1 basic

顶点数组对象：Vertex Array Object，VAO，定义的定点数据存储方式。顶点缓冲对象：Vertex Buffer Object，VBO，管理 GPU 上存储 VAO 的内存。可以一次发送尽量多的数据到 GPU 索引缓冲对象：Element Buffer Object，EBO 或 Index Buffer Object，IBO

3D 坐标转为 2D 坐标的处理过程是由 OpenGL 的图形渲染管线（Graphics Pipeline）管理。图形渲染管线可以被划分为几个阶段，每个阶段将会把前一个阶段的输出作为输入。所有这些阶段都是高度专门化的（它们都有一个特定的函数），并且很容易并行执行。

显卡具有成千上万可并行执行的处理核心，每个核心在渲染管线的各个阶段运行各自的叫做着色器 (Shader) 的小程序，从而在图形渲染管线中快速处理数据。

图元 (Primitive)：指定数据（坐标和颜色值）的渲染类型，如是一系列点还是一系列三角形等。任何一个绘制指令的调用都将把图元传递给 OpenGL。这是其中的几个：GL_POINTS $GL_TRIANGLES$ GL_LINE

以三维空间的一个三角形为例，输入的是三个 3d 点，作为顶点数据 (Vertex Data)。依次经过：1）顶点着色器 (Vertex Shader)，3d 坐标变成标准化设备坐标 (Normalized Device Coordinates, NDC),x y z 轴限制在 [-1, 1]，其余的被丢弃和剪裁，接着通过 glViewport 转化为屏幕空间坐标 (Screen-space Coordinates)；

2）图元装配 (Primitive Assembly)，装配成指定的图元形状，这里是三角形；

3) 几何着色器 (Geometry Shader)。几何着色器把图元形式的一系列顶点的集合作为输入，它可以通过产生新顶点构造出新的（或是其它的）图元来生成其他形状。例子中，它生成了另一个三角形。

4) 光栅化阶段 (Rasterization Stage)，这里它会把图元映射为最终屏幕上相应的像素，生成供片段着色器 (Fragment Shader) 使用的片段 (Fragment)。在片段着色器运行之前会执行裁切 (Clipping)。裁切会丢弃超出你的视图以外的所有像素，用来提升执行效率。OpenGL 中的一个片段是 OpenGL 渲染一个像素所需的所有数据。

5) 片段着色器的主要目的是计算一个像素的最终颜色，这也是所有 OpenGL 高级效果产生的地方。片段着色器包含 3D 场景的数据（比如光照、阴影、光的颜色等等）。

6)Alpha 测试和混合 (Blending) 阶段。最终阶段。这个阶段检测片段的对应的深度（和模板 (Stencil)）值，用它们来判断这个像素是其它物体的前面还是后面，决定是否应该丢弃。这个阶段也会检查 alpha 值（alpha 值定义了一个物体的透明度）并对物体进行混合 (Blend)。所以，即使在片段着色器中计算出来了一个像素输出的颜色，在渲染多个三角形的时候最后的像素颜色也可能完全不同。

对于大多数场合，只需要配置顶点和片段着色器。必须定义至少一个顶点着色器和一个片段着色器（因为 GPU 中没有默认的顶点/片段着色器)。几何着色器是可选的，通常按默认。

1.1 opengl 基本流程

1.2 shader

Uniform 是一种从 CPU 中的应用向 GPU 中的着色器发送数据的方式，全局变量，定义了不用会出问题

在片段着色器中进行的所谓片段插值 (Fragment Interpolation) 的结果。当渲染一个三角形时，光栅化 (Rasterization) 阶段通常会造成比原指定顶点更多的片段。光栅会根据每个片段在三角形形状上所处相对位置决定这些片段的位置。

1.3 纹理

使用纹理坐标获取纹理颜色叫做采样 (Sampling)。纹理坐标起始于 (0, 0)，也就是纹理图片的左下角，终始于 (1, 1)，即纹理图片的右上角。OpenGL 以这个顶点的纹理坐标数据去查找纹理图像上的像素，然后进行采样提取纹理像素的颜色。

1.3.1 纹理环绕方式

有 4 种，需要选择设置。需要设置 s、t、r 轴的环绕方式，border 需要设置 bordercolor

1.3.2 纹理过滤

需要进行一次设置。纹理坐标不依赖于分辨率 (Resolution)，纹理坐标所取像素的颜色，选择方式，称为纹理过滤。$GL_NEAREST$（也叫邻近过滤，Nearest Neighbor Filtering）是 OpenGL 默认的纹理过滤方式。选择中心点最接近纹理坐标的那个像素。GL_LINEAR（也叫线性过滤，(Bi)linear Filtering）它会基于纹理坐标附近的纹理像素，计算出一个插值，近似出这些纹理像素之间的颜色。通常设置纹理被缩小的时候使用邻近过滤，被放大时使用线性过滤。

1.3.3 多级渐远纹理 (Mipmap)

在纹理被缩小的情况下使用。远处的物体可能只产生很少的片段，OpenGL 从高分辨率纹理中为这些片段获取正确的颜色值就很困难. 创建完一个纹理后调用 glGenerateMipmaps 函数即可。不同多级渐远纹理级别之间指定过滤方式，消除产生不真实的生硬边界。比如: