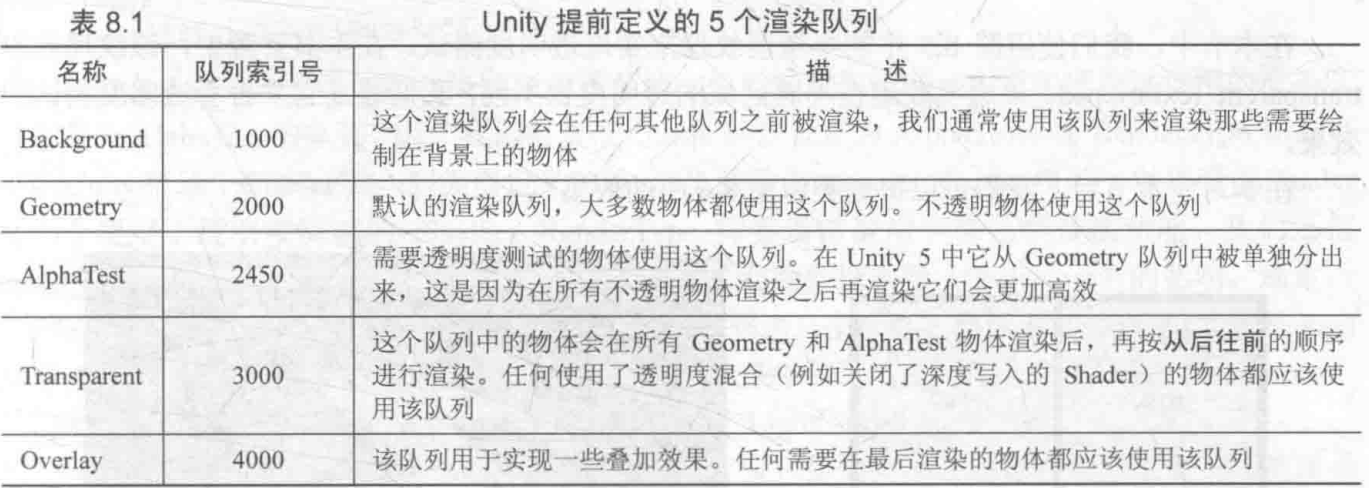
在unity中提前定义了5个渲染队列，队列索引号越小越先渲染



Unity中先将渲染队列中较前的进行渲染，然后再执行ZWrite、ZTest

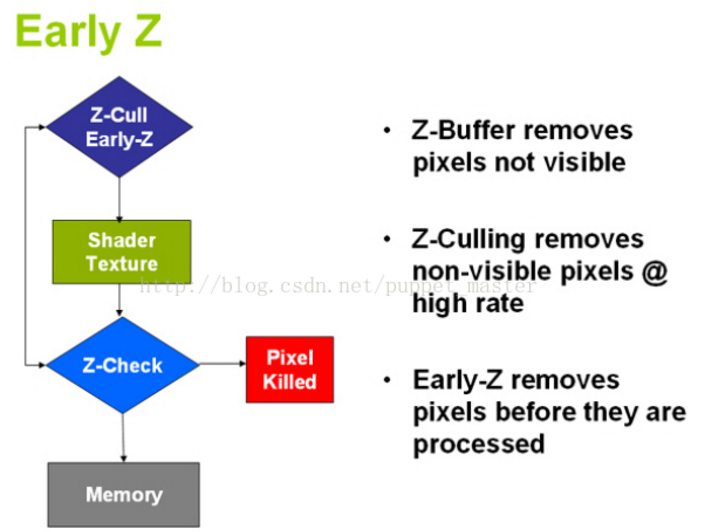
ZWrite可以取的值为：On/Off，默认值为On，代表是否要将像素的深度写入深度缓冲区（同时还要看ZTest是否通过）

当ZTest不通过时，像素的颜色值不会写入颜色缓存，像素的深度值也不会写入深度缓存

当ZTest通过时，像素的颜色值会写入颜色缓存，当ZWrite为On时，像素的深度值会写入深度缓存，当ZWrite为Off时，像素的深度值不会写入深度缓存

**先介绍一下Early-Z技术**

在传统的渲染管线中，ZTest其实是在Blending阶段，这时候进行深度测试，所有对象的像素着色器都会计算一遍，没有什么性能提升，仅仅是为了得到正确的遮挡结果，会造成大量的无用计算，因为每个像素点上肯定重叠了很多计算。因此，现代GPU中运行了Early-Z技术，在Vertex阶段和Fragment阶段之间（光栅化之后，fragment之前），进行一次深度测试，如果测试失败，就不必进行fragment阶段的计算了，因此在性能上会有很大提升。但是最终的ZTest仍然需要进行，以保证最终的遮挡关系结果正确。前面的一次主要是Z-Cull为了裁剪以达到优化的目的，后一次主要是Z-Check，为了检查，如下图：



Early-Z的实现，主要通过一个Z-pre-pass实现，简单来说，对于所有不透明的问题（透明的没有用，本身不会写深度），首先用一个超级简单的Shader进行渲染，这个shader不写颜色缓冲区，只写深度缓冲区，第二个pass关闭深度写入，开启深度测试，用正常的shader进行渲染。其中这种技术我们可以借鉴，在渲染透明物体时，因为关闭了深度写入，有时候会有其他不透明的部分挡住透明部分，而我们其实不希望他们被挡住，仅仅希望被遮挡的物体半透，这时我们就可以采用两个pass来渲染，第一个pass使用Color Mask屏蔽颜色写入，仅仅写入深度，第二个pass正常渲染半透，关闭深度写入。

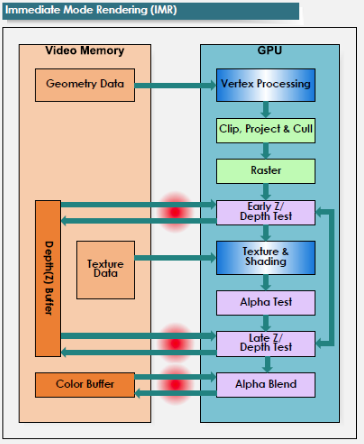
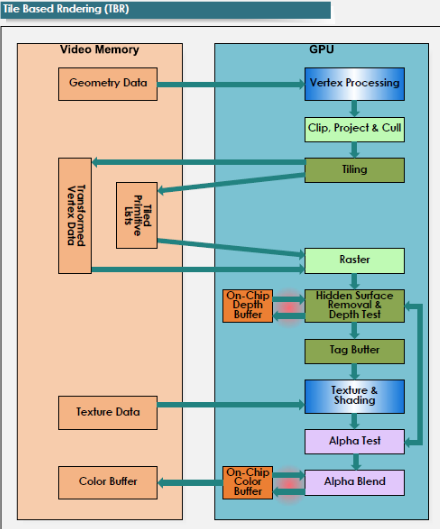
**1）由于alpha test在深度测试之前，因此alpha test和early-z是有冲突的，可能出现的一种情况是early-z测试得到的片元深度值小的并没有通过alpha test，而舍弃的片元深度值大的可能通过了alpha test，因此，对于部分显卡硬件，一旦关闭了alpha test，early-z culling就会自动开启。**

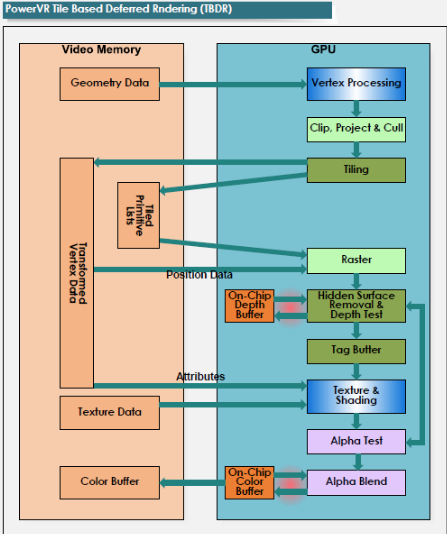
If Depth or Stencil writes are enabled, or Occlusion Queries are  
enabled, and one of the following is true:  
• Alpha-test is enabled  
• Pixel Shader kills pixels (clip(), texkil, discard)  
也就是说打开alphe test 或者有pixel clip操作，eraly z就自动关闭

**2）对于PowerVR系列而言会影响HSR（隐藏面消除），打断ISP处理覆盖同一像素的几何体（更具体一点，ISP要得到PS执行后的结果才能正确进行HSR，在这一过程中，所有覆盖了带有discard操作像素的几何体全部都要等待），具体看现在这种GPU架构基于Tile-Based Deffer Rendering光照渲染，如果开启了alpha test，那么像素的显隐在片元着色器之前不可知，也就在片元着色器之前没办法知道该像素应该对应片元的信息（法线、材质等等），就没办法对最终的像素进行光照计算。**



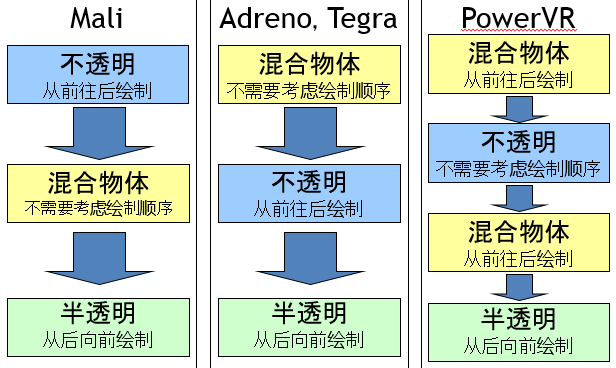
**Alpha Test（Discard）在移动平台消耗较大的原因**



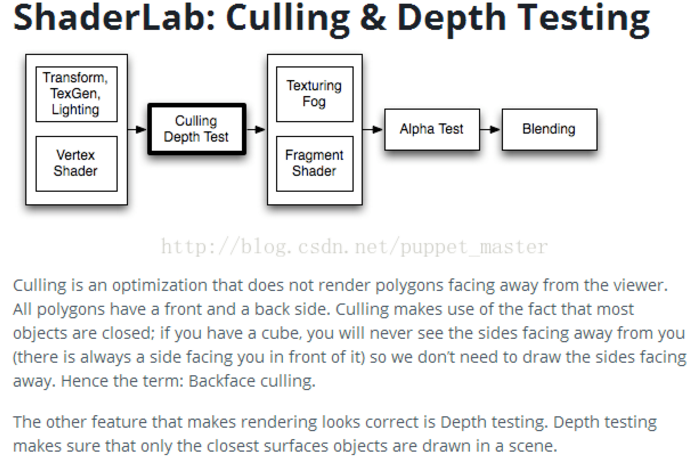
没加芯片的特性不同，根据向framebuffer的写法不同，分为tile-based和的mobile cpu，不补分来的Adreno。对于tile-based的GPU来说，一旦使用打开alpha test或者其他discard功能的指令就不能使用early-z技术了，就意味着这个fragment shader上不再只绘制一次像素了。这样会增加额外的性能消耗，**所以一般都建议不用alpha test（alpha test有discard）**，或者用alpha blend来代替

不同芯片推荐渲染顺序：



**Unity渲染顺序总结**

如果我们先绘制后面的物体，再绘制前面的物体，就会造成over draw；而通过early-z技术，我们就可以先绘制较近的物体，再绘制较远的物体（仅限于不透明物体），这样，通过先渲染前面的物体，让前面的物体先占坑，就可以让后面的物体深度测试失败，进而减少重复的fragment计算，达到优化的目的。Unity中默认应该就是按照最近距离的面进行绘制的，我们可以看看unity的官方文档显示：



从文档流程可以看出，这个Depth-Test发生在Vertex阶段和Fragment阶段之间，也就是上面所说的Early-Z优化

**简单总结一些Unity中的渲染顺序：先渲染不透明物体，顺序是总前到后，再渲染透明物体，顺序是从后到前**