OpenGL compute shaders

T Qi

December 17, 2023

1 Overview

Compute shaders are generalized shader programs, away from the standard OpenGL graphical pipeline, that are run on the GPU in parallel cores. Compared to alternatives like CUDA or OpenCL, compute shaders better in interfacing with an existing OpenGL project, with similar performance in GLSL code compared to that of CUDA or OpenCL.

The objective of compute shaders is to exploit the parallel execution of shaders on a GPU. This is suitable for a wide range of algorithms which are embarrassingly parallel that requires little branching or cross grid references.

The steps taken for the GPU to process data are

1. We send the data to the GPU
2. We give shader instruction code to the GPU, that performs our desired operations on the data
3. We wait for the operations to finish
4. We read the data from the GPU

2 The Theory

To create a compute shader program, we follow the same step as to create a regular vertex/fragment shader program (compile the shader, create the program, and link the program).

To allocate inputs and outputs for our compute shader, we simply pass the shader a texture. Textures in OpenGL are simply tightly packed pixel component values that are easy to work with; we can use textures to store arbitrary data values.

The difference between samplers and images in GLSL are that samplers represent a readable texture, with the sampling method set on the shader program by OpenGL; images are read and writable pieces of memory, where the shader can access individual texels. For the sake of passing structured data to the compute shader, we will use images.

To allocate an image buffer, we piggyback on an existing texture buffer object: we simply bind it to the image unit (activating the texture object beforehand). The image units are completely separate from the texture units, albeit having the same index format. On the compute shader, we can access the image unit by setting the layout property with "bindings = index" on a uniform input with type "image2D".

The compute shaders are run in both global workgroups and local workgroups. Globally, one compute shader instance is allocated for each workgroup, with each workgroup defined a group size in 3D space. Each compute shader instance is passed a global "gl_GlobalInvocationID" representing the 3D coordinates of the workgroup. This ID can be used to access the desired image texture pixel location for our inputs and outputs. For simplicity, we let the local workgroup size to be one.

3 In practice

Here are the steps to run a single computation using a compute shader:

1. Create and link the shader program against your compute shader source code.
2. Setup the texture and bind to the image unit.
3. Upload the data to the texture.
4. Activate the texture, shader, and call dispatch compute with the correct workgroup size.
5. Wait for the shader to finish computing by calling "glMemoryBuffer".
6. Download the data from the texture. The texture can be read all at once or partially through a "GL_PIXEL_PACK_BUFFER".

Of course, the image data will not be reset after every dispatch call; we can simply reuse the same texture data from the last dispatch and continue iterating if our algorithm is iterative.

The main bottleneck in this process is reading the texture image data from the GPU. It may be recommended to use such compute shaders only if each shader program runs on a computationally expensive task (some iterative algorithm) or if the state space is large.