Antialiasing and Multisampling

Mike Bailey
mjb@cs.oregonstate.edu
“Aliasing” is a signal-processing term for “under-sampled compared with the frequencies in the signal”.

![Diagram showing Aliasing](image)
Nyquist Criterion

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

4x

16x
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 points within a pixel, render the image into each of these sub-pixels (including depth and stencil tests), then average them together.

2. **Multisampling**: Pick some number of unique points within each pixel and perform a depth and stencil render there. Then, perform a single color render for that pixel. Assign that RGBA 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

<table>
<thead>
<tr>
<th>VK_SAMPLE_COUNT_1_BIT</th>
<th>VK_SAMPLE_COUNT_2_BIT</th>
<th>VK_SAMPLE_COUNT_4_BIT</th>
<th>VK_SAMPLE_COUNT_8_BIT</th>
<th>VK_SAMPLE_COUNT_16_BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.5,0.5)</td>
<td>(0.25,0.25)</td>
<td>(0.375,0.125)</td>
<td>(0.5625,0.3125)</td>
<td>(0.5625,0.5625)</td>
</tr>
<tr>
<td>(0.75,0.75)</td>
<td>(0.875,0.375)</td>
<td>(0.4375,0.6875)</td>
<td>(0.4375,0.3125)</td>
<td></td>
</tr>
<tr>
<td>(0.125,0.625)</td>
<td>(0.625,0.875)</td>
<td>(0.3125,0.1875)</td>
<td>(0.3125,0.625)</td>
<td></td>
</tr>
<tr>
<td>(0.9375,0.0625)</td>
<td>(0.1875,0.8125)</td>
<td>(0.1875,0.375)</td>
<td>(0.625,0.8125)</td>
<td>(0.1875,0.1875)</td>
</tr>
<tr>
<td>(0.0,0.5)</td>
<td>(0.0625,0.4375)</td>
<td>(0.0625,0.1875)</td>
<td>(0.8125,0.6875)</td>
<td></td>
</tr>
<tr>
<td>(0.375,0.875)</td>
<td>(0.9375,0.0625)</td>
<td>(0.6875,0.9375)</td>
<td>(0.6875,0.1875)</td>
<td></td>
</tr>
<tr>
<td>(0.5,0.0625)</td>
<td>(0.25,0.125)</td>
<td>(0.125,0.75)</td>
<td>(0.0,0.5)</td>
<td></td>
</tr>
<tr>
<td>(0.25,0.125)</td>
<td>(0.125,0.75)</td>
<td>(0.0,0.5)</td>
<td>(0.9375,0.25)</td>
<td></td>
</tr>
<tr>
<td>(0.875,0.9375)</td>
<td>(0.0625,0.0)</td>
<td>(0.9375,0.25)</td>
<td>(0.0625,0.0)</td>
<td></td>
</tr>
</tbody>
</table>
Consider Two Triangles Whose Edges Pass Through the Same Pixel
Supersampling

Final Pixel Color = \( \frac{\sum_{i=1}^{8} \text{Color sample from subpixel}_i}{8} \)

# Fragment Shader calls = 8
Multisampling

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

# Fragment Shader calls = 2
Setting up the Image

```c
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;
    ... 
    vgpci.pMultisampleState = &vpmsci;

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

*How dense is the sampling*

*VK_TRUE means to allow some sort of multisampling to take place*
VkPipelineMultisampleStateCreateInfo vpmsci;

\[ vpmsci.minSampleShading = 0.5; \]

**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
Setting up the Image

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;

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

VkAttachmentReference depthReference;
depthReference.attachment = 1;
depthReference.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
Setting up the Image

```
VkSubpassDescription vsd;
    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.preserveAttachmentCount = 0;
    vsd.pPreserveAttachments = (uint32_t*)nullptr;

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

result = vkCreateRenderPass(LogicalDevice, IN &vrpci, PALLOCATOR, OUT &RenderPass);
```
Resolving the Image:
Converting the multisampled image to a VK_SAMPLE_COUNT_1_BIT image

```c
VLOffset3D vo3;
    vo3.x = 0;
    vo3.y = 0;
    vo3.z = 0;

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

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

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 );
```