UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Benjamin Resch (benjamin.resch@uni-tuebingen.de)
Fabian Groh (fabian.groh@uni-tuebingen.de)
Raphael Braun (raphael.braun@uni-tuebingen.de)

Eberhard Karls
Universität
Tübingen

6. März 2017

Massively Parallel Computing
Assignment 6

Submission deadline for the exercises: 7. März 2017

Hints

Download the framework `exercise06.tar.gz` from the Ilias course web page.
Present your results to the exercise instructors to get a grading on this exercise sheet.

6.1 NBody Simulation (100 P)

- Start with the source code in the `NBody` folder.
- Create an empty CUDA C/C++ executable project (no Makefile!) from the source code in nsight. Generate PTX and GPU code for SM 6.1.
  You will have to link against libGL.so (-lGL flag) and libX11.so (-lX11 flag) to build the program.
- Implement the following ToDos:
  1. Allocate GPU memory for all the data you will need in your kernels.
  2. Free all GPU you allocated before program termination.
  3. Write a kernel that updates the accelerations of all bodies based on the gravitational attraction of all other bodies once per frame. This is the kernel which does most of the work.
     The acceleration a of a body i can be calculated with

$$a_i = G \cdot \sum_{j=0}^{N} \frac{m_j r_{ij}}{(\|r_{ij}\|^2 + \epsilon^2)^{3/2}} \tag{1}$$

     where
     - G is the gravitational constant given as `GRAVITY` in the code,
     - m_j is the mass of body j,
     - r_{ij} is the vector from body i to body j and
     - ϵ^2 is a damping factor given as `EPS_2` in the code.
  4. Write a kernel that updates the velocities and positions of all bodies based on their accelerations once per frame.
     - To update a body's velocity, just add its current acceleration to its velocity.
     - To update a body's position, just add its current velocity to the its position.
  5. In each frame, download the positions of all bodies to `hPositions` which will be used by OpenGL for visualization.

- *Hints:*
  - Calculate the acceleration of one body per thread.
  - Cooperatively load the data of many bodies per block and use them for the acceleration calculation in all threads.
  - You can find detailed information about how to achieve really good performance in https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch31.html.
  - The visualization will only work if you run the program locally. If you are not sitting on a cgpool* machine, do the debugging locally, then disable the GUI via macro and do the optimization remotely on a fast machine.
  - Reduce `NUM_FRAMES` to 1 while profiling.
- *Bonus:* Get your implementation as fast as possible. Beside optimizations in your code, also use compiler flags to optimize further!
  About 6 ms per frame with 50.000 bodies is quite good on a GeForce GTX 1080.