OpenGL学习笔记（幻骁版）

一、OpenGL 单缓冲与双缓冲

**1、说明**

相关函数：Void glutInitDisplayMode(unsigned int mode);

GLUT\_SINGLE 指定单缓存窗口

GLUT\_DOUBLE 指定双缓存窗口

应用程序使用单缓冲绘图时可能会存在图像闪烁的问题。这是因为生成的图像不是一下子被绘制出来的，而是按照从左到右，由上而下逐像素地绘制而成的。最终图像不是在瞬间显示给用户，而是通过一步一步生成的，这会导致渲染的结果很不真实。为了规避这些问题，我们应用双缓冲渲染窗口应用程序。前缓冲保存着最终输出的图像，它会在屏幕上显示；而所有的的渲染指令都会在后缓冲上绘制。当所有的渲染指令执行完毕后，我们交换(Swap)前缓冲和后缓冲，这样图像就立即呈显出来，之前提到的不真实感就消除了。

**2、原理**

GLUT\_SINGLE单缓冲，屏幕显示调用glFlush()，将图像在当前显示缓存中直接渲染，会有图形跳动(闪烁)问题

GLUT\_DOUBLE双缓冲，屏幕显示调用glutSwapBuffers()，将图像先绘制在另外的缓存中，渲染完毕之后，将其整个缓存贴到当前的窗口，能消除闪烁,一般动画要用双缓冲.

如果缓冲与函数不对应的话，则会出错。

二、有关深度缓冲区

接触到3D，就一定会接触到深度这个概念，最直接的就是看Z轴，两个人在场景中，你的Z轴是1，我的Z轴是2，摄像机位于Z -10的位置，那么我的深度比你的大。

在OpenGL中，默认是没有开启深度检测的，也就是说，后绘制的物体覆盖先绘制的物体(颜色缓冲区中，先绘制的物体被后绘制的物体覆盖)。为了避免这个问题，需要在创建窗口时启动深度缓冲区(glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE | GLUT\_DEPTH);)，同时在init()函数中开启深度测试，在每次绘制时清除深度缓冲区的内容。

三、有关glTranslate()和glRotate()函数

这两个函数与模型变换有关，使用其对模型进行平移或者旋转时，一个角度的理解是：模型自身的坐标系会随平移或旋转而发生相应的变换。

四、堆栈

OpenGL有三个矩阵堆栈，分别是GL\_MODELVIEW（模型视图矩阵堆栈）、GL\_PROJECTION（投影矩阵堆栈）、GL\_TEXTURE（纹理矩阵堆栈）。

对于矩阵的操作都是对于矩阵栈的栈顶来操作的。当前矩阵即为矩阵栈的栈顶元素，而对当前矩阵进行平移、旋转等的变换操作也同样是对栈顶矩阵的修改。所以我们在变换之前调用giPushMatrix()的话，就会把当前状态压入第二层，不过此时栈顶的矩阵也与第二层的相同。

五、混合

颜色混合，控制原来的颜色值与新片断的颜色值进行组合，产生透明的效果。可使用alpha混合来创建半透明片断，是透明化、数字合成、油漆这类技巧的核心。

六、绘制半透明物体的注意事项

深度缓冲是这样一段数据，它记录了每一个像素距离观察者有多近。在启用深度缓冲测试的情况下，如果将要绘制的像素比原来的像素更近，则像素将被绘制。否则，像素就会被忽略掉，不进行绘制。这在绘制不透明的物体时非常有用——不管是先绘制近的物体再绘制远的物体，还是先绘制远的物体再绘制近的物体，或者干脆以混乱的顺序进行绘制，最后的显示结果总是近的物体遮住远的物体。

然而在你需要实现半透明效果时，发现一切都不是那么美好了。如果你绘制了一个近距离的半透明物体，则它在深度缓冲区内保留了一些信息，使得远处的物体将无法再被绘制出来。虽然半透明的物体仍然半透明，但透过它看到的却不是正确的内容了。

要解决以上问题，需要在绘制半透明物体时将深度缓冲区设置为只读，这样一来，虽然半透明物体被绘制上去了，深度缓冲区还保持在原来的状态。如果再有一个物体出现在半透明物体之后，在不透明物体之前，则它也可以被绘制（因为此时深度缓冲区中记录的是那个不透明物体的深度）。以后再要绘制不透明物体时，只需要再将深度缓冲区设置为可读可写的形式即可。

void renderAlpha(){

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA,GL\_ONE\_MINUS\_SRC\_ALPHA);

glDepthMask(GL\_FALSE);

glDisable(GL\_LIGHTING);

//draw sth;需要绘制物体的代码

// 要设置透明需要使用颜色函数 glColor4f(r,g,b,a); a表示的是透明度

glDisable(GL\_BLEND);

glEnable(GL\_LIGHTING);

glDepthMask(GL\_TRUE);

}

七、设置着色模式

着色模式（glShadeModel）实际上是解决实心图形内部空间颜色填充的问题。默认情况下OpenGL是采用渐变着色模式（采用GL\_SMOOTH进行设置），若线段的两个端点颜色设置不同，在GL\_SMOOTH模式下，线段中间的颜色在两端点之间进行平稳过渡。启用渐变着色模式采用下面的命令：glShadeModel（GL\_SMOOTH）。当然我们也可以采用数学方法，完全按照自己的意愿进行颜色渐变。

与之相对的是单颜色着色模式（采用GL\_FLAT进行设置）。在这种模式下，图形内部颜色填充采用的是最后一个顶点的颜色。启用单颜色着色模式采用命令：glShadeModel（GL\_FLAT）。但是若所有的顶点颜色相同，GL\_SMOOTH、GL\_FLAT两种模式产生的效果是没有任何差别的。

两种着色模式：平滑着色与单调着色：

（1）平滑着色：glShadeModel（GL\_SMOOTH）平滑着色可以实现一种颜色到另一种颜色的平滑渐变。

（2）单调着色：glShadeModel（GL\_FLAT）单调着色即GL\_FLAT，单调着色表示图元在内部不进行计算。一般而言，使用单调着色，图元的内部颜色就是最后一个项点所指定的颜色。唯一例外的是GL\_POLYGON( 多边形 )图元，它的内部颜色是第一个顶点的颜色。

七、纹理贴图

glGenTextures(GLsizei n, GLuint \*textures)函数说明

n：用来生成纹理的数量

textures：存储纹理索引的第一个元素指针

（glGenTextures就是用来产生你要操作的纹理对象的索引的，比如你告诉OpenGL，我需要5个纹理对象，它会从没有用到的整数里返回5个给你）

glBindTexture实际上是改变了OpenGL的这个状态，它告诉OpenGL下面对纹理的任何操作都是对它所绑定的纹理对象的，比如glBindTexture(GL\_TEXTURE\_2D,1)告诉OpenGL下面代码中对2D纹理的任何设置都是针对索引为1的纹理的。