# Parallel Image Analysis

## Lab session 3: Using OpenCL for image processing on the GPU

In this lab you will be writing kernels and running them using OpenCL. Start by obtaining the example source code: `http://www.cb.uu.se/~cris/ParallelImageAnalysis/lab3.zip` The ZIP file contains one subdirectory, `gpgpu`, with two different programs: `first.c` and `simple.c`. You will also need some of the code and data that we used in last week's lab (lab 2).

## 1 Exercise 1: first

Open and examine the file `first.c`. Compare it to what David showed in class on Tuesday. Then compile and run the program using the commands

```
make first
./first
```

If everything is OK, you should be looking at the first 10 elements of each of the two input and the output arrays. If there is a compiling or linking error, you might not have the latest drivers for your GPU installed, or you might not have a GPU that supports OpenCL.

If you use Windows, you will have to figure out how to compile this program. Use Google!

## 2 Exercise 2: simple

Open and examine the file `simple.c`. This is very much like the first exercise, except we are using image objects instead of simple arrays. Also, the work is divided two-dimensionally instead of one-dimensionally. The program allocates two image objects, one as input and one as output. The kernel reads a pixel value from the input, divides it by the third input argument, and writes it to the output image. Compile the program and run it with the commands

```
make simple
./simple
```

You can use the same trick we used last week to read in the output file `result.ui8` into MATLAB to see it.

## 3 Exercise 3: difficult!

The main exercise for today is to write a useful filter that runs on the GPU. For example the Gaussian smoothing from last week's lab. Make a kernel out of the `filter_1D()` function. One of the parameters should be a boolean, indicating whether to filter along the rows or the columns of the image. That way, the same kernel can be enqueued twice, once for each of the two passes over the image (don't forget the barrier!). Each image line should be a work item, meaning that the kernel should loop over a whole line instead of processing a single pixel. Try your program on both a small image (`cermet.ui8`) and a large image (`europe.ui8`).

Things to think about:

- Do you want to use a simple array to store the image, or an image object?
- How can you avoid moving the data back and forth to the GPU in between the two passes?
- Try to keep the input and output of the filter as 8-bit integers, to minimize data transfer.
- Into how many work items do you need to divide the work?