**zhuanqiu 代码的详细注解**

**裴新凤**

**一、主函数的注解**

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

{

glutInit(&argc, argv);//对GLUT 进行初始化，这个函数必须在其它的 GLUT 使用之前调用一次，格式固定。

1．设置初始显示模式

第二行代码：glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE | GLUT\_DEPTH);

告诉GLUT库在创建窗口时使用RGB颜色模式（GLUT\_RGB），使用双缓冲区窗口（GLUT\_DOUBLE），窗口使用深度缓存（GLUT\_DEPTH）。

**注释**：RGB颜色模通过分别提供红、绿、蓝成分的浓度来指定颜色。双缓冲区窗口（GLUT\_DOUBLE），它使用了两个帧缓冲区，前台缓冲区用于显示图形，绘图命令用后台缓冲区进行绘图，绘图完毕后，在程序中交换这两个缓冲区指针就可以了，这种方法常用于产生动画的场合。使用深度缓存（GLUT\_DEPTH）用来产生远处的图形小，近处的图形大的三维效果。

2. 创建OpenGL窗口

glutInitWindowSize(640,480);//设置窗口大小，参数单位是像素

glutInitWindowPosition(10,10);//设置窗口位置，参数单位是像素

glutCreateWindow("zhuanqiu");//创建OpenGL窗口，参数将被作为窗口的标题。

3. 注册回调函数

glutReshapeFunc(resize);// 注册当前窗口的大小改变事件的回调函数

glutDisplayFunc(display);//注册当前窗口的显示回调函数

glutKeyboardFunc(key);//注册当前窗口的键盘回调函数

glutIdleFunc(idle);//注册空闲回调函数

glClearColor(0,0,1,1);// 设置清除窗口时的颜色为蓝色，即设定窗口内的背景色

4．开启各种需要的功能

glEnable用于开启各种功能。具体功能由参数决定，glDisable是用来关闭某种功能的。某种功能开启后一直有效，直到glDisable关闭该功能。

glEnable(GL\_CULL\_FACE);// 启用剔除操作效果

glCullFace(GL\_BACK);// 禁用多边形背面上的光照和颜色计算操作，即背面不显示。

**注释：**这两条命令完成了物体背面不显示。

**注释：**如不开启某功能，执行相应操作，则操作无效。

glEnable(GL\_DEPTH\_TEST);// 启用深度测试

glDepthFunc(GL\_LESS);// 指定深度缓冲比较值，深度值小于参考值，则通过。

**注释：**启用深度缓冲区的作用是根据当前图元的Z坐标来进行遮挡处理。如果该图元Z坐标小于其它图元的，说明它更靠近人的视线，就显示该图元，遮挡掉远处的图元。

glEnable(GL\_LIGHT0);//启用0号光源

glEnable(GL\_NORMALIZE);// 启用自动单位化法向量

glEnable(GL\_COLOR\_MATERIAL); //启用材质颜色

glEnable(GL\_LIGHTING);//启用光照

//设置0号光源

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

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

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

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

//设置材质

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);

glutMainLoop();//启动glut程序进入事件循环。

return EXIT\_SUCCESS;

}

5．OPENGL程序的执行过程

Main() 函数中的语句将被逐条执行，当执行glutMainLoop()时进入事件循环，一旦进入该循环，除非借助回调函数或一些外部干预（如按下一个“终止”键）才能结束该循环。

进入[GLUT](http://baike.baidu.com/subview/1082598/1082598.htm" \t "_blank)事件处理循环后，当“事件”（通常包括：鼠标事件，键盘事件，窗口事件（绘图，窗口尺寸的改变）发生时，系统自动调用对应的回调函数。

**注释：**

对OPENGL编程来说，开发人员主要任务是定义对具体事件响应方式的回调函数。

**6.** 在没有与用户相关的事件(如：键盘等事件）发生时，该程序的主要执行顺序如下：

从main()开始执行：

glutInit(&argc, argv);

……

glutCreateWindow("zhuanqiu") //创建一个窗口，**该函数的执行自动引发了以下两个函数的调用：**

1 resize(int width, int height)// 当前窗口大小改变事件的回调函数，

2 display(void)//显示回调，该函数用来画图

……//继续执行main()中的语句

glutMainLoop() //程序进入事件循环

idle(void) //空闲回调函数的执行调用glutPostRedisplay();标记当前窗口重新绘制,即调用display函数

display(void)//显示回调，

idle(void) //空闲回调函数的执行调用glutPostRedisplay();标记当前窗口重新绘制,即调用display函数

display(void)//显示回调，

......//重复调用idle(void),之后调用display(void)

**二、程序中定义的几个具体事件的回调函数**

static int slices = 16;//画二次曲面的经线数

static int stacks = 16; //画二次曲面的纬线数

**static void resize(int width, int height)//** 当前窗口大小改变事件的回调函数

{

const float ar = (float) width / (float) height;

glViewport(0, 0, width, height);//利用函数glViewport定义视区

glMatrixMode(GL\_PROJECTION);//投影矩阵

glLoadIdentity();// 重置坐标系统，使投影变换复位

glFrustum(-ar, ar, -1.0, 1.0, 2.0, 100.0);//利用函数glFrustum定义修剪区

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity() ;

}

**注释：这个函数是必须要有的，一般写法固定，不需要修改，在窗口首次创建时或当用户用鼠标改变窗口尺寸时该函数被调用，之后，显示回调函数将被自动调用。**

该函数主要完成**坐标系的变换**。

在窗口中绘图之前，必须告诉OpenGL把程序员绘图所使用的世界坐标转换为屏幕坐标（度量单位为像素）。为此，先指定物体在笛卡儿空间中占据的区域，称这一区域为“修剪区”。而以像素计算的话，修剪区的宽度和高度很少会恰好与窗口的宽度和高度一致。所以坐标系必须从逻辑笛卡儿坐标映射为物理屏幕坐标（度量单位为像素）。即把“修剪区”映射为“视区”或称“视口”。视区是窗口内用于绘制修剪区内容的区域（度量单位为像素），通常把视区定义为充满整个窗口。

当窗口的大小改变，函数**resize**就会收到新的宽度和高度，在函数中利用这些信息可以完成修剪区到视区（世界坐标系到实际屏幕坐标）的映射。

该函数的进一步解释见本文中的OPENGL的变换一节。

**static void display(void)//显示回调**

{

static double a =0;//定义a为每次旋转的角度

a=a+0.08;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);//清除颜色缓冲区和深度缓冲区的值

glColor3d(1,0,0);//画图的颜色为红

//绘制一个实心球，默认原点在屏幕中心与照相机重合

**glPushMatrix();**

glTranslated(0,0,-6); //x为0,y为0,z为-6（z为负才可以显示），向照相机前方移6

glRotated(60,1,0,0);//为了使三维效果更好，绕x轴转动60，不动点原点 glRotated(a,0,0,1);//绕z轴旋转, 角度a，不动点原点，注意每次调用display时a的值都不一样，故实现了旋转。

glutSolidSphere(1,slices,stacks);//绘制圆球半径是1

**glPopMatrix();**

．．．．．．

**//绘制一个线框圆环**

**glPushMatrix();**

glTranslated(0,0,-6);

glRotated(60,0,1,0);//绕y轴转动,角度60

glRotated(a,0,0,1);//绕z轴旋转, 角度a，不动点原点

glutWireTorus(0.5,1.8,slices,stacks);//绘制圆环，内半径0.5，外半径1.8

**glPopMatrix();**

glutSwapBuffers();//交换缓冲区

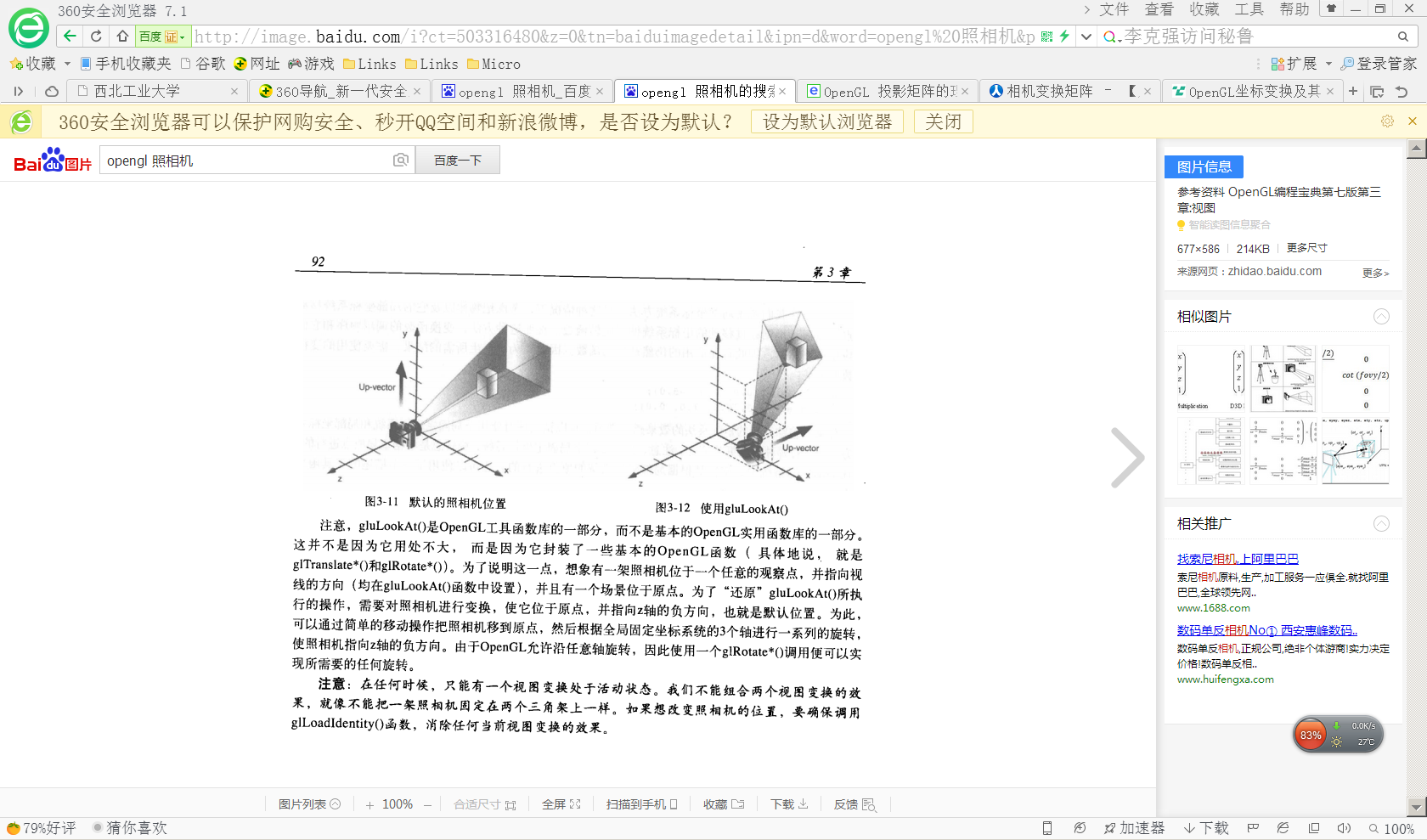
}

**注释：**

这个函数是用来绘图的，在窗口首次创建时当前窗口大小改变事件的回调函数（重绘回调函数）调用后显示回调函数将被自动调用，以后只要绘制窗口，就调用该函数。

**写绘制图形代码的重要提示：**

照相机的默认位置(0,0,0)（位于屏幕中心），指向-z，向上。



**例：绘制一个线框圆环**

**glPushMatrix();**

glTranslated(0,0,-6); //移动坐标

glRotated(60,0,1,0);//绕y轴转动,角度60

glRotated(a,0,0,1);//绕z轴旋转, 角度a，不动点原点

glutWireTorus(0.5,1.8,slices,stacks);//绘制圆环，内半径0.5，外半径1.8

**glPopMatrix();**

**提示1:**绘制图形代码要放在一下两个函数调用中间

**glPushMatrix();**

．．．．．．．

**glPopMatrix();**

**提示 2: glTranslated(0,0,-6);使用该函数移动坐标时，**

**x,y坐标值应小于２和-２，否则将移除窗口，**

**z的值应该为负,例:-2在-3的前面,写代码时注意前面的图像会挡住后面的图像。**

**static void key(unsigned char key, int x, int y)//键盘回调函数**

{

switch (key)

{

case 27 ://Esc键的ASCII值为27

case 'q':

exit(0); //结束程序

break;

case '+':

slices++;

stacks++;

break;

case '-':

if (slices>3 && stacks>3)

{

slices--;

stacks--;

}

break;

}

**glutPostRedisplay();//当前窗口重新绘制，，即调用display函数**

}

**注释：**

这个函数是当击键事件发生时调用的函数，所击的键作为实参，赋值给形参key. x和y则是键摁下时鼠标相对于窗口左上角的位置，由系统获得。

**static void idle(void)//空闲回调函数**

{

glutPostRedisplay();//当前窗口重新绘制，即调用display函数

}

**注释：**

这个函数是当系统空闲时调用的。

//定义全局CONST数组来存放默认光源的颜色值

const GLfloat light\_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };//光源中环境光的颜色，黑色，第四个分量1.0表示不透明。

const GLfloat light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };//光源中漫射光的颜色，白色

const GLfloat light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };//光源中镜面反射光的颜色，白色

const GLfloat light\_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };//定向光源，指定了光源的方向

//定义全局CONST数组来存放材质的颜色值

const GLfloat mat\_ambient[]= { 0.7f, 0.7f, 0.7f, 1.0f }; //材质的环境颜色

const GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f }; //材质的漫射光颜色

const GLfloat mat\_specular[]= { 1.0f, 1.0f, 1.0f, 1.0f };//材质镜面反射光的颜色

const GLfloat high\_shininess[] = { 100.0f };//材质的镜面反射指数

目录

裴新凤

[OPENGL重要概念介绍 8](#_Toc452367104)

[一．Opengl中的几个重要变换 8](#_Toc452367105)

[1．视点变换 9](#_Toc452367106)

[2．模型变换 10](#_Toc452367107)

[3．投影变换----主要用来确定视景体即裁剪体 10](#_Toc452367108)

[4．绘制图形的常用格式 11](#_Toc452367109)

[二.OPENGL绘图坐标系 13](#_Toc452367110)

[三．OpenGL 中的颜色 13](#_Toc452367111)

[四、zhuanqiu程序中动画的实现 13](#_Toc452367112)

[五. zhuanqiu程序中添加光线的步骤 14](#_Toc452367113)

# OPENGL重要概念介绍

**将一个三维物体要在屏幕上显示出来，需要经过几个步骤，每个步骤都是通过OPENGL变换完成的。**

# 一．Opengl中的几个重要变换

从三维物体到二维图象，就如同用相机拍照一样，通常都要经历以下几个步骤：

1）将相机置于三角架上，让它对准三维景物，用OpenGL的视点变换（Viewing Transformation）。

2）将三维物体放在场景中的适当位置，用OpenGL中的模型变换（Modeling Transformation），包括对模型进行旋转、平移和缩放。

3）选择相机镜头并调焦，使三维物体投影在二维胶片上，它相当于OpenGL中把三维模型投影到二维屏幕上的过程，用OpenGL的投影变换（Projection Transformation），OpenGL中投影的方法有两种，即正射投影和透视投影。正射投影时一般是一个长方体的视景体，透视投影时一般是一个棱台似的视景体。只有视景体内的物体能被投影在显示平面上，其他部分则不能。

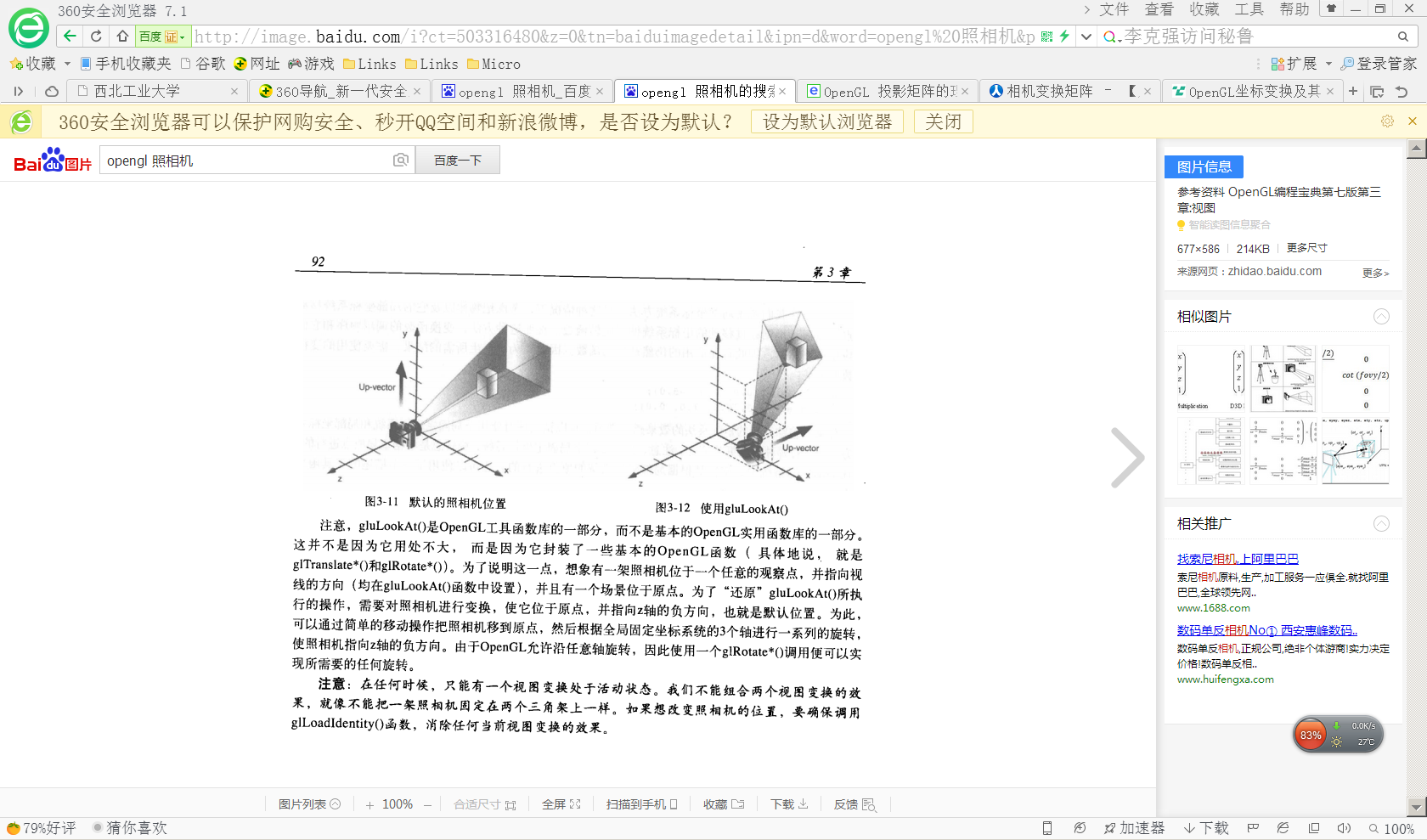
4）冲洗底片，决定二维相片的大小，它相当与OpenGL中的视口变换（Viewport Transformation）（在屏幕窗口内定义一个矩形，称为视口（Viewport），视景体投影后的图形就在视口内显示）规定屏幕上显示场景的范围和尺寸。

通过以上几个步骤，一个三维空间里的物体就可以用相应的二维平面物体表示了，也就能在二维的电脑屏幕上显示了。以上每一个步骤都是通过OPENGL的变换完成的。

**重要提示：Opengl中各种变换是通过矩阵运算实现的。**

### 1．视点变换

照相机的位置称为视点，如果在程序中不做视点变换，则视点的默认位置(0,0,0)，指向-z，向上。



用下函数来改变视点：

void gluLookAt(GLdouble eyex,GLdouble eyey,GLdouble eyez,GLdouble centerx,GLdouble centery,GLdouble upx,GLdouble upy,GLdouble upz);

参数：eyex,eyey,eyez定义了视点的位置；centerx、centery和centerz变量指定了参考点的位置，该点通常为相机所瞄准的场景中心轴线上的点；upx、upy、upz变量指定了向上向量的方向。

**注释：**在例子zhuanqiu中没有视点变换，采用的是默认视点。

### 2．模型变换

由于OpenGL变换是通过矩阵运算来实现，在进行模型变换变换前，应先设置当前操作的矩阵为“模型视图矩阵”。用下面两条语句进行设置：  
glMatrixMode(GL\_MODELVIEW);// 设置当前操作的矩阵为“模型视图矩阵”  
glLoadIdentity();//在进行变换前把当前矩阵设置为单位矩阵  
这两条语句之后，就可以进行模型变换了。模型变换，主要涉及到三个函数：  
1）平移变换函数：glTranslatef（GLfloat x，GLfloat y，GLfloat z），该函数生成一个表示移动物体的平移矩阵，并将这个矩阵乘以当前矩阵，从而生成新的当前矩阵。三个参数分别表示了在三个坐标上的位移值。  
2）旋转变换函数：glRotatef（GLfloat angle，GLfloat x，GLfloat y，GLfloat z），该函数生成一个表示旋转物体的旋转矩阵，并将这个矩阵乘以当前矩阵，从而生成新的当前矩阵。物体将绕着(0,0,0)到(x,y,z)的直线以逆时针旋转，参数angle表示旋转的角度。  
3）缩放函数：glScalef（GLfloat x，GLfloat y，GLfloat z），该函数生成一个表示缩放物体的缩放矩阵，并将这个矩阵乘以当前矩阵，从而生成新的当前矩阵。x,y,z分别表示在该方向上的缩放比例。

**注释：**做OPENGL模型变换之前，先设置当前矩阵的类型，然后将其设置为单位矩阵。

**glMatrixMode(GL\_MODELVIEW);// 设置当前操作的矩阵为“模型视图矩阵”  
glLoadIdentity();//在进行变换前把当前矩阵设置为单位矩阵**然后，才可以用上面三个函数进行相应的模型变换。

### 3．投影变换----主要用来确定视景体即裁剪体

OpenGL支持两种类型的投影变换，即透视投影（透视投影所产生的结果有近大远小的效果）和正投影（图形的大小不变）。投影变换也是使用矩阵来实现的，在进行投影变换前，应先设置当前操作的矩阵为“投影矩阵”。

glMatrixMode(GL\_PROJECTION); // 设置当前操作的矩阵为“投影矩阵”  
glLoadIdentity();//在进行变换前把当前矩阵设置为单位矩阵

然后可以调用下列函数进行投影变换了

1）正投影

void glOrtho(GLdouble left，GLdouble right，GLdouble bottom，GLdouble top，GLdouble near，GLdouble far);

它创建一个平行视景体。这个函数的操作是创建一个正射投影矩阵，并且用这个矩阵乘以当前矩阵。其中近裁剪平面是一个矩形，矩形左下角点三维空间坐标是（left，bottom，-near），右上角点是（right，top，-near）；远裁剪平面也是一个矩形，左下角点空间坐标是（left，bottom，-far），右上角点是（right，top，-far）。所有的near和far值同时为正或同时为负。如果没有其他变换，正射投影的方向平行于Z轴，且视点朝向Z负轴。

通常的使用方法：

glMatrixMode(GLPROJECTION); //设置投影矩阵

glLoadIdentity(); // // 当前矩阵设置为单位矩阵

glOrtho(-1,1,-1,1,-1,-8); //设置长方体的裁剪体

2）透视投影---**定义了一个四棱台的裁剪体（视景体），裁剪体外的物体将不出现在图像中。**

 void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

它创建一个透视视景体。其操作是创建一个透视投影矩阵，并且用这个矩阵乘以当前矩阵。这个函数的参数只定义近裁剪平面的左下角点和右上角点的三维空间坐标，即（left，bottom，-near）和（right，top，-near）；最后一个参数far是远裁剪平面的Z负值，其左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。 near和far表示离视点的远近，它们总为正值。

通常的使用方法：

glMatrixMode(GLPROJECTION); //设置投影矩阵

glLoadIdentity(); // 当前矩阵设置为单位矩阵

glFrustum(-1,1,-2,2,4,10) //一个4棱台（六面体）的裁剪体

**注释：**模型视图矩阵和投影矩阵都有相应的矩阵。使用glMatrixMode来指定当前操作的究竟是模型视图矩阵还是投影矩阵。

现在重新阅读zhuanqiu中的**重绘回调函数**

**static void resize(int width, int height)//重绘回调函数**

{

const float ar = (float) width / (float) height;

**//设置视口变换**

glViewport(0, 0, width, height);//利用函数glViewport定义视区（视口），在窗口内设置一个区域，这个区域用于将修剪空间（视景体）的坐标（世界坐标）映射到物理窗口的坐标（像素）。

**//设置投影矩阵**

glMatrixMode(GL\_PROJECTION);//设置投影矩阵

glLoadIdentity();// 将当前矩阵设置为单位矩阵

glFrustum(-ar, ar, -1.0, 1.0, 2.0, 100.0);//做透视投影变换，定义修剪区

**//设置模型视图矩阵**

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity() ;

}

**重要注释：**因为**重绘回调函数在窗口创建时被调用，模型视图矩阵在这里已经设置好，所以在程序中做模型变换时不必再设置模型视图矩阵，只需做变换就可以。**

### 4．绘制图形的常用格式

由于OpenGL只维护一个当前矩阵，每次变换都会作用在上次变换之后的当前矩阵上，换句话说，这种变换是累加的。如：当我们绘制了一个球并平移了之后，新的当前矩阵就已经包含了平移的操作，如果我们再在此基础上绘制立方体，那么该立方体并不是真的在我们给定参数的位置上。怎样消除这些累积呢，最好的办法是：

绘图之前先调用glPushMatrix（）将当前矩阵压入堆栈；

然后通过移动、旋转、缩放物体，使物体处在理想的位置和观察角度，然后进行绘制。

绘制完毕后，调用glPopMatrix（），将栈顶矩阵弹出来，恢复当前矩阵到原来的状态。

通过这一对操作后，如果再绘制其它图形，就仍然会参照原始的坐标系，而不是变化后的坐标系。

**Zhuanqiu中绘制图形的代码：**

采用视点默认位置，原点在屏幕中心与照相机重合

//绘制一个实心球

glPushMatrix();//将当前矩阵压入堆栈；

glTranslated(0,0,-6); //x中为0,y值为0,z为-6.z值取负值才可以显示

glRotated(60,1,0,0);//绕x轴旋转, 角度60，不动点原点

glRotated(a,0,0,1);//绕z轴旋转, 角度a，不动点原点

glutSolidSphere(1,slices,stacks);//圆球半径是1

glPopMatrix();//将栈顶矩阵弹出来，恢复当前矩阵到原来的状态

//绘制一个实心圆锥体

glPushMatrix();

glTranslated(0,1.6,-6);

glRotated(60,1,0,0);

glRotated(a,0,0,1);

glutSolidCone(1,1,slices,stacks);

//glutSolidCone(GLdouble radius， GLdouble height, GLint slices, GLint stacks)

glPopMatrix();

glColor3d(0,1,0);//画图的颜色为绿色

//绘制一个线框圆环

glPushMatrix();

glTranslated(0,0,-6);

glRotated(60,1,0,0);

glRotated(a,0,0,1);

glutWireTorus(0.5,1.8,slices,stacks);

//glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

glPopMatrix();

# 二.OPENGL绘图坐标系

  OpenGL中使用的坐标系主要有两种，分别为世界坐标系和屏幕坐标系。世界坐标系即OpenGL内部处理时使用的三维坐标系，而屏幕坐标系是在屏幕上显示图形时使用的坐标系（以像素为 单位） 。

OpenGL所使用的世界坐标系为右手型，如右图所示。 以屏幕中心为原点(0, 0, 0)。面对屏幕，右边是x正轴，上面是y正轴，屏幕指向你的为z正轴。

而进行旋转操作时需要指定的角度θ的方向则由右手法则来决定，即右手握拳，大拇指直向某个坐标轴的正方向，那么其余四指指向的方向即为该坐标轴上的θ角的正方向（即θ角增加的方向），在上图中用圆弧形箭头标出。

当前绘图坐标系是 绘制物体时的坐标系。程序刚初始化时，世界坐标系和当前绘图坐标系是重合的。当用glTranslatef()，glScalef(), glRotatef()对当前绘图坐标系进行平移、伸缩、旋转变换之后， 世界坐标系和当前绘图坐标系不再重合。改变以后，再用绘图函数绘图时，都是在当前绘图坐标系进行绘图，所有的函数参数也都是相 对当前绘图坐标系来讲的。

# 三．OpenGL 中的颜色

* RGB是表示色彩的一种方法。R代表红色，G代表绿色，B代表蓝色，利用这三种原色的配比来表示一种色彩是计算机上最常用的方式。
* 在OpenGL中，颜色的范围是在[0,1]之间，（1，1，1）为白色，（0，0，0）为黑色。
* RGBA: 多一个分量，用第4个分量来表示颜色的透明度，1.0表示该颜色完全不透明，0.0表示颜色完全透明。

# 四、zhuanqiu程序中动画的实现

以下两个回调函数配合完成图形的旋转的。

**static void idle(void)//空闲回调函数**

{

glutPostRedisplay();//当前窗口需要重新绘制

}

**static void display(void)//显示回调函数**

{

static double a =0;//定义a为每次旋转的角度

a=a+0.08;

//绘制一个实心球，默认原点在屏幕中心与照相机重合

glPushMatrix();

glTranslated(0,0,-7); //将对象移动到x为0,y值为0,z为-6的位置，z值取负值才可以显示

glRotated(60,1,0,0);//绕x轴转动,角度60，不动点原点

glRotated(a,0,0,1);//绕z轴旋转,每次调用display时角度a的值都增加了0.08，实现了旋转。

glutSolidSphere(1,slices,stacks);//画圆球半径是1，经度数，纬度数分别为16

glPopMatrix();

**注释：**每当系统空闲时，调用display函数，该函数中的变量a(角度）在每次调用中都将增值0.08，从而实现了旋转。

# 五. zhuanqiu程序中添加光线的步骤

向场景添加光线的步骤：

* 为物体的每一个定点定义法线向量
* 启用光照
* 创建光源并设置光源属性。
* 设置物体材质属性。

具体实现：

首先完成启用光照和材质的功能，在main()函数中用了如下语句：

glEnable(GL\_LIGHT0);//启用0号光源

glEnable(GL\_NORMALIZE);// 启用自动单位化法向量

glEnable(GL\_LIGHTING);//启用光照

glEnable(GL\_COLOR\_MATERIAL); //启用材质颜色

通过下面两段代码设置了光源的颜色和位置（属性）：

//定义默认光源属性值，使用数组来存放光源的颜色值

const GLfloat light\_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };//光源中环境光的颜色，黑色，第四个分量1.0表示不透明。

const GLfloat light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };//光源中漫射光的颜色，白色

const GLfloat light\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };//光源中镜面反射光的颜色

const GLfloat light\_position[] = { 2.0f, 5.0f, 5.0f, 0.0f };//定向光源，指定了光源的位置

//设置光源

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

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

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

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

通过下面两段代码设置材质的颜色和位置（属性）

//定义材质属性值，使用数组来存放材质的颜色值

const GLfloat mat\_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f }; //材质的环境颜色

const GLfloat mat\_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f }; //材质的漫射光颜色

const GLfloat mat\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };//材质镜面反射光的颜色

const GLfloat high\_shininess[] = { 100.0f };//材质的镜面反射指数

//设置材质

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);