**第二章渲染流水线笔记**

1. 应用阶段
   1. 把数据加载到显存中
   2. 设置渲染状态
      1. 什么是渲染状态？顶点着色器，片元着色器，材质，光源属性等
   3. 调用draw call
      1. 这个命令仅仅指向一个需要被渲染的图元（primitives）列表，不再包含任何材质信息
2. 几何阶段
   1. 顶点着色器（Vertex Shader）
      1. 坐标转换，逐顶点光照，输出后续阶段需要的数据
      2. 必须要把顶点从模型空间变换到齐次裁剪空间o.pos = mul(UNITY\_MVP, v.position)
      3. 顶点变换到齐次裁剪空间后，做透视除法，得到归一化设备坐标（normalized device coordinate，NDC）
      4. Unity、openGL，NDC中z分量[-1, 1]；directX，NDC中z分量[0, 1]
   2. 曲面细分着色器（Tessellation Shader）
      1. 可选着色器，用于细分图元
   3. 几何着色器（Geometry Shader）
      1. 可选着色器，用于逐图元的着色操作，或者产生更多图元
3. 裁剪（Clipping）
   1. 一个图元，顶点部分在NDC单位立方体中的被裁剪
   2. 裁剪三角图元的正面或者背面
4. 屏幕映射（Screen Mapping）
   1. 把每个图元的x和y坐标转换到屏幕坐标系（Screen Coordinates），这是一个二维坐标系
   2. 屏幕映射不会对输入的z坐标做任何处理
   3. 屏幕坐标和z坐标一起构成了窗口坐标系
   4. 这些值被传递到光栅化阶段
   5. OpenGL把屏幕左下角当成最小的窗口坐标（0， 0）；directX定义屏幕左上角为最小窗口坐标值
5. 三角形设置（triangle setup）
   1. 上一个阶段输出的是三角网格的顶点，即三角网格每条边的两个端点
   2. 为了得到整个三角形网格对像素的覆盖情况，必须计算每条边上的像素坐标
   3. 为了计算边界像素的坐标信息，需要得到三角形边界的表示方式
   4. 三角形设置，就是一个计算三角网格表示数据的过程
6. 三角形遍历（Triangle Traversal）
   1. 该阶段会检查每个像素是否被一个三角网格覆盖，如果被覆盖，会生成一个片元
   2. 三角形遍历，就是一个找到哪些像素被三角网格覆盖的过程
   3. 这个阶段也被称为扫描变换（Scan Conversion）
   4. 使用三角网格3个顶点的信息对整个覆盖区域的像素进行插值
   5. 该阶段的输出是一个片元序列
   6. 片元不是真正意义上的像素，而是包含了很多状态的集合，这些状态包括（但不限于）它的屏幕坐标，深度信息，法线，纹理坐标等，这些状态用于计算每个像素的最终颜色。
7. 片元着色器（Fragment Shader）
   1. 片元着色器可以访问到导数信息（gradient，或者说是derivative）
8. 逐片元操作（Pre-Fragment Operations）
   1. DirectX中叫输出合并阶段（Output-Merger）
   2. 决定每个片元的可见性，深度测试、模板测试
   3. 通过测试片元，颜色值和已存在颜色缓冲区中的颜色进行合并，或者说混合（Blend）
   4. 模板测试（Stencil Test）的比较函数、修改操作，可以由开发者指定
   5. 模板测试可以用于渲染阴影、轮廓
   6. 深度测试（Depth Test）的比较函数、通过测试是否写入深度缓冲区，可以由开发者指定
   7. Blend时，目标颜色指的是已经存在与颜色缓冲区中的颜色值
   8. Unity渲染流水线中，深度测试是在片元着色器之前（Early-Z）
   9. 如果在片元着色器进行透明度测试，就会导致GPU无法提前执行各种测试，禁用提前测试。这也是透明度测试会导致性能下降的原因。 ***（即使提前做了深度测试，不通过深度测试，就不做片元着色器，不进行透明度测试，这不也是对的吗？为什么要禁用Early-Z？）***
   10. 双重缓冲（Double Buffering），场景先渲染到后置缓冲（Back Buffer），完成后，GPU就会交换后置缓冲区和前置缓冲（Front Buffer）区中的内容。保证我们看到的图像总是连续的

其他：

1. GLSL跨平台是由于OpenGL没有提供编译器，由显卡驱动来完成着色器编译工作
   1. 好处在于硬件供应商完全了解自己的硬件构造，可以发挥最大作用；
   2. GLSL依赖硬件，而非操作系统。GLSL编译结果取决于硬件供应商（NVIDIA、ATI等），编译实现不尽相同，可能导致编译结果不一致
2. HLSL由微软控制着色器的编译，就算使用不同硬件，同一个着色器的编译结果一样（前提是版本相同）
   1. 但是其他平台没有编译HLSL的编译器，因此只有微软自己的产品支持HLSL
3. 在每次调用Draw Call之前，CPU需要向GPU发送很多内容，包括数据、状态、命令。在这一阶段，CPU需要完成很多工作，例如检查渲染状态等。如果Draw Call数量太多，CPU就会把大量时间花费在提交Draw Call上。
4. 参考文献：
   1. Real-time rendering
   2. What does it really mean?

5.

显存

CPU

显存

显卡

显卡驱动

OpenGL或DirectX

应用程序