Model类：assimp网格理解，还要附加另外的处理

Camera类：

光照：shader----物体的材质，漫反射，镜面

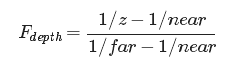
光的三个分量

多光源----平行光，点光源，手电光

I. 设置深度缓存：

glClearDepth(1.0f);// 设置深度缓存     
glEnable(GL\_DEPTH\_TEST);// 启用深度测试     
glDepthFunc(GL\_LEQUAL);// 所作深度测试的类型

若有深度矛盾，采用非线性转化对应



II. 设置模板缓冲（在片段着色器之后）：

用于给物体加边框



需要一系列对模板缓冲的操作，并且需要另加一段着色器渲染

III. 混合（比如玻璃）

一个物体的透明度被定义为它的alpha值

一．丢弃片段：在片段着色器重，if(texColor.a < 0.1) discard;

二．混合----渲染半透明纹理（无需丢弃片段了）



注意：深度测试不关系片段的透明度，故需要考虑绘制顺序

IV. 面剔除（片段着色器很昂贵）

默认情况下，**逆时针**的顶点连接顺序被定义为三角形的**正面**。

glEnable(GL\_CULL\_FACE);

**正但这只对封闭形状有效，比如草就两面都要能看到**

V. 帧缓冲

帧缓冲由于不会对屏幕产生影响，故被称为离屏渲染，需要

glBindFramebuffer(GL\_FRAMEBUFFER, 0)才能激活默认缓冲来对主窗口产生影响

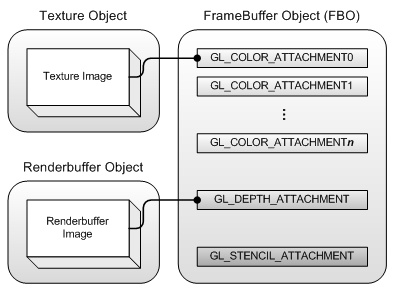
注意：1.必须往里面加入至少一个附件（颜色，深度，模板缓冲）

2.其中至少有一个是颜色附件

纹理附件：当把一个纹理附件添加到一个帧缓冲上的时候，所有渲染命令会写入到纹理上

渲染缓冲对象：存储数据、像素等，与深度和模板相关联

通常的规则是，如果你永远都不需要从特定的缓冲中进行采样，渲染缓冲对象对特定缓冲是更明智的选择。如果哪天需要从比如颜色或深度值这样的特定缓冲采样数据的话，你最好还是使用纹理附件。从执行效率角度考虑，它不会对效率有太大影响。



所以，为把场景绘制到一个单独的纹理，我们必须以下面步骤来做：

1. 使用新的绑定为激活帧缓冲的帧缓冲，像往常那样渲染场景。
2. 绑定到默认帧缓冲。
3. 绘制一个四边形，让它平铺到整个屏幕上，用新的帧缓冲的颜色缓冲作为他的纹理。

VI. 天空盒

要第一个绘制以作为背景（写入时关闭深度测试）

方便之处，可以通过反射向量对其进行采样，从而制造出物体的反射折射效果

制作水波效果（考虑反射+折射+顶点的移动）

动态环境映射： 需要先从各个面进行渲染

VII. 高级数据

使用glBufferSubData，将位置，法向量，纹理坐标等信息分开存储。

（实现基，方便调用）

VIII. 高级GLSL

gl\_FrontFacing:用于片段着色器中，给正面背面加上不同纹理（这样就要关闭面剔除）

gl\_FragCoord：读得当前片段的窗口坐标和深度值

gl\_FragDepth: 设置当前片段的深度值

如果把深度条件定义为greater或less，OpenGL会假定你只写入比当前的像素深度值的深度值大或小的。

接口块：如果从顶点着色器向片段着色器传递struct或数组之类：out VS\_OUT in VS\_OUT

**Uniform缓冲对象**（比如每个都要设置灯的参数方便化）

layout (std140) uniform Matrices { mat4 projection; mat4 view; };

layout(std140)意思是说当前定义的uniform块为它的内容使用特定的内存布局，这个声明实际上是设置uniform块布局(uniform block layout)。

GLuint lights\_index = glGetUniformBlockIndex(shaderA.Program, "Lights"); glUniformBlockBinding(shaderA.Program, lights\_index, 2);

IX.几何着色器（对输入的点，三角形处理，达到多简单图形、爆炸等效果）

对顶点着色器传入的每个点进行处理，将之拓展，放大等等

X.实例化

glDrawArrays或glDrawElements这样的函数(Draw call)过多。这样渲染顶点数据，会明显降低执行效率，这是因为OpenGL在它可以绘制你的顶点数据之前必须做一些准备工作(比如告诉GPU从哪个缓冲读取数据，以及在哪里找到顶点属性，所有这些都会使CPU到GPU的总线变慢)。

glDrawArraysInstanced(GL\_TRIANGLES, 0, 6, 100);

那么可以在顶点着色器中使用gl\_InstanceID来使用不同位置（在程序中生成并传入）

问题：会遇到uniform数据数量上限，方案：实例化数组

还要使用glVertexAttribDivisor(2, 1);

**高级光照效果：**

I. Blinn-Phong模型：稍微改进，使用半程向量计算specu light

II.Gamma校正：在片段着色器最后一行加上gamma校正计算

float gamma = 2.2;

fragColor.rgb = pow(fragColor.rgb, vec3(1.0/gamma));

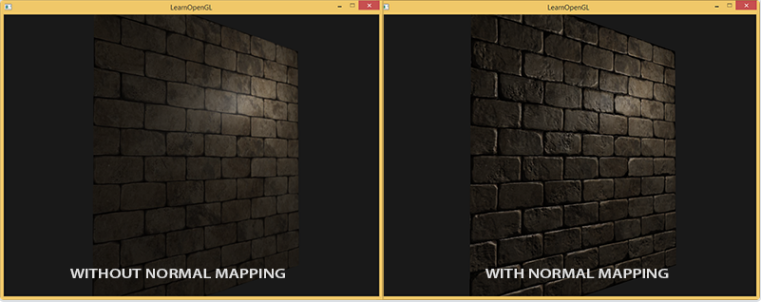


III.阴影：

使用深度贴图，渲染两次（第二次要调整视口）

一个是平行光，另一个是一定区域内点光源照射

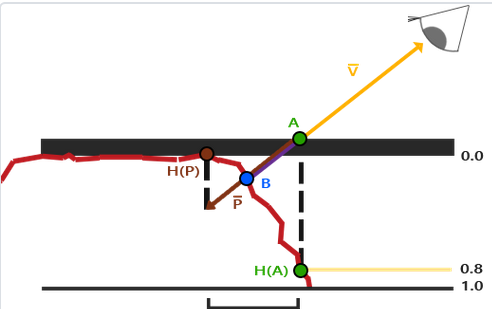
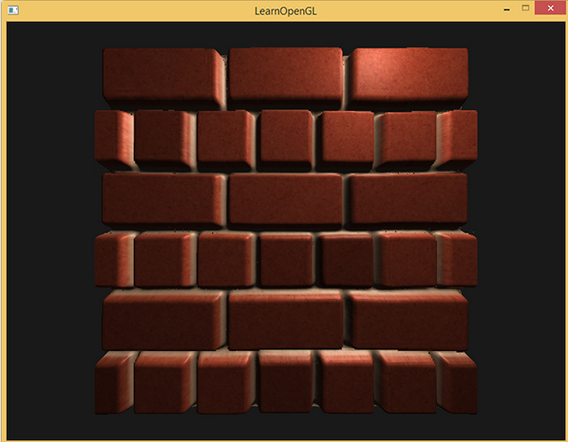
IV. 法线贴图！（用PS制作法线贴图）



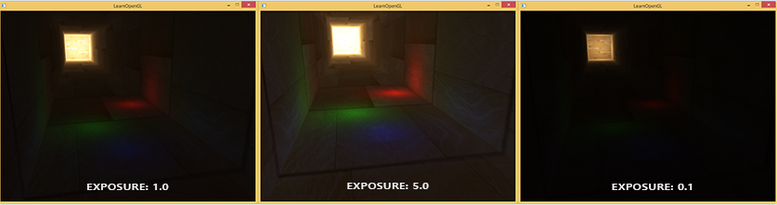
同时对应多面物体，需要用到切线空间

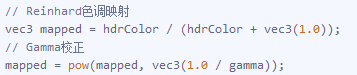
对于模型应用法线贴图（assimp里面自带函数）

V. 深度贴图（视差贴图）+多次取样提供准确度



VI. HDR颜色映射：将大色彩范围映射至0.0-1.0（人眼自带HDR功能）





需要帧缓冲，将颜色映射到纹理quad上再输出到屏幕

VII.泛光（以HDR为基础）



需要创建两个帧缓冲，然后将horizontal和vertical不断混合最后叠加到原图

IX. G缓冲（延迟着色法）：适用于成百上千光源

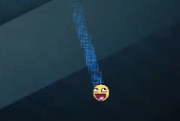
需要先将法线，反射系数等信息记录到帧缓冲二维纹理中

（若需要将光源加入，则需要结合正向渲染与延迟渲染）



X. SSAO：环境光遮蔽（较为麻烦，从点的周围采样）

XI.粒子效果



创造一些有趣的效果比如火焰,青烟,烟雾,魔法效果,炮火残渣

顶点着色器：gl\_Position = projection \* vec4((vertex.xy \* scale) + offset, 0.0, 1.0);

游戏类封装了一个初始化函数、一个更新函数、一个处理输入函数以及一个渲染函数

