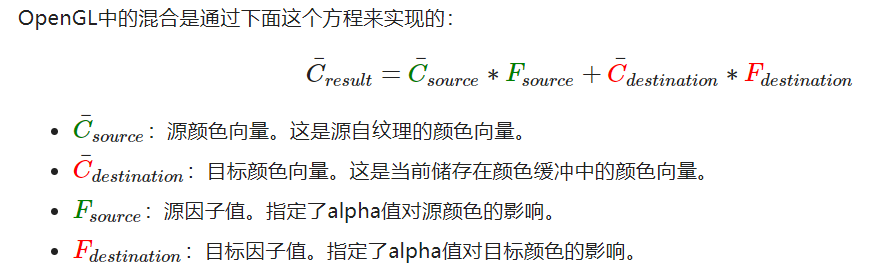
我们目前一直使用的纹理有三个颜色分量：红、绿、蓝。但一些材质会有一个内嵌的alpha通道，对每个纹素(Texel)都包含了一个alpha值。这个alpha值精确地告诉我们纹理各个部分的透明度。

GLSL给了我们discard命令，一旦被调用，它就会保证片段不会被进一步处理，所以就不会进入颜色缓冲。有了这个指令，我们就能够在片段着色器中检测一个片段的alpha值是否低于某个阈值，如果是的话，则丢弃这个片段，就好像它不存在一样：

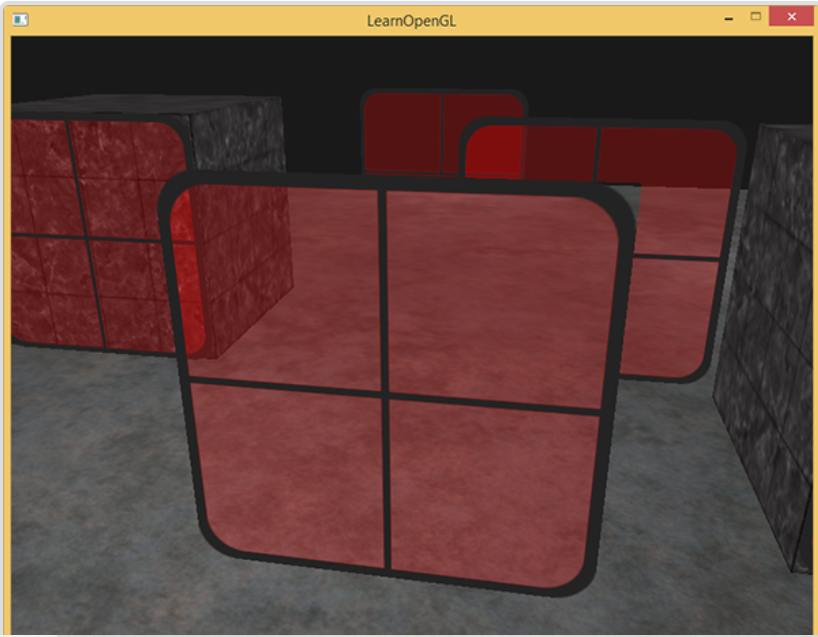
要想渲染有多个透明度级别的图像，我们需要启用混合(Blending)。和OpenGL大多数的功能一样，我们可以启用GL\_BLEND来启用混合：

glEnable(GL\_BLEND);



glBlendFunc(GLenum sfactor, GLenum dfactor)函数接受两个参数，来设置源和目标因子。OpenGL为我们定义了很多个选项，我们将在下面列出大部分最常用的选项。注意常数颜色向量C¯constantC¯constant可以通过glBlendColor函数来另外设置。

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



如果你仔细看的话，你可能会注意到有些不对劲。最前面窗户的透明部分遮蔽了背后的窗户？这为什么会发生呢？

发生这一现象的原因是，深度测试和混合一起使用的话会产生一些麻烦。当写入深度缓冲时，深度缓冲不会检查片段是否是透明的，所以透明的部分会和其它值一样写入到深度缓冲中。结果就是窗户的整个四边形不论透明度都会进行深度测试。即使透明的部分应该显示背后的窗户，深度测试仍然丢弃了它们。

所以我们不能随意地决定如何渲染窗户，让深度缓冲解决所有的问题了。这也是混合变得有些麻烦的部分。要想保证窗户中能够显示它们背后的窗户，我们需要首先绘制背后的这部分窗户。这也就是说在绘制的时候，我们必须先手动将窗户按照最远到最近来排序，再按照顺序渲染。

要想让混合在多个物体上工作，我们需要最先绘制最远的物体，最后绘制最近的物体。普通不需要混合的物体仍然可以使用深度缓冲正常绘制，所以它们不需要排序。但我们仍要保证它们在绘制（排序的）透明物体之前已经绘制完毕了。当绘制一个有不透明和透明物体的场景的时候，大体的原则如下：

1. 先绘制所有不透明的物体。
2. 对所有透明的物体排序。
3. 按顺序绘制所有透明的物体。

排序透明物体的一种方法是，从观察者视角获取物体的距离。这可以通过计算摄像机位置向量和物体的位置向量之间的距离所获得。接下来我们把距离和它对应的位置向量存储到一个STL库的map数据结构中。map会自动根据键值(Key)对它的值排序，所以只要我们添加了所有的位置，并以它的距离作为键，它们就会自动根据距离值排序了。

std::map<float, glm::vec3> sorted;

for (unsigned int i = 0; i < windows.size(); i++)

{

float distance = glm::length(camera.Position - windows[i]);

sorted[distance] = windows[i];

}

之后，这次在渲染的时候，我们将以逆序（从远到近）从map中获取值，之后以正确的顺序绘制对应的窗户：