TP05
GLSL Language

Robin Faury

01/29/20

Abstract

In this practical work, we will see how to write, compile and send a program to the GPU.

1 GLSL

The GLSL (OpenGL Shading Language) is the high-level language used to process data on a GPU. The syntax is based on the C programming language. This language includes built-in data types and functions for floating-point vectors and matrices. Be aware those new type work component-wise. For example, if you need the dot product, don't use the "*" operator but the "dot" function. You can also get value, subvector or swizzle a vector using .xyzw or .rgba.

```
vec4 a = vec4(vec2(1.0), 2.0, 3.0); // a = (1.0, 1.0, 2.0, 3.0)
vec4 b = vec4(a.yzw, a.x); // b = (1.0, 2.0, 3.0, 1.0)
vec4 c = a * b; // c = (1.0, 2.0, 6.0, 3.0)
```

1.1 A shader line by line

The shader is the program we send to the GPU to tell cores what they need to compute. First, create a new file named "shader.comp". The "comp" part means you want a compute shader. You can also request for a vertex shader (.vert) or a fragment shader (.frag). Both of them are used for a graphic pipeline. This isn't the case for us. The first line of the file should be the version of your shader. This version must match the GLSL number version.

```
#version 440
```

Next you can push constant values (or uniform values). If your data is huge, it's recommended using another buffer to send your constants. This buffer is called UBO (Uniform Buffer Object). In our case we send a buffer. So we just need its size as constant.

```
layout(push_constant) uniform Parameters {
        uint bufferSize;
} params;
```

The last line before the core of the shader is the layout of inputs buffers. Here, a buffer of vec4.

```
layout(std430, binding = 0) buffer lay0 { vec4 inputBuffer[]; };
```

As usual, any program start by a main function.

```
void main() {

}
```

This main function will be executed by all the core of the GPU at the same time. Fortunately, we can ask the ID of the thread to apply the same program on several data (Single Program Multiple Data (SPMD))

```
void main() {
        const uint id = gl_GlobalInvocationID.x;
}
```

Our Buffer is stored as a linear buffer so we can use directly the id as index.

```
void main() {
        const uint id = gl_GlobalInvocationID.x;
        inputBuffer[id] = vec4(1.0, 0.0, 1.0, 1.0);
}
```

Finally, we should take care of memory access. Read or Write into unallocated memory can cause crash. We must check our buffer size is greater than the id.

```
void main() {
        const uint id = gl_GlobalInvocationID.x;
        if (params.bufferSize <= id) {
                return;
        }
        inputBuffer[id] = vec4(1.0, 0.0, 1.0, 1.0);
}
```

For information gl_GlobalInvocationID is a uvec3. That means you can organize your data as a linear buffer, or as an image, or as a 3D texture.

1.2 compilation

We just wrote our compute shader. With OpenGL (before Vulkan), we can send the text as string to the GPU and the program is compiled by the driver during the transfer. It's a good thing for the development time, but not for the security. If you open your program with a hexa editor, you'll see our source code. With Vulkan, you can compile your shader as a binary file called SPIR-V (.spv).
Find the glslc.exe program on your Vulkan folder and run this command to compile the shader:

```
glslc.exe shader.comp -o shader.spv
```

2 Load the shader

Back to your C++ application. Load the .spv file using, for example, "std::ifstream(filename, std::ios::binary);"

2.1 Shader Module

The vkShaderModule is the Vulkan object use to wrap the shader code before sending it to the compute pipeline.
Create a VkShaderModuleCreateInfo with the sType VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO.
The member codeSize is the number of char and the pCode is a pointer on the code reinterpreted as uint32.

```
reinterpret_cast<const uint32_t*>(code.data());
```

The codeSize should be memory align. Resize your vector using this formula : $\frac{size+3}{4} * 4$.
Now create the vkShaderModule:

```
vkCreateShaderModule(logicalDevice, &shaderModuleCreateInfo, nullptr, &shaderModule);
```

Don't forget to check the return of the function and to destroy the object at the end.

3 Be familiar with GLSL

Going further. You can go on the website

```
https://www.shadertoy.com/
```

to try the GLSL language. As a suggestion, you can search "circle" and take a look to the code of a simple one. Try to modify some input.