Boids Flocking on the GPU

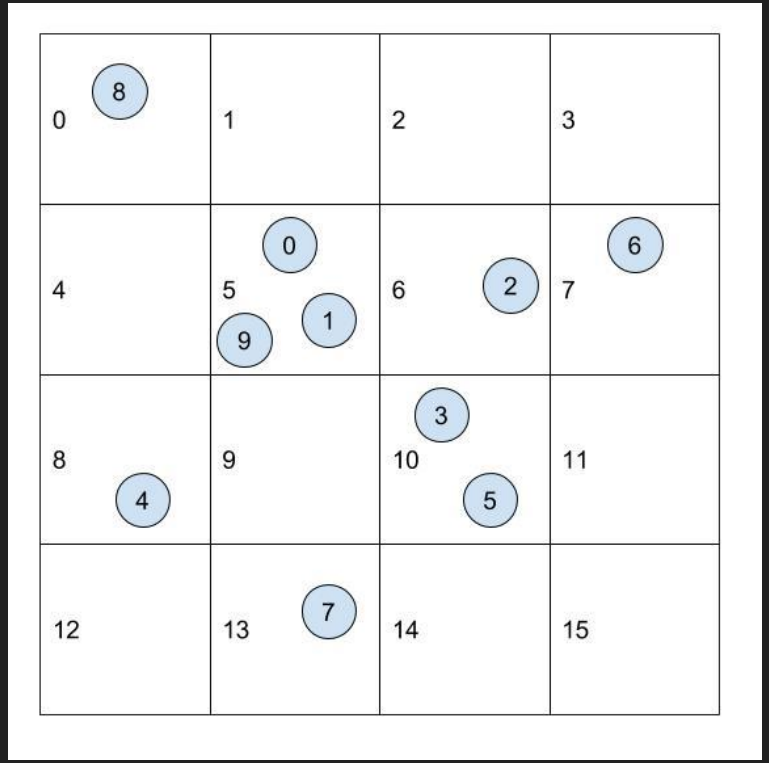
CIS 565 Fall 2017 Recitation 1
Sept 6 2017

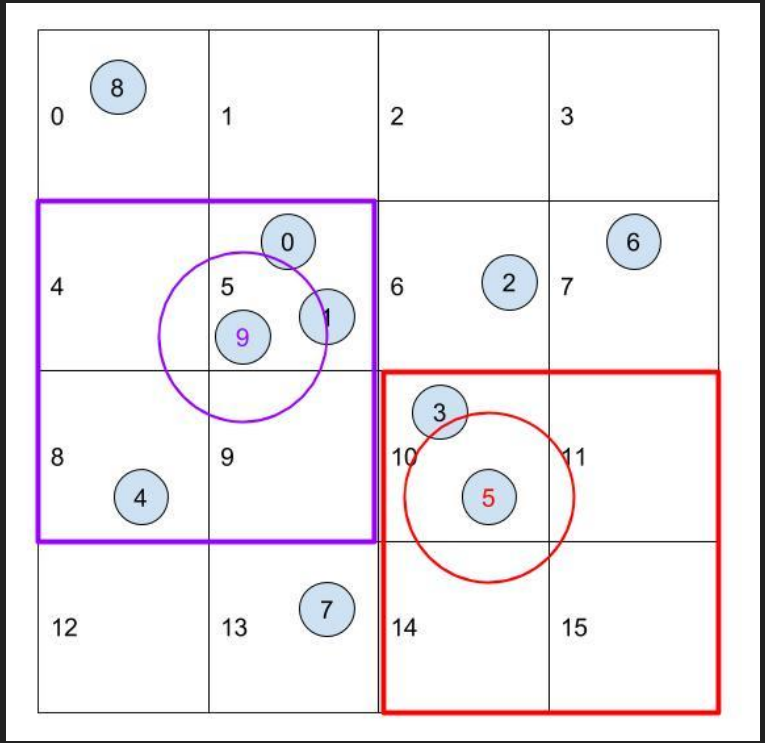
Flocking

- Flocking: each boid looks at neighboring “boids” within a search radius and determines what its new velocity should be
- Naive flocking: boids look at all other boids and decide if they are close enough for further computation

Flocking - The Uniform Grid

- To Algorithms people, spatial data structures are “cheating with both hands”
- Preprocess to reduce numerous boid-boid checks -> “culling”
- Can be performed using a key-value sort
  - Key: integer grid cell index, computed from boid position
  - Value: “pointer” to the boid’s data -> in our case, just an array index
- Then, walk over sorted keys to determine which pointers go in which cells


0
8
1
2
3
4
5
0
9
1
6
2
7
6
8
4
9
10
3
5
11
12
13
7
14
15


0
8
1
2
3
4
0
5
9
1
6
2
7
6
8
4
9
10
3
11
5
12
13
7
14
15

In-memory representation

- `dev_particleGridIndices`
  - buffer containing the grid index of each boid
- `dev_particleArrayIndices`
  - buffer containing a pointer for each boid to its data in dev_pos and dev_vel1 and dev_vel2
- `dev_gridCellStartIndices`
  - buffer containing a pointer for each cell to the beginning of its data in `dev_particleArrayIndices`
- `dev_gridCellEndIndices`
  - buffer containing a pointer for each cell to the end of its data in `dev_particleArrayIndices`

Grid cell index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cell data pointers	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Grid cell index	5	5	6	10	8	10	7	13	0	5
Boid index	0	1	2	3	4	5	6	7	8	9

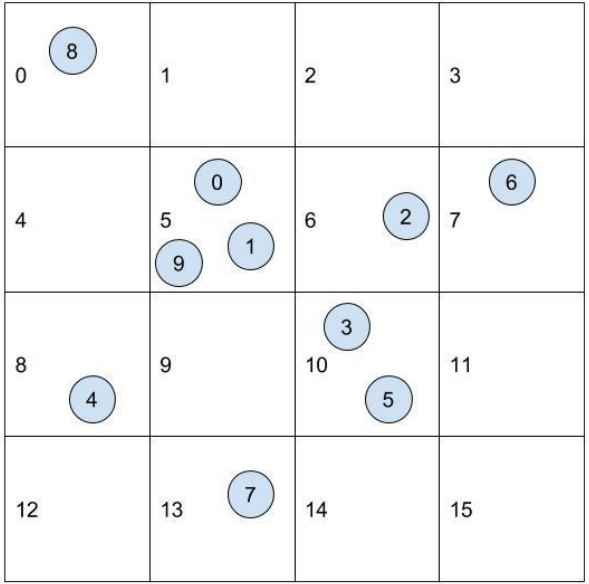
Boid index	0	1	2	3	4	5	6	7	8	9
Pos + Vel	a	b	c	d	e	f	g	h	i	j

- Sort [Grid cell index][Boid index] by [Grid Cell index]
- Walk over sorted [Grid cell index] to establish pointers with [Grid cell index][Cell data pointers]

Grid cell index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cell data pointers		x	x	x	x					x		x	x		x	x

Grid cell index	0	5	5	5	6	7	8	10	10	13
Boid index	8	0	1	9	2	6	4	3	5	7

Boid index	0	1	2	3	4	5	6	7	8	9
Pos + Vel	a	b	c	d	e	f	g	h	i	j



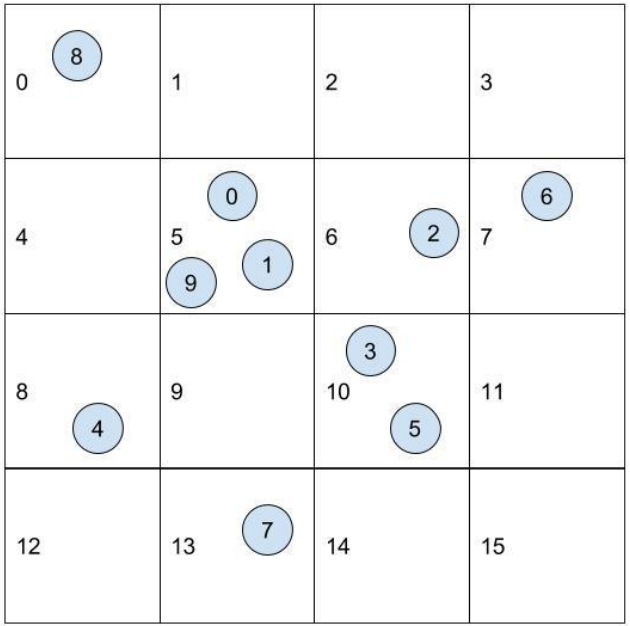
Semi-Coherent Uniform Grid

- **Naive Uniform Grid boid access pseudocode:**
  - For each cell containing potential neighbors to “thisBoid”:
  - Look at the cell’s start and end indices into the array of “pointers” -> data indices
  - For each “pointer,” chase down the data in the actual position/velocity buffers
- **Preprocess to avoid chasing pointers**
  - For each boid “pointer” at index “i” in the “pointer” array, move the pointer’s data to index “i” in a new array.
  - For each cell containing potential neighbors to “thisBoid”:
  - Look at the cell’s start and end indices into the array of “pointers” -> data indices
  - Since we have the boid data rearranged into an array that is parallel to the “pointers”, we access this directly

Grid cell index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cell data pointers	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Grid cell index	5	5	6	10	8	10	7	13	0	5
Boid index	0	1	2	3	4	5	6	7	8	9

Boid index	0	1	2	3	4	5	6	7	8	9
Pos + Vel	a	b	c	d	e	f	g	h	i	j



- Sort [Grid cell index][Boid index] by [Grid Cell index]
- Walk over sorted [Grid cell index] to establish pointers with [Grid cell index][Cell data pointers]
- Rearrange [Pos + Vel] to match [Boid index] and directly acces that

Grid cell index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cell data pointers		x	x	x	x					x		x	x		x	x

Grid cell index	0	5	5	5	6	7	8	10	10	13
Boid index	8	0	1	9	2	6	4	3	5	7
Pos + Vel	i	a	b	j	c	g	e	d	f	h

Tips

- We didn’t define all the kernels you will need… pseudocode gives hints
- Use “Boids::unitTest” to individually test stages described in pseudocode
- Naive uniform grid -> semi-coherent uniform grid should involve a LOT of copy-paste
- Consider the indexing order in “gridIndex3Dto1D”:

  x + y * gridResolution + z * gridResolution * gridResolution

  - If you use nested loops to walk over a “chunk” of the uniform grid, what order should you nest? Goal is to keep access as contiguous as possible!

Additional Reading

- All this neighbor-search stuff is drawn straight from SPH, particle-based fluid simulation!
- Muller, Charypar, Gross - *Particle-Based Fluid Simulation for Interactive Applications*
- Ihmsen, Akinci, Becker, Teschner - *A parallel SPH implementation on multi-core CPUs*

Notes adapted from Gary Li, 2016