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

Compute Shaders

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

lots less complicated than the Graphics Pipeline!

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.

Here is how you create a Compute Pipeline Data Structure

Start by Creating the Data Buffers

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
};
A Reminder about Data Buffers

vkCreateBuffer( )

VkBufferCreateInfo

bufferUsage

queueFamilyIndices

size (bytes)

LogicalDevice

vkGetBufferMemoryRequirements( )

Buffer

VkMemoryAllocateInfo

sizememoryType

vkAllocateMemory( )

vkBindBufferMemory( )

bufferMemoryHandle

vkMapMemory( )

gpuAddress

Creating a Shader Storage Buffer

 VkBuffer Buffer;

...

VkBufferCreateInfo vbci;

vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;

vbci.pNext = nullptr;

vbci.flags = 0;

vbci.size = << buffer size in bytes >>

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 &Buffer );

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

VkMemoryRequirements

result = vkGetBufferMemoryRequirements( LogicalDevice, Buffer, 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, Buffer, IN vdm, 0 ); // 0 is the offset ...

result = vkMapMemory( LogicalDevice, IN vdm, 0, VK_WHOLE_SIZE, 0, &ptr );

<< do the memory copy >>

result = vkUnmapMemory( LogicalDevice, IN vdm );

And, since we have Data Buffers, we will need Descriptor Sets to Create the Pipeline Layout

Descriptor Set Layouts

Push Constants

VkPipelineLayoutCreateInfo

vkCreatePipelineLayout( )
Create the Compute Pipeline Layout

```
VkPipelineLayout ComputePipelineLayout;

VkDescriptorSetLayout ComputeSetLayout;

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

VkDescriptorSetLayoutCreateInfo vdslc;
vdslc.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
vdslc.pNext = nullptr;
vdslc.flags = 0;
vdslc.bindingCount = 1;
vdslc.pBindings = &ComputeSet[0];
result = vkCreateDescriptorSetLayout(LogicalDevice, &vdslc, 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;
result = vkCreatePipelineLayout(LogicalDevice, &vplci, PALLOCATOR, OUT &ComputePipelineLayout);
```

Create the Compute Pipeline

```
VkPipeline ComputePipeline;

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 vcpci[1];
vcpci[0].sType = VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO;
vcpci[0].pNext = nullptr;
vcpci[0].flags = 0;
vcpci[0].stage = vpssci;
vcpci[0].layout = ComputePipelineLayout;
vcpci[0].basePipelineHandle = VK_NULL_HANDLE;
vcpci[0].basePipelineIndex = 0;
result = vkCreateComputePipelines(LogicalDevice, VK_NULL_HANDLE, 1, &vcpci[0], PALLOCATOR, OUT &ComputePipeline);
```

The Particle System Compute Shader -- Setup

```
#define POINT vec3
#define VELOCITY vec3
#define VECTOR vec3
#define SPHERE vec4

const VECTOR G = VECTOR(0., -9.8, 0.);
const float DT = 0.1;
const SPHERE Sphere = vec4(-100., -800., 0., 600.);
```

```
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;

p += p * v * DT + .5 * DT * DT * G;
vel += vel + G * DT;
```

```
POInt p = Positions[gid].xyz;
VELocity v = Velocities[gid].xyz;

p += p * v * DT + .5 * DT * DT * G;
vel += vel + G * DT;
```

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, groupCountX, groupCountY, groupCountZ)` function call in the application program.
VELOCITY
Bounce( VELOCITY vin, VECTOR n ) {
    VELOCITY vout = reflect( vin, n );
    return vout;
}
VELOCITY
BounceSphere( POINT p, VELOCITY v, SPHERE s ) {
    VECTOR n = normalize( p - s.xyz );
    return Bounce( v, n );
}
bool
IsInsideSphere( POINT p, SPHERE s ) {
    float r = length( p - s.xyz );
    return ( r < s.w );
}

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

Dispatching the Compute Shader from the Command Buffer

const int NUM_PARTICLES = 1024*1024;
const int NUM_WORK_ITEMS = 64;
const int NUM_X_WORK_GROUPS = NUM_PARTICLES / NUM_WORK_ITEMS;

vkCmdBindPipeline( CommandBuffer, VK_PIPELINE_BIND_POINT_COMPUTE, ComputePipeline );
vkCmdDispatch( CommandBuffer, NUM_X_WORK_GROUPS, 1, 1 );

Or,
vkCmdBindPipeline( CommandBuffer, VK_PIPELINE_BIND_POINT_COMPUTE, ComputePipeline );
vkCmdDispatchIndirect( CommandBuffer, Buffer, 0 ); // Buffer holds the 3 sizes, offset=0

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...