Aliasing

“Aliasing” is a signal-processing term for “under-sampled compared with the frequencies in the signal.”

What the signal really is: what we want
What we think the signal is: too often, what we get

Sampling Interval

Sampled Points

The Display We Want
Too often, the Display We Get
The Nyquist Criterion

“Nyquist [sampling] rate is twice the maximum component frequency of the function [i.e., signal] being sampled.” -- Wikipedia

MultiSampling

Multisampling is a computer graphics technique to improve the quality of your output image by looking inside every pixel to see what the rendering is doing there.

There are two approaches to this:

1. **Supersampling**: Pick some number of unique sub-pixels within a pixel, render the image at each of these sub-pixels (including depth and stencil tests), then average them together.

2. **Multisampling**: Perform a single color render for the one pixel. Then, pick some number of unique sub-pixels within that pixel and perform depth and stencil tests there. Assign the single color to all the sub-pixels that made it through the depth and stencil tests.

Vulkan Distribution of Sampling Points within a Pixel

![Vulkan Distribution of Sampling Points within a Pixel](image)
Consider Two Triangles Whose Edges Pass Through the Same Pixel

Supersampling

Final Pixel Color = \frac{3}{8} \text{ Color sample from subpixel, } A + \frac{5}{8} \text{ Color sample from subpixel, } B

# Fragment Shader calls = 8

Multisampling

Final Pixel Color = \frac{3}{8} \text{ One color sample from } A + \frac{5}{8} \text{ One color sample from } B

# Fragment Shader calls = 2

Setting up the Image

VkPipelineMultisampleStateCreateInfo vpmsci;
vpmsci.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
vpmsci.pNext = nullptr;
vpmsci.flags = 0;
vpmsci.rasterizationSamples = VK_SAMPLE_COUNT_8_BIT;
vpmsci.sampleShadingEnable = VK_TRUE;
vpmsci.minSampleShading = 0.5f;
vpmsci.pSampleMask = (VkSampleMask *)nullptr;
vpmsci.alphaToCoverageEnable = VK_FALSE;
vpmsci.alphaToOneEnable = VK_FALSE;

VkGraphicsPipelineCreateInfo vgpci;
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;

result = vkCreateGraphicsPipelines(LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci, PALLOCATOR, OUT pGraphicsPipeline);
At least this fraction of samples will get their own fragment shader calls (as long as they pass the depth and stencil tests).

0. produces simple multisampling
(0, 1.) produces partial supersampling
1. produces complete supersampling

```c
VkPipelineMultisampleStateCreateInfo vpmsci;

vpmsci.minSampleShading = 0.5;
...
```

### Setting up the Image

```c
VkAttachmentDescription vad[2];
vad[0].format = VK_FORMAT_B8G8R8A8_SRGB;
vad[0].samples = VK_SAMPLE_COUNT_8_BIT;
vad[0].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
vad[0].storeOp = VK_ATTACHMENT_STORE_OP_STORE;
vad[0].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
vad[0].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
vad[0].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
vad[0].finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
vad[0].flags = 0;

vad[1].format = VK_FORMAT_D32_SFLOAT_S8_UINT;
vad[1].samples = VK_SAMPLE_COUNT_8_BIT;
vad[1].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
vad[1].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
vad[1].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
vad[1].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
vad[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
vad[1].finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
vad[1].flags = 0;
```

### Resolving the Image:

Converting the Multisampled Image to a VK_SAMPLE_COUNT_1_BIT image

```c
VkImageResolve vir;
vir.srcSubresource = visl;
vir.srcOffset = vo3;
vir.dstSubresource = visl;
vir.dstOffset = vo3;
vir.extent = ve3;

vkCmdResolveImage( cmdBuffer, srcImage, srcImageLayout, dstImage, dstImageLayout, 1, &vir );
```