

**Vulkan.**

**Antialiasing and Multisampling**



**Oregon State University**  
Mike Bailey  
mb@cs.oregonstate.edu




This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).




MultiSampling.pptx

mb - December 31, 2022


**Aliasing**



The Display We Want



Too often,  
the Display We Get



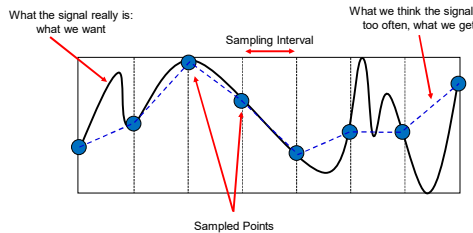
mb - December 31, 2022

**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

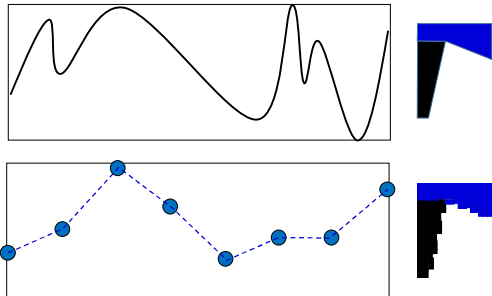



Sampled Points



mb - December 31, 2022

**Aliasing**

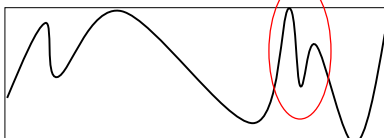





mb - December 31, 2022

**The Nyquist Criterion**

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





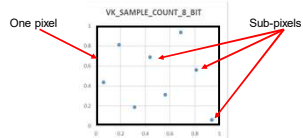
mb - December 31, 2022

**MultiSampling**

*Oversampling* 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:

- Supersampling:** Pick some number of sub-pixels within that pixel that pass the depth and stencil tests. Render the image at each of these sub-pixels.

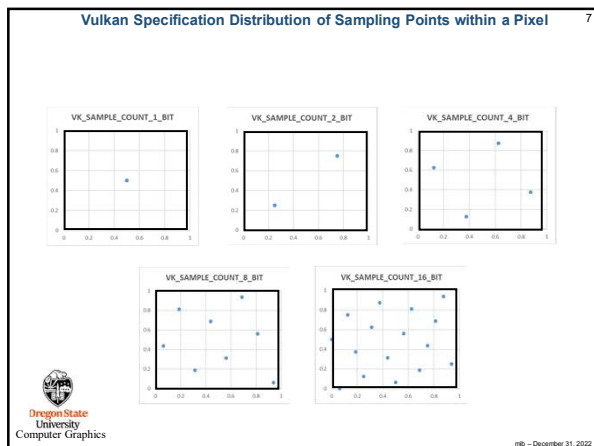


- Multisampling:** Pick some number of sub-pixels within that pixel that pass the depth and stencil tests. If any of them pass, then perform a single color render for the one pixel and assign that single color to all the sub-pixels that passed the depth and stencil tests.

The final step will be to average those sub-pixels' colors to produce one final color for this whole pixel. This is called **resolving** the pixel.



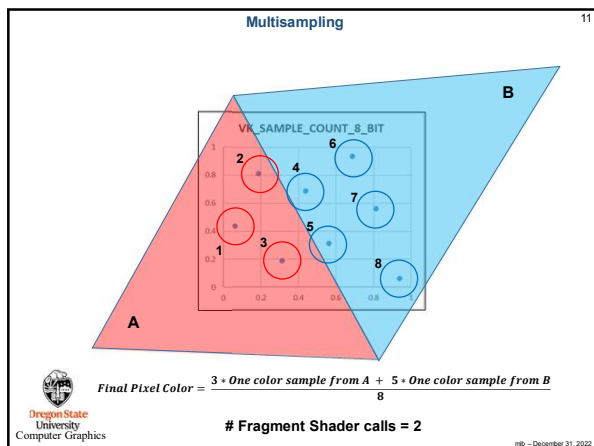
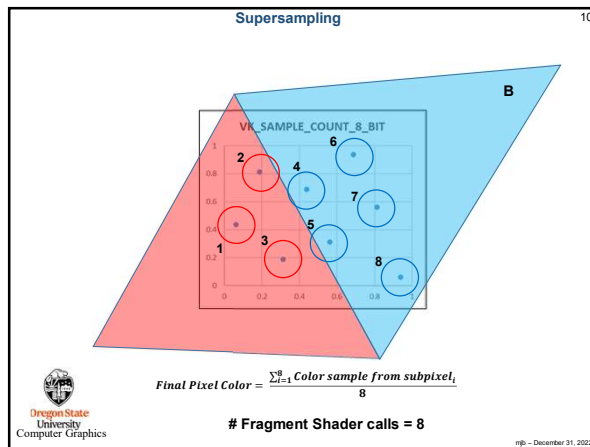
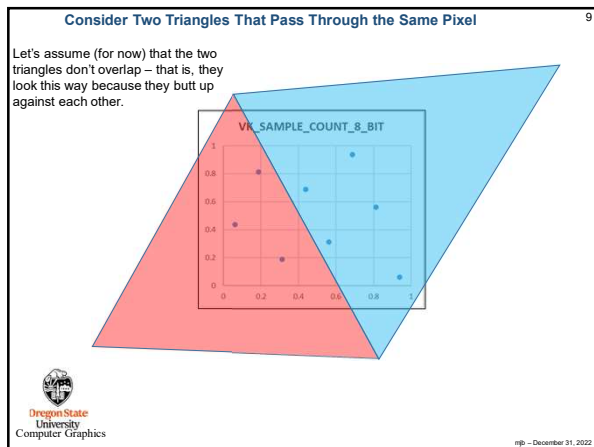
mb - December 31, 2022



### Vulkan Specification Distribution of Sampling Points within a Pixel

VK_SAMPLE_COUNT_2_BIT	VK_SAMPLE_COUNT_4_BIT	VK_SAMPLE_COUNT_8_BIT	VK_SAMPLE_COUNT_16_BIT
		(0.5625, 0.3125)	(0.5625, 0.5625)
	(0.375, 0.125)	(0.4375, 0.6875)	(0.4375, 0.3125)
(0.25, 0.25)		(0.8125, 0.5625)	(0.75, 0.4375)
	(0.875, 0.375)	(0.3125, 0.1875)	(0.1875, 0.375)
		(0.1875, 0.8125)	(0.625, 0.8125)
	(0.125, 0.625)	(0.6875, 0.1875)	(0.6875, 0.1875)
		(0.0625, 0.4375)	(0.375, 0.875)
(0.75, 0.75)		(0.0625, 0.4375)	(0.5, 0.0625)
		(0.6875, 0.9375)	(0.125, 0.125)
	(0.625, 0.875)		(0.0, 0.5)
		(0.9375, 0.0625)	(0.9375, 0.25)
			(0.875, 0.9375)
			(0.0625, 0.0)

OSU Oregon State University Computer Graphics  
mp - December 31, 2022



### Consider Two Triangles Who Pass Through the Same Pixel

Let's assume (for now) that the two triangles don't overlap – that is, they look this way because they butt up against each other.

	Multisampling	Supersampling
Blue fragment shader calls	1	5
Red fragment shader calls	1	3

OSU Oregon State University Computer Graphics  
mp - December 31, 2022

### Consider Two Triangles Who Pass Through the Same Pixel

13

Q: What if the blue triangle completely filled the pixel when it was drawn, and then the red one, which is closer to the viewer than the blue one, came along and partially filled the pixel?

A: The ideas are all still the same, but the blue one had to deal with 8 sub-pixels (instead of 5 like before). But, the red triangle came along and obsoleted 3 of those blue sub-pixels. Note that the "resolved" image will still turn out the same as before.

Oregon State University Computer Graphics

### Consider Two Triangles Who Pass Through the Same Pixel

14

What if the blue triangle completely filled the pixel when it was drawn, and then the red one, which is closer to the viewer than the blue one, came along and partially filled the pixel?

Number of Fragment Shader Calls

	Multisampling	Supersampling
Blue fragment shader calls	1	8
Red fragment shader calls	1	3

Oregon State University Computer Graphics

### Setting up the Image

15

```

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

VkGraphicsPipelineCreateInfo
  vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
  vgpci.pNext = nullptr;
  ...
  vgpci.pMultisampleState = &vpmisci;

result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
  PALLOCATOR, OUT pGraphicsPipeline );
    
```

Annotations: **vpmisci**, **vkCreateGraphicsPipelines**, **vkSampleCount\_8\_bit**, **vkTrue**, **0.5f**, **vkTrue** means to allow some sort of multisampling to take place.

Oregon State University Computer Graphics

### Setting up the Image

16

```

VkPipelineMultisampleStateCreateInfo  vpmisci;
...
vpmisci.minSampleShading = 0.5f;
...
    
```

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

Oregon State University Computer Graphics

### Setting up the Image

17

```

VkAttachmentDescription
  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;

VkAttachmentReference
  colorReference.attachment = 0;
  colorReference.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

VkAttachmentReference
  depthReference.attachment = 1;
  depthReference.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
    
```

Annotations: **vad[0]**, **vad[1]**, **vkSampleCount\_8\_bit**, **vkSampleCount\_8\_bit**, **vkAttachment\_0**, **vkAttachment\_1**, **vkImageLayout\_Color\_Attachment\_Optimal**, **vkImageLayout\_Depth\_Stencil\_Attachment\_Optimal**.

to next slide

Oregon State University Computer Graphics

### Setting up the Image

18

```

VkSubpassDescription
  vsd.flags = 0;
  vsd.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
  vsd.inputAttachmentCount = 0;
  vsd.pInputAttachments = (VkAttachmentReference*)nullptr;
  vsd.colorAttachmentCount = 1;
  vsd.pColorAttachments = &colorReference;
  vsd.pResolveAttachments = (VkAttachmentReference*)nullptr;
  vsd.pDepthStencilAttachment = &depthReference;
  vsd.pPreserveAttachments = 0;
  vsd.pPreserveAttachments = (uint32_t*)nullptr;

VkRenderPassCreateInfo
  vrpci.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
  vrpci.pNext = nullptr;
  vrpci.flags = 0;
  vrpci.attachmentCount = 2; // color and depth/stencil
  vrpci.attachments = vad;
  vrpci.subpassCount = 1;
  vrpci.subpasses = IN &vsd;
  vrpci.dependencyCount = 0;
  vrpci.pDependencies = (VkSubpassDependency*)nullptr;

result = vkCreateRenderPass( LogicalDevice, IN &vrpci, PALLOCATOR, OUT &RenderPass );
    
```

Annotations: **vsd**, **vkAttachment\_0**, **vkAttachment\_1**, **vkImageLayout\_Color\_Attachment\_Optimal**, **vkImageLayout\_Depth\_Stencil\_Attachment\_Optimal**, **vkImageLayout\_Depth\_Stencil\_Attachment\_Optimal**, **vkAttachment\_0**, **vkAttachment\_1**, **vkImageLayout\_Color\_Attachment\_Optimal**, **vkImageLayout\_Depth\_Stencil\_Attachment\_Optimal**.

from previous slide

Oregon State University Computer Graphics

19

**Resolving the Image:**  
**Converting the Multisampled Image to a VK\_SAMPLE\_COUNT\_1\_BIT image**

```
VkOffset3D
  vo3.x = 0;
  vo3.y = 0;
  vo3.z = 0;

VkExtent3D
  ve3.width = Width;
  ve3.height = Height;
  ve3.depth = 1;

VkImageSubresourceLayers
  visl.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
  visl.mipLevel = 0;
  visl.baseArrayLayer = 0;
  visl.layerCount = 1;

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

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

For the \*ImageLayout, use VK\_IMAGE\_LAYOUT\_GENERAL

Oregon State University Computer Graphics © - December 31, 2022