Conway's Game of Life in SYCL

André Cerqueira

November 29, 2024

REPÚBLICA PORTUGUESA | CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR

Fundação para a Ciência e a Tecnologia

Este trabalho foi financiado por: Projeto 10110190 – EUROCC2, com apoio financeiro da FCT/MCTES através de fundos nacionais (PIDDAC).

Introduction to Conway's Game of Life

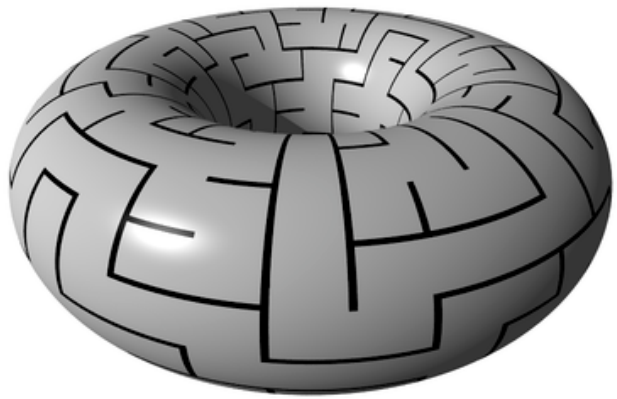
- Created by mathematician John Conway in 1970.
- Simulates how cells on a 2D grid evolve based on simple rules.
- Cells can either be:
  - **Alive (1)**, or
  - **Dead (0)**.
- Produces complex patterns despite its simplicity.

Rules of the Game

- Each cell's next state depends on its 8 neighbors.
- Rules:
  1. **Survival:** A live cell with 2 or 3 live neighbors stays alive.
  2. **Death:** A live cell with fewer than 2 or more than 3 live neighbors dies.
  3. **Birth:** A dead cell with exactly 3 live neighbors becomes alive.
- Key concept: Simple local rules lead to emergent global behavior.

Toroidal Grid Wrapping with Modulo Arithmetic

- **Grid wrapping** ensures that edges connect to the opposite sides, treating the grid as a torus.
- A cell at the edge (top, bottom, left, or right) interacts with cells on the opposite edge.
- **Key Insight**: Use modulo arithmetic to compute wrapped indices.



Example: Neighbors for cell **(0, 0)** at the top-left corner:

(N-1, N-1)	(N-1, 0)	(N-1, 1)
(0, N-1)	(0, 0)	(0, 1)
(1, N-1)	(1, 0)	(1, 1)

Formula: For a neighbor at offset (dx, dy):

$$nx = (x + dx + N)\%N, \quad ny = (y + dy + N)\%N$$

- **Explanation**:
  - Adding N ensures no negative indices.
  - Modulo N wraps the index to stay within the grid.

Exercise Overview

- **Objective:** Implement Conway's Game of Life in SYCL.
- **Tasks:**
  1. Initialize a random grid of size $N \times N$.
  2. Write a kernel to compute the next generation.
  3. Handle grid wrapping using modulo arithmetic.
  4. Implement Game of Life rules.
  5. You can use either:
     - Unified Shared Memory (USM), or
     - Buffers with Accessors.

SYCL Implementation Tips

- **Parallelism:**
  - Use a 2D ND-range for grid computation.
  - Each work item computes the next state of one cell.
- **Memory Management:**
  - USM: Use `malloc_shared` for shared memory.
  - Buffers: Use `buffer` objects with `accessor`.
- **Debugging:**
  - Use the **DEBUG** flag to enable optional debugging output (e.g., neighbor counts).

Example

Initial Grid:

1 0 0 1
0 1 1 0
0 1 0 0
1 0 1 1

Neighbor Counts:

1 2 3 1
2 2 3 3
2 3 4 2
3 2 3 2

Next Generation:

0 1 1 0
1 1 1 1
0 0 1 0
1 0 1 1