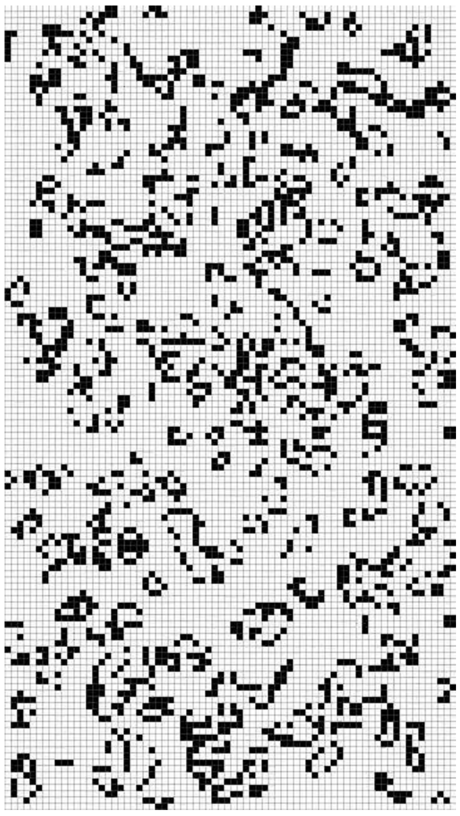
GPU Programming

TP – Game of Life



The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970. Its evolution is determined by its initial state, requiring no further input.
The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. The grid typically wraps around, so that there are no borders. At each step in time, the following transitions occur:

- *Any live cell with fewer than two live neighbors dies, as if caused by under-population.*
- *Any live cell with two or three live neighbors lives on to the next generation.*
- *Any live cell with more than three live neighbors dies, as if by overcrowding.*
- *Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.*

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed. The rules continue to be applied repeatedly to create further generations.

Exercices

Basic version (to pass the module "GPU Programming")

1) Write a CPU version of the Game of Life (GoL) algorithm.
2) GPU version 1. Write a kernel which receives a GoL grid and calculates the next generation. Fetch the computed data back to the host.
3) GPU version 2. Stop using a single grid on the GPU because you copy a grid configuration to the device every time you run the kernel. Work with two grids on the GPU! That way, you can copy the initial configuration to the GPU only once, then compute on the device from grid1 to grid2 and next time from grid2 back to grid1 ("double buffer"). Fetch the proper data back to the host.

Advanced version (for good marks)

4) GPU version 3: 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). Then preload the next tile, etc., until all cells are processed.
5) GPU version 4: render the result by CUDA/OpenGL interoperability.