浙江大学

《计算机图形学》
实验报告

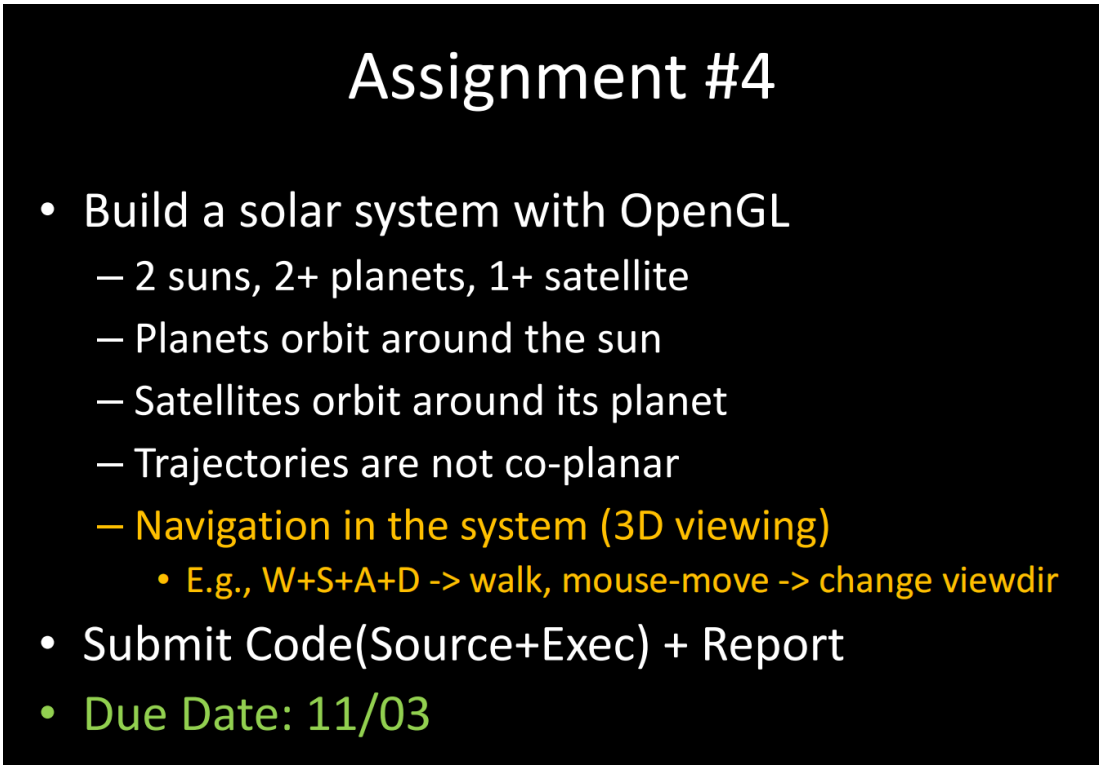
实验名称	：	Solar System
姓　　名	：	王晓宇
学　　号	：	3220104364
电子邮箱	：	3220104364@zju.edu.cn
联系电话	：	19550222634
授课教师	：	吴鸿智
助　　教	：	丁华铿

2024 年 11 月 1 日

Solar System

Solar System

1 实验内容及简要原理介绍



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

```
#define pi 3.1415926

// The angle of the planet
float planetAngle ;
// The speed of the planet

```

```
float moveSpeed = 0.1f;
// The window size
int windowWidth = 600;
int windowHeight = 600;
// The move(left/right) direction
bool isMoveLeft;
bool isMoveRight;
// The camera position for lookAt
float eyeX = 5.0f, eyeY = 0.0f, eyeZ = 5.0f; // The camera position
float centerX = 0.0f, centerY = 0.0f, centerZ = 0.0f; // The center position
float upX = 0.0f, upY = 0.0f, upZ = 1.0f; // The up vector
// The normal vector of the (center and eye vector)
#define normalX ((centerY-eyeY)*upZ-(centerZ-eyeZ)*upY)
#define normalY ((centerZ-eyeZ)*upX-(centerX-eyeX)*upZ)
#define normalZ ((centerX-eyeX)*upY-(centerY-eyeY)*upX)
// The standard normal vector
float normalLength;
float temp_normalX;
float temp_normalY;
float temp_normalZ;
```

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

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

  - 这里控制刷新微秒数为**7**，即每**7ms**刷新一次，刷新率为**1000/7=142.85Hz**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。

```
// The function of the reshape for the window size
void reshape(int width, int height)
{
    glViewport(0, 0, width, height); // Set the viewport size
    glMatrixMode(GL_PROJECTION);     // Reset the projection matrix
    glLoadIdentity();                // Load the identity matrix
    // Set the perspective projection matrix (field of view, aspect ratio, near clipping plane, far clipping plane)
    gluPerspective(45.0, (double)width / (double)height, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);      // Reset the modelview matrix
}
```

  - 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。

```
// The function of the passive motion for the mouse move
void passiveMotion(int x, int y)
{
    windowWidth = glutGet(GLUT_WINDOW_WIDTH);
    windowHeight = glutGet(GLUT_WINDOW_HEIGHT);
    if(isMoveLeft || isMoveRight)
    {
        return;
    }
    centerZ =  5.0*(windowHeight/2-y)/windowHeight;
    centerX =  (-eyeY)*0.5*(x-windowWidth/2)/windowWidth;
    centerY = (eyeX)*0.5*(x-windowWidth/2)/windowWidth;
}
```

- 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。

```
// The function of the keyboard for the key press
void keyboard(unsigned char key, int x, int y)
{
    normalLength =
sqrt(normalX*normalX+normalY*normalY+normalZ*normalZ);
    temp_normalX = normalX/normalLength;
    temp_normalY = normalY/normalLength;
    temp_normalZ = normalZ/normalLength;
    switch (key)
    {
        case 'w': // move forward
```

```
            eyeX += moveSpeed*(centerX-eyeX);
            eyeY += moveSpeed*(centerY-eyeY);
            eyeZ += moveSpeed*(centerZ-eyeZ);
            break;
        case 's': // move backward
            eyeX -= moveSpeed*(centerX-eyeX);
            eyeY -= moveSpeed*(centerY-eyeY);
            eyeZ -= moveSpeed*(centerZ-eyeZ);
            break;
        case 'a': // move left
            eyeX -= moveSpeed*temp_normalX;
            eyeY -= moveSpeed*temp_normalY;
            eyeZ -= moveSpeed*temp_normalZ;
            isMoveLeft = true;
            centerX -= moveSpeed*temp_normalX;
            centerY -= moveSpeed*temp_normalY;
            centerZ -= moveSpeed*temp_normalZ;
            break;
        case 'd': // move right
            eyeX += moveSpeed*temp_normalX;
            eyeY += moveSpeed*temp_normalY;
            eyeZ += moveSpeed*temp_normalZ;
            isMoveRight = true;
            centerX += moveSpeed*temp_normalX;
            centerY += moveSpeed*temp_normalY;
            centerZ += moveSpeed*temp_normalZ;
            break;
    }
    isMoveLeft = isMoveRight = false;
    glutPostRedisplay(); // Refresh the display
}
```

- 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。

```
void drawSolarSystem()
{
    // draw the sun's obrbit
    glColor3f(0.7f, 0.7f, 0.7f); // gray
    glPushMatrix();
    glutWireTorus(0.001, 0.7, 10, 100); // the orbit of radius 0.7 so thin that it looks like a circle
    glPopMatrix();

    // draw the first sun
    glPushMatrix();
    glRotatef( planetAngle *0.1, 0.0f, 0.0f, 1.0f); // rotate around z axis
    glTranslatef(0.7f, 0.0f, 0.0f); // move to the orbit position
    glColor3f(1.0f, 1.0f, 0.0f); // yellow
    glutSolidSphere(0.2f, 50, 50); // the solid sphere of radius 0.2
    glPopMatrix();
    // draw the second sun
    glPushMatrix();
    glRotatef(planetAngle*0.1, 0.0f, 0.0f, 1.0f); // rotate around z axis
    glTranslatef(-0.7f, 0.0f, 0.0f); //  move to the orbit position
    glColor3f(1.0f, 1.0f, 0.0f); // yellow
    glutSolidSphere(0.2f, 50, 50); // the solid sphere of radius 0.2
    glPopMatrix();

    // 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 second planet's satellite orbit of radius 2.5
41      glColor3f(0.7f, 0.7f, 0.7f); // gray
42      glPushMatrix();
43      // apply the rotation transformation (rotate 45 degrees
around the x axis)
44      glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
45      glutWireTorus(0.001, 2.5, 10, 100); // the torus of radius
2.5 so thin that it looks like a circle
46      glPopMatrix();
47
48      // draw the second planet's satellite
49      glPushMatrix();
50      glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate
around (0,-1,1) axis
51      glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit
position
```

```
    glColor3f(1.0f, 0.0f, 0.0f); // red
    glutSolidSphere(0.10f, 50, 50); // draw the solid sphere of radius 0.1
    glPopMatrix();

    // draw the second planet's satellite's satellite orbit of radius 0.5

    glColor3f(0.7f, 0.7f, 0.7f); // gray
    glPushMatrix();
    glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate around (0,-1,1) axis
    glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit position
    glutWireTorus(0.001, 0.5, 10, 100); // the torus of radius 0.5 so thin that it looks like a circle
    glPopMatrix();

    // draw the second planet's satellite's satellite
    glPushMatrix();
    glRotatef(planetAngle*0.5, 0.0f, -1.0f, 1.0f); // rotate around (0,-1,1) axis
    glTranslatef(-2.5f, 0.0f, 0.0f); // move to the orbit position
    // glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(planetAngle*0.1, 0.0f, 0.0f, 1.0f); // rotate around z axis
    glTranslatef(-0.5f, 0.0f, 0.0f); // move to the orbit position
    glColor3f(0.0f, 1.0f, 0.0f); // green
    glutSolidSphere(0.03f, 50, 50); // draw the solid sphere of radius 0.03
    glPopMatrix();

}
```

- 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。

```
void display()
{
    float green[3]={0,0,1};
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    // Set the camera position
    gluLookAt(eyeX, eyeY, eyeZ,     // The camera position
              centerX, centerY, centerZ, // The center position
              upX, upY, upZ);       // The up vector

    glColor3fv(green);
    drawSolarSystem();
    glFlush();
}
```

  - 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两个行星，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。