Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Delaunay triangulations for fixed-radius nearest neighbors on GPU

Propuesta de tema de tesis para optar al grado de Magíster en Ciencias, Mención Computación

Heinich Porro Sufan

Profesora guía:
Nancy Hitschfeld

Santiago, Chile
Mayo 2020

1 Introduction

In the last years, the world of computer graphics and high performance computing (HPC) has seen the emergence of new and more powerful graphics processing units (GPUs). That development, along with low cost commercial GPUs, has lead to an increasing interest of the scientific community towards novel algorithms that take advantage of its architecture.

In the domain of particle based fluids simulations, the operation to find the neighbors of each particle is the most costly operation that has to be performed at each iteration of the simulation. The problem of finding such a neighborhood in a simulation setup, where each particle moves at each timestep depending on its neighborhood, is called kinetic fixed-radius nearest neighbors (FRNN), and the most utilized approaches to solve it are inherited from other computer graphics domains.

This work will focus on the development of a new data structure that solves the kinetic version of fixed-radius nearest neighbors using a GPU parallel Delaunay triangulation algorithm [1], making its calculation faster in interactive simulation settings.

2 Problem Statement

Smoothed particle hydrodynamics (SPH) is a simulation method used in interactive graphics to simulate realistic fluids. In this method, the simulated fluid is discretized as a set of particles. The more particles the simulation can handle, the more precise it gets. Therefore, it is desired that the simulation has as much particles as possible. For an extensive explanation on how SPH works, refer to [2, 3].

Each of those particles represent the fluid motion, maintaining some properties about it, such as mass, density, velocity and position, which changes among time depending on its properties and their neighbors. To approximate those properties over time, it is required to calculate each particle's neighborhood. That is defined by most of the simulation methods, as the set of particles that are closer than a given threshold to itself. In other words, this is a variation of the fixed radius nearest neighbors (FRNN) in a dynamic setting.

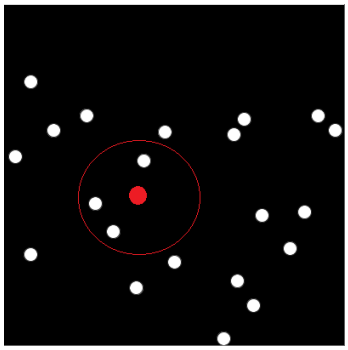


Figure 1: The FRNN of the red particle is the set of white particles within the red circle.

The main aim of this work is to speed up the calculation of that FRNN for real time fluid simulation using a Delaunay triangulation kept in the GPU as particles move in 2D. Then, explore the possibility to extend the same approach to a sequential 3D FRNN calculation using Delaunay tetrahedralizations, following the work done in [4].

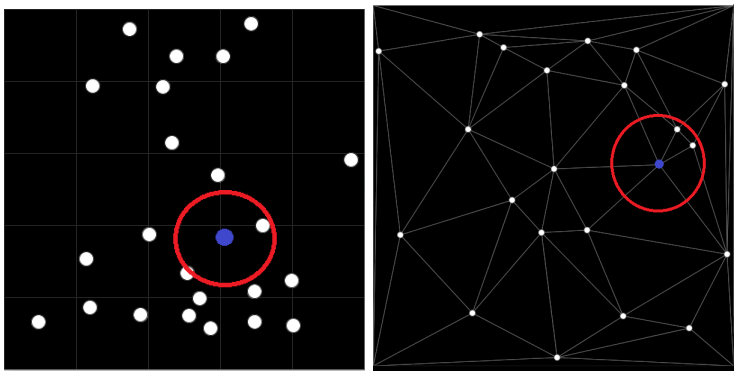
3 Related work

There have been a few ways to solve the problem, and most of them have been inherited from the computer graphics field. For example, some data structures designed for accelerate ray-object intersections have been used to calculate FRNN in fluid simulations. Some of these approaches have been also parallelized to speed up neighborhood computation, in CPU, and also in GPU.

The most commonly used data structures to solve real time FRNN for SPH are the cell based ones, sometimes called cell lists, or cell linked lists. The idea of these data structures is to discretize the 2D or 3D space in boxes with a size equal to the SPH threshold of influence between particles, so that for each particle is known that only particles located in the same cell and the neighboring cells could be part of the FRNN. Commonly,

this data structure is built once per frame of the simulation taking $O(n)$ time, and taking a query time, for a given particle, proportional to the number of particles inside the cell where is located the particle, plus the particles located in cells around it (8 cells in 2D and 26 in 3D). There are approaches to parallelize the build, update and query operations, as in [5], where the cell lists algorithm was adapted to run in GPU, and also approaches in which the cells are not built again but updated, according to the displacement of the particles, as noted in [6], which saves some computation time. There are also improvements in the structure to decrease the cache miss rate sorting the particles data as close as possible to its neighbors [6, 7].

A common data structure used to calculate short range forces in simulations, as SPH forces, is the Verlet list [8]. It stores a list for each particle, enumerating every particle within a cutoff radius around it (usually bigger than the FRNN radius). These lists are not updated in every iteration of the simulation, given that it is known that those particles will be the only relevant ones in the following steps to calculate short range forces. If we consider that there are n particles in the simulation, the Verlet list is built in $O(n^2)$ time, queried in $O(k)$ time -with k the number of particles within the cutoff radius-, and updated in $O(n^2)$ time, but as the structure is not updated at every simulation step, it works faster than other data structures in some simulation scenarios, specially in which all the particles move very slow.



(a) Grid representing the Cell list (b) Delaunay triangulation

Figure 2: Different version of neighborhood search. The red circle represents the radius of the neighborhood that corresponds to the blue particle.

We propose to extend the algorithm developed in [9], which is able to maintain a Delaunay triangulation in GPU for slowly moving particles, to compute and update the FRNN of each particle in parallel in a SPH fluid simulation, where particles might move faster. This algorithm, described in [9], uses the Delaunay triangulation to efficiently detect the overlaps between particles and correct them but not for computing the forces.

First, the algorithm builds a Delaunay triangulation of the starting position of the particles in the host device, using a $O(nlog(n))$ time sequential algorithm [10]. Then, it transfers the triangulation data to the GPU, starting the simulation loop. For each timestep of the simulation, it updates each particle's position applying long-range forces. The integration leads to two important events regarding the Delaunay triangulation:

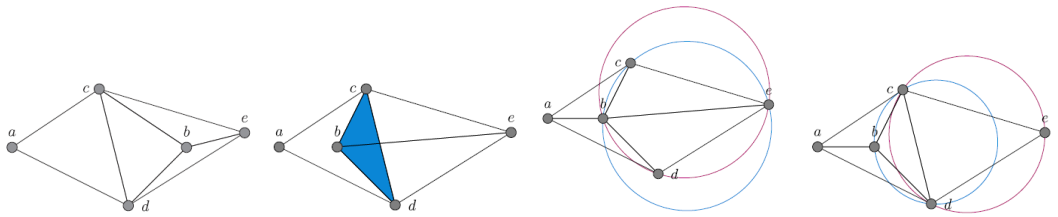
1. It might get invalidated due a particle going through an edge or another particle, in which case it can be fixed by flipping the triangles as shown in figure 3. The approach used in [9] to solve this issue only works with slow moving particles, as it only handles the case showed in 3, not cases where the particle goes through more than 1 edge or particle, as the one shown in figure 4, so it will be necessary to make some modifications to the triangulation data structure in order to handle fast moving particles (very commonly found in fluid simulations). Once inverted triangles are corrected, the second event described can also happen.

2. Even thought still valid, the triangulation might no longer be Delaunay, in which case is corrected using the procedure described in [1], which flips in parallel each edge that does not fulfill the Delaunay condition.

As a Delaunay triangulation is kept and updated in the GPU, the FRNN can be calculated performing

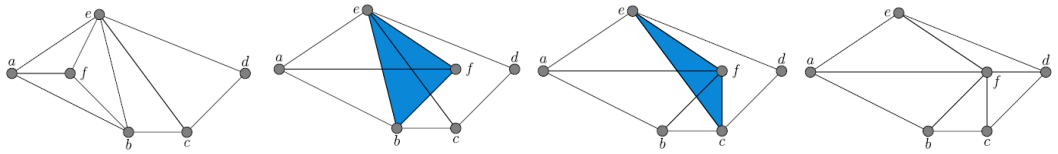
a breath first search starting at each node in parallel. For each particle, this search is bounded by the number of calculated fixed radius neighbors. A similar idea in a sequential static setting, was developed before in [11].

To summarize, the data structure is built in the host device in $O(nlog(n))$ time, updated in close to $O(1)$ time [9], depending on the particles movement on each time step (if a particle moves too much, it might cause many inverted triangles), and queried for all particles in parallel in $O(k)$ time for each, where k is the size of the answer, or number of neighbors.



(a) Starting simulation of 5 particles. (b) Particle b moves over the edge **dc** (c) Edge flip between **ab** and **cd** (d) Edge flip between **cd** and **be**

Figure 3: Inverted triangle correction with one edge flip. Images took from [9] with author's permission.



(a) Starting situation of two triangles. (b) Particle f after integration. (c) Edge flip between **be** and **cf** (d) Edge flip between **ce** and **df**

Figure 4: Inverted triangle correction with two edge flips. Images took from [9] with author's permission.

4 Research Questions

- Is a Delaunay based data structure a competitive approach in GPU based simulations which need the construction and update of each particle neighborhood? Is it as fast as other methods in some simulation domains (fluids, granular materials, etc.)?
- What is the scope of the proposed approach? Which simulation applications benefit from it? (if it is not competitive in fluid simulation, it may be useful in, for example, granular material simulation, or molecular dynamics simulation, or maybe in some fluids simulations where the particles do not move so fast, etc).
- If it is true that the proposed method is faster than cell based ones only in slower or denser particle regions. What is the threshold of that benefit? How dense or slow has to be a set of particles to take advantage of this approach?

5 Hypothesis

A Delaunay GPU approach will be faster than all other state of the art methods for solving fixed-radius nearest neighbors in an interactive setting where particles move slow or have a dense neighborhood.

6 Objectives

6.1 General Objectives

The general objective of this thesis is to design and build a parallel algorithm based in Delaunay triangulations, to calculate kinetic fixed radius nearest neighbors. The algorithm specified in [9] will be changed, in order to handle several inverted triangle corrections and to do it faster than the current approach. The designed algorithm will be evaluated in different SPH fluid simulations to find in which ones, and under which circumstances, works better than current methods.

6.2 Specific Objectives

- Design and implement an algorithm to solve the 2D version of fixed radius nearest neighbors based in GPU Delaunay triangulation, using the library developed in [1], extending the algorithm designed in [9].
- Compare and show that the algorithm designed is faster than a sequential solution of the problem.
- Compare and show that the designed algorithm is faster than other parallel approaches -as Verlet lists or cell lists- in some simulation scenarios, for example, in static or low velocity areas in fluids.
- Develop an open source library to do benchmarks and to ease the integration of this new algorithm and data structure to other people code.
- Explore the extension of the 2D approach to 3D, using the data structures and functions that TetGen library provides, specifically the ones described in [4].

7 Methodology

7.1 Research

The first step in the research is to do a short review of the state of the art methods for solving the dynamic FRNN problem (last 5 years). There are not many approaches to solve the problem listed in the related work section, which is a reflect of the state of the art of the problem in the sense that there are not a lot of ways to solve it -even thought there are a lot of different variations of cell lists and Verlet lists used-. On top of that, those methods are hidden in papers in the fluid simulation community (and other fields of simulation), so it may be hard to do a review based in FRNN keywords. Instead, it is proposed to do a review in the field of SPH interactive fluid simulation looking for how this problem is solved in this particular domain.

7.2 FRNN library

It will be useful to develop a library in order to integrate the algorithm to develop simulations, do benchmarking and integrate it in other people's code. It is planned to be based on the library cleap, developed in [1], and the extensions done in [9]. It will be developed in CUDA/C++ because of the maturity of the API, familiarity with the language and also because said libraries are written in those languages.

7.3 Experimentation

- Gather and make sets of experiments with different settings of particles varying in velocity and density. Experiment with different sized particles might be interesting too. Also, classic simulation scenarios of the area of fluid simulation will be included, such as dam break flood, fluid flowing around obstacles, etc.
- Use those sets to compare the proposed approach with other state of the art algorithms for solving FRNN in a simulation setting. The author already counts with a parallel cell list and parallel verlet list implementation. Some interesting comparison criteria are: Update and query times of the neighborhoods with a fixed number of particles, and the time took to add and eliminate particles during the simulation.

8 Expected Results

- A new algorithm that speeds up the computation of the FRNN for 2D interactive simulation.
- An open source library to facilitate the integration of the algorithm into simulation development.
- Determination of the scope of application of the algorithm (which simulation scenarios take most advantage from it).
- A short review of the state of the art methods (last 5 years) for solving the dynamic FRNN problem.

References

[1] Cristóbal A. Navarro, Nancy Hitschfeld-Kahler, and Eliana Scheihing. "A GPU-based Method for Generating quasi-Delaunay Triangulations based on Edge-flips". In: *GRAPP/IVAPP*. 2013.

[2] Micky Kelager. "Lagrangian fluid dynamics using smoothed particle hydrodynamics". In: *University of Copenhagen: Department of Computer Science* (2006).

[3] Matthias Müller, David Charypar, and Markus Gross. "Particle-based fluid simulation for interactive applications". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2003, pp. 154–159.

[4] Franco Dassi, Lennard Kamenski, Patricio Farrell, and Hang Si. "Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction". In: *Computer-Aided Design* 103 (2018), pp. 2–13.

[5] Rama Hoetzlein. "Fast fixed-radius nearest neighbors: interactive million-particle fluids". In: *GPU Technology Conference*. Vol. 18. 2014.

[6] Zhenhua Yao, Jian-Sheng Wang, Gui-Rong Liu, and Min Cheng. "Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method". In: *Computer physics communications* 161.1-2 (2004), pp. 27–35.

[7] Pedro Gonnet. "A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations". In: *Journal of Computational Chemistry* 28.2 (2007), pp. 570–573.

[8] Loup Verlet. "Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules". In: *Physical review* 159.1 (1967), p. 98.

[9] Francisco Carter, Nancy Hitschfeld, Cristóbal A Navarro, and Rodrigo Soto. "GPU parallel simulation algorithm of Brownian particles with excluded volume using Delaunay triangulations". In: *Computer Physics Communications* 229 (2018), pp. 148–161.

[10] Steven Fortune. "A sweepline algorithm for Voronoi diagrams". In: *Algorithmica* 2.1-4 (1987), p. 153.

[11] Volker Turau. "Fixed-radius near neighbors search". In: *Information processing letters* 39.4 (1991), pp. 201–203.

[12] Daniel Price. "Smoothed particle hydrodynamics and magnetohydrodynamics". In: *Journal of Computational Physics* 231.3 (2012), pp. 759–794.