Assignment 4 Solar System

Report

姓名：胡心卓　　学号：3150100520

实验目的

1. 搭建一个类似太阳系的星系系统，要求有除了恒星（太阳）之外有 2 个及以上的行星，1 个及以上的卫星
2. 实现行星能够绕着星系中的恒星（太阳）公转
3. 实现卫星能够绕着自己的行星公转
4. 各行星和卫星的公转轨道不能共面
5. 实现通过键盘和鼠标的交互使得观察者能够在你搭建的星系中进行 3D 观察（例如：使用 WSAD 键进行视角的移动，使用鼠标改变观察视角……）

实验环境

Visual Studio 2017+NupenGL 包+SOIL 库(简易 OpenGL 图像库)

实现方法

太阳系的实现(行星和卫星、轨道、公转)

行星和卫星

使用 OpenGL 自带的实体 API 来实现球体的绘制，具体的函数名称为 glusphere（），以及 glutSolidSphere()和 glutWireSphere(),每个函数的参数略有不同，具体可以在代码中查看。绘制好球体后，使用 OpenGL 自带的 glPushMatrix()和 glPopMatrix()函数来对当前的坐标系进行压栈和出栈，并同时使用 Translate 和 Rotate 函数移动球体，改变每个球体的位置以及旋转的角度，例如太阳地球月球系统的绘制如下:

```
    glPushMatrix();

    glColor3f(1.0, 0.0, 0.0);
    glEnable(GL_TEXTURE_2D);          //防止纹理只显示一次就被冲掉，开启 2D
纹理
    glGenerateMipmap(GL_TEXTURE_2D); //为 target 相关联的纹理生成一组完
整的 mipmap
    gluQuadricTexture(quadPlanet, GLU_TRUE); //设置自动计算纹理
    gluSphere(quadPlanet, 40, 1000, 1000);          //绘制太阳
    //glutWireSphere(40, 1000.0, 1000.0);
    glDisable(GL_TEXTURE_2D);     //关闭 2D 纹理
    DrawOrbit(80, 0, 1, 0);       //绘制地球轨道
    glRotatef(earthAngle, 0.0, 1.0, 0.0);  //设置地球旋转的轨道平面

    glPushMatrix();
    glTranslatef(80, 0.0, 0.0);       //移动地球到相应位置

    glColor3f(0.0, 0.0, 1.0);
    glutSolidSphere(10, 30, 30); //绘制地球
    DrawOrbit(14.14213562, 0, 0, 1);
    glRotatef(moonAngle, 0.0, 0.0, 1.0);

    glPushMatrix();
    glTranslated(10.0, 10.0, 0.0);

    glColor3f(0.5, 0.5, 0.5);
    glutSolidSphere(5, 100, 100);//绘制月球;
    glPopMatrix();
```

大致操作为先绘制太阳，然后压栈，然后使用 translate 和 rotate 将坐标系移动到地球的位置，绘制地球，然后继续压栈，将坐标系移动到月球位置，绘制月球。又由于需要实现公转的效果，需要在太阳的坐标系下实现旋转效果，压栈后就能实现地球绕太阳公转了。

公转

上述操作只能通过人为地改变旋转角的大小来改变各行星和卫星公转的角度，而且观察时是静止的。要想连续实现旋转的效果，必须调用 glutTimerFunc 这个回调函数，相当于使用一个定时器，在每个定时器的时间内各行星和卫星各自转过一个角度，然后使用 glutPostRedisplay()标记窗口需要重新绘制。回调函数的代码如下：

轨道

为了使绘制的星系系统运动可视性更好，我们可以在空间坐标系中绘制各行星和卫星公转运动的轨道，通过 glBegin(GL_LINE_LOOP)根据圆在空间中的参数方程绘制。我们除了需要知道所绘制的圆的半径，还要提供一个垂直轨道平面的法向量。我们根据法向量可以计算得到两个即同时垂直法向量，又互相垂直的单位向量，根据这两个单位向量 a,b 的坐标就可以画圆。具体的函数代码可以在工程文件中查看。

3D 观察的实现

其实在绘制星系的初始化阶段我们就要设置好 3D 观察的相关函数，这里使用 gluPerspective()设置观察的视景体大小，使用 gluLookAt()设置相机的相关参数。

为了实现通过键盘和鼠标和观察视角进行交互，我们使用 glutKeyboardFunc()以及 glutPassiveMotionFunc()和自定义的回调函数。前一个 Func 处理的是普通的按键消息，有三个形参，第一个表示按下的键的 ASCII 码，其余两个提供当按键按下时鼠标的位置，相对于当前客户窗口的左上角。而第二个 Func 处理的是函数响应鼠标没有被按下去的时候移动鼠标的情形。

```
void timerFunc(int value) {
    if (earthAngle >= 360.0)
        earthAngle = 0;
    if (moonAngle >= 360.0)
        moonAngle = 0;
    if (JupiterAngle >= 360.0)
        JupiterAngle = 0;
    if (MarsAngle >= 360.0)
        MarsAngle = 0;

    earthAngle += earthstep;
    moonAngle += moonstep;
    JupiterAngle += jupiterstep;
    MarsAngle += Marsstep;

    glutPostRedisplay();

    glutTimerFunc(100, timerFunc, 1);

}实现公转动画效果的回调函数代码
```

对于键盘的交互处理，当按下相应的按键时，我们首先将 glMatrixMode 设置为 ModelView 模式，然后移动相机在世界空间坐标系中的位置，然后使用 glutPostRedisplay() 函数重绘画面，就可以实现通过按键漫游的效果，具体代码可以在工程文件源码的相应部分查看。

对于通过鼠标改变视角的效果，我们定义一个屏幕坐标系的中心点，当鼠标移动时，鼠标的坐标改变，同时与坐标系中心点的坐标的距离也发生了改变。我们将距离差的改变通过乘以一定的系数映射到 gluLookAt 函数观察点坐标，通过改变 gluLookAt 函数观察点的坐标使得实现移动鼠标能够有改变视角的效果。相关代码如下：

```
void MouseMove(int x, int y) {

    const int x0 = 700;
    const int y0 = 300;

    int dx = x - x0;
    int dy = y0 - y;

    xref += 0.01*dx;
    yref += 0.01*dy;

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(eyex, eyey, eyez, xref, yref, zref, Vx, Vy, Vz);
    glutPostRedisplay();
}
```

背景贴图和纹理的实现

背景贴图的实现

使用 glDrawPixels()函数来实现绘制背景贴图的绘制，把内存中的一些数据作为像素数据进行绘制。我们从 BMP 文件中读取像素数据，然后使用这个函数绘制到屏幕上，在程序开始时就读取该文件，当得到图像的大小后，然后进行绘制。

我们使用的是 Windows 平台上的 24 位色，不使用压缩的 BMP 文件。通过网上搜索相关资料可知，Windows 使用的 BMP 文件，在开始处有一个文件头，大小为 54 字节，保存了包括文件格式标识，颜色数，图像大小，压缩方式等信息。我们需要读取的主要是文件头

中的“大小”这一信息，图像的宽度和高度都是一个 32 位整数，在文件中的地址分别为 0x0012 和 0x0016。

另外值得注意的是，BMP 文件采用了一种“对齐”机制，每一行像素数据的长度若不是 4 的倍数，则填充一些数据使他是 4 的倍数。所以分配内存时图像每行的字节数要补充到 4 的倍数为止，再乘以图像的高度。否则会导致分配的内存空间不足，造成越界访问，代码：

```
//读写方式打开背景贴图的 bmp 文件
    FILE* pFile = fopen("universe.bmp", "rb");

    if (pFile == 0)

    {
        cout << "FILE open error!" << endl;
        Sleep(3000);
        exit(0);

    }
    // 移动到 0x0012 位置
    fseek(pFile, 0x0012, SEEK_SET);
    //读取宽度
    fread(&ImageWidth, sizeof(ImageWidth), 1, pFile);
    //读取长度
    fread(&ImageHeight, sizeof(ImageHeight), 1, pFile);

    //每行数据长度
    PixelLength = ImageWidth * 3;

    //修正 Length 使其为 4 的倍数
    while (PixelLength % 4 != 0)

        ++PixelLength;

    PixelLength *= ImageHeight;
    PixelData = (GLubyte*)malloc(PixelLength);
    if (PixelData == 0)

        exit(0);

    //读取文件头后的数据
    fseek(pFile, 54, SEEK_SET);
    fread(PixelData, PixelLength, 1, pFile);
    //关闭文件
    fclose(pFile);
```

绘制背景贴图的代码如下：

```
    glDisable(GL_DEPTH_TEST);          //关闭深度测试

    glDrawPixels(ImageWidth, ImageHeight, GL_RGB, GL_UNSIGNED_BYTE,
PixelData);    //绘制背景贴图
    glEnable(GL_DEPTH_TEST);
```

值得注意的是，绘制背景贴图之前需要先关闭深度测试。

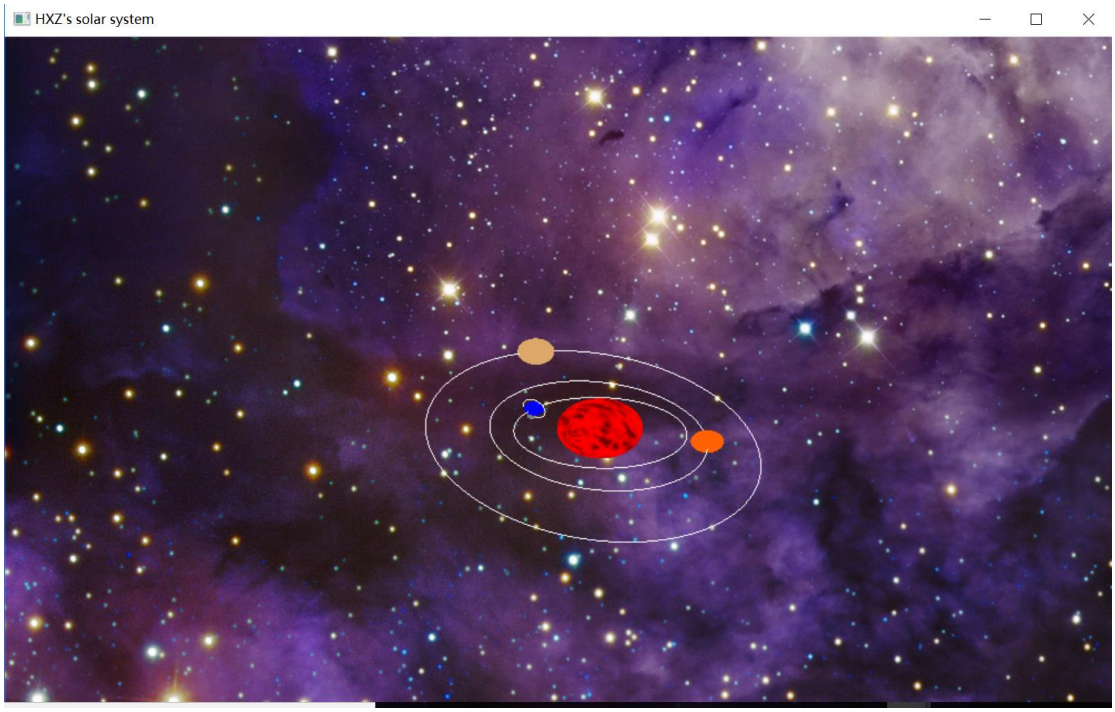
星球纹理的实现

这里我们使用了 SOIL 库(简易 OpenGL 图像库)来实现快速的读取图像文件，生成纹理，然后给指定的二次曲面贴上纹理，读取纹理部分的代码如下：

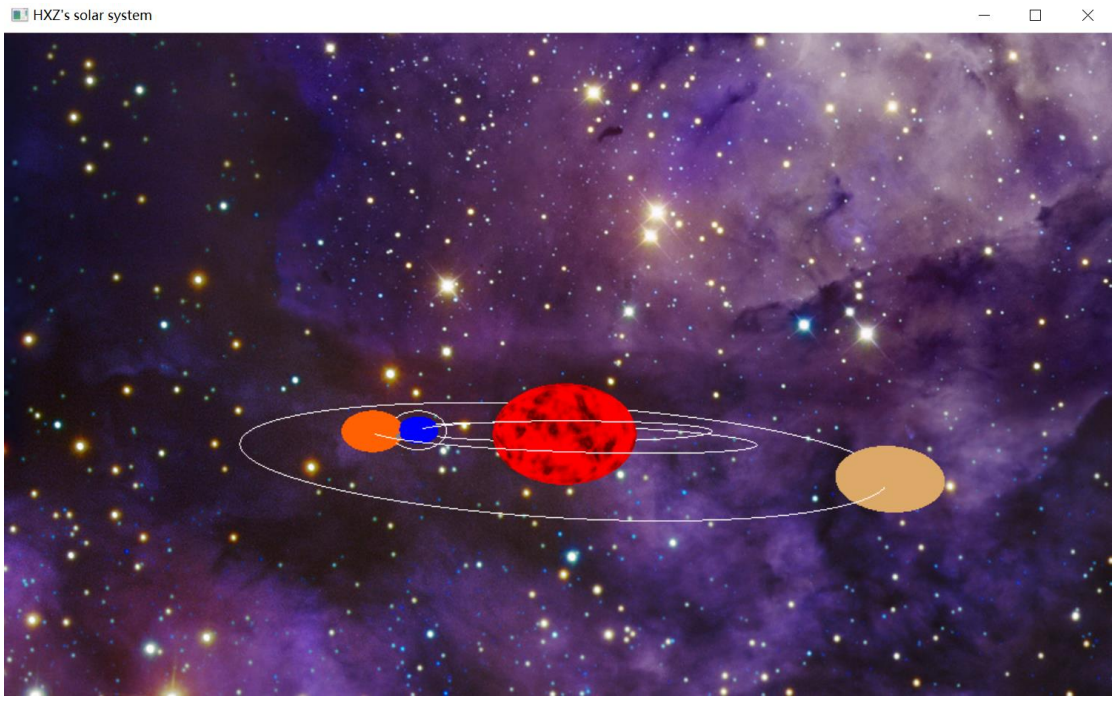
```
int sunHeight, sunWidth;    //纹理贴图的长与宽
    unsigned char *sun = SOIL_load_image("sun.bmp", &sunWidth,
&sunHeight, 0,SOIL_LOAD_RGB);     //使用 SOIL 库的载入纹理图片的函数
    glGenTextures(1, &texture_sun);   //生成一个纹理对象
    glBindTexture(GL_TEXTURE_2D, texture_sun);  //绑定该纹理对象
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, sunWidth, sunHeight, 0,
GL_RGB, GL_UNSIGNED_BYTE, sun); //生成太阳的纹理
    glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
    SOIL_free_image_data(sun);        //释放图像缓存
```

实验结果

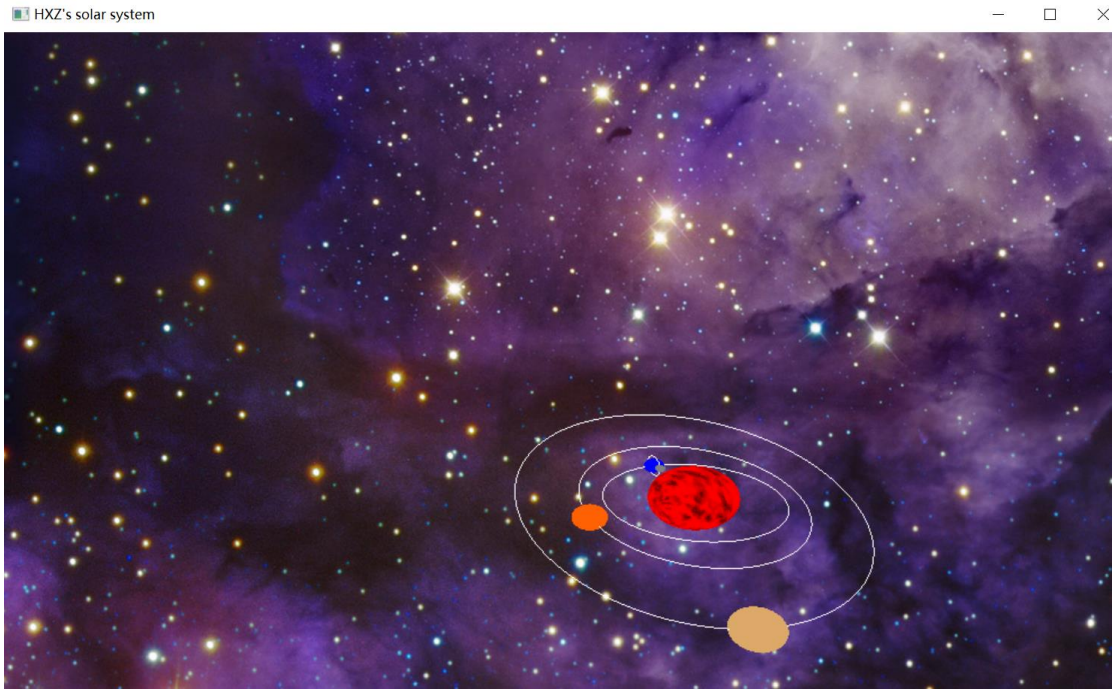
运行效果：



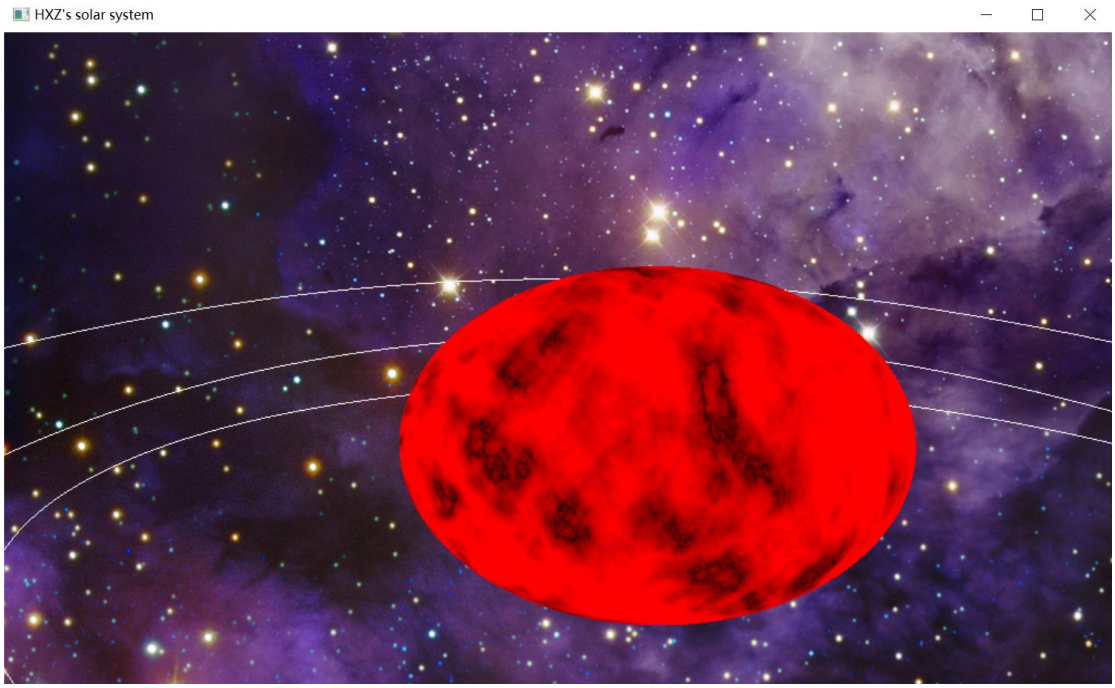
使用 WSADQE 键改变相机位置:



移动鼠标改变视角：



太阳表面纹理：



心得体会

这次实验是我们学习计算机图形学课程以来第一次上手实践的一个小型 project。其实，构造一个 solar system 需要运用到计算机图形学里的很多综合知识。本次实现的 solar system

也有很多不足的地方，比如依赖 SOIL 库从 bmp 图像文件直接导入的太阳纹理虽然步骤简便，但纹理看上去比较简单，细节比较差，下次真正学习纹理这一章时可能会考虑使用别的函数和方法。其次，没有处理其他行星和卫星的纹理。除此之外，没有在场景内布置光源和实现光线跟踪效果，使得这个太阳系缺乏真实性，在各行星绕太阳旋转的时候应该会出现阳面和阴面效果。最后，个人对通过移动鼠标改变视角的效果不太满意，本次实验只实现了观察点在一个二维平面上的改变，而非在一些 3D 游戏中的观察点可以随鼠标在三维空间内的效果。相关的操作可能可以借助 GLFW 库来实现，需要我进一步的学习和研究。

总体上来说，这次的 Assignment 加深了我对于老师上课所讲的很多知识点的理解，对我的计算机图形学的相关知识和编程能力水平来说是一次不小的提升。

参考资料

1. 三维空间中圆的参数方程——登山客的博客
   http://blog.sina.com.cn/s/blog_6496e38e0102vi7e.html
2. BMP 文件与像素操作(glReadPixels，glDrawPixels 和 glCopyPixels 应用举例)——psklf
   https://blog.csdn.net/u012515661/article/details/52046784
3. openGL 球面贴图的一种简单办法——evilkant
   https://www.cnblogs.com/evilkant/p/5978572.html
4. glPushMatrix()和 glPopmatirx()——tyxkzzf
   https://blog.csdn.net/tyxkzzf/article/details/40907273