混合是什么呢？混合就是把两种颜色混在一起。具体一点，就是把某一像素位置原来的颜色和将要画上去的颜色，通过某种方式混在一起，从而实现特殊的效果。

假设我们需要绘制这样一个场景：透过红色的玻璃去看绿色的物体，那么可以先绘制绿色的物体，再绘制红色玻璃。在绘制红色玻璃的时候，利用“混合”功能，把将要绘制上去的红色和原来的绿色进行混合，于是得到一种新的颜色，看上去就好像玻璃是半透明的。

要使用OpenGL的混合功能，只需要调用：glEnable(GL\_BLEND);即可。

要关闭OpenGL的混合功能，只需要调用：glDisable(GL\_BLEND);即可。

注意：只有在RGBA模式下，才可以使用混合功能，颜色索引模式下是无法使用混合功能的。

一、源因子和目标因子

前面我们已经提到，混合需要把原来的颜色和将要画上去的颜色找出来，经过某种方式处理后得到一种新的颜色。这里把将要画上去的颜色称为“源颜色”，把原来的颜色称为“目标颜色”。

OpenGL 会把源颜色和目标颜色各自取出，并乘以一个系数（源颜色乘以的系数称为“源因子”，目标颜色乘以的系数称为“目标因子”），然后相加，这样就得到了新的颜 色。（也可以不是相加，新版本的OpenGL可以设置运算方式，包括加、减、取两者中较大的、取两者中较小的、逻辑运算等，但我们这里为了简单起见，不讨 论这个了）  
 下面用数学公式来表达一下这个运算方式。假设源颜色的四个分量（指红色，绿色，蓝色，alpha值）是(Rs, Gs, Bs,  As)，目标颜色的四个分量是(Rd, Gd, Bd, Ad)，又设源因子为(Sr, Sg, Sb, Sa)，目标因子为(Dr, Dg, Db,  Da)。则混合产生的新颜色可以表示为：  
(Rs\*Sr+Rd\*Dr, Gs\*Sg+Gd\*Dg, Bs\*Sb+Bd\*Db, As\*Sa+Ad\*Da)  
当然了，如果颜色的某一分量超过了1.0，则它会被自动截取为1.0，不需要考虑越界的问题。  
  
 源因子和目标因子是可以通过glBlendFunc函数来进行设置的。glBlendFunc有两个参数，前者表示源因子，后者表示目标因子。这两个参数可以是多种值，下面介绍比较常用的几种。  
GL\_ZERO：     表示使用0.0作为因子，实际上相当于不使用这种颜色参与混合运算。  
GL\_ONE：      表示使用1.0作为因子，实际上相当于完全的使用了这种颜色参与混合运算。  
GL\_SRC\_ALPHA：表示使用源颜色的alpha值来作为因子。  
GL\_DST\_ALPHA：表示使用目标颜色的alpha值来作为因子。  
GL\_ONE\_MINUS\_SRC\_ALPHA：表示用1.0减去源颜色的alpha值来作为因子。  
GL\_ONE\_MINUS\_DST\_ALPHA：表示用1.0减去目标颜色的alpha值来作为因子。

除 此以外，还有GL\_SRC\_COLOR（把源颜色的四个分量分别作为因子的四个分量）、GL\_ONE\_MINUS\_SRC\_COLOR、 GL\_DST\_COLOR、GL\_ONE\_MINUS\_DST\_COLOR等，前两个在OpenGL旧版本中只能用于设置目标因子，后两个在OpenGL 旧版本中只能用于设置源因子。新版本的OpenGL则没有这个限制，并且支持新的GL\_CONST\_COLOR（设定一种常数颜色，将其四个分量分别作为 因子的四个分量）、GL\_ONE\_MINUS\_CONST\_COLOR、GL\_CONST\_ALPHA、 GL\_ONE\_MINUS\_CONST\_ALPHA。另外还有GL\_SRC\_ALPHA\_SATURATE。新版本的OpenGL还允许颜色的alpha 值和RGB值采用不同的混合因子。但这些都不是我们现在所需要了解的。毕竟这还是入门教材，不需要整得太复杂~  
  
举例来说：

如果设置了glBlendFunc(GL\_ONE, GL\_ZERO);，则表示完全使用源颜色，完全不使用目标颜色，因此画面效果和不使用混合的时候一致（当然效率可能会低一点点）。如果没有设置源因子和目标因子，则默认情况就是这样的设置。  
 如果设置了glBlendFunc(GL\_ZERO, GL\_ONE);，则表示完全不使用源颜色，因此无论你想画什么，最后都不会被画上去了。（但这并不是说这样设置就没有用，有些时候可能有特殊用途）

如 果设置了glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);，则表示源颜色乘以自身的alpha 值，目标颜色乘以1.0减去源颜色的alpha值，这样一来，源颜色的alpha值越大，则产生的新颜色中源颜色所占比例就越大，而目标颜色所占比例则减 小。这种情况下，我们可以简单的将源颜色的alpha值理解为“不透明度”。这也是混合时最常用的方式。  
如果设置了glBlendFunc(GL\_ONE, GL\_ONE);，则表示完全使用源颜色和目标颜色，最终的颜色实际上就是两种颜色的简单相加。例如红色(1, 0, 0)和绿色(0, 1, 0)相加得到(1, 1, 0)，结果为黄色。

注意：

所 谓源颜色和目标颜色，是跟绘制的顺序有关的。假如先绘制了一个红色的物体，再在其上绘制绿色的物体。则绿色是源颜色，红色是目标颜色。如果顺序反过来，则 红色就是源颜色，绿色才是目标颜色。在绘制时，应该注意顺序，使得绘制的源颜色与设置的源因子对应，目标颜色与设置的目标因子对应。不要被混乱的顺序搞晕 了。

二、实现三维混合

也许你迫不及待的想要绘制一个三维的带有半透明物体的场景了。但是现在恐怕还不行，还有一点是在进行三维场景的混合时必须注意的，那就是深度缓冲。

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。嗯？你问我怎么绘制一个一部分半透明一部分不透明的物体？这个好办，只需要把物体分为两个部分，一部分全是半透明 的，一部分全是不透明的，分别绘制就可以了。

即使使用了以上技巧，我们仍然不能随心所欲的按照混乱顺序来进行绘制。必须是先绘制不透明的物体， 然 后绘制透明的物体。否则，假设背景为蓝色，近处一块红色玻璃，中间一个绿色物体。如果先绘制红色半透明玻璃的话，它先和蓝色背景进行混合，则以后绘制中间 的绿色物体时，想单独与红色玻璃混合已经不能实现了。  
总结起来，绘制顺序就是：首先绘制所有不透明的物体。如果两个物体都是不透明的，则谁先谁 后 都没有关系。然后，将深度缓冲区设置为只读。接下来，绘制所有半透明的物体。如果两个物体都是半透明的，则谁先谁后只需要根据自己的意愿（注意了，先绘制 的将成为“目标颜色”，后绘制的将成为“源颜色”，所以绘制的顺序将会对结果造成一些影响）。最后，将深度缓冲区设置为可读可写形式。

调用glDepthMask(GL\_FALSE);可将深度缓冲区设置为只读形式。调用glDepthMask(GL\_TRUE);可将深度缓冲区设置为可读可写形式。  
一 些网上的教程，包括大名鼎鼎的NeHe教程，都在使用三维混合时直接将深度缓冲区禁用，即调用glDisable(GL\_DEPTH\_TEST);。这样 做并不正确。如果先绘制一个不透明的物体，再在其背后绘制半透明物体，本来后面的半透明物体将不会被显示（被不透明的物体遮住了），但如果禁用深度缓冲， 则它仍然将会显示，并进行混合。NeHe提到某些显卡在使用glDepthMask函数时可能存在一些问题，但可能是由于我的阅历有限，并没有发现这样的 情况。

那么，实际的演示一下吧。我们来绘制一些半透明和不透明的球体。假设有三个球体，一个红色不透明的，一个绿色半透明的，一个蓝色 半透明的。红色最远，绿色 在中间，蓝色最近。根据前面所讲述的内容，红色不透明球体必须首先绘制，而绿色和蓝色则可以随意修改顺序。这里为了演示不注意设置深度缓冲的危害，我们故 意先绘制最近的蓝色球体，再绘制绿色球体。

为了让这些球体有一点立体感，我们使用光照。在(1, 1, -1)处设置一个白色的光源。代码如下：

void setLight(void)  
{  
    static const GLfloat light\_position[] = {1.0f, 1.0f, -1.0f, 1.0f};  
    static const GLfloat light\_ambient[]  = {0.2f, 0.2f, 0.2f, 1.0f};  
    static const GLfloat light\_diffuse[]  = {1.0f, 1.0f, 1.0f, 1.0f};  
    static const GLfloat light\_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};  
  
    glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);  
    glLightfv(GL\_LIGHT0, GL\_AMBIENT,  light\_ambient);  
    glLightfv(GL\_LIGHT0, GL\_DIFFUSE,  light\_diffuse);  
    glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);  
  
    glEnable(GL\_LIGHT0);  
    glEnable(GL\_LIGHTING);  
    glEnable(GL\_DEPTH\_TEST);  
}

每一个球体颜色不同。所以它们的材质也都不同。这里用一个函数来设置材质。  
void setMatirial(const GLfloat mat\_diffuse[4], GLfloat mat\_shininess)  
{  
    static const GLfloat mat\_specular[] = {0.0f, 0.0f, 0.0f, 1.0f};  
    static const GLfloat mat\_emission[] = {0.0f, 0.0f, 0.0f, 1.0f};  
  
    glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, mat\_diffuse);  
    glMaterialfv(GL\_FRONT, GL\_SPECULAR,  mat\_specular);  
    glMaterialfv(GL\_FRONT, GL\_EMISSION,  mat\_emission);  
    glMaterialf (GL\_FRONT, GL\_SHININESS, mat\_shininess);  
}

有了这两个函数，我们就可以根据前面的知识写出整个程序代码了。这里只给出了绘制的部分，其它部分大家可以自行完成。

void myDisplay(void)  
{  
    // 定义一些材质颜色  
    const static GLfloat red\_color[] = {1.0f, 0.0f, 0.0f, 1.0f};  
    const static GLfloat green\_color[] = {0.0f, 1.0f, 0.0f, 0.3333f};  
    const static GLfloat blue\_color[] = {0.0f, 0.0f, 1.0f, 0.5f};  
  
    // 清除屏幕  
    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
  
    // 启动混合并设置混合因子  
    glEnable(GL\_BLEND);  
    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  
  
    // 设置光源  
    setLight();  
  
    // 以(0, 0, 0.5)为中心，绘制一个半径为.3的不透明红色球体（离观察者最远）  
    setMatirial(red\_color, 30.0);  
    glPushMatrix();  
    glTranslatef(0.0f, 0.0f, 0.5f);  
    glutSolidSphere(0.3, 30, 30);  
    glPopMatrix();  
  
    // 下面将绘制半透明物体了，因此将深度缓冲设置为只读  
    glDepthMask(GL\_FALSE);  
  
    // 以(0.2, 0, -0.5)为中心，绘制一个半径为.2的半透明蓝色球体（离观察者最近）  
    setMatirial(blue\_color, 30.0);  
    glPushMatrix();  
    glTranslatef(0.2f, 0.0f, -0.5f);  
    glutSolidSphere(0.2, 30, 30);  
    glPopMatrix();  
  
    // 以(0.1, 0, 0)为中心，绘制一个半径为.15的半透明绿色球体（在前两个球体之间）  
    setMatirial(green\_color, 30.0);  
    glPushMatrix();  
    glTranslatef(0.1, 0, 0);  
    glutSolidSphere(0.15, 30, 30);  
    glPopMatrix();  
  
    // 完成半透明物体的绘制，将深度缓冲区恢复为可读可写的形式  
    glDepthMask(GL\_TRUE);  
  
    glutSwapBuffers();  
}

大家也可以将上面两处glDepthMask删去，结果会看到最近的蓝色球虽然是半透明的，但它的背后直接就是红色球了，中间的绿色球没有被正确绘制。

小结：

本课介绍了OpenGL混合功能的相关知识。  
混合就是在绘制时，不是直接把新的颜色覆盖在原来旧的颜色上，而是将新的颜色与旧的颜色经过一定的运算，从而产生新的颜色。新的颜色称为源颜色，原来旧的颜色称为目标颜色。传统意义上的混合，是将源颜色乘以源因子，目标颜色乘以目标因子，然后相加。  
源 因子和目标因子是可以设置的。源因子和目标因子设置的不同直接导致混合结果的不同。将源颜色的alpha值作为源因子，用1.0减去源颜色alpha值作 为目标因子，是一种常用的方式。这时候，源颜色的alpha值相当于“不透明度”的作用。利用这一特点可以绘制出一些半透明的物体。

在进行混合时，绘制的顺序十分重要。因为在绘制时，正要绘制上去的是源颜色，原来存在的是目标颜色，因此先绘制的物体就成为目标颜色，后来绘制的则成为源颜色。绘制的顺序要考虑清楚，将目标颜色和设置的目标因子相对应，源颜色和设置的源因子相对应。  
在进行三维混合时，不仅要考虑源因子和目标因子，还应该考虑深度缓冲区。必须先绘制所有不透明的物体，再绘制半透明的物体。在绘制半透明物体时前，还需要将深度缓冲区设置为只读形式，否则可能出现画面错误。