COS301 Project for 2015: *Motion Simulation* in CA

Offered by the Research Group for System Specifications and Formal Methods (SSFM)
Background

• Cellular Automata (CA) are an Abstract Model of massive-parallel computation, similar to the Turing Machine being an Abstract Model of sequential computation.

• From a theoretical point of view, CA are Turing-equivalent, i.e.: they can compute no less and no more than what Turing machines, too, can compute.

• From a practical point of view, however, CA are interesting because they can show clearly the possibilities –as well as the limits– of parallelism.
• The principle of CA computation is very simple:
  – Every cell has some internal state $S$ at time $T$
  – A clock is ticking
  – At time $T'$, every cell has an internal state $S'$
    which is defined as $S' = f(S, S(N_1), ..., S(N_n))$, whereby each $N_i$ is an immediate neighbour cell, $S(N_i)$ was its own internal state at time $T$ before $T'$, and $f$ can be any computable total function.
Problem

• One of the “tricky” technical problems in such a setting is the simulation of motion.
• This is quite difficult because the cells themselves are fixed in their positions.
• “Motion” must thus be simulated cleverly by means of “copying-and-deleting” the contents of neighbour cells into adjacent cells, under preservation of previous info.
• If not implemented very carefully, wrong behaviour of the CA is typically the effect.
HOW?

Earlier

Later
The SSFM Research Group is currently in possession of a “home-made” CA programming and simulation environment (called CAL+) which is unsatisfactory from three points of view:

- The implementation of the Motion-Simulation is too naive, hence wrong
- The entire run-time environment is too slow
- Numerous “bugs” are still sitting in the system due to hitherto insufficient, sloppy testing
Project

• **Remedy is needed** for those flaws, along the following lines:
  
  – Each cell needs to be *internally multi-layered*, such that the internal states of neighbour cells can be copied into each other without deleting each other
  
  – Speed shall be achieved by re-configuring the system’s “architecture” in such a manner that it becomes *executable with CUDA / GPU on an external graphics card*
  
  – *Thorough Testing (with many many test cases!)* must eliminate a large number of “legacy bugs” and lead to high reliability
• **The group** of students bidding for this project must have:
  – Good understanding of what is “**software evolution**” from legacy software
  – The ability to “**re-factor**” an existing system’s **architecture** for another run-time environment
  – **Excellent knowledge in Algorithms and Data Structures** (e.g.: “distinction” in COS212)
  – Access to some suitable CUDA equipment
  – Basic Experience with utilising a GPU for user-defined computational purposes
  – **Good knowledge also in Computer Graphics**
Most projects in the software industry are maintenance and evolution projects – not development from scratch:

This important lesson can thus be learned with this difficult project.

Moreover, the students will get additional insights into the area of Parallelism,

as well as some exposure to recent trends such as GPU-programming, or CUDA

The high importance of thorough testing will also be learned as another important lesson from this project.