



Vulkan.

Textures



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

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



Oregon State University Computer Graphics

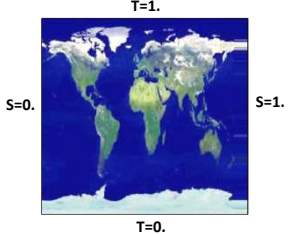

mjb - December 29, 2022

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**.

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.

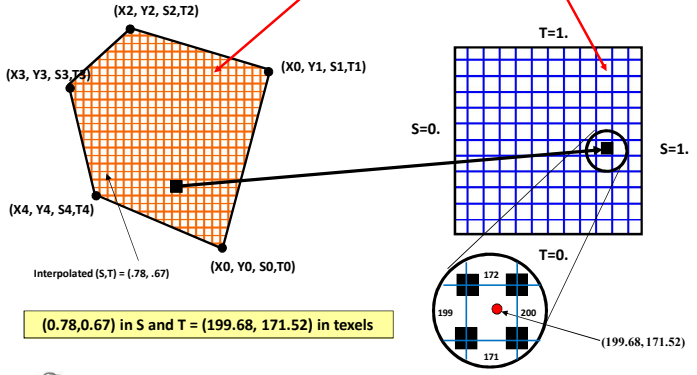



Oregon State University Computer Graphics

mjb - December 29, 2022


The Basic Idea

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



(0.78,0.67) in S and T = (199.68, 171.52) in texels

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 look up a color in the texture image.



Oregon State University Computer Graphics

mjb - December 29, 2022

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 - December 29, 2022

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. },
    },
};
    
```

Computer Graphics
mjb - December 29, 2022

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,

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

Oregon State University
Computer Graphics
mjb - December 29, 2022

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

Or, for a sphere,

$$s = \frac{\theta - (-\pi)}{2\pi} \quad t = \frac{\Phi - (-\frac{\pi}{2})}{\pi}$$

$$s = (\text{lng} + M_PI) / (2 * M_PI);$$

$$t = (\text{lat} + M_PI/2.) / M_PI;$$

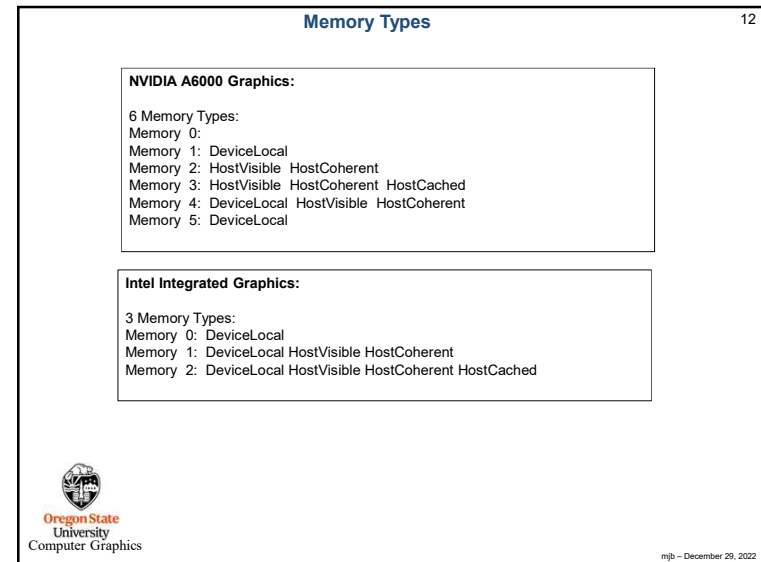
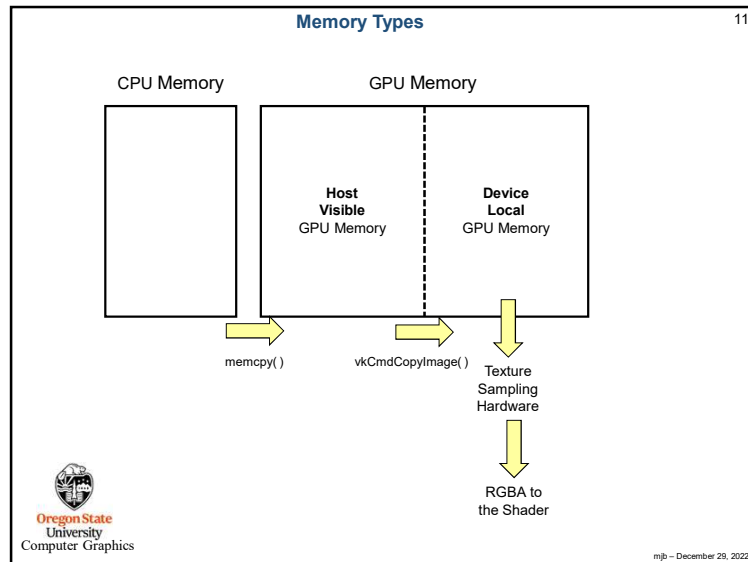
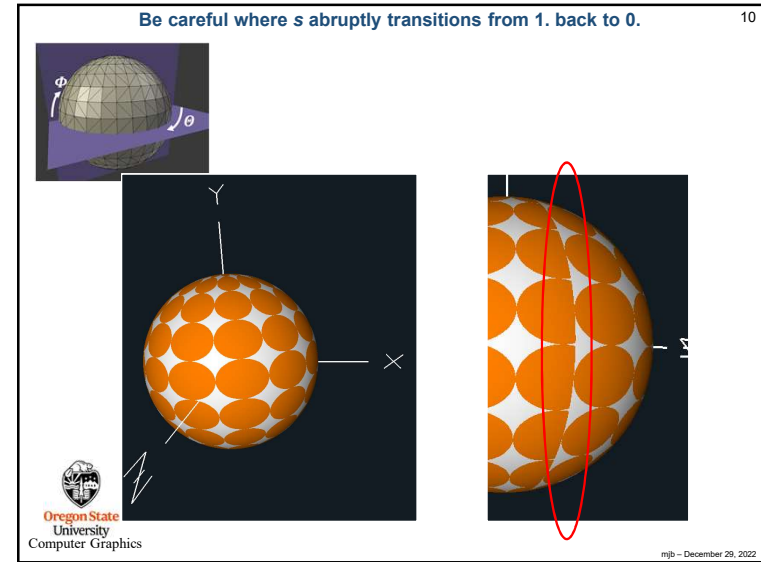
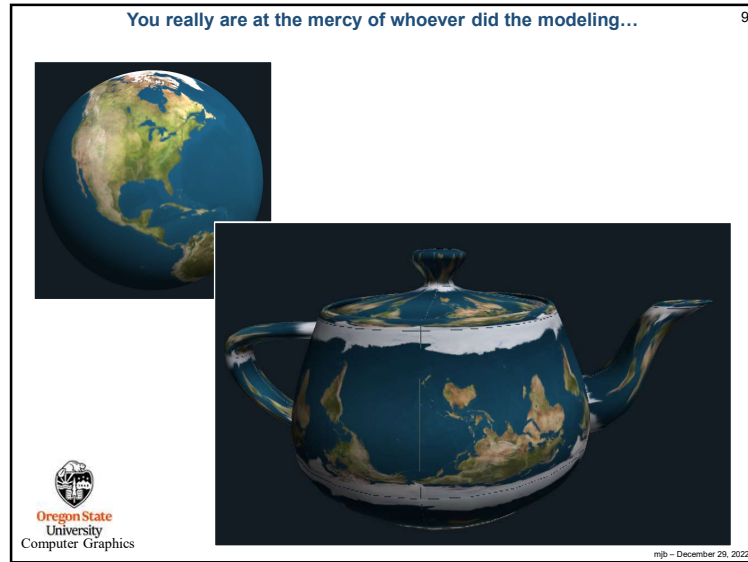
Oregon State University
Computer Graphics
mjb - December 29, 2022

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...

$s = ?$ $t = ?$

Oregon State University
Computer Graphics
mjb - December 29, 2022



Something I've Found Useful


13

I find it handy to encapsulate texture information in a struct, just like I do with buffer information:

```

// holds all the information about a data buffer so it can be encapsulated in one variable:
typedef struct MyBuffer
{
    VkDataBuffer    buffer;
    VkDeviceMemory vdm;
    VkDeviceSize    size;
} MyBuffer;

// holds all the information about a texture so it can be encapsulated in one variable:
typedef struct MyTexture
{
    uint32_t        width;
    uint32_t        height;
    unsigned char * pixels;
    VkImage          texImage;
    VkImageView      texImageView;
    VkSampler        texSampler;
    VkDeviceMemory  vdm;
} MyTexture;
    
```



mb - December 29, 2022

Texture Sampling Parameters

14

OpenGL

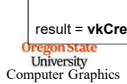
```

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 );
    
```

Vulkan

```

MyTexture MyPuppyTexture;
...
VkSamplerCreateInfo
    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, OUT &MyPuppyTexture->texSampler);
    
```

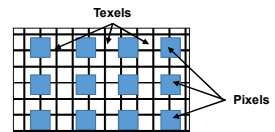


mb - December 29, 2022

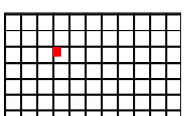

Textures' Undersampling Artifacts

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.



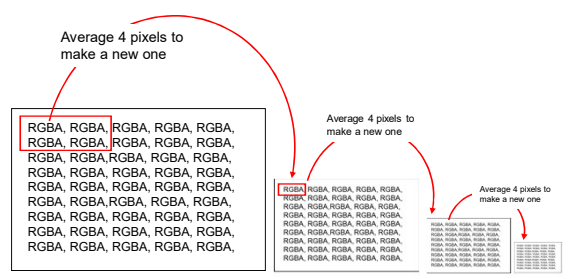
Consider a texture that consists of one red texel and all the rest white. It is easy to imagine an object rendered with that texture as ending up all *white*, with the red texel having never been included in the final image. The solution is to create lower-resolutions of the same texture so that the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


mb - December 29, 2022

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 Texture:Pixel 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"
mb - December 29, 2022

```

VkResult
Init0TextureSampler(MyTexture * pMyTexture)
{
    VkResult result;

    VkSamplerCreateInfo
    {
        vci.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
        vci.pNext = nullptr;
        vci.flags = 0;
        vci.magFilter = VK_FILTER_LINEAR;
        vci.minFilter = VK_FILTER_LINEAR;
        vci.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
        vci.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
        vci.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
        vci.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
    };

    #ifdef CHOICES
    vci.samplerAddressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
    vci.samplerAddressModeV = VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT;
    vci.samplerAddressModeW = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE;
    #endif

    vci.mipmapLodBias = 0;
    vci.anisotropyEnable = VK_FALSE;
    vci.maxAnisotropy = 1;
    vci.compareEnable = VK_FALSE;
    vci.compareOp = VK_COMPARE_OP_NEVER;

    #ifdef CHOICES
    vci.compareOp = VK_COMPARE_OP_NEVER;
    vci.compareOpLess = VK_COMPARE_OP_LESS;
    vci.compareOpEqual = VK_COMPARE_OP_EQUAL;
    vci.compareOpLessOrEqual = VK_COMPARE_OP_LESS_OR_EQUAL;
    vci.compareOpGreater = VK_COMPARE_OP_GREATER;
    vci.compareOpNotEqual = VK_COMPARE_OP_NOT_EQUAL;
    vci.compareOpGreaterOrEqual = VK_COMPARE_OP_GREATER_OR_EQUAL;
    vci.compareOpAlways = VK_COMPARE_OP_ALWAYS;
    #endif

    vci.minLod = 0;
    vci.maxLod = 0;
    vci.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;

    #ifdef CHOICES
    vci.borderColor = VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK;
    vci.borderColor = VK_BORDER_COLOR_INT_TRANSPARENT_BLACK;
    vci.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
    vci.borderColor = VK_BORDER_COLOR_INT_OPAQUE_BLACK;
    vci.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE;
    vci.borderColor = VK_BORDER_COLOR_INT_OPAQUE_WHITE;
    #endif

    vci.unnormalizedCoordinates = VK_FALSE; // VK_TRUE means we are using raw texels as the index
                                           // VK_FALSE means we are using the usual 0..1.

    result = vkCreateSampler(LogicalDevice, IN &vci, PALLOCATOR, OUT &MyPuppyTexture->texSampler);
}

```

17

```

VkResult
Init0TextureBuffer(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
        {
            vci.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
            vci.pNext = nullptr;
            vci.flags = 0;
            vci.imageType = VK_IMAGE_TYPE_2D;
            vci.format = VK_FORMAT_R8G8B8A8_UNORM;
            vci.extent.width = texWidth;
            vci.extent.height = texHeight;
            vci.extent.depth = 1;
            vci.mipLevels = 1;
            vci.arrayLayers = 1;
            vci.samples = VK_SAMPLE_COUNT_1_BIT;
            vci.tiling = VK_IMAGE_TILING_LINEAR;
        };

        #ifdef CHOICES
        vci.imageTiling = VK_IMAGE_TILING_OPTIMAL;
        vci.imageTiling = VK_IMAGE_TILING_LINEAR;
        #endif

        vci.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT;

        #ifdef CHOICES
        vci.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT;
        vci.usage = VK_IMAGE_USAGE_TRANSFER_DST_BIT;
        vci.usage = VK_IMAGE_USAGE_SAMPLED_BIT;
        vci.usage = VK_IMAGE_USAGE_STORAGE_BIT;
        vci.usage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
        vci.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;
        vci.usage = VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT;
        vci.usage = VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT;
        #endif

        vci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
    };
}

```

18

```

#ifdef CHOICES
VK_IMAGE_LAYOUT_UNDEFINED;
VK_IMAGE_LAYOUT_PREINITIALIZED;
#endif

vci.queueFamilyIndexCount = 0;
vci.pQueueFamilyIndices = (const uint32_t *)nullptr;

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

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

if (Verbose)
{
    fprintf(FpDebug, "Image vmr.size = %ldx%ld", vmr.size);
    fprintf(FpDebug, "Image vmr.alignment = %ld", vmr.alignment);
    fprintf(FpDebug, "Image vmr.memoryTypeBits = %0x08x", vmr.memoryTypeBits);
    fflush(FpDebug);
}

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

VkDeviceMemory
{
    vdm;
};

result = vkAllocateMemory(LogicalDevice, IN &vma, 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
{
    vrs.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
    vrs.mipLevel = 0;
    vrs.arrayLayer = 0;
};

VkSubresourceLayout
{
    vsl;
};

vkGetImageSubresourceLayout(LogicalDevice, stagingImage, IN &vrs, OUT &vsl);

if (Verbose)
{
    fprintf(FpDebug, "Subresource Layout:\n");
    fprintf(FpDebug, "  offset = %ld", vsl.offset);
    fprintf(FpDebug, "  size = %ld", vsl.size);
    fprintf(FpDebug, "  rowPitch = %ld", vsl.rowPitch);
    fprintf(FpDebug, "  arrayPitch = %ld", vsl.arrayPitch);
    fprintf(FpDebug, "  depthPitch = %ld", vsl.depthPitch);
    fflush(FpDebug);
}

```

19

```

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);
}

```

20

21

```

// .....
// 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_R32G8B8_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 eventually sample from it
vici.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
vici.initialLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
vici.queueFamilyIndices = 0;
vici.queueFamilyIndices = (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 = %ld\n", vmr.size);
    fprintf(FpDebug, "Texture vmr.alignment = %ld\n", vmr.alignment);
    fprintf(FpDebug, "Texture vmr.memoryTypeBits = %x\n", vmr.memoryTypeBits);
    fflush(FpDebug);
}

VkMemoryAllocateInfo vmai;
vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
vmai.pNext = nullptr;
vmai.allocatorSize = 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
// .....

```

Oregon State University Computer Graphics

mjb - December 29, 2022

22

```

// 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 vsr;
vsr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
vsr.baseMipLevel = 0;
vsr.levelCount = 1;
vsr.baseArrayLayer = 0;
vsr.layerCount = 1;

VkImageMemoryBarrier vmb;
vmb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
vmb.pNext = nullptr;
vmb.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
vmb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL;
vmb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vmb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vmb.image = stagingImage;
vmb.srcAccessMask = VK_ACCESS_HOST_WRITE_BIT;
vmb.dstAccessMask = 0;
vmb.subresourceRange = vsr;

VkCmdPipelineBarrier vcpb;
vcpb.sType = VK_STRUCTURE_TYPE_PIPELINE_BARRIER;
vcpb.pNext = nullptr;
vcpb.flags = VK_PIPELINE_STAGE_HOST_BIT | VK_PIPELINE_STAGE_HOST_BIT, 0;
vcpb.srcAccessMask = 0;
vcpb.dstAccessMask = 0;
vcpb.srcStageMask = 0;
vcpb.dstStageMask = 0;
vcpb.srcQueueFamilyIndex = 1;
vcpb.dstQueueFamilyIndex = 1;
vcpb.subresourceRange = vmb.subresourceRange;
// .....

```

Oregon State University Computer Graphics

mjb - December 29, 2022

23

```

// .....
// transition the texture buffer layout:
// .....
VkImageSubresourceRange vsr;
vsr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
vsr.baseMipLevel = 0;
vsr.levelCount = 1;
vsr.baseArrayLayer = 0;
vsr.layerCount = 1;

VkImageMemoryBarrier vmb;
vmb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
vmb.pNext = nullptr;
vmb.oldLayout = VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL;
vmb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
vmb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vmb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vmb.image = textureImage;
vmb.srcAccessMask = 0;
vmb.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
vmb.subresourceRange = vsr;

VkCmdPipelineBarrier vcpb;
vcpb.sType = VK_STRUCTURE_TYPE_PIPELINE_BARRIER;
vcpb.pNext = nullptr;
vcpb.flags = VK_PIPELINE_STAGE_TRANSFER_BIT | VK_PIPELINE_STAGE_TRANSFER_BIT, 0;
vcpb.srcAccessMask = 0;
vcpb.dstAccessMask = 0;
vcpb.srcStageMask = 0;
vcpb.dstStageMask = 0;
vcpb.srcQueueFamilyIndex = 1;
vcpb.dstQueueFamilyIndex = 1;
vcpb.subresourceRange = vmb.subresourceRange;

// now do the final image transfer:
VkImageSubresourceLayers vsl;
vsl.sType = VK_STRUCTURE_TYPE_IMAGE_SUBRESOURCE_LAYERS;
vsl.pNext = nullptr;
vsl.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
vsl.baseMipLevel = 0;
vsl.levelCount = 1;
vsl.baseArrayLayer = 0;
vsl.layerCount = 1;

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

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

```

Oregon State University Computer Graphics

mjb - December 29, 2022

24

```

VkImageCopy vic;
vic.srcSubresource = vsr;
vic.srcOffset = vo3;
vic.dstSubresource = vsl;
vic.dstOffset = vo3;
vic.extent = ve3;

VkCmdCopyImage vcki;
vcki.sType = VK_STRUCTURE_TYPE_COPY_COMMAND_BUFFER_INFO;
vcki.pNext = nullptr;
vcki.srcImage = stagingImage;
vcki.dstImage = textureImage;
vcki.srcSubresource = vsr;
vcki.dstSubresource = vsl;
vcki.extent = ve3;
vcki.offset = vo3;
vcki.flags = 0;
// .....

```

Oregon State University Computer Graphics

mjb - December 29, 2022

25

```

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

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

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
mb - December 29, 2022

26

```

// create an image view for the texture image:
// (an "image view" is used to indirectly access an image)

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

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

result = vkCreateImageView( LogicalDevice, IN &vvc, PALLOCATOR, OUT &MyTexture->texImageView );

return result;
    
```

Diagram: Access to an Image ↔ Image View ↔ The Actual Image Data

Color Channels: 8 bits Red, 8 bits Green, 8 bits Blue, 8 bits Alpha

Note that, at this point, the Staging Buffer is no longer needed, and can be destroyed.

Oregon State University Computer Graphics
mb - December 29, 2022

27


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 ( "puppy1.bmp", &MyPuppyTexture );
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
mb - December 29, 2022