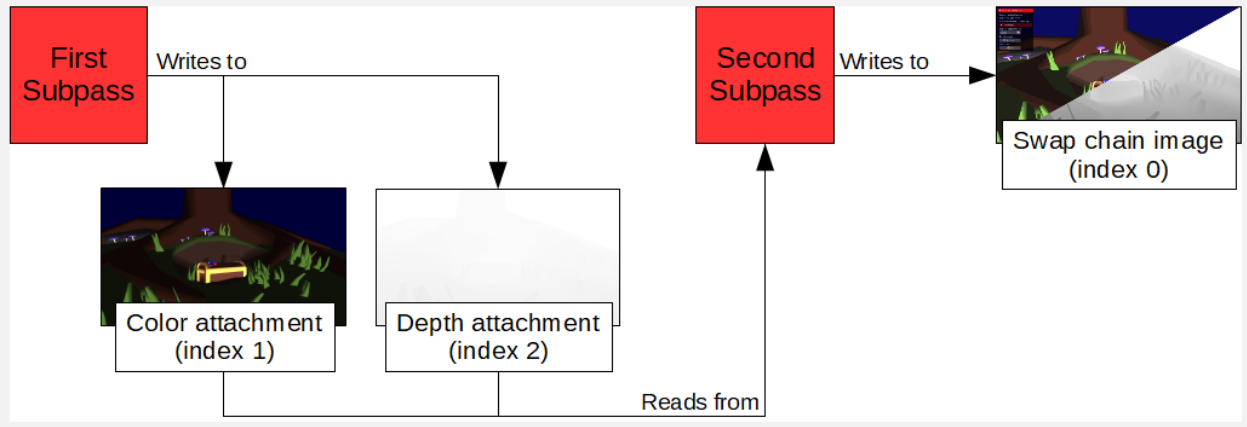
Subpass

https://www.saschawillems.de/blog/2018/07/19/vulkan-input-attachments-and-sub-passes/

https://gavinkg.github.io/ILearnVulkanFromScratch-CN/mdroot/Vulkan 进阶/Subpass/Subpass 初步.html

后一个 subpass 所使用的 attachments 的像素可以直接从前一个 subpass 的 attachments 中的对应位置读取，从而免去了收集的过程。当然需要注意的是，下一个 subpass 不能对临近像素进行采样了，只能使用同一位置的像素信息。这些 attachments 便被称为 input attachments。

在该案例中，我们在第一个 subpass 渲染出两个 input attachments：一个 color attachment 用来记录颜色，以及一个 depth attachment 用来记录深度。在第二个 subpass 我们将上述两个 input attachments 进行采样，输出处理后的结果到 swap chain image attachment，工作流是通过VkAttachmentReference实现与Attachment的关联：

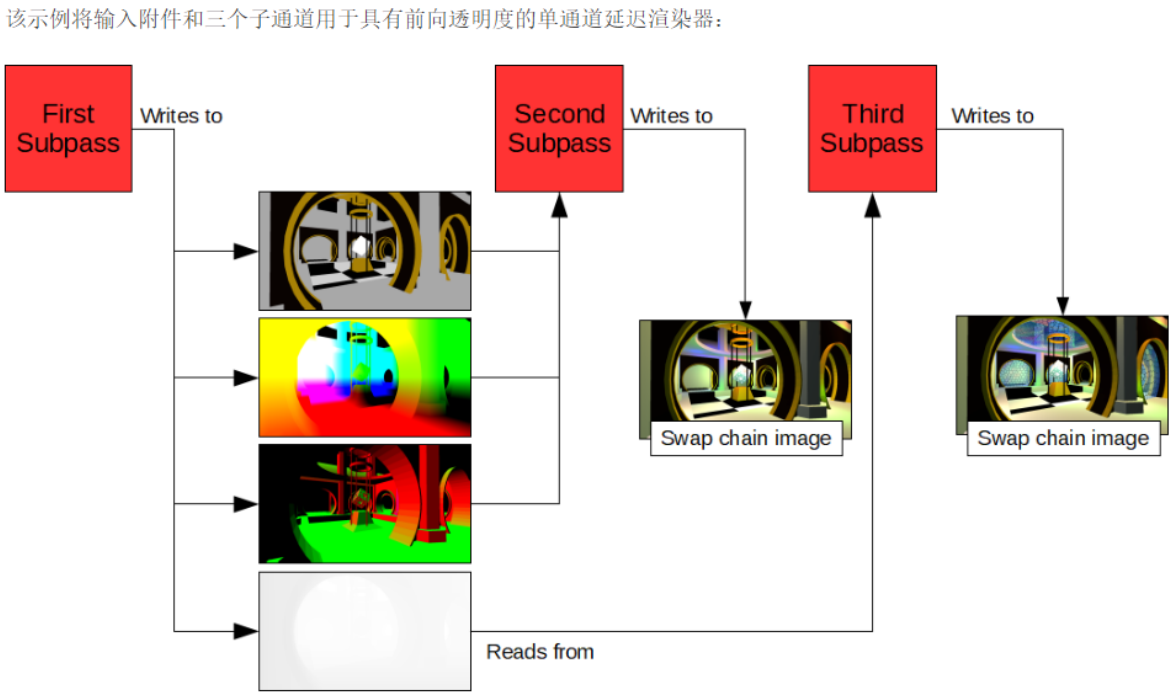


Subpass是Vulkan专门针对移动端TBDR架构设计的特性，在Vulkan中，每个渲染的RenderPass由多个Subpass组成，对于每个Subpass，我们会显式指定其输入域输出，在条件满足的时候，GPU驱动会自动将符合条件的多个subpass组合成一个renderpass，而经过组合后的单个RenderPass中的多个Subpass才是真正意义上的Subpass，否则在实际执行中，那些非真正意义上的Subpass最终会被当成单个的Renderpass来执行，并没有达到Subpass的效率。

同时，只有经过组合后的单个RenderPass中的多个Subpass，其输出结果才会被保留在On-Chip Memory上，其他的Subpass都是按照RenderPass的执行逻辑，在结束的时候将数据写回到System Memory，在下一个RenderPass再从System Memory读回。

基于上述信息，我们在实践中使用Subpass的时候，就需要注意，不是手动指定一个subpass，就能够充分利用TBDR架构了，而是需要确保subpass能够成功的被组合到一个RenderPass中才行，也就是需要注意哪些情况下，subpass不会被组合到一个RenderPass中，下面给出其中一些无法组合的情形：

- subpass之间并不存在输入输出上的依赖关系
- subpass占用了过高的On-Chip Memory，导致无法全部保留在On-Chip上，只能写入到System Memory
- Subpass使用了此前Resolve后的Attachment（导致MultiSample Num变化）
- Subpass之间具有不同的MultiSample Num
- Subpass没有设置Color/Depth-Stencil Attachment
- Subpass使用了不同的ViewMasks
- 使用了Alias Attachment



总的来说，Vulkan的Subpass是从Arm早期的PLS（Pixel Local Storage）做法中来的，其目的在于如果前后两次绘制存在输出数据-输入数据之间的转换，那么就可以考虑将第一次绘制的输出数据放到显存等Local Storage上，之后绘制的时候就可以直接取用，避免数据从Local Storage写入到内存之后再重新读回来导致的消耗。不过这种做法是有限制的，因为Local Storage（更正式的说法叫做On-Chip Cache，简称OCC）是有限的，在早期的时候，每个像素只有16bits，而后面做了升级，比如mali G71，每个pixel有128bits，那么在MSAA x2的情况下，color/depth都可以塞入到OCC上面，可以实现基本free（轻微sample平均的消耗）的抗锯齿效果，而对于MSAA x4的情况，depth就不能塞入到OCC上，导致消耗会有比较明显的增加，但是后面做了进一步升级，在mali G72以及更新的硬件中，每个pixel的OCC提高到256bit，所以在前向渲染下，即使是msaa x4，也是almost free的了。