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

H. Porro[1], B. Crespin[1], N. Hitschfeld-Kahler[2] and C. 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

Abstract

We propose to explore a GPU solution to the fixed-radius nearest-neighbor problem in 2D based on Delaunay triangulations. This problem is crucial for many particle-based simulation techniques for collision detection or momentum exchange between particles. Our method computes the neighborhood of each particle at each iteration without neighbor lists or grids, using a Delaunay triangulation whose consistency is preserved by edge flipping. We study how this approach compares to a grid-based implementation on a flocking simulation with variable parameters.

CCS Concepts

• ***Computing methodologies*** → ***Physical simulation; Massively parallel and high-performance simulations;***

1. Problem statement

Particle-based techniques are widely used for simulations in various scientific fields, for example in material science at different scales, from classical molecular dynamics to Brownian dynamics, up to the discrete element method. Two important characteristics of particle-based techniques are: (i) the main computational bottleneck is related to neighbours search, hence optimizing this task is crucial as it is needed for the computation of collision detection or momentum exchange between particles; (ii) there is a much larger need of data processing than storage, making the problem suitable for a massively parallel solution, for example with GPUs.

In this paper we propose to explore a GPU acceleration technique for the fixed-radius near neighbors problem (or FRNN), using Delaunay triangulations in 2D. Unlike traditional approaches based on cell-linked lists or neighbor lists, we extend the work presented by Carter et al. [CHNS18], where edge flips are used to preserve the Delaunay condition for all triangles at each step of the simulation. Our contributions include an enhanced data structure (see Fig. 1) for storing additional topological relationships with the Delaunay triangles, which are maintained consistently when particles move during the simulation. This structure is then used to compute the neighborhood of all particles within a fixed radius of every particle in parallel, and calculate their interactions.

We study the benefits of our implementation on the famous "Boids" simulation of the flocking behaviour of birds [Rey87] (see snapshots on Fig. 7 on the accompanying poster). By varying different parameters such as the interaction radius and the number of particles, we are able to spot the differences in performance and memory consumption compared to a grid-based approach.

2. Previous work

Many GPU implementations of FRNN search are already available in existing simulation software. Cell-linked lists or grid-based approaches are the most common, sorting the particles placed in a uniform grid and then exploring the neighborhood in contiguous cells [Hoe14, GKK19]. Such a method is used for example in [HKG*] for molecular dynamics, as well as in most particle-based fluid simulations in computer graphics.

In molecular simulations and other scientific fields, an important feature is the limited displacement of particles at each iteration, either because the timestep is very short or because the simulated motion involves small displacements, as in Brownian dynamics. Specific GPU approaches based on neighbor lists (or Verlet lists) have therefore been developed which aim at maintaining, for each particle, a list of neighboring particles during several iterations of the calculation [TCCV18]. Delaunay triangulations have also been applied to track particles affected by short-range forces under small displacements in 2D [CHNS18]. This type of graph connects all the particles with its nearest neighbor by an edge, and this property is used to efficiently check and solve overlaps between particles. After particles change their position, the Delaunay condition is restored at each iteration by applying edge-flips in parallel [NHKS]. The triangulation is not intended to speed up the displacement calculation; a Verlet list is used to keep neighboring particles within a given radius for several iterations [CHNS18].

DOI: 10.2312/egp.20221002

Our method uses the same principle of a Delaunay triangulation whose coherence is preserved by edge flips, but this time to compute directly the neighborhood of each particle without using neighbor lists. This operation is similar to various graph or tree traversal problems, for example to determine connected components in an undirected graph using DFS (Depth-First Search) or BFS (Breadth-First) algorithms. Several GPU implementations were proposed for this problem [TWS19, LWH10], however our case is different since we have to search, for each vertex of the Delaunay triangulation, the set of vertices reachable within a given radius.

3. Mesh update

The main idea of our approach is to maintain a consistent Delaunay triangulation of the particles at each iteration of the simulation, i.e. when particles are displaced to new locations because of physical interactions. Different situations can occur after a particle moves. If the displacement is very small, the Delaunay triangulation may be still valid and no further action is required. If the particle moves inside the polygon formed by the points of its one ring neighbourhood or cross only one edge, then the Delaunay condition can be restored using edge flips [NHKS].

The most difficult situation occurs when a particle crosses more than one edge, as shown on the accompanying poster. In this case we determine an intermediate position and restore the Delaunay condition with this smaller displacement. This algorithm is guaranteed to stop since we assume that there are no points going through each other, which is a common constraint in physically-based simulations where particles usually repel each other and are not allowed to overlap.

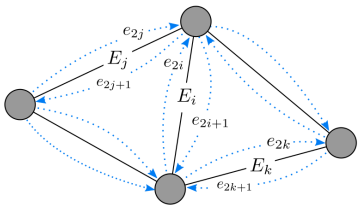

Figure 1: *Example of triangulation stored in our data structure. Every edge is represented by 2 half-edges storing their respective next edge index, keeping a CCW order on every triangle.*

4. Breadth first search to calculate the FRNN

We apply a BFS on the Delaunay triangulation by mapping one thread to every vertex. The upper bound for the number of points to visit has to be set manually depending on the simulation. We make use of this maximal value to allocate a fixed amount of memory per vertex and use it as a buffer to check if a point has been already visited by the BFS.

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 even further. For example 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.

5. Experiments

We implemented a 2D version of the Boids model simulation [Rey87] and compared the performances of our method against the grid-based acceleration described in [Gre10]. Both methods are parallel and run completely on the GPU in double precision.The code is available at `gitlab.com/hporro01/mcleap`, and was tested on Windows and Linux with CUDA 11.2. Table 1 in the accompanying poster shows the computation times (in ms) for one timestep, averaged over a simulation of 100 timesteps tested on two GPUs; NVIDIA RTX3090 (24GB) and NVIDIA A100 (40GB). Different configurations were applied, with a varying number of particles and a scale factor controlling the size of the simulation box and the average number of neighbours inside the interaction radius.

Acknowledgements

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).

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. 1

[GKK19] GROSS J., KÖSTER M., KRÜGER A.: Fast and Efficient Nearest Neighbor Search for Particle Simulations. In *Computer Graphics and Visual Computing (EG UK)* (2019). 1

[Gre10] GREEN S.: Particle simulation using cuda. *NVIDIA whitepaper 6* (2010), 121–128. 2

[HKG*] HERMOSILLA P., KRONE M., GUALLAR V., VÁZQUEZ P.-P., VINACUA À., ROPINSKI T.: Interactive gpu-based generation of solvent-excluded surfaces. *The Visual Computer 33*, 6. 1

[Hoe14] HOETZLEIN R. C.: Fast fixed-radius nearest neighbors: interactive million-particle fluids. In *GPU Technology Conference* (2014), vol. 18, p. 2. 1

[LWH10] LUO L., WONG M., HWU W.-M.: An effective gpu implementation of breadth-first search. In *Design Automation Conference* (2010), pp. 52–55. 2

[NHKS] NAVARRO C., HITSCHFELD-KAHLER N., SCHEIHING E.: A parallel gpu-based algorithm for delaunay edge-flips. In *The 27th European Workshop on Computational Geometry, EuroCG*. 1, 2

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph. 21*, 4 (1987). 1, 2

[TCCV18] TRAN C. T., CRESPIN B., CERBELAUD M., VIDECOQ A.: Colloidal suspension by srd–md simulation on gpu. *Computer Physics Communications 232* (2018), 35–45. 1

[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. 2