1

Textures

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

Textures.pptx

mjb – September 18, 2018

2

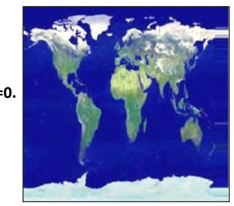
The Basic Idea

Texture mapping is a computer graphics operation in which a separate image, referred to as the **texture**, is stretched onto a piece of 3D geometry and follows it however it is transformed. This image is also known as a ***texture map***. This can be most any image. At one time, some graphics hardware required the image's pixel dimensions to be a ***power of two***. This restriction has been lifted on most (all?) graphics cards, but just to be safe… The X and Y dimensions did not need to be the *same* power of two, just a power of two. So, a 128x512 image would have been OK; a 129x511 image might not have.

Also, to prevent confusion, the texture pixels are not called ***pixels***. A pixel is a dot in the final screen image. A dot in the texture image is called a ***texture element***, or ***texel***. Similarly, to avoid terminology confusion, a texture's width and height dimensions are not called *X* and *Y*. They are called **S** and **T**. A texture map is not generally indexed by its actual resolution coordinates. Instead, it is indexed by a coordinate system that is resolution-independent. The left side is always **S=0.**, the right side is **S=1.**, the bottom is **T=0.**, and the top is **T=1.** Thus, you do not need to be aware of the texture's resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.

T=1.

S=0.



S=1.

T=0.

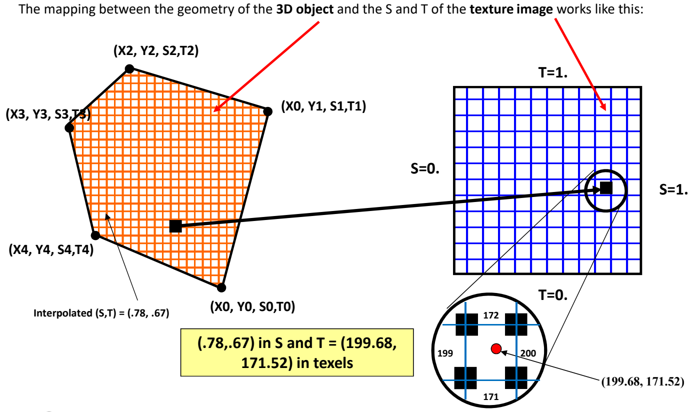
Oregon State University
Computer Graphics

mjb – September 18, 2018

3

The Basic Idea

The mapping between the geometry of the **3D object** and the S and T of the **texture image** works like this:



You specify an (s,t) pair at each vertex, along with the vertex coordinate. At the same time that the rasterizer is interpolating the coordinates, colors, etc. inside the polygon, it is also interpolating the (s,t) coordinates. Then, when it goes to draw each pixel, it uses that pixel's interpolated (s,t) to lookup a color in the texture image.

Oregon State University
Computer Graphics

mjb – September 18, 2018

4

In OpenGL terms: assigning an (s,t) to each vertex

Enable texture mapping:

```
glEnable( GL_TEXTURE_2D );
```

Draw your polygons, specifying **s** and **t** at each vertex:

```
glBegin( GL_POLYGON );
        glTexCoord2f( s0, t0 );
        glNormal3f( nx0, ny0, nz0 );
        glVertex3f( x0, y0, z0 );

        glTexCoord2f( s1, t1 );
        glNormal3f( nx1, ny1, nz1 );
        glVertex3f( x1, y1, z1 );

        . . .
glEnd( );
```

Disable texture mapping:

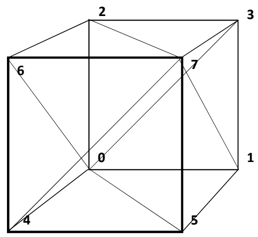
```
glDisable( GL_TEXTURE_2D );
```

Oregon State University
Computer Graphics

mjb – September 18, 2018

Triangles in an Array of Structures

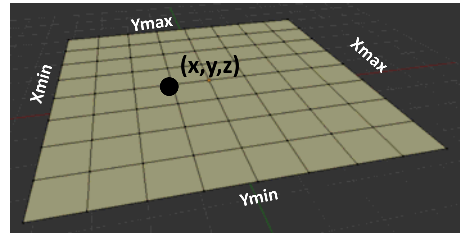

```
struct vertex
{
	glm::vec3	position;
	glm::vec3	normal;
	glm::vec3	color;
	glm::vec2	texCoord;
};

struct vertex VertexData[ ] =
{
	// triangle 0-2-3:
	// vertex #0:
	{
		{ -1., -1., -1. },
		{  0.,  0., -1. },
		{  0.,  0.,  0. },
		{  1., 0. }
	},

	// vertex #2:
	{
		{ -1.,  1., -1. },
		{  0.,  0., -1. },
		{  0.,  1.,  0. },
		{  1., 1. }
	},

	// vertex #3:
	{
		{  1.,  1., -1. },
		{  0.,  0., -1. },
		{  1.,  1.,  0. },
		{  0., 1. }
	},
```



Oregon State University Computer Graphics

Using a Texture: How do you know what (s,t) to assign to each vertex?

The easiest way to figure out what s and t are at a particular vertex is to figure out what fraction across the object the vertex is living at. For a plane,

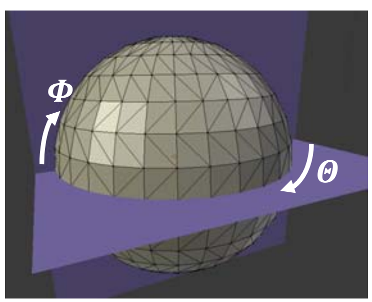


$$s = \frac{x - Xmin}{Xmax - Xmin} \qquad t = \frac{y - Ymin}{Ymax - Ymin}$$

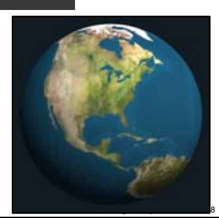
Oregon State University Computer Graphics

Using a Texture: How do you know what (s,t) to assign to each vertex?

Or, for a sphere,



$$s = \frac{\Theta - (-\pi)}{2\pi} \qquad t = \frac{\Phi - (-\frac{\pi}{2})}{\pi}$$



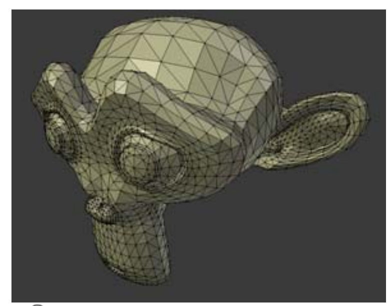
From the Sphere code:

```
s = ( lng + M_PI    ) / ( 2.*M_PI );
t  = ( lat + M_PI/2. ) / M_PI;
```

Oregon State University Computer Graphics

Using a Texture: How do you know what (s,t) to assign to each vertex?

Uh-oh. Now what? Here's where it gets tougher…,

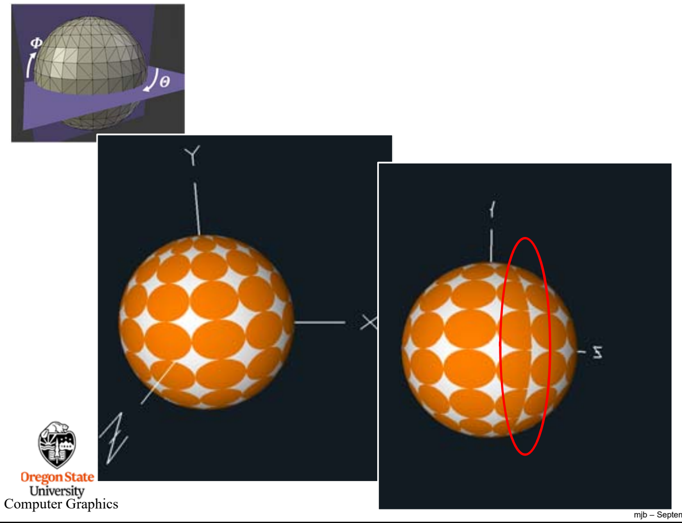


$$s = ? \qquad t = ?$$

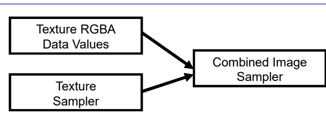
Oregon State University Computer Graphics

You really are at the mercy of whoever did the modeling…

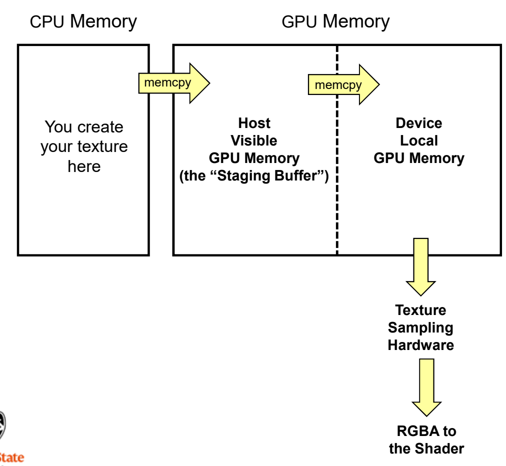
Be careful where *s* abruptly transitions from 1. back to 0.




```
VkDescriptorSetLayoutBinding          TexSamplerSet[1];
        TexSamplerSet[0].binding              = 0;
        TexSamplerSet[0].descriptorType     = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
                                        // uniform sampler2D uSampler
                                        // vec4 rgba = texture( uSampler, vST );
        TexSamplerSet[0].descriptorCount    = 1;
        TexSamplerSet[0].stageFlags           = VK_SHADER_STAGE_FRAGMENT_BIT;
        TexSamplerSet[0].pImmutableSamplers = (VkSampler *)nullptr;

    . . .

VkDescriptorImageInfo                   vdii0;
        vdii0.sampler      = MyPuppyTexture.texSampler;
        vdii0.imageView = MyPuppyTexture.texImageView;
        vdii0.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;

    . . .

VkWriteDescriptorSet                    vwds3;
        vwds3.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
        vwds3.pNext = nullptr;
        vwds3.dstSet = DescriptorSets[3];
        vwds3.dstBinding = 0;
        vwds3.dstArrayElement = 0;
        vwds3.descriptorCount = 1;
        vwds3.descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
        vwds3.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
        vwds3.pImageInfo = &vdii0;
        vwds3.pTexelBufferView = (VkBufferView *)nullptr;
```



Memory Types



Memory Types

13

NVIDIA Discrete Graphics:

11 Memory Types:
Memory 0:
Memory 1:
Memory 2:
Memory 3:
Memory 4:
Memory 5:
Memory 6:
Memory 7: DeviceLocal
Memory 8: DeviceLocal
Memory 9: HostVisible HostCoherent
Memory 10: HostVisible HostCoherent HostCached

Intel Integrated Graphics:

3 Memory Types:
Memory 0: DeviceLocal
Memory 1: DeviceLocal HostVisible HostCoherent
Memory 2: DeviceLocal HostVisible HostCoherent HostCached

Oregon State University Computer Graphics

mjb – September 18, 2018

Texture Sampling Parameters

14

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
```

OpenGL

```
VkSamplerCreateInfo                 vsci;
        vsci.magFilter = VK_FILTER_LINEAR;
        vsci.minFilter  = VK_FILTER_LINEAR;
        vsci.mipmapMode    = VK_SAMPLER_MIPMAP_MODE_LINEAR;
        vsci.addressModeU  = VK_SAMPLER_ADDRESS_MODE_REPEAT;
        vsci.addressModeV  = VK_SAMPLER_ADDRESS_MODE_REPEAT;
        vsci.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;

        . . .

result = vkCreateSampler( LogicalDevice, IN &vsci, PALLOCATOR, pTextureSampler );
```

Vulkan

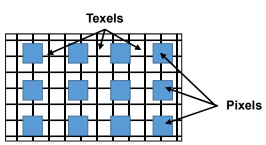
Oregon State University Computer Graphics

mjb – September 18, 2018

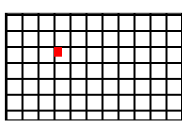
Textures' Undersampling Artifacts

15

As an object gets farther away and covers a smaller and smaller part of the screen, the **texels : pixels ratio** used in the coverage becomes larger and larger. This means that there are pieces of the texture leftover in between the pixels that are being drawn into, so that some of the texture image is not being taken into account in the final image. This means that the texture is being undersampled and could end up producing artifacts in the rendered image.



Consider a texture that consists of one red texel and all the rest white. It is easy to imagine an object rendered with that texture as ending up all *white*, with the red texel having never been included in the final image. The solution is to create lower-resolutions of the same texture so that the red texel gets included somehow in all resolution-level textures.

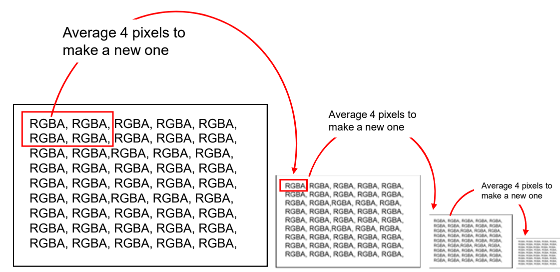


Oregon State University Computer Graphics

mjb – September 18, 2018

Texture Mip*-mapping

16



- Total texture storage is ~ 2x what it was without mip-mapping
- Graphics hardware determines which level to use based on the texels : pixels ratio.
- In addition to just picking one mip-map level, the rendering system can sample from two of them, one less that the T:P ratio and one more, and then blend the two RGBAs returned. This is known as **VK_SAMPLER_MIPMAP_MODE_LINEAR**.

* Latin: *multim in parvo*, "many things in a small place"

Oregon State University Computer Graphics

mjb – September 18, 2018

```
VkResult
Init07TextureSampler( MyTexture * pMyTexture )
{
	VkResult result;

	VkSamplerCreateInfo			vsci;
		vsci.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
		vsci.pNext = nullptr;
		vsci.flags = 0;
		vsci.magFilter = VK_FILTER_LINEAR;
		vsci.minFilter = VK_FILTER_LINEAR;
		vsci.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
		vsci.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
		vsci.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
		vsci.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
#ifdef CHOICES
VK_SAMPLER_ADDRESS_MODE_REPEAT
VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER
VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE
#endif
		vsci.mipLodBias = 0.;
		vsci.anisotropyEnable = VK_FALSE;
		vsci.maxAnisotropy = 1.;
		vsci.compareEnable = VK_FALSE;
		vsci.compareOp = VK_COMPARE_OP_NEVER;
#ifdef CHOICES
VK_COMPARE_OP_NEVER
VK_COMPARE_OP_LESS
VK_COMPARE_OP_EQUAL
VK_COMPARE_OP_LESS_OR_EQUAL
VK_COMPARE_OP_GREATER
VK_COMPARE_OP_NOT_EQUAL
VK_COMPARE_OP_GREATER_OR_EQUAL
VK_COMPARE_OP_ALWAYS
#endif
		vsci.minLod = 0.;
		vsci.maxLod = 0.;
		vsci.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
#ifdef CHOICES
VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK
VK_BORDER_COLOR_INT_TRANSPARENT_BLACK
VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK
VK_BORDER_COLOR_INT_OPAQUE_BLACK
VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE
VK_BORDER_COLOR_INT_OPAQUE_WHITE
#endif
		vsci.unnormalizedCoordinates = VK_FALSE;	// VK_TRUE means we are use raw texels as the index
							// VK_FALSE means we are using the usual 0. - 1.

	result = vkCreateSampler( LogicalDevice, IN &vsci, PALLOCATOR, OUT &pMyTexture->texSampler );
```

enable comparison against a reference value during lookups


```
VkResult
Init07TextureBuffer( INOUT MyTexture * pMyTexture)
{
	VkResult result;

	uint32_t texWidth = pMyTexture->width;;
	uint32_t texHeight = pMyTexture->height;
	unsigned char *texture = pMyTexture->pixels;
	VkDeviceSize textureSize = texWidth * texHeight * 4;		// rgba, 1 byte each

	VkImage  stagingImage;
	VkImage  textureImage;

	// *******************************************************************************
	// this first {...} is to create the staging image:
	// *******************************************************************************
	{
		VkImageCreateInfo			vici;
			vici.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
			vici.pNext = nullptr;
			vici.flags = 0;
			vici.imageType = VK_IMAGE_TYPE_2D;
			vici.format = VK_FORMAT_R8G8B8A8_UNORM;
			vici.extent.width = texWidth;
			vici.extent.height = texHeight;
			vici.extent.depth = 1;
			vici.mipLevels = 1;
			vici.arrayLayers = 1;
			vici.samples = VK_SAMPLE_COUNT_1_BIT;
			vici.tiling = VK_IMAGE_TILING_LINEAR;
#ifdef CHOICES
VK_IMAGE_TILING_OPTIMAL
VK_IMAGE_TILING_LINEAR
#endif
			vici.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT;
#ifdef CHOICES
VK_IMAGE_USAGE_TRANSFER_SRC_BIT
VK_IMAGE_USAGE_TRANSFER_DST_BIT
VK_IMAGE_USAGE_SAMPLED_BIT
VK_IMAGE_USAGE_STORAGE_BIT
VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
#endif
			vici.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
```



```
#ifdef CHOICES
VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_PREINITIALIZED
#endif
			vici.queueFamilyIndexCount = 0;
			vici.pQueueFamilyIndices = (const uint32_t *)nullptr;

		result = vkCreateImage(LogicalDevice, IN &vici, PALLOCATOR, OUT &stagingImage);	// allocated, but not filled

		VkMemoryRequirements			vmr;
		vkGetImageMemoryRequirements( LogicalDevice, IN stagingImage, OUT &vmr);

		if (Verbose)
		{
			fprintf(FpDebug, "Image vmr.size = %lld\n", vmr.size);
			fprintf(FpDebug, "Image vmr.alignment = %lld\n", vmr.alignment);
			fprintf(FpDebug, "Image vmr.memoryTypeBits = 0x%08x\n", vmr.memoryTypeBits);
			fflush(FpDebug);
		}

		VkMemoryAllocateInfo			vmai;
			vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
			vmai.pNext = nullptr;
			vmai.allocationSize = vmr.size;
			vmai.memoryTypeIndex = FindMemoryThatIsHostVisible();	// because we want to mmap it

		VkDeviceMemory				vdm;
		result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm);
		pMyTexture->vdm = vdm;

		result = vkBindImageMemory( LogicalDevice, IN stagingImage, IN vdm, 0);	// 0 = offset

		// we have now created the staging image -- fill it with the pixel data:

		VkImageSubresource			vis;
			vis.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
			vis.mipLevel = 0;
			vis.arrayLayer = 0;

		VkSubresourceLayout			vsl;
		vkGetImageSubresourceLayout( LogicalDevice, stagingImage, IN &vis, OUT &vsl);

		if (Verbose)
		{
			fprintf(FpDebug, "Subresource Layout:\n");
			fprintf(FpDebug, "\toffset = %lld\n", vsl.offset);
			fprintf(FpDebug, "\tsize = %lld\n", vsl.size);
			fprintf(FpDebug, "\trowPitch = %lld\n", vsl.rowPitch);
			fprintf(FpDebug, "\tarrayPitch = %lld\n", vsl.arrayPitch);
			fprintf(FpDebug, "\tdepthPitch = %lld\n", vsl.depthPitch);
			fflush(FpDebug);
		|
```



```
		void * gpuMemory;
		vkMapMemory( LogicalDevice, vdm, 0, VK_WHOLE_SIZE, 0, OUT &gpuMemory);
						// 0 and 0 = offset and memory map flags

		if (vsl.rowPitch == 4 * texWidth)
		{
			memcpy(gpuMemory, (void *)texture, (size_t)textureSize);
		}
		else
		{
			unsigned char *gpuBytes = (unsigned char *)gpuMemory;
			for (unsigned int y = 0; y < texHeight; y++)
			{
				memcpy(&gpuBytes[y * vsl.rowPitch], &texture[4 * y * texWidth], (size_t)(4*texWidth) );
			}
		}

		vkUnmapMemory( LogicalDevice, vdm);

	}
	// *******************************************************************************
```

Oregon State University
Computer Graphics

```
// ********************************************************************************
// this second {...} is to create the actual texture image:
// ********************************************************************************
{
	VkImageCreateInfo			vici;
		vici.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
		vici.pNext = nullptr;
		vici.flags = 0;
		vici.imageType = VK_IMAGE_TYPE_2D;
		vici.format = VK_FORMAT_R8G8B8A8_UNORM;
		vici.extent.width  = texWidth;
		vici.extent.height = texHeight;
		vici.extent.depth = 1;
		vici.mipLevels = 1;
		vici.arrayLayers = 1;
		vici.samples = VK_SAMPLE_COUNT_1_BIT;
		vici.tiling = VK_IMAGE_TILING_OPTIMAL;
		vici.usage = VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT;
				// because we are transferring into it and will eventual sample from it
		vici.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
		vici.initialLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vici.queueFamilyIndexCount = 0;
		vici.pQueueFamilyIndices = (const uint32_t *)nullptr;

	result = vkCreateImage(LogicalDevice, IN &vici, PALLOCATOR, OUT &textureImage); // allocated, but not filled

	VkMemoryRequirements			vmr;
	vkGetImageMemoryRequirements( LogicalDevice, IN textureImage, OUT &vmr);

	if( Verbose )
	{
		fprintf( FpDebug, "Texture vmr.size = %lld\n", vmr.size );
		fprintf( FpDebug, "Texture vmr.alignment = %lld\n", vmr.alignment );
		fprintf( FpDebug, "Texture vmr.memoryTypeBits = 0x%08x\n", vmr.memoryTypeBits );
		fflush( FpDebug );
	}
	VkMemoryAllocateInfo			vmai;
		vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
		vmai.pNext = nullptr;
		vmai.allocationSize = vmr.size;
		vmai.memoryTypeIndex = FindMemoryThatIsDeviceLocal( );  // because we want to sample from it

	VkDeviceMemory				vdm;
	result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm);

	result = vkBindImageMemory( LogicalDevice, IN textureImage, IN vdm, 0 );	// 0 = offset
}
// ********************************************************************************
```



```
// copy pixels from the staging image to the texture:

VkCommandBufferBeginInfo		vcbbi;
	vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
	vcbbi.pNext = nullptr;
	vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
	vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

result = vkBeginCommandBuffer( TextureCommandBuffer, IN &vcbbi);

// ********************************************************************************
// transition the staging buffer layout:
// ********************************************************************************
{
	VkImageSubresourceRange			visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageMemoryBarrier			vimb;
		vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
		vimb.pNext = nullptr;
		vimb.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vimb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL;
		vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.image = stagingImage;
		vimb.srcAccessMask = 0;
		vimb.dstAccessMask = 0T;
		vimb.subresourceRange = visr;

	vkCmdPipelineBarrier( TextureCommandBuffer,
			VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT, VK_PIPELINE_STAGE_FRAG<EMT_SHADER_BIT, 0,
			0, (VkMemoryBarrier *)nullptr,
			0, (VkBufferMemoryBarrier *)nullptr,
			1, IN &vimb );
}
// ********************************************************************************
```



```
// ********************************************************************************
// transition the texture buffer layout:
// ********************************************************************************
{
	VkImageSubresourceRange			visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageMemoryBarrier			vimb;
		vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
		vimb.pNext = nullptr;
		vimb.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vimb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
		vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.image = textureImage;
		vimb.srcAccessMask = 0;
		vimb.dstAccessMask = 0;
		vimb.subresourceRange = visr;

	vkCmdPipelineBarrier( TextureCommandBuffer,
		VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT, VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, 0,
		0, (VkMemoryBarrier *)nullptr,
		0, (VkBufferMemoryBarrier *)nullptr,
		1, IN &vimb);

	// now do the final image transfer:

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

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

	VkExtent3D				ve3;
		ve3.width = texWidth;
		ve3.height = texHeight;
		ve3.depth = 1;
```



```
	VkImageCopy				vic;
		vic.srcSubresource = visl;
		vic.srcOffset = vo3;
		vic.dstSubresource = visl;
		vic.dstOffset = vo3;
		vic.extent = ve3;

	vkCmdCopyImage(TextureCommandBuffer,
		stagingImage, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL,
		textureImage, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1, IN &vic);
}
// ********************************************************************************
```

```
// ******************************************************************************
// transition the texture buffer layout a second time:
// ******************************************************************************
	{
		VkImageSubresourceRange		visr;
			visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
			visr.baseMipLevel = 0;
			visr.levelCount = 1;
			visr.baseArrayLayer = 0;
			visr.layerCount = 1;

		VkImageMemoryBarrier			vimb;
			vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
			vimb.pNext = nullptr;
			vimb.oldLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
			vimb.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
			vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
			vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
			vimb.image = textureImage;
			vimb.srcAccessMask = 0;
			vimb.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
			vimb.subresourceRange = visr;

		vkCmdPipelineBarrier(TextureCommandBuffer,
			VK_PIPELINE_STAGE_TRANSFER_BIT, VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, 0,
			0, (VkMemoryBarrier *)nullptr,
			0, (VkBufferMemoryBarrier *)nullptr,
			1, IN &vimb);
	}
	// ******************************************************************************

	result = vkEndCommandBuffer( TextureCommandBuffer );

	VkSubmitInfo				vsi;
		vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
		vsi.pNext = nullptr;
		vsi.commandBufferCount = 1;
		vsi.pCommandBuffers = &TextureCommandBuffer;
		vsi.waitSemaphoreCount = 0;
		vsi.pWaitSemaphores = (VkSemaphore *)nullptr;
		vsi.signalSemaphoreCount = 0;
		vsi.pSignalSemaphores = (VkSemaphore *)nullptr;
		vsi.pWaitDstStageMask = (VkPipelineStageFlags *)nullptr;

	result = vkQueueSubmit( Queue, 1, IN &vsi, VK_NULL_HANDLE );
	result = vkQueueWaitIdle( Queue );
```

Oregon State University Computer Graphics


```
	// create an image view for the texture image:

	VkImageSubresourceRange		visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageViewCreateInfo			vivci;
		vivci.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
		vivci.pNext = nullptr;
		vivci.flags = 0;
		vivci.image = textureImage;
		vivci.viewType = VK_IMAGE_VIEW_TYPE_2D;
		vivci.format = VK_FORMAT_R8G8B8A8_UNORM;
		vivci.components.r = VK_COMPONENT_SWIZZLE_R;
		vivci.components.g = VK_COMPONENT_SWIZZLE_G;
		vivci.components.b = VK_COMPONENT_SWIZZLE_B;
		vivci.components.a = VK_COMPONENT_SWIZZLE_A;
		vivci.subresourceRange = visr;

	result = vkCreateImageView( LogicalDevice, IN &vivci, PALLOCATOR, OUT &pMyTexture->texImageView);

	return result;
}
```

Note that, at this point, the CPU buffer and the GPU Staging Buffer are no longer needed, and can be destroyed.

Oregon State University Computer Graphics

Reading in a Texture from a BMP File


```
typedef struct MyTexture
{
	uint32_t			width;
	uint32_t			height;
	VkImage			texImage;
	VkImageView		texImageView;
	VkSampler			texSampler;
	VkDeviceMemory		vdm;
} MyTexture;

•••

MyTexture	MyPuppyTexture;
```

```
result = Init06TextureBufferAndFillFromBmpFile ( "puppy.bmp", &MyTexturePuppy);
Init06TextureSampler( &MyPuppyTexture.texSampler );
```

This function can be found in the **sample.cpp** file. The BMP file needs to be created by something that writes uncompressed 24-bit color BMP files, or was converted to the uncompressed BMP format by a tool such as ImageMagick's *convert*, Adobe *Photoshop,* or GNU's *GIMP*.

Oregon State University Computer Graphics

Anisotropic Texture Filtering

https://en.wikipedia.org/wiki/Anisotropic_filtering

Oregon State University Computer Graphics