作业2 实验报告

简介

在本次作业中，我们基于OpenGL模拟了太阳系的运动，包括地球、太阳以及其他恒星。模拟过程包含以下组件：

- 球体网格的生成
- 可交互相机（我们实现了键盘和鼠标事件）
- 基于B-Spline的行星轨迹和基于Bresenham的行星环绘制

我们使用了以下工具和第三方库：

- Glut (The OpenGL Utility Toolkit)：跨屏台得到OpenGL实用库，提供了I/O、窗口管理、上下文管理和事件管理功能。
- Visual Studio 2019/2022 和对应的 Windows SDK 及工具，SDK中包含了必要的GL.h等接口定义。

本次作业中，我们使用了OpenGL的兼容模式。

关键点

B-Spline

为了演示行星轨迹，我们实现了B-Spline的算法，通过使用分段的三次样条曲线，在保证C^2连续性的同时，构造样条曲线。需要提及的是，这需要至少4个输入点。

由数学知识可知，这样构造的曲线将位于控制点的闭包内，因而只要提供足够多的控制点，我们将得到收敛的光滑曲线。这足以表示行星的轨迹。

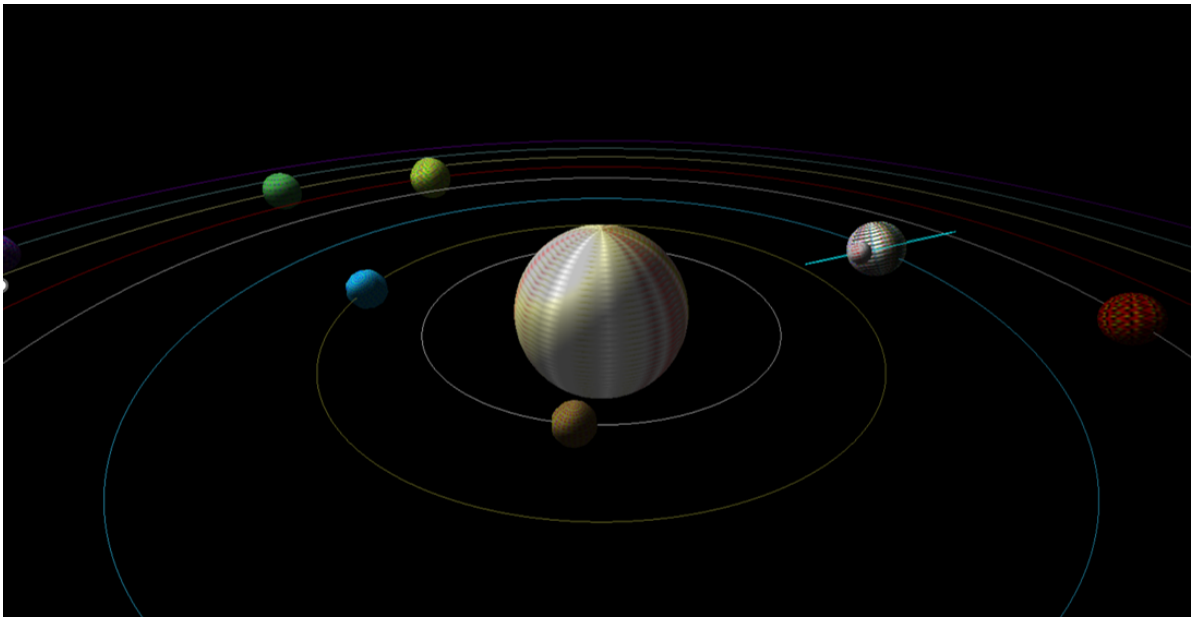
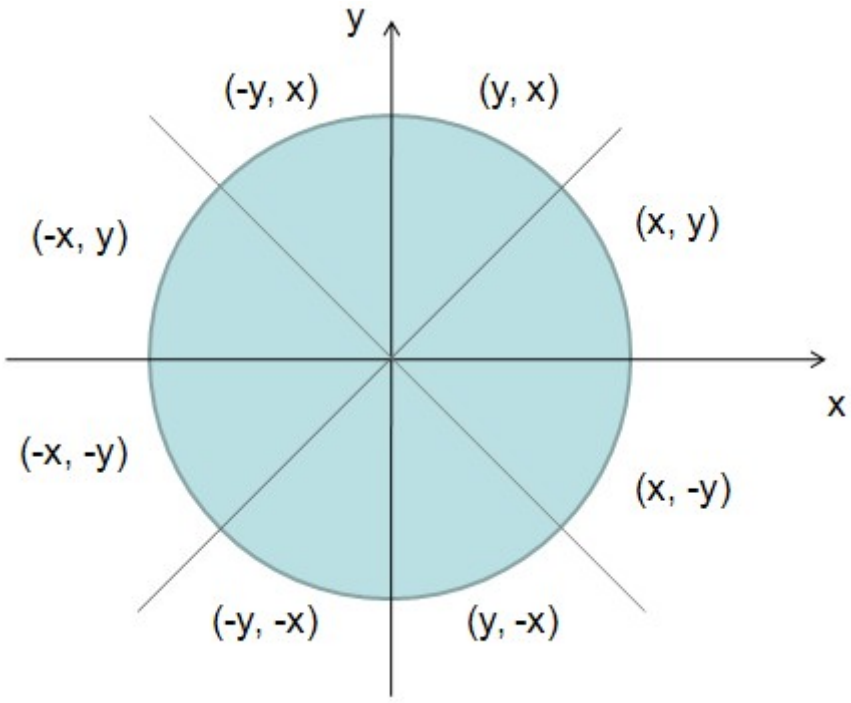


Image: The trajectories of the planets

Bresenham 算法

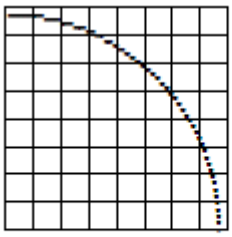
Bresenham 画圆算法又称中点画圆算法，其基本的方法是利用判别变量来判断选择最近的像素点，判别变量的数值仅仅用一些加、减和移位运算就可以计算出来。为了简便起见，考虑一个圆心在坐标原点的圆，而且只计算八分圆周上的点，其余圆周上的点利用对称性就可得到。

基于圆的**“八对称性”**，如下图所示，

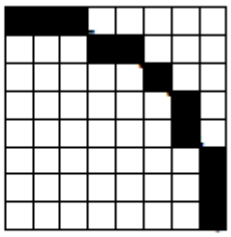


我们只需要知道圆上的一个点的坐标 (x, y)，就能得到另外七个对称点的坐标。

结果上，Bresenham 画圆算法用一系列离散的点来近似描述一个圆，如下图。



(a).圆弧曲线



(b). 圆弧的离散表示

Bresenham 画圆算法示意

关键代码

```
#pragma comment(lib, "legacy_stdio_definitions.lib")
extern "C" { FILE __iob_func[3] = { *stdin,*stdout,*stderr }; }

AUX_RGBImageRec* Images[10];//图片数组容器
GLuint ImagesIDs[11];  //索引
const char* szFiles[10] = {//各个材质贴图的相对路径
    ("../rec/Sun.bmp"),
    ("../rec/Mercury.bmp"),
    ("../rec/Venus.bmp"),
    ("../rec/Earth.bmp"),
    ("../rec/Mars.bmp"),
    ("../rec/Jupiter.bmp"),
    ("../rec/Saturn.bmp"),
```

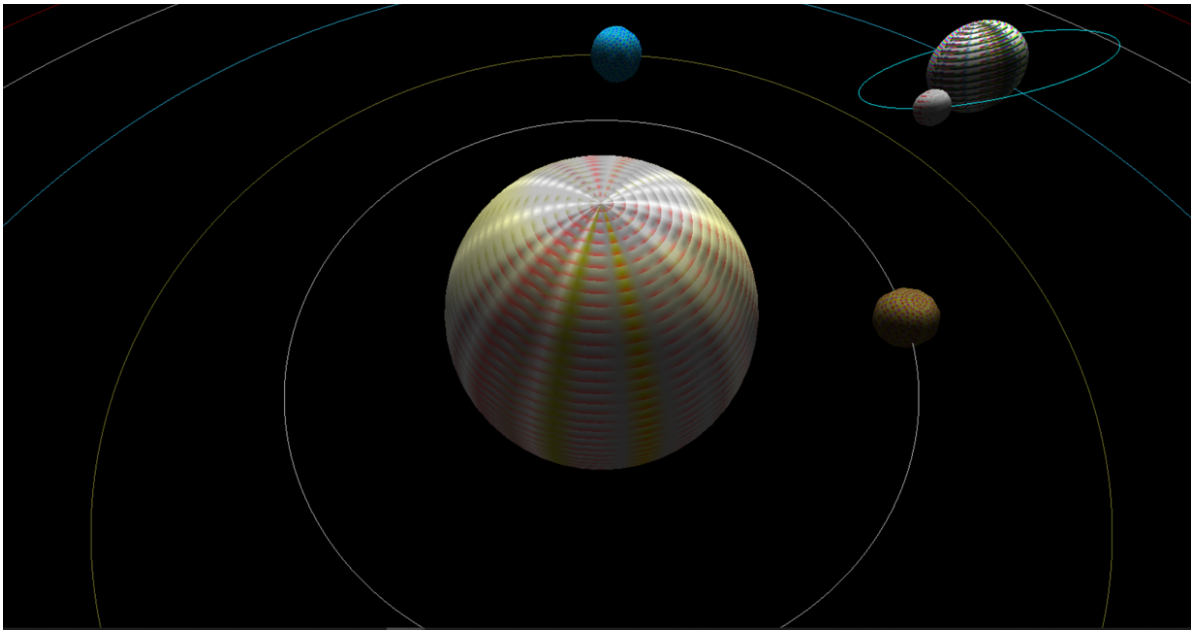
```
    ("../rec/Uranus.bmp"),
    ("../rec/Neptune.bmp"),
    ("../rec/Moon.bmp"), };
GLubyte* pImg; //opengl图片处理类
GLint iWidth, iHeight;//窗口大小

#define PI 3.1415926//圆周率
static float  year = 0, month = 0, day = 0, angle = 30;//初始化时间 以及旋转角

GLint W, H, width, height;//窗口长宽属性
float pox = 2, poy = 3, poz = 8;  //照相机的位置
GLint fovy = 60;//相机的视场角

void Init();//初始化函数
void OnDisplay();//opengl绘图函数
void OnReshape(int, int);//刷新函数
void OnTimer(int);//定时器函数，定时刷新
void DrawCircle(GLdouble);//绘制星球轨道
void gltDrawSphere(GLfloat, GLint, GLint);//绘制星球
void keyDownEvent(unsigned char key, int x, int y);//按键响应函数

int main(int argc, char* argv[]) {//程序主函数
    glutInit(&argc, argv);//opengl初始化函数
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);//创建窗口，指定显示模
式
    glutInitWindowSize(1000, 800);//窗口大小
    glutInitWindowPosition(100, 100);//窗口位置
    glutCreateWindow("太阳系");//窗口标题
    Init();//调用初始化函数
    glutReshapeFunc(OnReshape);//绑定OpenGL刷新函数
    glutDisplayFunc(&OnDisplay);//绑定OpenGL绘制函数
    glutKeyboardFunc(&keyDownEvent);//绑定OpenGL按键函数
    glutTimerFunc(100, OnTimer, 1);//绑定OpenGL刷新函数
    glutMainLoop();//进去opengl主循环
    return 0;
}

void OnReshape(int w, int h) {
    W = w; H = h;//传入窗口大小
    width = W; height = H;//窗口大小
    glViewport(0, 0, w, h);//设计opengl视口大小
    glMatrixMode(GL_PROJECTION);//将当前矩阵指定为投影矩阵然后把矩阵设为单位矩阵：
    glLoadIdentity();//重置当前指定矩阵为单位矩阵
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1, 20);//用新矩阵替换单位矩阵
    glMatrixMode(GL_MODELVIEW);//对模型视景矩阵操作
    glLoadIdentity();//重置当前指定矩阵为单位矩阵
}

void Init() {
    glClearColor(0.0, 0.0, 0.0, 0.0);//初始化背景颜色为黑色
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);//清空窗口绘制
    for (int i = 0; i < 10; i++) {//循环取各个纹理图片
        glGenTextures(1, &ImagesIDs[i]);  //生成纹理
        glBindTexture(GL_TEXTURE_2D, ImagesIDs[i]);   //绑定纹理

        //vs2019下必须进行转化
        WCHAR wfilename[256];//字符格式转化为256大小的数组
```

```
        memset(wfilename, 0, sizeof(wfilename));//初始化一串连续的内存，大小为
wfilename
        //该函数映射一个字符串到一个宽字符（unicode）的字符串
        MultiByteToWideChar(CP_ACP, 0, szFiles[i], strlen(szFiles[i]) + 1,
wfilename, sizeof(wfilename) / sizeof(wfilename[0]));

        Images[i] = auxDIBImageLoad(wfilename);  //加载图片

        iWidth = Images[i]->sizeX;//设置图片宽度
        iHeight = Images[i]->sizeY;//设置图片长度
        pImg = Images[i]->data;
        //装载纹理
        glTexImage2D(GL_TEXTURE_2D, 0, 3, iWidth, iHeight, 0, GL_RGB,
GL_UNSIGNED_BYTE, pImg);

        //纹理过滤、纹理裁剪
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        //纹理环境
        glTexEnvf(GL_TEXTURE_2D, GL_TEXTURE_ENV_MODE, GL_REPLACE);

        //启动2D纹理
        glEnable(GL_TEXTURE_2D);
        glEnable(GL_DEPTH_TEST);
    }
    glEnable(GL_LIGHTING);                              // 开启光照效果
    glEnable(GL_LIGHT0);
}

void OnTimer(int value) {
    day += angle;//模拟各个星球公转
    glutPostRedisplay();//重新刷新页面绘图
    glutTimerFunc(100, OnTimer, 1);//再次执行下一帧
}

//轨道函数
void DrawCircle(GLdouble R) {
    glPushMatrix();//指定为当前矩阵
    glRotatef(90.0, 1.0, 0.0, 0.0);//初始化旋转量
    glColor3f(1.0, 1.0, 1.0);//初始化颜色
    glBegin(GL_LINE_LOOP);//开始换闭环线
    for (int i = 0; i < 1000; i++) {
        GLdouble angle1 = i * 2 * PI / 1000;
        glVertex2d(R * cos(angle1), R * sin(angle1));//不停计算下一位置的坐标，绘制线
条，1000为圆的光滑度
    }
    glEnd();
    glPopMatrix();//结束绘画，刷新回主矩阵
}

//绘制星球
void gltDrawSphere(GLfloat fRadius, GLint iSlices, GLint iStacks) {
    GLfloat drho = (GLfloat)(3.141592653589) / (GLfloat)iStacks;//斯皮尔曼相关系数
    GLfloat dtheta = 2.0f * (GLfloat)(3.141592653589) / (GLfloat)iSlices;//计算时
间消耗度
```

```
    GLfloat ds = 1.0f / (GLfloat)iSlices;//位置量
    GLfloat dt = 1.0f / (GLfloat)iStacks;//位置量
    GLfloat t = 1.0f;
    GLfloat s = 0.0f;
    GLint i, j;

    GLfloat rho=0.0f ;//初始化各个斯皮尔曼相关系数
    GLfloat srho = 0.0f;
    GLfloat crho = 0.0f;
    GLfloat srhodrho = 0.0f;
    GLfloat crhodrho = 0.0f;

    for (i = 0; i < iStacks; i++) {
        rho = (GLfloat)i * drho;
        srho = (GLfloat)(sin(rho));
        crho = (GLfloat)(cos(rho));
        srhodrho = (GLfloat)(sin(rho + drho));
        crhodrho = (GLfloat)(cos(rho + drho));

        glBegin(GL_TRIANGLE_STRIP);
        s = 0.0f;
        for (j = 0; j <= iSlices; j++) {
            GLfloat theta = (j == iSlices) ? 0.0f : j * dtheta;
            GLfloat stheta = (GLfloat)(-sin(theta));
            GLfloat ctheta = (GLfloat)(cos(theta));

            GLfloat x = stheta * srho;
            GLfloat y = ctheta * srho;
            GLfloat z = crho;

            glTexCoord2f(s, t);
            glNormal3f(x, y, z);
            glVertex3f(x * fRadius, y * fRadius, z * fRadius);

            x = stheta * srhodrho;
            y = ctheta * srhodrho;
            z = crhodrho;
            glTexCoord2f(s, t - dt);
            s += ds;
            glNormal3f(x, y, z);
            glVertex3f(x * fRadius, y * fRadius, z * fRadius);
        }
    }

    glEnd();

    t -= dt;
}

void keyDownEvent(unsigned char key, int x, int y)
{
    // 处理键盘事件，略
}

void OnDisplay() {

    glColor3f(1.0, 0.0, 0.0);//初始化颜色
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);//清空画布
```

```
year = day / 365;//当前年的时间
month = day / 30;//当前月的时间

glMatrixMode(GL_PROJECTION);//指定当前矩阵绘制矩阵
glLoadIdentity(); //重置当前指定矩阵为单位矩阵
gluPerspective(fovy, (GLfloat)W / (GLfloat)H, 2, 60.0);//矩阵变换

glMatrixMode(GL_MODELVIEW);//指定当前矩阵绘制矩阵
glLoadIdentity();//重置当前指定矩阵为单位矩阵
gluLookAt(pox, poy, poz, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//相机位置

//太阳光
GLfloat sun_mat_ambient[4] = { 1.0,1.0,1.0,0.0 };
GLfloat sun_mat_diffuse[4] = { 1.0,1.0,0.5,1.0 };
GLfloat sun_mat_specular[4] = { 1.0,1.0,1.0,1.0 };
GLfloat sun_mat_shininess[] = { 10.0 };
GLfloat sun_mat_emission[4] = { 0.1,0.1,0.1,1.0 };
GLfloat mat_ambient[4] = { 0.2,0.2,0.2,1.0 };
GLfloat mat_diffuse[4] = { 1.0,1.0,1.0,1.0 };
GLfloat mat_specular[4] = { 0.5,0.5,0.5,1.0 };
GLfloat mat_shininess[] = { 5.0 };
GLfloat light_position[] = { -10.0,10.0,0.0, 1.0 };//光源位置
glLightfv(GL_LIGHT0, GL_POSITION, light_position);     // 创建光源
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, sun_mat_ambient);   // 材质设定
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, sun_mat_diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, sun_mat_specular);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, sun_mat_shininess);
glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, sun_mat_emission);

//绘制太阳
glPushMatrix();
glLightfv(GL_LIGHT0, GL_POSITION, light_position);             // 创建光源
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, sun_mat_ambient);   // 材质设定
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, sun_mat_diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, sun_mat_specular);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, sun_mat_shininess);
glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, sun_mat_emission);
glBindTexture(GL_TEXTURE_2D, ImagesIDs[0]);//添加纹理
glRotatef((GLfloat)month, 0.0, 1.0, 0.0);  //太阳自转
glRotatef(90.0, -1.0, 0.0, 0.0);//设置当前时间的旋转量
gltDrawSphere(1.0, 50, 50);//画星球
glPopMatrix();//结束绘画，刷新回主矩阵

//绘制水星
glPushMatrix();
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);   // 材质设定
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
DrawCircle(2.0);//绘制水星的轨道
glRotatef((GLfloat)month / 2, 0.0, 1.0, 0.0);  //水星围绕太阳转
glTranslatef(2, 0.0, 0.0);  //向x轴右移，让其在轨道上
glBindTexture(GL_TEXTURE_2D, ImagesIDs[1]);//添加纹理
glRotatef((GLfloat)month, 0.0, 1.0, 0.0);//公转
glRotatef(90.0, -1.0, 0.0, 0.0);//自转
gltDrawSphere(0.2, 10, 10);//画星球
glPopMatrix(); // 结束绘画，刷新回主矩阵，之后以相同方法绘制不同星球。
```

Results



Debates

1. OpenGL为一个典型的状态机，包含诸如裁剪模式、绘制模式、Alpha通道语义等诸多开关，同时具有栈结构，例如变换栈。OpenGL兼容模式（Compatibility Profile）为较陈旧的接口，使用固定管线，在渲染前的准备过程中，数据以流式从Client发送到Server端，效率较低，同时光照等关键特性也固定到了管线内，不便于定制使用。虽然有简单易用、模型简单的特点，但以不适合图形学发展和工程需要。
2. 在渲染过程中，一个关键点是从世界坐标系到相机的View Volumn的变换，这一变换使用仿射坐标可以表达为线性映射，因而可以通过矩阵表达。一般实现中使用三个变换矩阵相乘，分别为投影矩阵、View矩阵、Model矩阵，分别对应到View Volumn的变换、相机位置和姿态决定的变换、模型自身的刚性变换。
3. 绘制轨道过程中，其实际形状是不符合物理的，这也启示了我们下一步的改进方向：使用ODE算法，结合引力公式进行实际模拟。事实上，这也是图形学和物理模拟目的不同的体现 - 前者需要尽可能提高计算效率、降低计算消耗的基础上保证渲染效果，后者则期望更贴近物理实际，并指导物理应用。
4. GLUT 其实是 OpenGL Utility Toolkit 的缩写，它是一个处理 OpenGL 程序的工具库，主要负责处理与底层操作系统的调用及 I/O操作。使用 GLUT 可以屏蔽掉底层操作系统 GUI 实现上的一些细节，仅使用 GLUT 的 API 即可跨平台的创建应用程序窗口、处理鼠标键盘事件等等。
5. 在实验中发现，由于各图形卡的OpenGL接口实现不同，其表现亦有差异。