**OpenCL, final assignment**

# Introduction

In this assignment, you are invited to think of an application where the GPU can be used to speed up calculations. It should be a calculation that involves a lot of similar operations that can be done at the same time, typically on (big) vectors, matrices or images.

Some suggestions (but feel free to think of others):

* Find another type of fractal that can be parallelized (like Pythagoras tree, or Koch fractal). Or make the Mandelbrot considerably faster.
* Implement a common matrix operation as efficiently as possible, for example multiplying two matrices.
* Implement an image processing operation, like mean filter, Sobel edge detection, Gaussian-blur, etc.
* Implement an encryption algorithm (like AES) on the GPU.
* Implement a compression algorithm.
* Implement some simulation involving many objects (for example game-of-life, particle system, n-body simulation).

You have to make a solution for the CPU, and one or more for the GPU.

Your solutions should of course give correct answers to the problem, but that is not enough.   
As important is that you experiment a bit to find a fast implementation on your GPU. Therefore, you should try out different settings, like:

* Try out different workgroup sizes.
* Try different memory allocations (intermediate results in private, local or global memory).
* Minimize data transport between host and GPU.
* Minimize divergence.
* Coalesce memory access.

It is not necessary to find the fastest possible implementation (that would not be realistic after such a short course), but you must think about how you can optimize your program, make different versions of it, and do measurements that show the speed differences. Here, we must note that speed improvements are very difficult to predict, because they depend on many factors, so it can happen that you expected a speed improvement, which did not work out.

# Grading

The mark will be decided along the following guidelines:

* If he programs do not give the correct results: 5 or lower.
* If the programs give correct results: 6 or higher.
* 6: You made only one version of your GPU program which is faster than the CPU program, and did no experiments to see if you can make it faster.
* 7: You made a few GPU versions to experiment with one of the settings mentioned above, and choose the “best” version.
* 8: You made several versions to experiment with two or more of the settings mentioned above, and choose the “best” version.
* 9: You made a remarkably clever implementation, and experimented with other versions to proof that it is optimal.
* 10 (Only in very exceptional cases): You correctly implemented a remarkably difficult application in a very efficient way.

For situations where the above criteria do not apply, the teacher decides.

# Delivery

The following has to be delivered:

* A short description of the problem you solved.
* The CPU implementation (zipped project).
* One or more GPU implementations (zipped project(s)).
* Tables in which you compare the results of the different CPU/GPU versions.
* A short explanation of which version is the fastest, and why.   
  As mentioned above, sometimes a program does not behave as expected (you think it will be faster, but in practice it isn’t). That can happen because of the complexity of a GPU; just mention that, and say that it is not as expected.
* The mark that you think you earn, and why.