**《计算机图形学实验》综合实验报告**

**题目 OpenGL对三维图形的渲染**

**学 号 20201120508**

**姓 名 李进林**

**指导教师 钱文华**

**日 期 2022/6/12**

摘要：

计算机图形学是计算机科学领域最为年轻和活跃的学科之一，它同时也是一门交叉性很强的学科，不仅与数学、光学、结构设计等非计算机科学范畴的学科有紧密联系，还与数字图像处理、计算几何、模式识别等计算机科学领域的二级学科不可分割。其中三维图形又是计算机图形学研究的最主要的部分，OpenGL又对图形学领域提供了强大的库函数支持，来方便我们的开发使用，本次实验也是基于OpenGL来对三维图形进行渲染，用光线跟踪、光照明模型、纹理贴图、纹理映射等算法为三维图形添加纹理、色彩、光照、阴影、透明等效果。

关键词：计算机图形学、三维图形、OpenGL。

目录

[1. 实验背景 7](#_Toc105876825)

[2. 实验内容 7](#_Toc105876826)

[3. 开发工具及实现目的 7](#_Toc105876827)

[4. 关键代码及算法理论 7](#_Toc105876828)

[5. 实验心得及小结 11](#_Toc105876829)

[参考文献： 11](#_Toc105876830)

[附录： 11](#_Toc105876831)

1. 实验背景

渲染是三维计算机图形学中的最重要的研究课题之一，是模型成像的过程。渲染在建筑，游戏，模拟，电影或电视视觉效果以及设计可视化的各种场景中都被充分的使用，而每个行业都采用不同的特征和技术平衡。而三维渲染是指在[电子计算机](https://zh.m.wikipedia.org/wiki/%E7%94%B5%E5%AD%90%E8%AE%A1%E7%AE%97%E6%9C%BA)上将[三维模型](https://zh.m.wikipedia.org/wiki/%E4%B8%89%E7%BB%B4%E6%A8%A1%E5%9E%8B)转换为[二维计算机图形](https://zh.m.wikipedia.org/wiki/%E4%BA%8C%E7%BB%B4%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2)的[三维计算机图形](https://zh.m.wikipedia.org/wiki/%E4%B8%89%E7%BB%B4%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2)制图过程，也就是从准备的场景创建实际的二维景象或动画的最后阶段。其中渲染的过程为：首先，必须定位三维场景中的摄像机，这和真实的摄影是一样的。一般来说，三维软件已经提供了四个默认的摄像机，那就是软件中四个主要的窗口，分为顶视图、正视图、侧视图和透视图。我们大多数时候渲染的是透视图而不是其它视图，透视图的摄像机基本遵循真实摄像机的原理，所以我们看到的结果才会和真实的三维世界一样，具备立体感。之后，为了体现空间感，渲染程序要做一些“特殊”的工作，就是决定哪些物体在前面、哪些物体在后面和那些物体被遮挡等。渲染程序通过摄像机获取了需要渲染的范围之后，就要计算光源对物体的影响，这和真实世界的情况又是一样的。许多三维软件都有默认的光源，否则，我们是看不到透视图中的着色效果的，更不用说渲染。因此，渲染程序就是要计算我们在场景中添加的每一个光源对物体的影响。和真实世界中光源不同的是，渲染程序往往要计算大量的辅助光源。在场景中，有的光源会照射所有的物体，而有的光源只照射某个物体，这样使得原本简单的事情又变得复杂起来。在这之后，是使用深度贴图阴影还是使用光线追踪阴影来渲染往往取决于在场景中是否使用了透明材质的物体计算光源投射出来的阴影。另外，使用了面积光源之后，渲染程序还要计算一种特殊的阴影－－软阴影（只能使用光线追踪），场景中的光源如果使用了光源特效，渲染程序还将花费更多的系统资源来计算特效的结果，特别是体积光，也称为灯光雾，它会占用代量的系统资源，使用的时候一定要注意。在这之后，渲染程序还要根据物体的材质来计算物体表面的颜色，材质的类型不同，属性不同，纹理不同都会产生各种不同的效果。而且，这个结果不是独立存在的，它必须和前面所说的光源结合起来。如果场景中有粒子系统，比如火焰、烟雾等，渲染程序都要加以“考虑”。

1. 实验内容

利用Visual Studio, OpenGL,等工具，实现三维图形渲染，自定义三维图形，三维图形不能仅仅是简单的茶壶、球体、圆柱体、圆锥体等图形，渲染过程须加入纹理、色彩、光照、阴影、透明等效果，可采用光线跟踪、光照明模型、纹理贴图、纹理映射等算法。

1. 开发工具及实现目的

开发工具：Visual Studio, OpenGL。

目的：用相关开发工具画出一个球体，并运用光线跟踪、光照明模型、纹理贴图、纹理映射等算法为将球体渲染成地球，并添加纹理、色彩、光照、阴影、透明等效果。

1. 关键代码及算法理论

fopen(filename, "rb"),以只读写的方式打开二进制文件，调用该函数主要是为了将图片信息文件打开；

glGenTextures(1, &textures)，根据纹理参数返回1个纹理索引，根据纹理参数返回n个纹理名称，纹理名称集合不必是一个连续的整数集合，和glBindTexture(GL\_TEXTURE\_2D, texture)一起搭配使用，其函数功能为绑定纹理，在用GL渲染的时候，纹理是很常见的东西，使用纹理之前，必须执行这句命令为你纹理分配一个ID（即函数的第二个参数），然后绑定这个纹理，加载纹理图像，这之后，这个纹理才可以使用；

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE)，指定纹理贴图与材质的混合模式，前面两个参数时固定的，第三个参数可以是GL\_MODULATE、GL\_DECAL、GL\_BLEND或GL\_REPLACE之一，都是颜色组合操作类型，用于产生特定的绘制效果。

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, width, height, 0, GL\_BGR\_EXT, GL\_UNSIGNED\_BYTE, data); //根据指定的参数，生成一个2D纹理，纹理将指定纹理图像的一部分映射到纹理化为活动的每个图形基元上。当前片段着色器或顶点着色器使用内置纹理查找函数时，纹理处于活动状态。定义纹理图像时需要调用glTexImage2D，

glRotatef(-90, 1.0, 0.0, 0.0),将物体旋转，第一个参数为旋转角度，后面三个参数为对应xyz轴的布尔值变量，及本例子中为绕着x轴顺时针旋转90°，函数底层原理为三维几何变换当中的旋转的实现，及用相应的变换矩阵与规范化齐次坐标相乘而得到；

gluSphere(qobj, 1.5, 60, 60)，函数绘制以原点为中心的给定半径的球体，第一个参数为四元对象，第二个参数为球体的半径，第三个参数为球体在z 轴周围的细分数 (类似于经度线) ，第四个参数为沿 z 轴的细分数 (类似于纬度) 线。

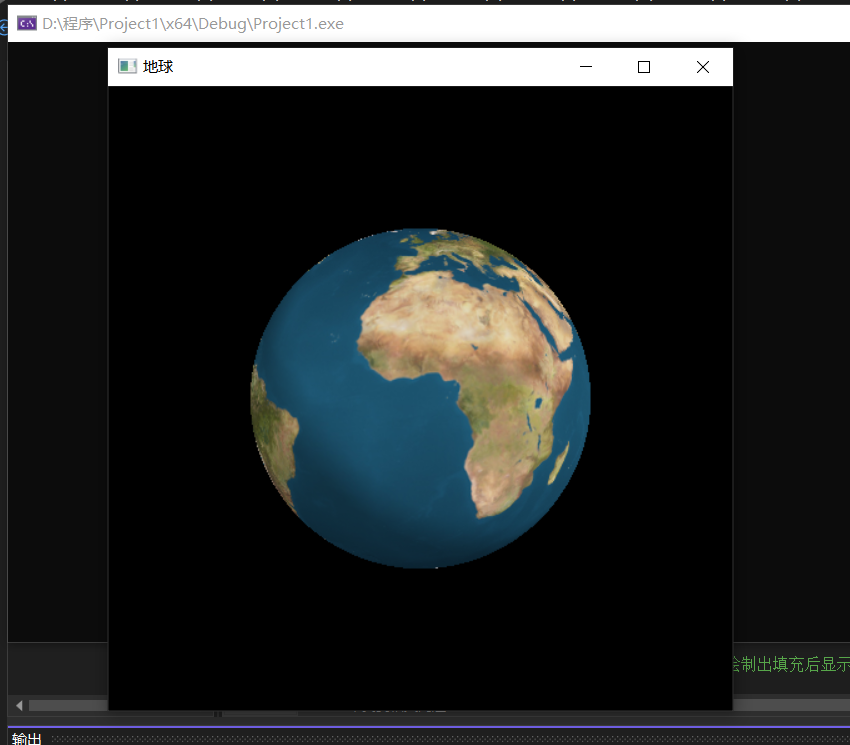
glViewport(0, 0, (GLsizei)w, (GLsizei)h);使视区大小保持与窗口的显示区域大小一致， 在默认情况下，视口被设置为占据打开窗口的整个像素矩形，窗口大小和设置视口大小相同，所以为了选择一个更小的绘图区域，就可以用glViewport函数来实现这一变换，在窗口中定义一个像素矩形，最终将图像映射到这个矩形中。例如可以对窗口区域进行划分，在同一个窗口中显示分割屏幕的效果，以显示多个视图。

gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);设定观察视点和观察方向，函数设定了观察视点，和观察方向，函数原型：gluLookAt(GLdoble eyex,GLdouble eyey,GLdouble eyez,GLdouble centerx,GLdouble centery,GLdouble centerz,GLdouble upx,GLdouble upy,GLdouble upz);它选取 eyex,eyey,eyez做为原点（观察坐标系的坐标（用世界坐标系来表示的）），然后centerx, centery,centerz　指定了观察方向（ｚ轴的反方向），upx,upy,upz　指定了ｙ轴的正方向的近似方向（它不一定和ｚ轴方向正交，但可以通过相关的运算，找到正交的Ｙ轴的正方向从而也可以找到ｘ轴的正方向．主要就是通过向量的点乘和叉乘来计算的））． 使用了gluLookAt函数，实际上是建立了从世界坐标系到观察坐标系的转换的矩阵，后面的投影变换将在观察坐标系下进行。

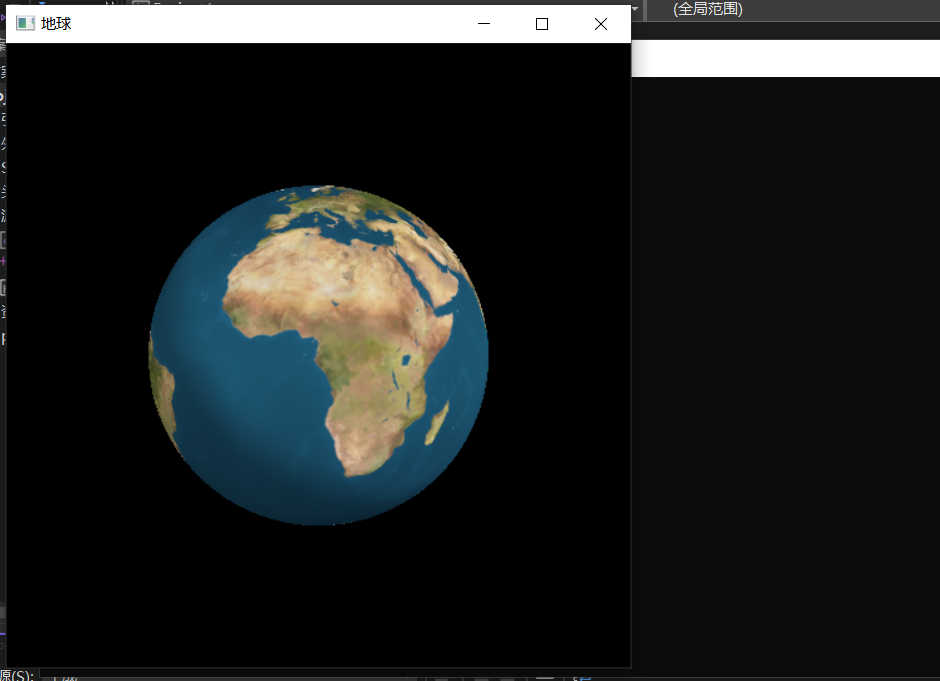
glMatrixMode(GL\_PROJECTION); 使用投影矩阵栈，准备设定投影矩阵，这个函数其实就是对接下来要做什么进行一下声明，参数有GL\_PROJECTION 投影, GL\_MODELVIEW 模型视图, GL\_TEXTURE [纹理](https://so.csdn.net/so/search?q=%E7%BA%B9%E7%90%86&spm=1001.2101.3001.7020)，本程序用到投影，就是把物体投影到一个平面上，就像我们照相一样，把3维物体投到2维的平面上。

gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 0.5, 30.0);定义视景体,根据四个参数计算用以投影变换的透视投影矩阵，可以简单的理解为相机的内在镜头参数，第一个参数定义可视角的大小，若值小，表示从相机（人眼）出发的光线的角度小，此时同等距离下，可观察到的视野范围较小，反之则大。从物体显示在画板的大小来反映，若值较大，则物体在画板中所占比例就较少，看起来比较小，反之则显示比较大。第二个参数定义物体显示在画板上的x和y方向上的比例。若值小于1，则物体显示出来比实际更高，若值大于1，显示出来比实际看起来更宽，若为1，会按实际反应长宽比。第三个参数定义距离相机（人眼）最近处物体截面相距的距离；这个值越大，表示观测点距离物体距离越远，看起来物体就比较小，反之则比较大。如果物体运动到距离观测点的距离小于了设定的值,则物体不会被绘制在画板上；第四个参数定义可观测到的物体的最远处截面相距相机的距离，如果物体运动到距离观测点的距离大于了设定的值则物体不会被绘制的画板上。

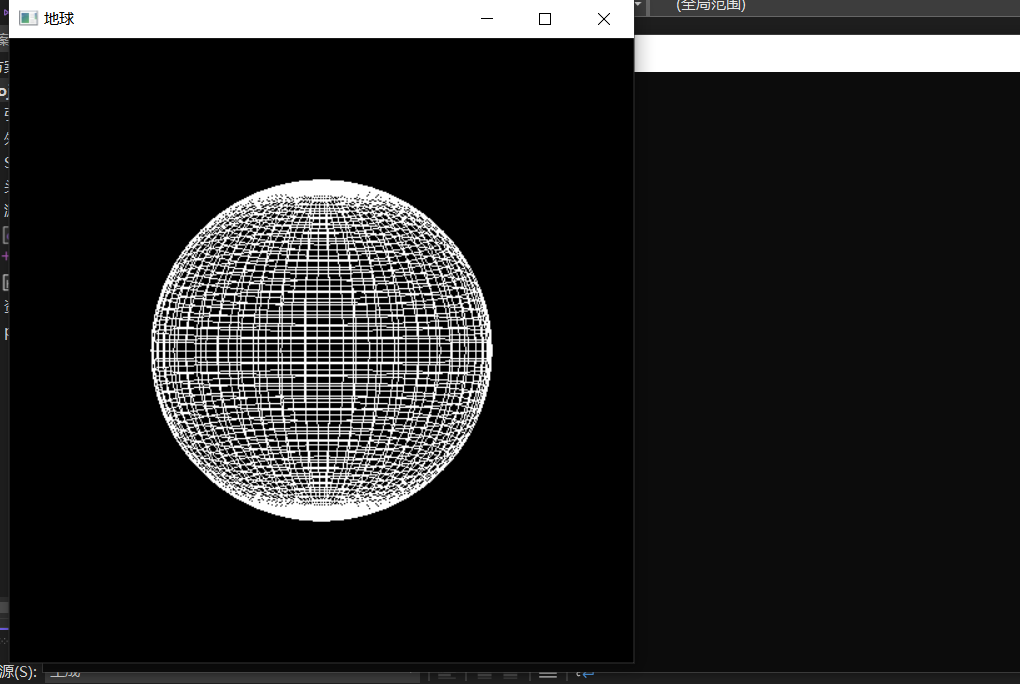
运行结果图为：



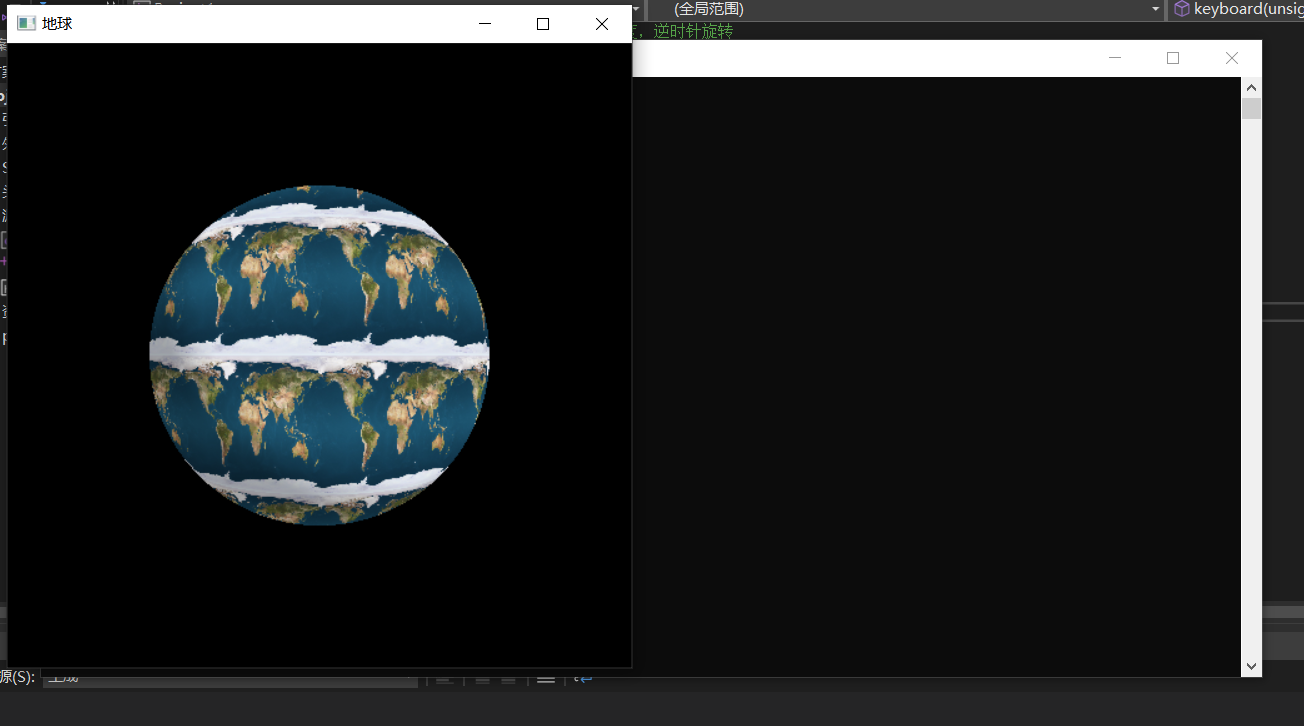
顺时针旋转10度后结果图为：



线框图为：



填充图为：



1. 实验心得及小结

本次实验我完成了一个通过纹理贴图实现的地球，通过键盘上的R,Z,L,F四个按键可以对地球进行操作，注意是大写状态，其中R是使地球顺时针旋转10度，Z是使地球逆时针旋转10度，L是出现地球线框图，F是出现填充纹理图。本次实验是一个比较综合的实验，将我们本学期学到的很多知识都结合在了一起，例如纹理贴图、光照、线框图、键盘交互等等知识点，和之前实验有明显的区别，之前的实验是完成单个知识点即可，这次的复合了许多知识点，在实验过程中我也遇到了很多问题，程序不是报错就是一直出bug，这也让我深刻的意识到图形学中的很多知识点并不是独立分开的，很多知识点都是有机的串联在一起，我们要有总体的知识框架，弄清楚我们要干什么，要用到哪些知识点，OpenGL中的哪些函数可以帮助我们实现我们想要的功能，将这些弄清楚之后我们来写程序就很顺畅，也能在我们实现基础功能后，我们能在这些基础功能上添加一些更复杂的功能。本次实验另一个深刻体会是遇到问题也不要担心，毕竟人生总是在经历各种磨难，这些磨难也是在我们变得越来越强。此次我在网上看了很多人写的博客以及一些相关书籍，才将我的问题逐一化解，让我受益匪浅，我也领悟到OpenGL以及计算机图形学相关知识领域的魅力，加深了我对图形学相关知识的理解程度，我也深刻认识到，我们计算机专业讲究的理论与实践结合的学习方法一定要贯彻落实，不能光说不练，一定要在学好理论理论知识的基础上，多动手敲代码来验证相关的理论知识，这样我们才有进步，才会更加深刻的理解理论知识。虽然在学习的过程中走过很多弯路,遇到很多挫折与问题，但回头想想，这些挫折与问题也是人生的必经路，遇到时要有耐心，迎难而上，找出最终的解决办法，这样才会有进步。在今后的学习中，我需要不断努力，主动探索，主动学习，为将来有着更深的学术造诣而努力。以后想继续深入相关领域学习更广的知识，将来也想在图形学相关领域有着更高的造诣。

参考文献：

[1]禹仁贵,任磊,王宁.基于OpenGL的纹理映射的实现与应用[J].西南民族大学学报：自然科学版,2008,34(6):1261-1263.

[2]兰一麟涛,钱伟,田明银.基于OpenGL的三维纹理贴图绘制技术研究与实现[J].甘肃科技,2015,31(22):32-34.

[3]童立靖,陈静.OpenGL纹理函数应用方法研究[J].软件导刊,2015,14(07):33-36.

[4]杨键,张敏.OpenGL中的光照技术研究[J].软件导刊,2011,10(04):84-86.

附录：

#include <windows.h>

#include <GL/freeglut.h>

#include <stdlib.h>

#include<stdio.h>

#include <io.h>

#include <process.h>

static int year = 0, day = 0;

static int longitude = 200, latitude = 200;

static GLuint texture[3] = { 0,0,0 };

static GLfloat xequalzero[] = { 1.0, 0.0, 0.0, 0.0 };

static GLfloat yequalzero[] = { 0.0, 1.0, 0.0, 0.0 };

static GLfloat slanted[] = { 1.0, 1.0, 1.0, 0.0 };

static GLfloat\* currentCoeff;

static GLenum currentPlane;

static GLint currentGenMode;

GLubyte\* readImg(char\* filename, int\* imagewidth, int\* imageheight)

{

int pixellength;

GLubyte\* pixeldata;

FILE\* pfile = fopen(filename, "rb");//打开文件

if (pfile == 0) exit(0);

//读取图像大小

fseek(pfile, 0x0012, SEEK\_SET);

fread(imagewidth, sizeof(\*imagewidth), 1, pfile);

fread(imageheight, sizeof(\*imageheight), 1, pfile);

//计算像素数据长度

pixellength = (\*imagewidth) \* 3;

while (pixellength % 4 != 0)pixellength++;

pixellength \*= (\*imageheight);

//读取像素数据

pixeldata = (GLubyte\*)malloc(pixellength);

if (pixeldata == 0) exit(0);

fseek(pfile, 54, SEEK\_SET);

fread(pixeldata, pixellength, 1, pfile);

fclose(pfile);//关闭文件

return pixeldata;

}

void LoadTexture(char\* filename, GLuint& texture)

{

GLubyte\* data;

GLint width, height;

FILE\* file;

//读文件

data = readImg(filename, &width, &height);

glGenTextures(1, &texture);//生成纹理的数量并存储存储纹理索引

glBindTexture(GL\_TEXTURE\_2D, texture);//绑定纹理

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);//指定纹理贴图与材质的混合模式

//线性滤图

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

//自动生成纹理坐标，使用球面映射

currentCoeff = xequalzero;

currentGenMode = GL\_OBJECT\_LINEAR;

currentPlane = GL\_OBJECT\_PLANE;

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1); //指定OpenGL如何从数据缓存区中解包图像数据

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, width, height, 0, GL\_BGR\_EXT, GL\_UNSIGNED\_BYTE, data); //根据指定的参数，生成一个2D纹理

glEnable(GL\_TEXTURE\_2D);//打开纹理映射

free(data); //释放纹理图像数据

}

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);//设置好清除颜色

glShadeModel(GL\_SMOOTH); //对多边形内部所有点的颜色通过对多边形顶点颜色插值产生

glEnable(GL\_DEPTH\_TEST);//开启深度测试

glDepthFunc(GL\_LESS);//目标像素与当前像素在z方向上值大小比较

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);//设置两面均为填充

GLfloat mat\_diffuse[] = { 1.0, 1.0, 1.0, 1.0 }; //漫放射系数，对应RGBA四个分量

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 }; //镜面放射系数,对应RGBA四个分量

GLfloat mat\_shininess[] = { 5.0 };//材料反射指数

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, mat\_diffuse);//设置环境光系数和漫反射光系数

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);//设置镜面光反射

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);//设置材料反射指数

glEnable(GL\_LIGHTING);// 开启光照计算

glEnable(GL\_LIGHT0);//开启0号光源，OpenGL中有8种光源

LoadTexture("D:\\develop\\earth.bmp", texture[0]);//创建并绑定纹理对象

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

GLfloat light\_position[] = { 1.0, 1.0, 1.0, 0.0 };//光源位置

GLfloat light\_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f }; // 环境强度

GLfloat light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 散射强度

GLfloat light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 镜面强度

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);//光源位置

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient); //光源强度,不写这三句将使用默认值，

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glColor3f(1.0, 1.0, 1.0); //启用光源后，不再使用颜色函数对物体着色

glMatrixMode(GL\_MODELVIEW); //使用模型视图矩阵栈保存后面的几何变换

//用二次曲面画球体

GLUquadricObj\* qobj;

qobj = gluNewQuadric();

glPushMatrix();

glRotatef((GLfloat)day, 0.0, 1.0, 0.0);

glRotatef(-90, 1.0, 0.0, 0.0); //看地球的侧面

glBindTexture(GL\_TEXTURE\_2D, texture[0]);//绑定纹理

gluQuadricTexture(qobj, GL\_TRUE);//纹理函数

gluSphere(qobj, 1.5, 60, 60);//二次曲面qobj

glPopMatrix();

glutSwapBuffers();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h); //使视区大小保持与窗口的显示区域大小一致

glMatrixMode(GL\_MODELVIEW);//使用模型视图矩阵栈

glLoadIdentity();//重置当前指定的矩阵为单位矩阵

gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//设定观察视点和观察方向

glMatrixMode(GL\_PROJECTION); //使用投影矩阵栈，准备设定投影矩阵

glLoadIdentity(); //初始化投影矩阵为单位矩阵

gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 0.5, 30.0);//定义视景体,根据四个参数计算用以投影变换的透视投影矩阵

}

void keyboard(unsigned char key, int x, int y)

{

switch (key) {

case 'Z':

day = (day + 10) % 360; //增加自转的角度，逆时针旋转

glutPostRedisplay();// 标记当前窗口并重新绘制出地球逆时针自转后的图

break;

case 'R':

day = (day - 10) % 360; //减少自转的角度，顺时针旋转

glutPostRedisplay();// 标记当前窗口并重新绘制出地球逆时针自转后的图

break;

case 'L':

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); //使用线框模型显示物体

glDisable(GL\_LIGHTING);//关闭光照计算

glDisable(GL\_TEXTURE\_2D);//关闭纹理映射

longitude = 20;

latitude = 16;

glutPostRedisplay();//标记当前窗口并重新绘制出地球的线框图

break;

case 'F':

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL); //使用多边形填充模型型显示物体

glEnable(GL\_LIGHTING);//开启光照计算

glEnable(GL\_TEXTURE\_2D);//打开纹理映射

glEnable(GL\_TEXTURE\_GEN\_S); //打开纹理坐标自动映射，纹理图像中的S相当于x坐标

glEnable(GL\_TEXTURE\_GEN\_T); //打开纹理坐标自动映射，纹理图像中的T相当于y坐标

longitude = 200;

latitude = 200;

glutPostRedisplay();// 标记当前窗口并重新绘制出填充后显示的图

break;

case 27:

exit(0);

break;

default:

break;

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);//初始化OpenGL

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);//设置显示参数(双缓存，RGB格式)

glutInitWindowSize(500, 500);//设置窗口尺寸

glutInitWindowPosition(100, 100);//设置窗口显示位置，在屏幕左上角像素值(100,100)处

glutCreateWindow("地球");//设置窗口名称

init();//程序初始化

glutDisplayFunc(display);//显示函数，参数display为上文中的函数

glutReshapeFunc(reshape);//当窗口尺寸改变时，图形比例不发生变化

glutKeyboardFunc(keyboard);//键盘交互事件函数

glutMainLoop();//多次调用glut函数

return 0;

}