到目前为止，我们已经使用了很多屏幕缓冲了：用于写入颜色值的颜色缓冲、用于写入深度信息的深度缓冲和允许我们根据一些条件丢弃特定片段的模板缓冲。这些缓冲结合起来叫做帧缓冲(Framebuffer)，它被储存在内存中。OpenGL允许我们定义我们自己的帧缓冲，也就是说我们能够定义我们自己的颜色缓冲，甚至是深度缓冲和模板缓冲。

我们目前所做的所有操作都是在默认帧缓冲的渲染缓冲上进行的。默认的帧缓冲是在你创建窗口的时候生成和配置的（GLFW帮我们做了这些）。有了我们自己的帧缓冲，我们就能够有更多方式来渲染了。

和OpenGL中的其它对象一样，我们会使用一个叫做glGenFramebuffers的函数来创建一个帧缓冲对象(Framebuffer Object, FBO)：

unsigned int fbo;

glGenFramebuffers(1, &fbo);

这种创建和使用对象的方式我们已经见过很多次了，所以它的使用函数也和其它的对象类似。首先我们创建一个帧缓冲对象，将它绑定为激活的(Active)帧缓冲，做一些操作，之后解绑帧缓冲。我们使用glBindFramebuffer来绑定帧缓冲。

glBindFramebuffer(GL\_FRAMEBUFFER, fbo);

一个完整的帧缓冲需要满足以下的条件：

* 附加至少一个缓冲（颜色、深度或模板缓冲）。
* 至少有一个颜色附件(Attachment)。
* 所有的附件都必须是完整的（保留了内存）。
* 每个缓冲都应该有相同的样本数。

帧缓冲是否完整

if(glCheckFramebufferStatus(GL\_FRAMEBUFFER) == GL\_FRAMEBUFFER\_COMPLETE) // 执行胜利的舞蹈

现在我们已经创建好一个纹理了，要做的最后一件事就是将它附加到帧缓冲上了：

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, texture, 0);

创建一个渲染缓冲对象的代码和帧缓冲的代码很类似：

unsigned int rbo;

glGenRenderbuffers(1, &rbo);

类似，我们需要绑定这个渲染缓冲对象，让之后所有的渲染缓冲操作影响当前的rbo：

glBindRenderbuffer(GL\_RENDERBUFFER, rbo);

解释：

因为计算本身是消耗时间的，所以如果不采用双缓冲，则会出现图像闪烁的现象，有了双缓冲之后，显示前缓冲区，在后缓冲区绘图。然后交替交换显示，只要有一个拷贝的过程，就可以完成绘图。

这个后缓冲区，就是OpenGL（或者DX也是一样）的默认缓冲区。诸如深度测试，模板测试，颜色混合这些，成果通过测试，保留的字段（或者颜色混合得到的结果）都是默认写到这个缓冲区里面的。所谓的通过测试的片段，说白了就是写到这个后缓冲区里，并等待交换并显示于显示设备。

我对帧缓冲的理解是：为了执行某些特殊操作，实现某些特殊效果，在片段程序和后缓冲之间，增加了一个中间层，这个层就是帧缓冲。

由于GPU的计算都是并行的，所以在你计算A点的时候，你怎么保证A点周围的8个点都已经计算完了？这是不可能做到的。

如果你不用帧缓冲，直接在后缓冲区计算模糊效果，那么当你计算A-1这个点时，就会把包含A在内的八个邻接点全部污染掉。

举例说明：

1 1 1

你期望的结果是 中间的点是左右点的加和，也就是

1 2 1

好，那么是个点呢？

1111，你期望的结果是，1221

然而实际结果呢？一定不是1221

而是：1 2 3 1

数据被污染了，算2号点的时候，用的是1,3点，而算3号点的时候，用的是2,4号点。

但是如果有帧缓冲，结果就不一样了。我们先用帧缓冲，将场景全部渲染到一张和屏幕等大小的贴图上

1111

而目标，也就是最终渲染目标是，默认的缓冲区

我们先从帧缓冲区取1,3计算2号位 得到2，写到默认缓冲区

我们先从帧缓冲区取2,43计算3号位 得到2，写到默认缓冲区

最后结果是1 2 2 1，就是我们想要的结果（假定不考虑边界值的计算）

这就是帧缓冲区的作用

这就是为什么实现后期效果，需要用到帧缓冲的原因