Parallel Visual Computing 2012/13
Assignment 7

For all tasks that you address, you have to provide a short documentation as well as present it to the class in our exercise meeting. Please make sure to report absolute timing in terms of "Hz" (frames per second) as well as speed improvements w.r.t. your first measurement of the initial code on your hardware.

Please note that the provided code is provided by Rodrigo Benenson and comes with a license. Have a look at the included "license.text" file.

This code has 3 additional dependencies that you have to install:

- protobuffer: `http://code.google.com/p/protobuf/`
- opencv $\geq$ 2.4: `http://opencv.org`
- boost: `http://www.boost.org`

The build system is based on cmake and tested on linux. A readme file is included. We recommend linux for this exercise. Please have a look into "doppia/commen_settings.cmake" and "doppia/src/applications/objects_detection_exercise/CMakeLists.txt" and change the path to opencv and protobuf. You should be able to build and run the code using:

```
cd doppia/src/applications/objects_detection_exercise
cmake .
make -j 4
./objects_detection_exercise
       --save_detections true
       --video_input.images_folder ../../../data/eth-linthescher
```

The "-j" option sets the number of threads for compilation.

1 Mandatory Tasks

For this exercise you start of with a working but slow code ... which we slowified on purpose. All the work you have to do is within the function "integral_channels_detector_over_all_scales_impl_v0" and the functions called within

that can be found in the file
src/objects_detection/gpu/integral_channels_detector.cu.

Below you find the recommended order of doing the exercises. The things that matter most come first. But the steps do not depend on each other.

1.1 Threads (2 Point)

Figure out the appropriate number of threads. Use the CUDA occupancy calculator in order to determine the number: `http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls`
The compute capability can be figure out from this webpage: `http://en.wikipedia.org/wiki/CUDA`
You can determine the number of registers by compiling the CUDA file with the following options "–ptxas-options=-v". Repeat this procedure if you change your code significantly.

1.2 Cascade (6 Points)

The current version evaluates the classifier as a sum over decision tree stumps. Use the cascade idea in order to speed up the code. Each data structure "stage" contains a value "stage.cascade_threshold". If your current detection score sum is below that threshold at this stage, we assume that there is no object at this location and we don't need to evaluate the cascade (sum) furthermore.

1.3 Parallelize over Scales (8 Points)

The kernel "integral_channels_detector_over_all_scales_kernel_v0" loops over all scales. This turns out to not be a great idea performancewise. Change the code, so that it also parallelizes over the scales.
Hint: Notice that block_dimensions and grid_dimensions are 3D.

1.4 Preread decision values for decision stumps (2 Point)

The function "update_detection_score_v0" evaluates the decision stumps and updates the detection score. First one features is read from the feature map. According to the outcome of the comparison, a second feature is read that yields a score. Try pre-reading all 3 features.

1.5 Global vs. texture memory (2 Point)

The function "get_feature_value" implements reading features from the global and texture memory. Compare them in terms of performance. Explain your observation.

2 Optional Assignment

2.1 Profiling (5 Points)

Profile your best code. Provide insights why the code is not running faster. Where are the bottlenecks? What is the fastest speed you would expect to achieve assuming you can fix the bottlenecks? What is the max throughput of your card? How close are you?

2.2 Additional Speed-Ups (5 Points)

Propose additional speedups beyond what is mentioned in the mandatory part. Explain your approach and argue why you think it should work. Implement and measure. Explain why you succeed ... or didn't.

2.3 Experiment with varying stride (5 Points)

An easy way to vary the accuracy-speed-trade-off is to vary the stride in the spatial sampling of positions. Experiment with the stride and plot number of detections vs. computation time spend. Also investigate the quality of detections. Make sure to very the stride in dependence of the scale. Intuitively, you can afford to make larger steps when searching for large pedestrians.

2.4 Fix bug ;) (2 Points)

The application terminates abnormally with an exception. Find the problem. Beware: this might be tricky to find out. We don't know the answer.