Multi-core programming homework 3

César Leblic

Langage version and libraries used

C++11

I used C++11 to write the program. This choice was motivated by mutliple reasons :

- new coding standards

FreeGlut and SDL2

I used Freeglut in association with SDL2 instead of Glut :

- Glut is not maintained since a lot of time
- Glut do not permit to exit the game loop properly. In fact you cannot dealocate memory or destroy objects at the end of the program, it just autorize you to `exit()` the program.

Imgui

I used Imgui (https://github.com/ocornut/imgui) which permit you to display easily a GUI.

Very important notice

I do **not** use Cuda for this exercise. In fact I spent a lot of time to make it work on my personnal computer. My first program didn't compule on Visual 2013. This was a Visal compiler error due to Cuda call. So I install visual 2012 and port the project on it. Still the same error. So I reinstall Cuda. It didn't work too. Then I try to open and run the Cuda samples provided by Nvidia. On Visual 2013 they didn't compile, with the same error as my project. Visual 2012 refuse to open them.

I spent a lot of time trying to understand and to fix the problem. I didn't resolve it. That's why I decide to use OpenGL compute shader. I'm aware that it's not the same technology but it's very close to it and the architecture of my program and the algorithm stay the same.

I'll try to explain you in that document how it work in OpenGL compute shader and how I would have done it in Cuda.

Architecture

Class descrition

All the computation is done on the Gpu, for that reason the C++ code is very short and simple. That's why there is just one class `App` which contain all the code.
Another class `Shader` is used to load and compile shaders.
Note : you can edit and recompile shaders at runtime, using the key `r`.

Workflow explaination

I use 4 shader buffer storage :

- One composed of vector4 for cells, it's used to read cells status in previous state.
- A second one composed of vector4 is used to write new cells status
- Another, very small, composed of 4 `unsigned int` is used to communicate counter result (number of each type of cell) from the gpu to the cpu.
- A last one is used for the rendering, it contain a vector4 for each cell position on the screen. This buffer is specific to the openGl implementation, in Cuda we would have done differently, in fact, we would map a `cudaGraphicsResource` to an OpenGl buffer data (created with `glGenBuffer`).

In Cuda I would use three array either and allocate them on the Gpu. For that I would use`cudaGraphicsResource` for the computation and some OpenGl pixels buffer for the render. I would map the `cudaGraphicsResource` data to the OpenGl buffer with commands `cudaGraphicsGLRegisterBuffer`and `cudaGraphicaMapResources` so that OpenGl can render the Cuda result.
Here, compute shaders, replace Cuda code, they are not compiled the same way (the first one by the OpenGl driver, the second by the cuda compiler) but the code is the same.

A first pass, copy the data of the result of the last frame from the buffer storage used for read to the one used to write. So that if the state does not change, we will not have to rewrite them. We use also the same pass to count cells type. For that we use in glsl `AtomicAdd` on the counter array, we would call the exact same function in Cuda. The second pass is the computation pass, it's where we compute the new state. For that, for each thread, one pixel (cell) is computed. We read data from the read only buffer and write result to the write buffer.

Then we render the result using or shader buffer storage containing positions and the one containing the result (each cell type is represented by a vector4, corresponding to it's color).

Difficulties

Except problems encountered with Cuda compilation on my computer, I do not encounter lots of difficulties. The only one is due to the cell type counter. In fact, it's not a difficulty to implement, it's just that it require communication between the cpu and the gpu at each frame. This feature droped the FPS from 400 to 200.

I think that using OpenGL compute shader instead of Cuda may offer better performance (I do not test, because I do not manage to make Cuda work on my computer) because their is no need of synchronisation or data exchange between Cuda and OpenGL for the rendering. In fact all datas are managed by the OpenGL driver and nothing else.

Conclusion

I understand that I do not respect what was asked by the teacher doing my homework in openGL and not Cuda. But I'm totaly honest when I say that the architecture would have been the same in Cuda.