Fixed-radius near neighbors searching for 2D simulations on the GPU using Delaunay triangulations

Heinich Porro[1], Benoit Crespin[1], Nancy Hitschfield[2] and Cristobal Navarro[3].

1 XLIM UMR CNRS 7252, University of Limoges, France
2 Computer Science Department (DCC), University of Chile
3 Instituto de Informática, Facultad de Ciencias de la Ingeniería, Universidad Austral de Chile

1. Problem statement

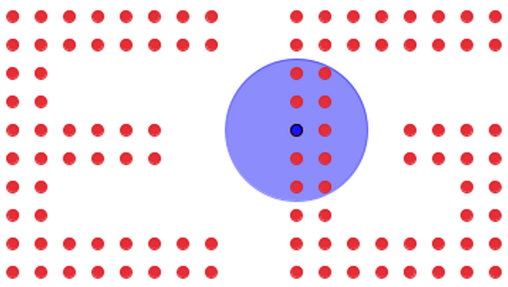

Fig 1. Fixed radius near neighbors (FRNN) problem.

Particle-based techniques are widely used for simulations in various scientific fields. We focus our research on simulations were particles move slowly.
The main computational bottleneck on those simulation scenarios is related to the neighbors search. Our contribution is an improved data structure based on the work of [CHNS18], which allows to find the neighbors of any given vertex within a fixed radius using a BFS approach.

2. Related Work

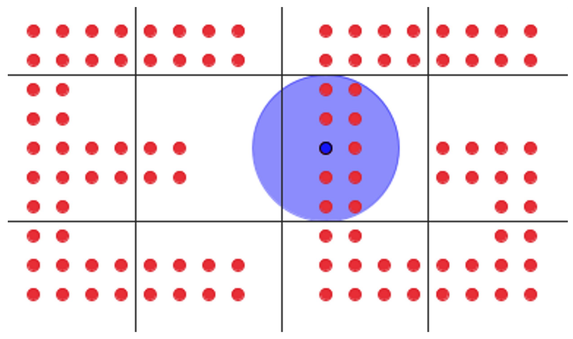

Fig 2. Grid approach to solve FRNN.

Grid-based approaches are commonly used to solve this problem in the GPU, because they are robust and easy to program.
In simulations were particles move slowly, GPU approaches based on neighbor lists (or Verlet lists) have been developed too.
GPU Delaunay triangulations have been used in [CHNS18] to check collisions in particle systems affected by short-range forces under small displacements in 2D.

3. Our method

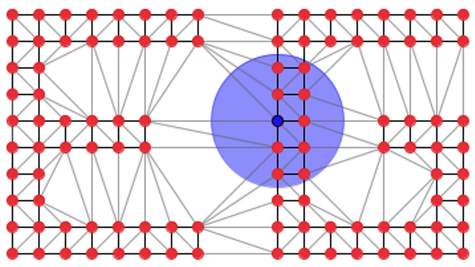

Fig 3. Delaunay approach to solve FRNN.

We propose a GPU acceleration technique for the fixed-radius near neighbors problem (or FRNN), using Delaunay triangulation in 2D.
We maintain a Delaunay triangulation of the set of moving points, using a half edge data structure adapted to perform parallel flips efficiently in the GPU, in a similar way as Carter et. al. did in [CHNS18]. Later we use the data structure to calculate the neighborhood of every vertex in parallel.

4. Parallel Delaunay triangulation update

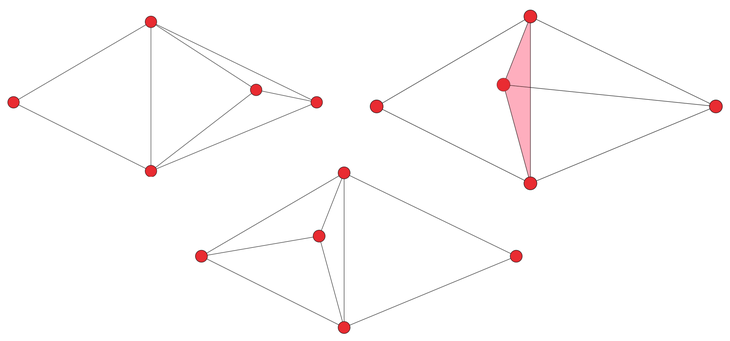

Fig 4. If a point goes through one edge, we can flip that edge in order to update the triangulation.

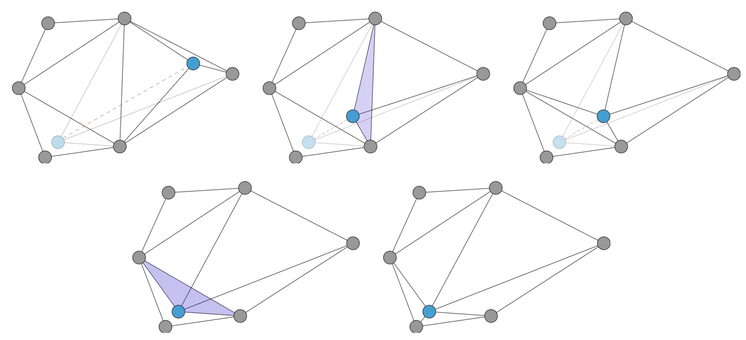

Fig 5. If a point goes trough more than one edge, we divide the displacement so that it crosses only one edge per iteration of the algorithm.

In order to update the Delaunay triangulation we first check if the triangulation is still valid after one iteration of the simulation, and then we check and fix the Delaunay condition. To fix the triangulation we map a thread to every edge and check if one of the opposite vertices went trough that edge. If that's the case, we proceed as depicted in figures 4 and 5. Finally, we check the Delaunay condition mapping a thread to every edge, and flipping the ones that have to be flipped.

5. Parallel BFS starting from every vertex

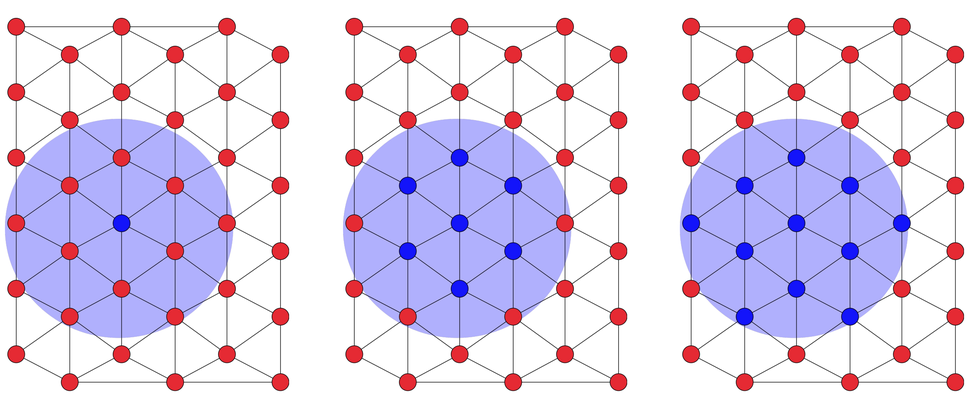

Fig 6. BFS on a Delaunay defined graph with an increasing **frontier** vertex array.

We start the algorithm by allocating an array per vertex called **frontier** with a fixed size, where we are going to store all the neighboring vertices that have been discovered so far. The size is chosen based on the maximum number of neighbors in the simulation.

To compute the neighborhood, we map one thread to every vertex of the triangulation, and start keeping track of the current frontier we are growing using 2 pointers to the **frontier** array. The first frontier is only the vertex we are starting from (Fig 6. left) and the next one is its 1-ring neighborhood (Fig 6. middle). Then the next frontiers are computed by iterating over every vertex in the current frontier, calculating its 1-ring neighborhood, and adding to the **frontier** array every point that has not been found before. We stop the search when we find points that are outside the fixed radius (Fig 6. Right).

The parallelization in our work is different from other parallel BFS approaches, as for example [TWS19], where the BFS starts only from one vertex. However, it could be possible to apply some techniques described there and parallelize ever further the BFS. We could compute the 1-ring neighborhood of every point in the current frontier and check if it discovers new points in parallel in order to compute the next frontier. This could be done in the same block so that we could utilize effectively shared memory to check if a point has been discovered before, and write to global memory only once per block.

6. Experimentation and discussion

We implemented a 2D version of the Boids model simulation [Rey87] and compared the performance of our method against the grid-based acceleration described in [Gre10]. Both methods are parallel and run completely on the GPU in double precision. Our code was tested on Windows and Linux with CUDA 11.2. Table 1 shows the computation times (in ms) for one timestep, averaged over a simulation of 100 timesteps on two GPUs; NVIDIA RTX3090 (24GB) and NVIDIA A100 (40GB). We experimented with different configurations with a varying number of particles and a scale factor controlling the size of the simulation box and the average number of neighbors inside the interaction radius. These preliminary results show that our approach is faster than grid based methods in various situations, but more research is needed to explore other types of particle based simulations.

Table 1: Performance results for RTX3090 and A100 GPUs on Boids simulations.

Particles	Scale	Avg Neighbours	GPU	Delaunay			Grid		
				Mesh update	Neighb. search	Total (ms)	Construction	Neighb. search	Total (ms)
100K	64	287	RTX 3090	2	318	320	1	253	254
			A100	1	114	115	1	23	24
100K	128	89	RTX 3090	1	101	**102**	1	233	**234**
			A100	0	31	31	1	20	21
1M	64	22	RTX 3090	4	218	**222**	2	270	**273**
			A100	2	61	63	2	20	22
1M	128	8	RTX 3090	4	40	**44**	2	262	**264**
			A100	2	8	**10**	1	19	**20**

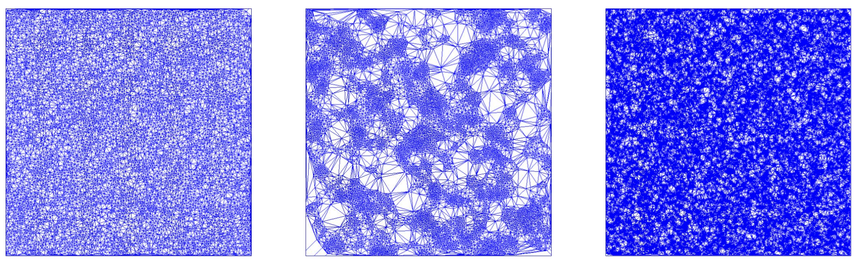

Fig 7. Delaunay triangulation of boids. From left to right: Initial configuration of 10K boids, 10K boids after 100 iterations and 100K boids.

References

[CHNS18] CARTER F., HITSCHFELD N., NAVARRO C. A., SOTO R.: Gpu parallel simulation algorithm of brownian particles with excluded volume using delaunay triangulations. Computer Physics Communications 229 (2018), 148–161.
[Gre10] GREEN S.: Particle simulation using cuda. NVIDIA whitepaper 6 (2010), 121–128.
[TWS19] TÖDLING D., WINTER M., STEINBERGER M.: Breadth-first search on dynamic graphs using dynamic parallelism on the gpu. In 2019 IEEE High Performance Extreme Computing Conference (HPEC) (2019), pp. 1–7.
[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput. Graph. 21, 4 (1987), 25–34.

This research was supported by the Patagón supercomputer of Universidad Austral de Chile (FONDEQUIP EQM180042), ANID FONDECYT grants #1211484, #1221457 and the SOMA-DNS project (ANR-20-CE46-0004).

Source code: gitlab.com/hporro01/mcleap

EG'22 TheEvent