


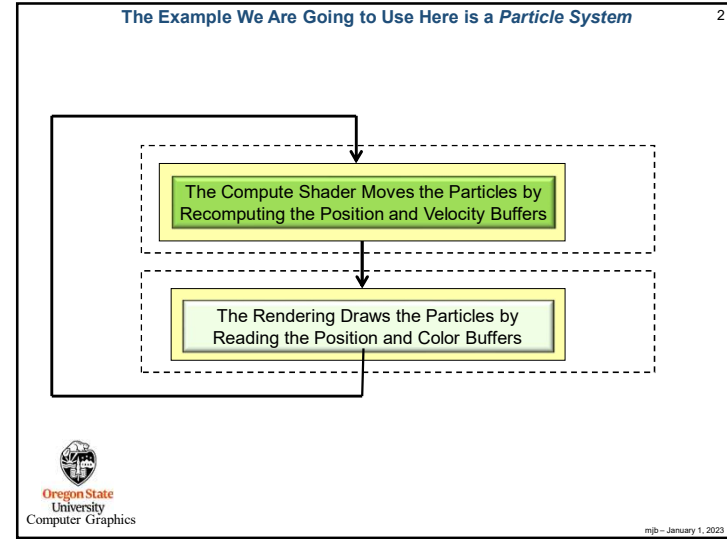

Compute Shaders


Oregon State University
 Mike Bailey
 mjb@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/)


 Oregon State University
 Computer Graphics

ComputeShaders.pptx mjb - January 1, 2023



The Data in your C/C++ Program will look like This

This is a Particle System application, so we need Positions, Velocities, and (possibly) Colors

```


#define NUM_PARTICLES      (1024*1024) // total number of particles to move
#define NUM_WORK_ITEMS_PER_GROUP 64 // # work-items per work-group
#define NUM_X_WORK_GROUPS (NUM_PARTICLES / NUM_WORK_ITEMS_PER_GROUP)

struct pos
{
    glm::vec4; // positions
};

struct vel
{
    glm::vec4; // velocities
};

struct col
{
    glm::vec4; // colors
};
    
```

Note that .w and .vw are not actually needed. But, by making these structure sizes a multiple of 4 floats, it doesn't matter if they are declared with the std140 or the std430 qualifier. I think this is a good thing.


 Oregon State University
 Computer Graphics

mjb - January 1, 2023

The Data in your Compute Shader will look like This


```

layout( std140, set = 0, binding = 0 ) buffer Pos
{
    1  vec4 Positions[ ]; // array of structures
};

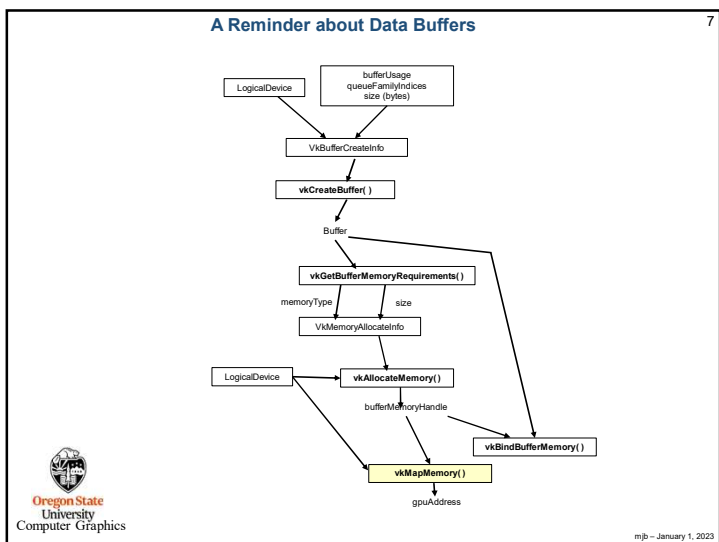
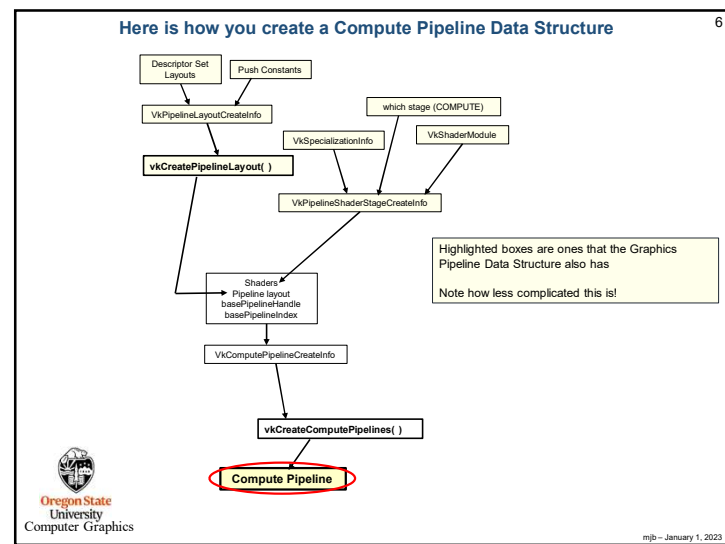
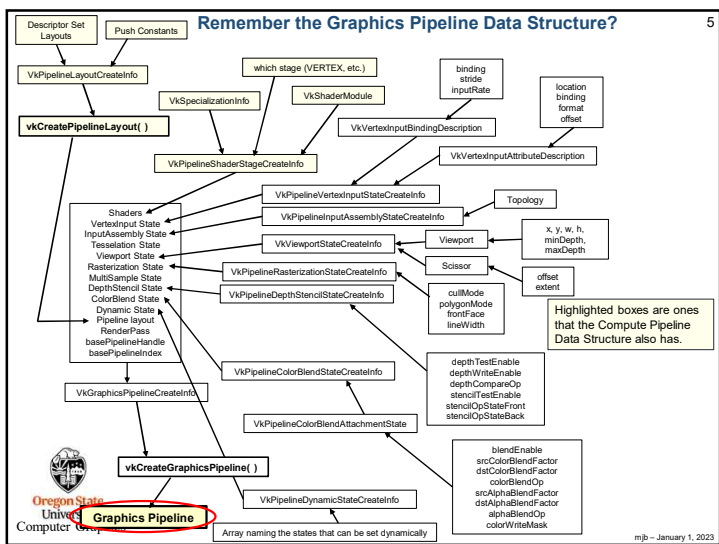
layout( std140, set = 0, binding = 1 ) buffer Vel
{
    2  vec4 Velocities[ ]; // array of structures
};

layout( std140, set = 0, binding = 2 ) buffer Col
{
    3  vec4 Colors[ ]; // array of structures
};
    
```

You can use the empty brackets, but only on the *last* element of the buffer. The actual dimension will be determined for you when Vulkan examines the size of this buffer's data store.


 Oregon State University
 Computer Graphics

mjb - January 1, 2023



Creating a Shader Storage Buffer

```

VkBuffer PosBuffer;
...
VkBufferCreateInfo vbci;
vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
vbci.pNext = nullptr;
vbci.flags = 0;
vbci.size = NUM_PARTICLES * sizeof(glm::vec4);
vbci.usage = VK_USAGE_STORAGE_BUFFER_BIT;
vbci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
vbci.queueFamilyIndexCount = 0;
vbci.pQueueFamilyIndices = (const uint32_t*) nullptr;

result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR, OUT &PosBuffer );
  
```

Oregon State University Computer Graphics
mjb - January 1, 2023

Allocating Memory for a Buffer, Binding a Buffer to Memory, and Filling the Buffer

```

VkMemoryRequirements
result = vkGetBufferMemoryRequirements( LogicalDevice, PosBuffer, OUT &vmr );

VkMemoryAllocateInfo
vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
vmai.pNext = nullptr;
vmai.flags = 0;
vmai.allocationSize = vmr.size;
vmai.memoryTypeIndex = FindMemoryThatIsHostVisible();

...

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

result = vkBindBufferMemory( LogicalDevice, PosBuffer, IN vdm, 0 ); // 0 is the offset
  
```

mjb - January 1, 2023

Create the Compute Pipeline Layout

```

VkDescriptorSetLayoutBinding ComputeSet[3];
ComputeSet[0].binding = 0;
ComputeSet[0].descriptorType = VK_DESCRIPTOR_TYPE_STORAGE_BUFFER;
ComputeSet[0].descriptorCount = 1;
ComputeSet[0].stageFlags = VK_SHADER_STAGE_COMPUTE_BIT;
ComputeSet[0].pImmutableSamplers = (VkSampler *)nullptr;

ComputeSet[1].binding = 1;
ComputeSet[1].descriptorType = VK_DESCRIPTOR_TYPE_STORAGE_BUFFER;
ComputeSet[1].descriptorCount = 1;
ComputeSet[1].stageFlags = VK_SHADER_STAGE_COMPUTE_BIT;
ComputeSet[1].pImmutableSamplers = (VkSampler *)nullptr;

ComputeSet[2].binding = 2;
ComputeSet[2].descriptorType = VK_DESCRIPTOR_TYPE_STORAGE_BUFFER;
ComputeSet[2].descriptorCount = 1;
ComputeSet[2].stageFlags = VK_SHADER_STAGE_COMPUTE_BIT;
ComputeSet[2].pImmutableSamplers = (VkSampler *)nullptr;

VkDescriptorSetLayoutCreateInfo vdsic;
vdsic.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
vdsic.pNext = nullptr;
vdsic.flags = 0;
vdsic.bindingCount = 3;
vdsic.pBindings = &ComputeSet[0];
  
```

mjb - January 1, 2023

Create the Compute Pipeline Layout

```

VkPipelineLayout ComputePipelineLayout;
VkDescriptorSetLayout ComputeSetLayout;

...

result = vkCreateDescriptorSetLayout( LogicalDevice, IN &vdsic, PALLOCATOR, OUT &ComputeSetLayout );

VkPipelineLayoutCreateInfo vplci;
vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
vplci.pNext = nullptr;
vplci.flags = 0;
vplci.setLayoutCount = 1;
vplci.pSetLayouts = &ComputeSetLayout;
vplci.pushConstantRangeCount = 0;
vplci.pPushConstantRanges = (VkPushConstantRange *)nullptr;

result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR, OUT &ComputePipelineLayout );
  
```

mjb - January 1, 2023

Create the Compute Pipeline

```

VkPipelineShaderStageCreateInfo vpssci;
vpssci.sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
vpssci.pNext = nullptr;
vpssci.flags = 0;
vpssci.stage = VK_SHADER_STAGE_COMPUTE_BIT;
vpssci.module = computeShader;
vpssci.pName = "main";
vpssci.pSpecializationInfo = (VkSpecializationInfo *)nullptr;

VkComputePipelineCreateInfo vcpici[1];
vcpici[0].sType = VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO;
vcpici[0].pNext = nullptr;
vcpici[0].flags = 0;
vcpici[0].stage = vpssci;
vcpici[0].layout = ComputePipelineLayout;
vcpici[0].basePipelineHandle = VK_NULL_HANDLE;
vcpici[0].basePipelineIndex = 0;

result = vkCreateComputePipelines( LogicalDevice, VK_NULL_HANDLE, 1, &vcpici[0], PALLOCATOR, &ComputePipeline );
  
```

mjb - January 1, 2023

Creating a Vulkan Data Buffer


13

VkBuffer Buffer;

```

VkBufferCreateInfo vbci;
vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
vbci.pNext = nullptr;
vbci.flags = 0;
vbci.size = NUM_PARTICLES * sizeof( glm::vec4 );
vbci.usage = VK_USAGE_STORAGE_BUFFER_BIT;
vbci.sharingMode = VK_SHARING_MODE_CONCURRENT;
vbci.queueFamilyIndexCount = 0;
vbci.pQueueFamilyIndices = (const int32_t) nullptr;

result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR, OUT &posBuffer );
        
```



mjb - January 1, 2023

Allocating Memory and Binding the Buffer

14

```


VkMemoryRequirements
result = vkGetBufferMemoryRequirements( LogicalDevice, posBuffer, OUT &vmr );

VkMemoryAllocateInfo
vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
vmai.pNext = nullptr;
vmai.flags = 0;
vmai.allocationSize = vmr.size;
vmai.memoryTypeIndex = FindMemoryThatIsHostVisible();

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

result = vkBindBufferMemory( LogicalDevice, posBuffer, IN vdm, 0 ); // 0 is the offset

MyBuffer myPosBuffer;
myPosBuffer.size = vbci.size;
myPosBuffer.buffer = PosBuffer;
myPosBuffer.vdm = vdm;
        
```



mjb - January 1, 2023

Fill the Buffers


15

```

struct pos * positions;
vkMapMemory( LogicalDevice, IN myPosBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT (void *) &positions );
for( int i = 0; i < NUM_PARTICLES; i++ )
{
    positions[ i ].x = Ranf( XMIN, XMAX );
    positions[ i ].y = Ranf( YMIN, YMAX );
    positions[ i ].z = Ranf( ZMIN, ZMAX );
    positions[ i ].w = 1.;
}
vkUnmapMemory( LogicalDevice, IN myPosBuffer.vdm );

struct vel * velocities;
vkMapMemory( LogicalDevice, IN myVelBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT (void *) &velocities );
for( int i = 0; i < NUM_PARTICLES; i++ )
{
    velocities[ i ].x = Ranf( VXMIN, VXMAX );
    velocities[ i ].y = Ranf( VYMIN, VYMAX );
    velocities[ i ].z = Ranf( VZMIN, VZMAX );
    velocities[ i ].w = 0.;
}
vkUnmapMemory( LogicalDevice, IN myVelBuffer.vdm );

struct col * colors;
vkMapMemory( LogicalDevice, IN myColBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT (void *) &colors );
for( int i = 0; i < NUM_PARTICLES; i++ )
{
    colors[ i ].r = Ranf( .3f, 1. );
    colors[ i ].g = Ranf( .3f, 1. );
    colors[ i ].b = Ranf( .3f, 1. );
    colors[ i ].a = 1.;
}
vkUnmapMemory( LogicalDevice, IN myColBuffer.vdm );
        
```



mjb - January 1, 2023

Fill the Buffers

16


```

#include <stdlib.h>

#define TOP 2147483647. // 2^31 - 1

float
Ranf( float low, float high )
{
    long random( ); // returns integer 0 - TOP

    float r = (float)rand();
    return low + r * ( high - low ) / (float)RAND_MAX;
}
        
```



mjb - January 1, 2023

The Particle System Compute Shader

```


layout( std140, set = 0, binding = 0 ) buffer Pos
{
    vec4 Positions[ ]; // array of structures
};

layout( std140, set = 0, binding = 1 ) buffer Vel
{
    vec4 Velocities[ ]; // array of structures
};

layout( std140, set = 0, binding = 2 ) buffer Col
{
    vec4 Colors[ ]; // array of structures
};

layout( local_size_x = 64, local_size_y = 1, local_size_z = 1 ) in;
    
```

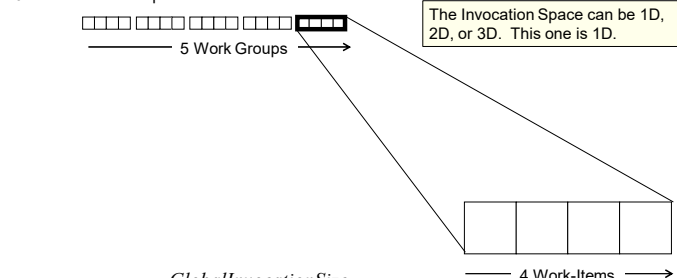
This is the number of **work-items per work-group**, set in the compute shader. The number of work-groups is set in the `vkCmdDispatch(commandBuffer, workGroupCountX, workGroupCountY, workGroupCountZ);` function call in the application program.




mjb - January 1, 2023

The Data gets Divided into Large Quantities call *Work-Groups*, each of which is further Divided into Smaller Units Called *Work-Items*

20 total items to compute:



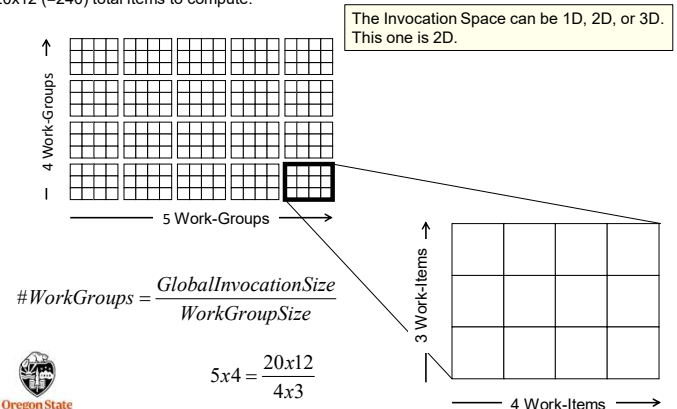
$$\#WorkGroups = \frac{GlobalInvocationSize}{WorkGroupSize}$$

$$5 = \frac{20}{4}$$



mjb - January 1, 2023

The Data Needs to be Divided into Large Quantities call *Work-Groups*, each of which is further Divided into Smaller Units Called *Work-Items*

20x12 (=240) total items to compute:

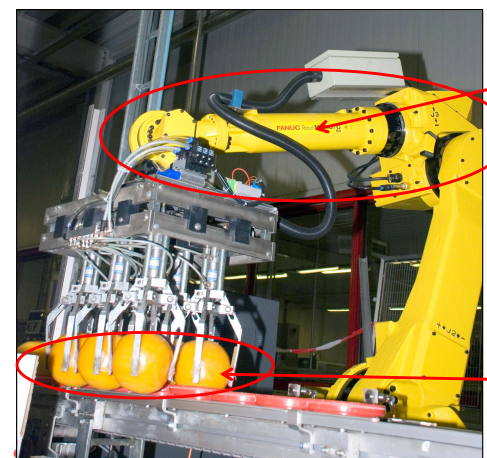


$$\#WorkGroups = \frac{GlobalInvocationSize}{WorkGroupSize}$$

$$5 \times 4 = \frac{20 \times 12}{4 \times 3}$$


mjb - January 1, 2023

A Mechanical Equivalent...



University Computer Graphics <http://news.cision.com>

mjb - January 1, 2023

The Particle System Compute Shader – The Physics

21

```

#define POINT          vec3
#define VELOCITY      vec3
#define VECTOR        vec3
#define SPHERE        vec4 // xc, yc, zc, r
#define PLANE         vec4 // a, b, c, d

const VECTOR G       = VECTOR( 0., -9.8, 0. );
const float  DT      = 0.1;

const SPHERE Sphere = vec4( -100., -800., 0., 600. ); // x, y, z, r

...
uint gid = gl_GlobalInvocationID.x; //where I am in the global dataset (6 in this example)
// (as a 1d problem, the .y and .z are both 1)

POINT  p = Positions[ gid ].xyz;
VELOCITY v = Velocities[ gid ].xyz;

POINT  pp = p + v*DT + .5*DT*DT*G;
VELOCITY vp = v + G*DT;

Positions[ gid ].xyz = pp;
Velocities[ gid ].xyz = vp;
    
```

$$p' = p + v \cdot t + \frac{1}{2} G \cdot t^2$$

$$v' = v + G \cdot t$$

Oregon State University Computer Graphics mjb - January 1, 2023

The Particle System Compute Shader – How About Introducing a Bounce?

22

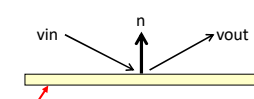
```

VELOCITY
Bounce( VELOCITY vin, VECTOR n )
{
    VELOCITY vout = reflect( vin, n );
    return vout;
}

// plane equation: Ax + By + Cz + D = 0
// ( it turns out that (A,B,C) is the normal )

VELOCITY
BouncePlane( POINT p, VELOCITY vin, PLANE pl )
{
    VECTOR n = normalize( VECTOR( pl.xyz ) );
    return Bounce( vin, n );
}

bool
IsUnderPlane( POINT p, PLANE pl )
{
    float r = pl.x*p.x + pl.y*p.y + pl.z*p.z + pl.w;
    return ( r < 0. );
}
    
```



Note: a surface in the x-z plane has the equation:
 $0x + 1y + 0z + 0 = 0$
 and thus its normal vector is (0,1,0)

Oregon State University Computer Graphics mjb - January 1, 2023

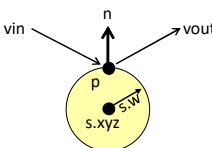
The Particle System Compute Shader – How About Introducing a Bounce?

23

```

VELOCITY
BounceSphere( POINT p, VELOCITY vin, SPHERE s )
{
    VECTOR n = normalize( p - s.xyz );
    return Bounce( vin, n );
}

bool
IsInsideSphere( POINT p, SPHERE s )
{
    float r = length( p - s.xyz );
    return ( r < s.w );
}
    
```



Oregon State University Computer Graphics mjb - January 1, 2023

The Particle System Compute Shader – How About Introducing a Bounce?

24

```

uint gid = gl_GlobalInvocationID.x; // the .y and .z are both 1 in this case

POINT  p = Positions[ gid ].xyz;
VELOCITY v = Velocities[ gid ].xyz;

POINT  pp = p + v*DT + .5*DT*DT*G;
VELOCITY vp = v + G*DT;

if( IsInsideSphere( pp, Sphere ) )
{
    vp = BounceSphere( p, v, S );
    pp = p + vp*DT + .5*DT*DT*G;
}

Positions[ gid ].xyz = pp;
Velocities[ gid ].xyz = vp;
    
```

$$p' = p + v \cdot t + \frac{1}{2} G \cdot t^2$$

$$v' = v + G \cdot t$$

Graphics Trick Alert: Making the bounce happen from the surface of the sphere is time-consuming. Instead, bounce from the previous position in space. If DT is small enough (and it is), nobody will ever know...

Oregon State University Computer Graphics mjb - January 1, 2023

Dispatching the Compute Shader from the Command Buffer 25


```

#define NUM_PARTICLES (1024*1024)
#define NUM_WORK_ITEMS_PER_GROUP 64
#define NUM_X_WORK_GROUPS (NUM_PARTICLES / NUM_WORK_ITEMS_PER_GROUP)
...
vkCmdBindPipeline( CommandBuffer, VK_PIPELINE_BIND_POINT_COMPUTE, ComputePipeline );
vkCmdDispatch( CommandBuffer, NUM_X_WORK_GROUPS, 1, 1 );
    
```

This is the number of work-groups, set in the application program.
 The number of work-items per work-group is set in the layout in the compute shader:

```

layout( local_size_x = 64, local_size_y = 1, local_size_z = 1 ) in;
        
```



mjb - January 1, 2023

Displaying the Particles 26

```


VkVertexInputBindingDescription vvibd[3]; // one of these per buffer data buffer
vvibd[0].binding = 0; // which binding # this is
vvibd[0].stride = sizeof( struct pos ); // bytes between successive structs
vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

vvibd[1].binding = 1;
vvibd[1].stride = sizeof( struct vel );
vvibd[1].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

vvibd[2].binding = 2;
vvibd[2].stride = sizeof( struct col );
vvibd[2].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
    
```

```

layout( location = 0 ) in vec4 aPosition;
layout( location = 1 ) in vec4 aVelocity;
layout( location = 2 ) in vec4 aColor;
        
```



mjb - January 1, 2023


Displaying the Particles 27

```

VkVertexInputAttributeDescription vviad[3]; // array per vertex input attribute
// 3 = position, velocity, color
vviad[0].location = 0; // location in the layout decoration
vviad[0].binding = 0; // which binding description this is part of
vviad[0].format = VK_FORMAT_VEC4; // x, y, z, w
vviad[0].offset = offsetof( struct pos, pos ); // 0

vviad[1].location = 1;
vviad[1].binding = 0;
vviad[1].format = VK_FORMAT_VEC4; // nx, ny, nz
vviad[1].offset = offsetof( struct vel, vel ); // 0

vviad[2].location = 2;
vviad[2].binding = 0;
vviad[2].format = VK_FORMAT_VEC4; // r, g, b, a
vviad[2].offset = offsetof( struct col, col ); // 0
    
```




mjb - January 1, 2023

Telling the Pipeline about its Input 28

```

VkPipelineVertexInputStateCreateInfo vpvisci; // used to describe the input vertex attributes
vpvisci.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vpvisci.pNext = nullptr;
vpvisci.flags = 0;
vpvisci.vertexBindingDescriptionCount = 3;
vpvisci.pVertexBindingDescriptions = vvibd;
vpvisci.vertexAttributeDescriptionCount = 3;
vpvisci.pVertexAttributeDescriptions = vviad;

VkPipelineInputAssemblyStateCreateInfo vpiasci;
vpiasci.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
vpiasci.pNext = nullptr;
vpiasci.flags = 0;
vpiasci.topology = VK_PRIMITIVE_TOPOLOGY_POINT_LIST;
    
```



mjb - January 1, 2023

Telling the Pipeline about its Input 29

We will come to the Pipeline later, but for now, know that a Vulkan Pipeline is essentially a very large data structure that holds (what OpenGL would call) the state, including how to parse its vertex input.

```

VkGraphicsPipelineCreateInfo
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;
vgpci.flags = 0;
vgpci.stageCount = 2; // number of shader stages in this pipeline
vgpci.pStages = vpscsi;
vgpci.pVertexInputState = &vpscisci;
vgpci.pInputAssemblyState = &vpiascisci;
vgpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr; // &vptscisci
vgpci.pViewportState = &vpvscisci;
vgpci.pRasterizationState = &vprscisci;
vgpci.pMultisampleState = &vpmscisci;
vgpci.pDepthStencilState = &vpdscisci;
vgpci.pColorBlendState = &vpcbscisci;
vgpci.pDynamicState = &vpdscisci;
vgpci.layout = IN GraphicsPipelineLayout;
vgpci.renderPass = IN RenderPass;
vgpci.subpass = 0; // subpass number
vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
vgpci.basePipelineIndex = 0;

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

Oregon State University Computer Graphics mjb - January 1, 2023

Setting a Pipeline Barrier so the Drawing Waits for the Compute 30

```

VkBufferMemoryBarrier
vbmb.sType = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER;
vbmb.pNext = nullptr;
vbmb.srcAccessFlags = VK_ACCESS_SHADER_WRITE_BIT;
vbmb.dstAccessFlags = VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT;
vbmb.srcQueueFamilyIndex = 0;
vbmb.dstQueueFamilyIndex = 0;
vbmb.buffer =
vbmb.offset = 0;
vbmb.size = NUM_PARTICLES * sizeof( glm::vec4 );

const uint32 bufferMemoryBarrierCount = 1;
vkCmdPipelineBarrier
(
    commandBuffer,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT, VK_PIPELINE_STAGE_VERTEX_INPUT_BIT,
    VK_DEPENDENCY_BY_REGION_BIT, 0, nullptr, bufferMemoryBarrierCount, IN &vbmb, 0, nullptr
);
    
```

Oregon State University Computer Graphics mjb - January 1, 2023

Drawing 31

```

VkBuffer buffers[ ] = MyPosBuffer.buffer, MyVelBuffer.buffer, MyColBuffer.buffer;
size_t offsets[ ] = { 0, 0, 0 };

vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 3, buffers, offsets );

const uint32_t vertexCount = NUM_PARTICLES;
const uint32_t instanceCount = 1;
const uint32_t firstVertex = 0;
const uint32_t firstInstance = 0;

vkCmdDraw( CommandBuffers[nextImageIndex], NUM_PARTICLES, 1, 0, 0 );
// vertexCount, instanceCount, firstVertex, firstInstance
    
```

Oregon State University Computer Graphics mjb - January 1, 2023

Setting a Pipeline Barrier so the Compute Waits for the Drawing 32

```

VkBufferMemoryBarrier
vbmb.sType = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER;
vbmb.pNext = nullptr;
vbmb.srcAccessFlags = 0;
vbmb.dstAccessFlags = VK_ACCESS_UNIFORM_READ_BIT;
vbmb.srcQueueFamilyIndex = 0;
vbmb.dstQueueFamilyIndex = 0;
vbmb.buffer =
vbmb.offset = 0;
vbmb.size = ??

const uint32 bufferMemoryBarrierCount = 1;
vkCmdPipelineBarrier
(
    commandBuffer,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT, VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT,
    VK_DEPENDENCY_BY_REGION_BIT, 0, nullptr, bufferMemoryBarrierCount,
    IN &vbmb, 0, nullptr
);
    
```

Oregon State University Computer Graphics mjb - January 1, 2023