近144Hz。

- **reshape** 函数

窗口大小调整函数，用于设置视口大小和投影矩阵。

```

1 // The function of the reshape for the window size
2 void reshape(int width, int height)
3 {
4     glViewport(0, 0, width, height); // Set the viewport size
5     glMatrixMode(GL_PROJECTION); // Reset the projection
6     matrix
7     glLoadIdentity(); // Load the identity
8     matrix
9     // Set the perspective projection matrix (field of view,
10    aspect ratio, near clipping plane, far clipping plane)
11    gluPerspective(45.0, (double)width / (double)height, 0.1,
12    100.0);
13    glMatrixMode(GL_MODELVIEW); // Reset the modelview
14    matrix
15 }

```

- 这里设置了透视投影矩阵，视角为45度，宽高比为窗口宽高比，近裁剪面为0.1，远裁剪面为100.0。
- 使用透视投影矩阵可以使得远处的物体看起来比较小，近处的物体看起来比较大，更符合人眼的视觉感受。

- **passiveMotion** 函数

鼠标移动处理函数，用于根据鼠标位置更新观察向量，实现视角的旋转。

```

1 // The function of the passive motion for the mouse move
2 void passiveMotion(int x, int y)
3 {
4     windowHeight = glutGet(GLUT_WINDOW_HEIGHT);
5     windowWidth = glutGet(GLUT_WINDOW_WIDTH);
6     if(isMoveLeft || isMoveRight)
7     {
8         return;
9     }
10    centerZ = 5.0*(windowHeight/2-y)/windowHeight;
11    centerX = (-eyeY)*0.5*(x-windowWidth/2)/windowWidth;
12    centerY = (eyeX)*0.5*(x-windowWidth/2)/windowWidth;
13 }

```

- 这里根据鼠标位置更新观察向量，实现视角的旋转。
- 这里使用`passiveMotion`函数处理鼠标移动事件，根据鼠标位置更新观察向量，实现视角的旋转，不需要按下鼠标键。
- 这里的视线方向其实是在以`z`轴为中心轴的圆柱截面上，通过鼠标的移动，改变视线方向，实现视角的旋转。
- 通过鼠标相对于窗口的位置，可以计算出视线方向的变化，从而实现视角的旋转。

- `keyboard` 函数

键盘处理函数，用于处理键盘输入，实现摄像机的前后左右移动。

```

1 // The function of the keyboard for the key press
2 void keyboard(unsigned char key, int x, int y)
3 {
4     normalLength =
5     sqrt(normalX*normalX+normalY*normalY+normalZ*normalZ);
6     temp_normalX = normalX/normalLength;
7     temp_normalY = normalY/normalLength;
8     temp_normalZ = normalZ/normalLength;
9     switch (key)
10    {
11        case 'w': // move forward

```

```

11         eyeX += moveSpeed*(centerX-eyeX);
12         eyeY += moveSpeed*(centerY-eyeY);
13         eyeZ += moveSpeed*(centerZ-eyeZ);
14         break;
15     case 's': // move backward
16         eyeX -= moveSpeed*(centerX-eyeX);
17         eyeY -= moveSpeed*(centerY-eyeY);
18         eyeZ -= moveSpeed*(centerZ-eyeZ);
19         break;
20     case 'a': // move left
21         eyeX -= moveSpeed*temp_normalX;
22         eyeY -= moveSpeed*temp_normalY;
23         eyeZ -= moveSpeed*temp_normalZ;
24         isMoveLeft = true;
25         centerX -= moveSpeed*temp_normalX;
26         centerY -= moveSpeed*temp_normalY;
27         centerZ -= moveSpeed*temp_normalZ;
28         break;
29     case 'd': // move right
30         eyeX += moveSpeed*temp_normalX;
31         eyeY += moveSpeed*temp_normalY;
32         eyeZ += moveSpeed*temp_normalZ;
33         isMoveRight = true;
34         centerX += moveSpeed*temp_normalX;
35         centerY += moveSpeed*temp_normalY;
36         centerZ += moveSpeed*temp_normalZ;
37         break;
38     }
39     isMoveLeft = isMoveRight = false;
40     glutPostRedisplay(); // Refresh the display
41 }

```

- 这里根据键盘输入更新摄像机位置，实现摄像机的前后左右移动。
- 这里计算出了左右平移方向(即`upVector`和视线方向的叉乘)，并根据键盘输入更新摄像机位置，实现摄像机的前后左右移动。

- 左右移动时，需要同时更新视线方向，以保持视线方向与视线方向的叉乘垂直，从而保持视线方向的正交性。

- `drawSolarSystem` 函数

OpenGL绘图函数，用于绘制太阳、行星及其轨道。

```
1 void drawSolarSystem()
2 {
3     // draw the sun's orbit
4     glColor3f(0.7f, 0.7f, 0.7f); // gray
5     glPushMatrix();
6     glutWireTorus(0.001, 0.7, 10, 100); // the orbit of
    radius 0.7 so thin that it looks like a circle
7     glPopMatrix();
8
9     // draw the first sun
10    glPushMatrix();
11    glRotatef(planetAngle * 0.1, 0.0f, 0.0f, 1.0f); // rotate
    around z axis
12    glTranslatef(0.7f, 0.0f, 0.0f); // move to the orbit
    position
13    glColor3f(1.0f, 1.0f, 0.0f); // yellow
14    glutSolidSphere(0.2f, 50, 50); // the solid sphere of
    radius 0.2
15    glPopMatrix();
16    // draw the second sun
17    glPushMatrix();
18    glRotatef(planetAngle * 0.1, 0.0f, 0.0f, 1.0f); // rotate
    around z axis
19    glTranslatef(-0.7f, 0.0f, 0.0f); // move to the orbit
    position
20    glColor3f(1.0f, 1.0f, 0.0f); // yellow
21    glutSolidSphere(0.2f, 50, 50); // the solid sphere of
    radius 0.2
22    glPopMatrix();
23
24    // draw the first planet's orbit
```

```

25     glColor3f(0.7f, 0.7f, 0.7f); // gray
26     glPushMatrix();
27     // apply the rotation transformation (rotate 45 degrees
    around the x axis)
28     glRotatef(135.0f, 1.0f, 0.0f, 0.0f);
29     glutWireTorus(0.001, 1.5, 10, 100); // the torus of
    radius 1.5 so thin that it looks like a circle
30     glPopMatrix();
31
32     // draw the first planet
33     glPushMatrix();
34     glRotatef(planetAngle, 0.0f, 1.0f, 1.0f); // rotate
    around (0,1,1) axis
35     glTranslatef(1.5f, 0.0f, 0.0f); // move to the orbit
    position
36     glColor3f(0.0f, 0.0f, 1.0f); // blue
37     glutSolidSphere(0.08f, 50, 50); // the solid sphere of
    radius 0.08
38     glPopMatrix();
39
40     // draw the first planet's satellite orbit of radius 0.5
41     glColor3f(0.7f, 0.7f, 0.7f); // gray
42     glPushMatrix();
43     glRotatef(planetAngle, 0.0f, 1.0f, 1.0f); // rotate
    around (0,1,1) axis
44     glTranslatef(1.5f, 0.0f, 0.0f); // move to the orbit
    position
45     glutWireTorus(0.001, 0.5, 10, 100); // the torus of
    radius 0.5 so thin that it looks like a circle
46     glPopMatrix();
47
48     // draw the first planet's satellite
49     glPushMatrix();
50     glRotatef(planetAngle, 0.0f, 1.0f, 1.0f); // rotate
    around (0,1,1) axis

```

```

51     glTranslatef(1.5f, 0.0f, 0.0f); // move to the orbit
    position
52     // glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
53     glRotatef(planetAngle*0.2, 0.0f, 0.0f, 1.0f); // rotate
    around z axis
54     glTranslatef(-0.5f, 0.0f, 0.0f); // move to the orbit
    position
55     glColor3f(0.0f, 1.0f, 0.0f); // green
56     glutSolidSphere(0.03f, 50, 50); // draw the solid sphere
    of radius 0.03
57     glPopMatrix();
58
59     // draw the second planet's orbit of radius 2.5
60     glColor3f(0.7f, 0.7f, 0.7f); // gray
61     glPushMatrix();
62     // apply the rotation transformation (rotate 45 degrees
    around the x axis)
63     glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
64     glutWireTorus(0.001, 2.5, 10, 100); // the torus of
    radius 2.5 so thin that it looks like a circle
65     glPopMatrix();
66
67     // draw the second planet
68     glPushMatrix();
69     glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate
    around (0,-1,1) axis
70     glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit
    position
71     glColor3f(1.0f, 0.0f, 0.0f); // red
72     glutSolidSphere(0.10f, 50, 50); // draw the solid sphere
    of radius 0.1
73     glPopMatrix();
74
75     // draw the second planet's satellite orbit of radius 0.5
76
77     glColor3f(0.7f, 0.7f, 0.7f); // gray

```

```

78     glPushMatrix();
79     glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate
    around (0,-1,1) axis
80     glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit
    position
81     glutWireTorus(0.001, 0.5, 10, 100); // the torus of
    radius 0.5 so thin that it looks like a circle
82     glPopMatrix();
83
84     // draw the second planet's satellite
85     glPushMatrix();
86     glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate
    around (0,-1,1) axis
87     glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit
    position
88     // glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
89     glRotatef(planetAngle*0.1, 0.0f, 0.0f, 1.0f); // rotate
    around z axis
90     glTranslatef(-0.5f, 0.0f, 0.0f); // move to the orbit
    position
91     glColor3f(0.0f, 1.0f, 0.0f); // green
92     glutSolidSphere(0.03f, 50, 50); // draw the solid sphere
    of radius 0.03
93     glPopMatrix();
94
95     // draw the third planet's orbit
96     glColor3f(0.7f, 0.7f, 0.7f); // gray
97     glPushMatrix();
98     glRotatef(0.0f, 0.0f, 0.0f, 1.0f);
99     glutWireTorus(0.001, 3.5, 10, 100); // the torus of
    radius 3.5 so thin that it looks like a circle
100    glPopMatrix();
101
102    // draw the third planet
103    glPushMatrix();

```

```

104     glRotatef(planetAngle*0.05, 0.0f, 0.0f, 1.0f); // rotate
        around (0,0,1) axis
105     glTranslatef(3.5f, 0.0f, 0.0f); // move to the orbit
        position
106     glColor3f(0.7f, 0.5f, 0.5f);
107     glutSolidSphere(0.08f, 50, 50); // the solid sphere of
        radius 0.08
108     glPopMatrix();
109
110 }
111

```

- 这里使用 `glutWireTorus` 和 `glutSolidSphere` 绘制太阳、行星及其轨道。当圆环足够细时，看起来就像一个圆。
- 这里使用 `glRotatef` 和 `glTranslatef` 函数实现了太阳、行星的旋转和平移。
- 这里使用 `glutSolidSphere` 函数绘制了太阳、行星和卫星的球体，实现了太阳系的模拟。
- 这里使用了多个 `glPushMatrix` 和 `glPopMatrix` 函数，实现了对不同物体的独立变换，保证了物体之间的相对位置关系。
- 可以注意到变换矩阵是左结合的，即先执行的变换在右边，后执行的变换在左边，这样可以保证变换的顺序正确。

- **display** 函数

显示函数，用于设置摄像机位置并调用绘图函数绘制整个场景。

```

1 void display()
2 {
3     float green[3]={0,0,1};
4     glClearColor(0,0,0,0);
5     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
6     glLoadIdentity();
7     // Set the camera position
8     gluLookAt(eyeX, eyeY, eyeZ,      // The camera position
9              centerX, centerY, centerZ, // The center
        position

```

```

10         upX, upY, upZ);          // The up vector
11
12     glColor3fv(green);
13     drawSolarSystem();
14     glFlush();
15 }

```

- 使用 `gluLookAt` 函数设置摄像机位置，实现摄像机的移动和视角的旋转。
- 使用 `glClearColor` 和 `glClear` 函数清空颜色缓冲区和深度缓冲区，实现场景的刷新。
- 使用 `glLoadIdentity` 函数加载单位矩阵，重置模型视图矩阵。
- 使用 `glFlush` 函数刷新绘图命令，实现场景的显示。

3 实验结果与分析

运行程序后，我们得到了一个动态的太阳系模拟窗口。窗口中展示了双星和三个行星，以及它们的卫星。太阳位于窗口中心，行星和卫星围绕太阳旋转。用户可以通过键盘控制摄像机的移动，通过鼠标移动实现视角的旋转。

