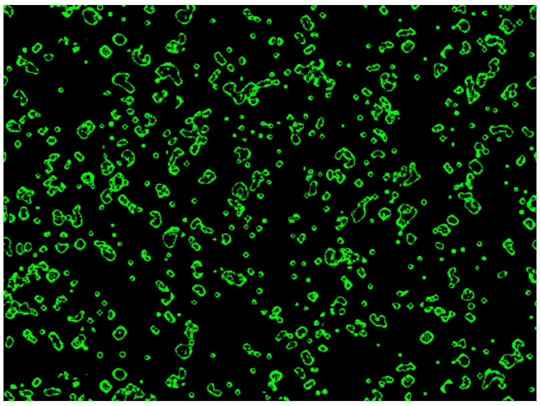
GPU Programming

TP 3 – Bugs



A cellular automaton is a collection of cells on a grid that evolves through discrete time steps, according to a set of rules based on the states of neighboring cells. Cellular automata have been extensively studied since the 1950s as a model for biological systems. The most celebrated automaton is the "The Game of Life" devised by the British mathematician John Conway in 1970, because it features the emergence of remarkable patterns. If you have never heard of cellular automata, take some minutes to read about this fascinating topic on the internet, until you have fully understood their concept.

The generalization of the Game of Life called "Larger than Life" (LtL). A LtL automaton is specified by the following five parameters: *(RANGE, SURVIVELO, SURVIVEHI, BIRTHLO, BIRTHHI)*

For each cell update, we consider a Moore neighborhood of range RANGE, i.e. a (2*RANGE+1)x(2*RANGE+1) square of neighboring cells. If the center cell is alive, and the square features between SURVIVELO and SURVIVEHI living cells, it remains alive. If the center cell is dead, it is born if the square features between BIRTHLO and BIRTHHI living cells. In every other case, the cell dies or remains dead. With this generalization, the "Game of Life" can for example be specified by the parameters (1,3,4,3,3).

Most parameter sets however do not produce interesting automata. They rapidly die out or show a completely chaotic behavior. In this exercise, we implement another really remarkable LtL automaton: "Bugs". Its rules are (5,34,58,34,45).

Exercises (1 point per question)

1) Write a CPU version of the Bug algorithm. The edges should wrap around so that you do not need to worry about borders and corners.
2) GPU version 1. Allocate two grids on the device: grid1 and grid2. Copy the initial grid from the host to grid1. Then compute on the device from grid1 to grid2 and fetch grid2 back for the OpenGL rendering call. At the next frame, compute from grid2 back to grid1 and fetch grid1 back for the OpenGL rendering call, and so on.
3) GPU version 2: use shared memory for faster memory access. In each thread block, preload a "tile", i.e. a subgrid of cells, into shared memory and calculate the next generation of that tile (you need to figure out how to resolve the problem of computing the tile borders).