GL\_PROJECTION（投影）

GL\_MODELVIEW（平移，旋转，缩放）

GL\_TEXTURE（纹理）

这两个都是glMatrixMode()函数的参数，那就先说说glMatrixMode吧~，这个函数其实就是对接下来要做什么进行一下声明，也就是在要做下一步之前告诉计算机我要对“什么”进行操作了，这个“什么”在glMatrixMode的“()”里的选项(参数)有，GL\_PROJECTION，GL\_MODELVIEW和GL\_TEXTURE；

如果参数是GL\_PROJECTION，这个是投影的意思，就是要对投影相关进行操作，也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。这样，接下来的语句可以是跟透视相关的函数，比如glFrustum()或gluPerspective()；

如果参数是GL\_MODELVIEW，这个是对模型视景的操作，接下来的语句描绘一个以模型为基础的适应，这样来设置参数，接下来用到的就是像gluLookAt()这样的函数；

若是GL\_TEXTURE，就是对纹理相关进行操作；

顺便说下，OpenGL里面的操作，很多是基于对矩阵的操作的，比如位移，旋转，缩放，所以，这里其实说的规范一点就是glMatrixMode是用来指定哪一个矩阵是当前矩阵，而它的参数代表要操作的目标，GL\_PROJECTION是对投影矩阵操作，GL\_MODELVIEW是对模型视景矩阵操作，GL\_TEXTURE是对纹理矩阵进行随后的操作。

OpenGL视点变换，模型变换，投影变换，视口变换详解

作者：luck\_net | 出处：博客园 | 2012/2/22 14:46:49 | 阅读112次

OpenGL通过相机模拟、可以实现计算机图形学中最基本的三维变换，即几何变换、投影变换、裁剪变换、视口变换等，同时，OpenGL还实现了矩阵堆栈等。理解掌握了有关坐标变换的opengl图片内容，就算真正走进了精彩地三维世界。

一、OpenGL中的三维物体的显示

（一）坐标系统

在现实世界中，所有的物体都具有三维特征，但计算机本身只能处理数字，显示二维的图形，将三维物体及二维数据联系在一起的唯一纽带就是坐标。

为了使被显示的三维物体数字化，要在被显示的物体所在的空间中定义一个坐标系。这个坐标系的长度单位和坐标轴的方向要适合对被显示物体的描述，这个坐标系称为世界坐标系。世界坐标系是始终固定不变的。

OpenGL还定义了局部坐标系的概念，所谓局部坐标系，也就是坐标系以物体的中心为坐标原点，物体的旋转或平移等操作都是围绕局部坐标系进行的，这时，当物体模型进行旋转或平移等操作时，局部坐标系也执行相应的旋转或平移操作。需要注意的是，如果对物体模型进行缩放操作，则局部坐标系也要进行相应的缩放，如果缩放比例在案各坐标轴上不同，那么再经过旋转操作后，局部坐标轴之间可能不再相互垂直。无论是在世界坐标系中进行转换还是在局部坐标系中进行转换，程序代码是相同的，只是不同的坐标系考虑的转换方式不同罢了。

计算机对数字化的显示物体作了加工处理后，要在图形显示器上显示，这就要在图形显示器屏幕上定义一个二维直角坐标系，这个坐标系称为屏幕坐标系。这个坐标系坐标轴的方向通常取成平行于屏幕的边缘，坐标原点取在左下角，长度单位常取成一个象素。

（二）三维物体的相机模拟

为了说明在三维物体到二维图象之间，需要经过什么样的变换，我们引入了相机（Camera）模拟的方式，假定用相机来拍摄这个世界，那么在相机的取景器中，就存在人眼和现实世界之间的一个变换过程。

图一、相机模拟OpenGL中的各种坐标变换

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，它相当于OpenGL中调整视点的位置，即视点变换（Viewing Transformation）。

2、将三维物体放在场景中的适当位置，它相当于OpenGL中的模型变换（Modeling Transformation），即对模型进行旋转、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，即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。为了使显示的物体能以合适的位置、大小和方向显示出来，必须要通过投影。有时为了突出图形的一部分，只把图形的某一部分显示出来，这时可以定义一个三维视景体（Viewing Volume）。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）（在屏幕窗口内可以定义一个矩形，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。

二、OpenGL中的几种变换

OpenGL中的各种转换是通过矩阵运算实现的，具体的说，就是当发出一个转换命令时，该命令会生成一个4X4阶的转换矩阵（OpenGL中的物体坐标一律采用齐次坐标，即(x, y, z, w)，故所有变换矩阵都采用4X4矩阵），当前矩阵与这个转换矩阵相乘，从而生成新的当前矩阵。例如，对于顶点坐标v ，转换命令通常在顶点坐标命令之前发出，若当前矩阵为C，转换命令构成的矩阵为M，则发出转换命令后，生成的新的当前矩阵为CM，这个矩阵再乘以顶点坐标v，从而构成新的顶点坐标CMv。上述过程说明，程序中绘制顶点前的最后一个变换命令最先作用于顶点之上。这同时也说明，OpenGL编程中，实际的变换顺序与指定的顺序是相反的。

（一）视点变换

视点变换确定了场景中物体的视点位置和方向，就向上边提到的，它象是在场景中放置了一架照相机，让相机对准要拍摄的物体。确省时，相机（即视点）定位在坐标系的原点（相机初始方向都指向Z负轴），它同物体模型的缺省位置是一致的，显然，如果不进行视点变换，相机和物体是重叠在一起的。

执行视点变换的命令和执行模型转换的命令是相同的，想一想，在用相机拍摄物体时，我们可以保持物体的位置不动，而将相机移离物体，这就相当于视点变换；另外，我们也可以保持相机的固定位置，将物体移离相机，这就相当于模型转换。这样，在OpenGL中，以逆时针旋转物体就相当于以顺时针旋转相机。因此，我们必须把视点转换和模型转换结合在一起考虑，而对这两种转换单独进行考虑是毫无意义的。

除了用模型转换命令执行视点转换之外，OpenGL实用库还提供了gluLookAt()函数，该函数有三个变量，分别定义了视点的位置、相机瞄准方向的参考点以及相机的向上方向。该函数的原型为：

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

该函数定义了视点矩阵，并用该矩阵乘以当前矩阵。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。

通常，视点转换操作在模型转换操作之前发出，以便模型转换先对物体发生作用。场景中物体的顶点经过模型转换之后移动到所希望的位置，然后再对场景进行视点定位等操作。模型转换和视点转换共同构成模型视景矩阵。

（二）模型变换

模型变换是在世界坐标系中进行的。缺省时，物体模型的中心定位在坐标系的中心处。OpenGL在这个坐标系中，有三个命令，可以模型变换。

1、模型平移

glTranslate{fd}(TYPE x,TYPE y,TYPE z);

该函数用指定的x,y,z值沿着x轴、y轴、z轴平移物体（或按照相同的量值移动局部坐标系）。

2、模型旋转

glRotate{fd}(TYPE angle,TYPE x,TYPE,y,TYPE z);

该函数中第一个变量angle制定模型旋转的角度，单位为度，后三个变量表示以原点（0,0,0）到点(x,y,z)的连线为轴线逆时针旋转物体。例如，glRotatef(45.0,0.0,0.0,1.0)的结果是绕z轴旋转45度。

3、模型缩放

glScale{fd}(TYPE x,TYPE y,TYPE z);

该函数可以对物体沿着x,y,z轴分别进行放大缩小。函数中的三个参数分别是x、y、z轴方向的比例变换因子。缺省时都为1.0，即物体没变化。程序中物体Y轴比例为2.0，其余都为1.0，就是说将立方体变成长方体。

（三）投影变换

经过模型视景的转换后，场景中的物体放在了所希望的位置上，但由于显示器只能用二维图象显示三维物体，因此就要靠投影来降低维数（投影变换类似于选择相机的镜头）。

事实上，投影变换的目的就是定义一个视景体，使得视景体外多余的部分裁剪掉，最终进入图像的只是视景体内的有关部分。投影包括透视投影（Perspective Projection）和正视投影（Orthographic Projection）两种。

透视投影，符合人们心理习惯，即离视点近的物体大，离视点远的物体小，远到极点即为消失，成为灭点。它的视景体类似于一个顶部和底部都被进行切割过的棱椎，也就是棱台。这个投影通常用于动画、视觉仿真以及其它许多具有真实性反映的方面。

OpenGL透视投影函数有两个，其中函数glFrustum()的原型为：

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。

另一个透视函数是：

void gluPerspective(GLdouble fovy,GLdouble aspect,GLdouble zNear, GLdouble zFar);

它也创建一个对称透视视景体，但它的参数定义于前面的不同，参数fovy定义视野在X-Z平面的角度，范围是[0.0, 180.0]；参数aspect是投影平面宽度与高度的比率；参数zNear和Far分别是远近裁剪面沿Z负轴到视点的距离，它们总为正值。

以上两个函数缺省时，视点都在原点，视线沿Z轴指向负方向。

正射投影，又叫平行投影。这种投影的视景体是一个矩形的平行管道，也就是一个长方体，如图五所示。正射投影的最大一个特点是无论物体距离相机多远，投影后的物体大小尺寸不变。这种投影通常用在建筑蓝图绘制和计算机辅助设计等方面，这些行业要求投影后的物体尺寸及相互间的角度不变，以便施工或制造时物体比例大小正确。

OpenGL正射投影函数也有两个，一个函数是：

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

它创建一个平行视景体。实际上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。这意味着物体在视点前面时far和near都为负值，物体在视点后面时far和near都为正值。

另一个函数是：

void gluOrtho2D(GLdouble left,GLdouble right,GLdouble bottom,GLdouble top)

它是一个特殊的正射投影函数，主要用于二维图像到二维屏幕上的投影。它的near和far缺省值分别为-1.0和1.0，所有二维物体的Z坐标都为0.0。因此它的裁剪面是一个左下角点为（left，bottom）、右上角点为（right，top）的矩形。

（四）视口变换。

视口变换就是将视景体内投影的物体显示在二维的视口平面上。运用相机模拟方式，我们很容易理解视口变换就是类似于照片的放大与缩小。在计算机图形学中，它的定义是将经过几何变换、投影变换和裁剪变换后的物体显示于屏幕窗口内指定的区域内，这个区域通常为矩形，称为视口。OpenGL中相关函数是：

glViewport(GLint x,GLint y,GLsizei width, GLsizei height);

这个函数定义一个视口。函数参数(x, y)是视口在屏幕窗口坐标系中的左下角点坐标，参数width和height分别是视口的宽度和高度。缺省时，参数值即(0, 0, winWidth, winHeight) 指的是屏幕窗口的实际尺寸大小。所有这些值都是以象素为单位，全为整型数。

（5）裁剪变换

在OpenGL中，除了视景体定义的六个裁剪平面（上、下、左、右、前、后）外，用户还可自己再定义一个或多个附加裁剪平面，以去掉场景中无关的目标。

附加平面裁剪函数为：

void glClipPlane(GLenum plane,Const GLdouble \*equation);

函数参数equation指向一个拥有四个系数值的数组，这四个系数分别是裁剪平面Ax+By+Cz+D=0的A、B、C、D值。因此，由这四个系数就能确定一个裁剪平面。参数plane是GL\_CLIP\_PLANEi(i=0,1,...)，指定裁剪面号。

在调用附加裁剪函数之前，必须先启动glEnable(GL\_CLIP\_PLANEi)，使得当前所定义的裁剪平面有效；当不再调用某个附加裁剪平面时，可用glDisable(GL\_CLIP\_PLANEi)关闭相应的附加裁剪功能。

下面这个例子不仅说明了附加裁剪函数的用法，而且调用了gluPerspective()透视投影函数，读者可以细细体会其中的用法。例程如下：

#include "glos.h"

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glaux.h>

void myinit(void);

void CALLBACK myReshape(GLsizei w, GLsizei h);

void CALLBACK display(void);

void CALLBACK display(void)

{

GLdouble eqn[4] = {1.0, 0.0, 0.0, 0.0};

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f (1.0, 0.0, 1.0);

glPushMatrix();

glTranslatef (0.0, 0.0, -5.0);

/\* clip the left part of wire\_sphere : x<0 \*/

glClipPlane (GL\_CLIP\_PLANE0, eqn);

glEnable (GL\_CLIP\_PLANE0);

glRotatef (-90.0, 1.0, 0.0, 0.0);

auxWireSphere(1.0);

glPopMatrix();

glFlush();

}

void myinit (void)

{

glShadeModel (GL\_FLAT);

}

void CALLBACK myReshape(GLsizei w, GLsizei h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

}

void main(void)

{

auxInitDisplayMode (AUX\_SINGLE | AUX\_RGB);

auxInitPosition (0, 0, 500, 500);

auxInitWindow ("Arbitrary Clipping Planes");

myinit ();

auxReshapeFunc (myReshape);

auxMainLoop(display);

}

（六）矩阵栈的操作

在讲述矩阵栈之前，首先介绍两个基本OpenGL矩阵操作函数：

void glLoadMatrix{fd}(const TYPE \*m)

设置当前矩阵中的元素值。函数参数\*m是一个指向16个元素(m0, m1, ..., m15)的指针，这16个元素就是当前矩阵M中的元素，其排列方式如下：

M = | m0 m4 m8 m12 |

| m1 m5 m9 m13 |

| m2 m6 m10 m14 |

| m3 m7 m11 M15 |

2、void glMultMatrix{fd}(const TYPE \*m)

用当前矩阵去乘\*m所指定的矩阵，并将结果存放于\*m中。当前矩阵可以是用glLoadMatrix() 指定的矩阵，也可以是其它矩阵变换函数的综合结果。

OpenGL的矩阵堆栈指的就是内存中专门用来存放矩阵数据的某块特殊区域。一般说来，矩阵堆栈常用于构造具有继承性的模型，即由一些简单目标构成的复杂模型。矩阵堆栈对复杂模型运动过程中的多个变换操作之间的联系与独立十分有利。因为所有矩阵操作函数如glLoadMatrix()、glMultMatrix()、glLoadIdentity()等只处理当前矩阵或堆栈顶部矩阵，这样堆栈中下面的其它矩阵就不受影响。堆栈操作函数有以下两个：

·void glPushMatrix(void);

该函数表示将所有矩阵依次压入堆栈中，顶部矩阵是第二个矩阵的备份；压入的矩阵数不能太多，否则出错。

·void glPopMatrix(void);

该函数表示弹出堆栈顶部的矩阵，令原第二个矩阵成为顶部矩阵，接受当前操作，故原顶部矩阵被破坏；当堆栈中仅存一个矩阵时，不能进行弹出操作，否则出错。