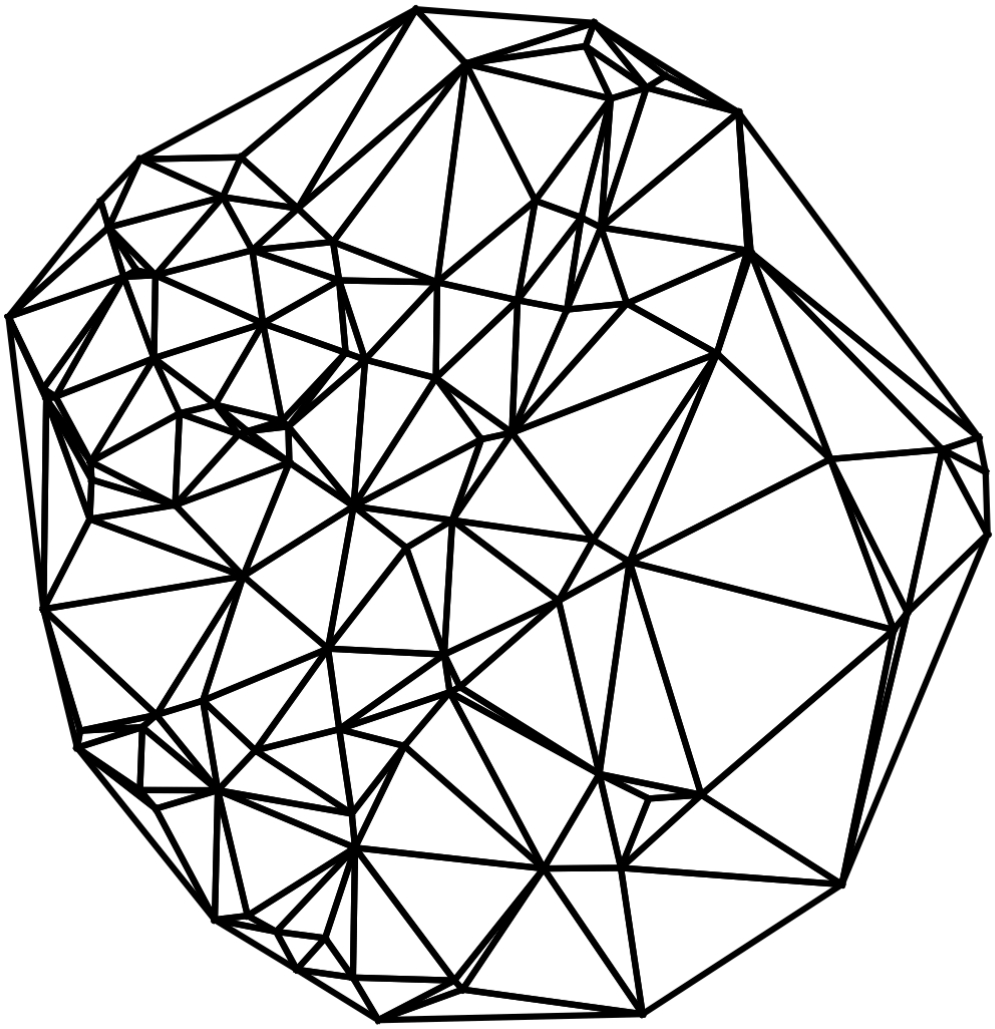
Delaunay Triangulations on the GPU

Patryk Drozd
Trinity College Dublin
drozdp@tcd.ie



Contents

Abstract

Triangulations of a set of points are a very useful mathematical construction to describe properties of discretised physical systems, such as modelling terrains, cars and wind turbines which are a commonly uses for simulations such as compulational fluid dynamics or other physical properties, and even have use in video games for rendiring and visualising complex geometries. To paint a picure you may think of a triangulation given a set of points P to be a bunch of line segments connecting each point in P in a way such that the edges are non instersecting. A particulary interesting subset of triangulations are Delaunay triangulations (DT). The Delaunay triangulation is a triangulation which maximises all angles in each triangle of the triangulation. Mathematically this gives us an interesting optimization problem which leads to some rich mathematical properties, at least in 2 dimensions, and for the applied size we have a good way to discretize space for the case of simulations with the aid of methods such as Finite Element and Finite Volume methods. Delaunay triangulatinos in particular are a good candiate for these numerical methods as they provide us with fat triangles, as opposed to skinny triangles, which can serve as good elements in the Finite Element method as they tend to improve accuracy [1].

There are many algorithms which compute Delaunay triangulations (cite some overview paper), however alot of them use the the operation of 'flipping' or originally called an 'exchange' [2]. This is a fundamental property of moving through triangulations of a set of points to with the goal of optaining the optimal Delaunay triangulation. This flipping operation involves a configuration of two triangles sharing an edge, forming a quadrilateral with its boundary. The shared egde between these two triangles will be swapped or flipped from the two points at its end to the other two points on the quadrilateral. The original agorithm motivated by ([2]) is hinted to be us this flipping operation to iterate through different triangulations and eventually arrive at the Delaunay trianglution which we desire.

With the flippig operation being at the core of the algorithm, we can notice that is has the possibility of being parallelized. This is desirable as problems which commonly use the DT are run with large datasets and can benefit from the highly parallelisable nature of this algorithm. If we wish to parallize this idea, and start with some initial triangulation, conflicts would only occur if we chose to flip a configuration of triangles which share a triangle. With some care, this is an avoidable situation leads to a highly scalable algorithm. In our case the hardware of choice will be the GPU which is designed with the SIMT model which is particularly well suited for this algorithm as we are mostly performing the same operations in each iteration of the algorithm in parallel.

The goad of this project is to explore the Delaunay trianglion through both serial and parallel algorithms with the goal of presenting a easy to understand, sufficiently complex parallel algorthm designed with with Nvidia's CUDA programming model for runnig software on their GPUs.

1. Delaunay triangulations

In this section I aim to introduce triangulations and Delaunay triangulations from a mathematical perspective with the foresight to help present the motivation and inspiration for the key algorithms used in this project. For the entirety of this project we only focus on 2 dimensional Delaunay triangulations.

In order to introduce indroduce the Delaunay Traingulation we first must define what we mean when we say triangultion. In order to create a triangualtion we need a set of points P which will make up the vertices of the triangles.

Definition 1.1: For a point set P, the term edge is used to indicate any segment that includes precisely two points of S at its endpoints.

Definition 1.2: A *triangulation* of a planar point set P is a subdivision of the plane determined by a maximal set of noncrossing edges whose vertex set is P [3].

(a) (b)

Figure 2: Examples of two traingulations on the same set of points. Triangulations are not unique!

A fact about triangulations is that we know how many triangles our triangulation will contains only given a set of points. This will be usefull when we will be storing triangles as we will allways know the number that will be created. For our purposes the convex hull is a boundary enclosing our triangulation will allways be known in algorithms in the following chapters.

Theorem 1.1: [4] Let P be a set of n points in the plane, not all collinear, and let k denote the number of points in P that lie on the boundary of the convex hull of P. Then any triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

A key feature of all of the Delaunay triangulation theorems we will be considering is that no three points from the set of points P which will make up our triangulation will lie on a shared line. This leads us to the following definition.

Definition 1.3: A set of points P is in *general position* if no 3 points in P are colinear and that no 4 points are cocircular.

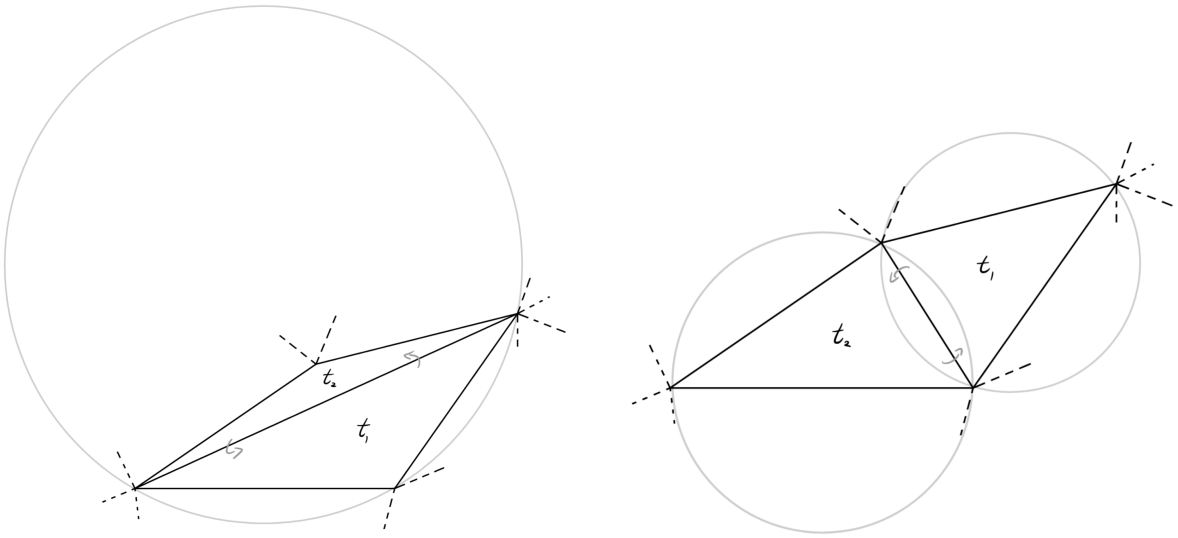
From this point onwards we will allways assume that the point set P from which we obtain our triangulation will be in *general position*. This is neccesary for the definitions and theorems we will define.

Definition 1.4: Let e be an edge of a triangulation T_1 , and let Q be the quadrilateral in T_1 formed by the two triangles having e as their common edge. If Q is convex, let T_2 be the triangulation after flipping edge e in T_1. We say e is a *legal edge* if $T_1 \geq T_2$ and e is an *illegal edge* if $T_1 < T_2$ [3]

Definition 1.5: For a point set P, a *Delaunay triangulation* of P is a triangulation that only has legal edges. [3]

Theorem 1.2 *(Empty Circle Property)*: Let P be a point set in general position, where no four points are cocircular. A triangulation T is a Delaunay triangulation if and only if no point from P is in the interior of any circumcircle of a triangle of T. [3]

Theorem 1.2 is the key ingredient in the the Delaunay triangulation algorithms we are going to use. This is because instead of having to compare angles, as is defined by Definition 1.5 we are allowed to only perform a computation, involving finding a circumcicle and performing one comparison which would involve determining whether the point not shared by triangles circumcircle is contained inside the circumcircle or not.



(a) (b)

Figure 3: Demonstration of the flipping operation. In (a) A configuration that needs to be flipped illustrated by the circumcircle of t_1 containg the auxillary point of t_2 in its interior. In (b) configuration (a) which has been flipped and no longer needs to be flipped as illustrated by the both circumcirles of t_1 and t_2.

Theorem 1.3 *(Lawson)*: [2] Given any two triangulations of a set of points P, T_1 and T_2, there exist a finite sequence of exchanges (flips) by which T_1 can be transformed to T_2.

2. The GPU

The Graphical Processing Unit (GPU) is a type of hardware accelerator originally used to significantly improve running video rendering tasks such for example in video games through visualizing the two or three dimensional environments the “gamer” would be interacting with or rendering vidoes in movies. Many different hardware accelerators have been tried and tested for more general use, like Intels Xeon Phis, however the more purpose oriented GPU has prevailed in

the market mainly lead by Nvidia and AMD, with intel now recently entering the GPU market with their Arc series. Today, the GPU has gained a more general purporse status with the rise of General Purpose GPU (GPGPU) programming as more and more people have notices that GPUs are very useful as a general hardware accelerator.

The triadional CPU (based on the Von Neumann architecture) which is built to perform *serial* tasks with helful features such as branch predicion, for dealing with if statements and vairable lenght instructions like for loops with variable lengh, The CPU is built to be a general purpose hardware for performing all tasks a user would demand from the computer. In contrast the GPU can't run alone and must be used in conjuction to the CPU. The CPU sends compute intructions for the GPU to perform and data is commonly passes between the CPU and GPU.

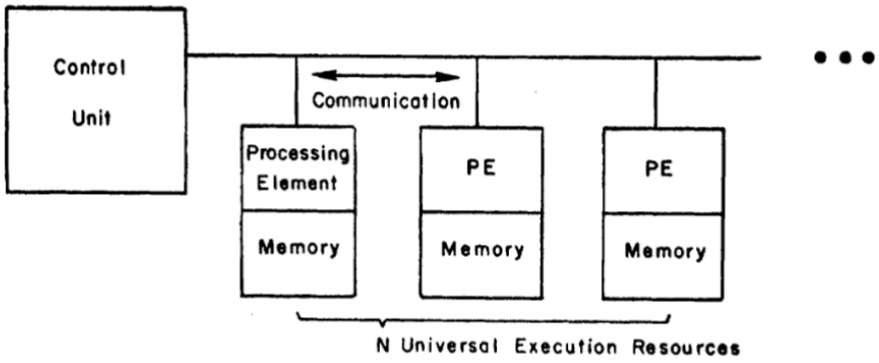


Figure 4: The *Single Instruction Multiple Threads (SIMT)* classification, originally known as an *Array Processor* as illustriated by Michael J. Flynn [5]. The control unit communicates instructions to the N processing element with each processing element having its own memory.

What makes the GPU increadibly usefull in certain usecases (like the one of this report) is its architecture which is build to enable massively parallelisable tasks. In Flynn's Taxonomy [5], the GPUs architecture is based a subcategory of the Single Instruction Multiple Data (SIMD) classification known as Single Instruction Multiple Threads (SIMT) also known as an Array Processor. The SIMD classification allows for many proccessing units to perform the same tasks on a shared dataset with the SIMT classification additionally allowing for each processing unit having its own memory allowing for more diverse proccessing of data.

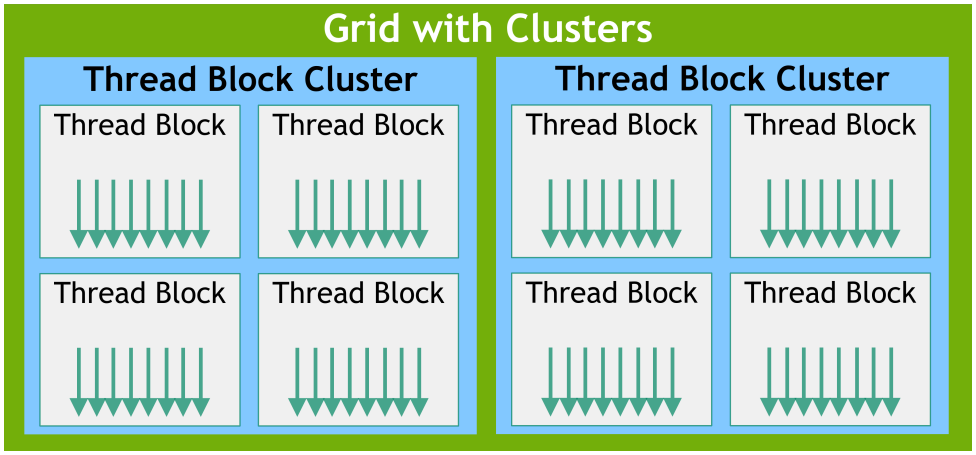


Figure 5: [6]

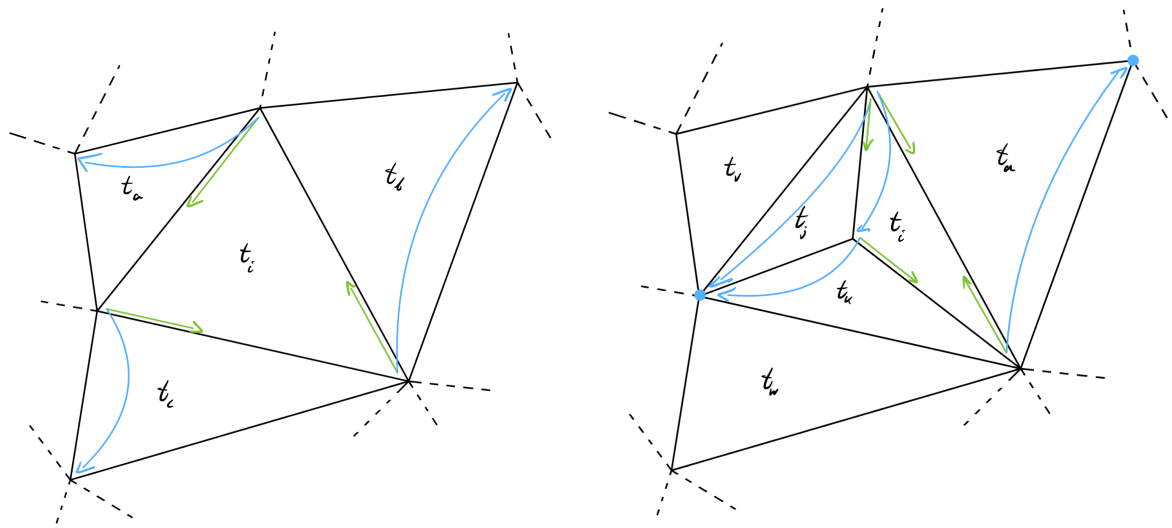
Nvida's GPUs take the SIMT model and further develop it. There are three core abstractions which allow Nvidia's GPU model to be succesfull, a hierarchy of thread groups, shared memories and

synchronization [6]. The threads, which represent the a theoretical proccesses which encode programmed instrcutions, are launched together in groups of 32 known as *warps*. This is the smallest unit of instructions that is executed on the GPU. The threads are further grouped into *thread blocks* which are used as a way of organising the shared memory to be used by each thread in this thread block. And once more the *thread blocks* grouped into a *grid*.

3. Algorithms

3.1. Serial

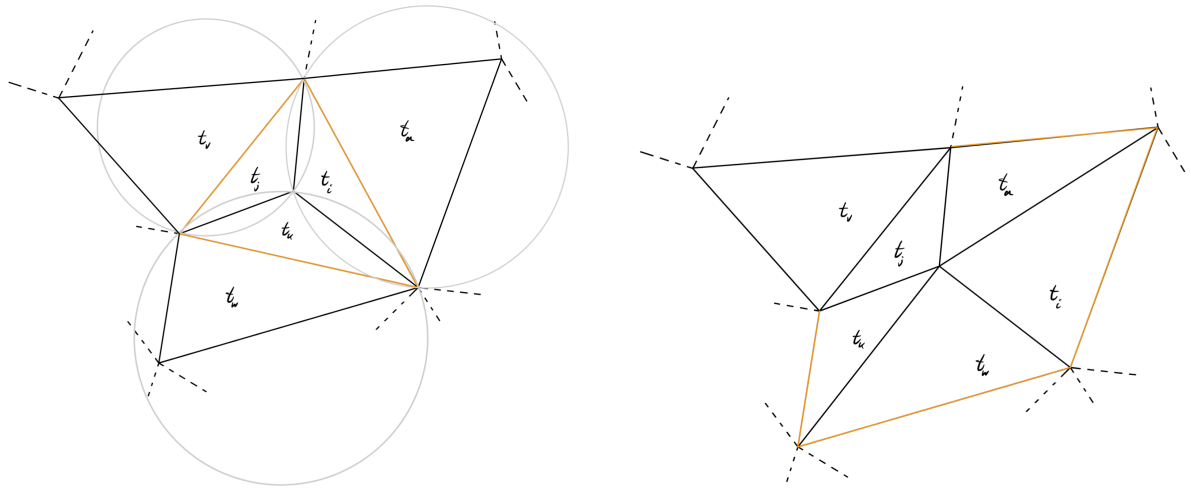
3.1.1. Point insertion



(a) (b)

Figure 6:

3.1.2. Flipping



(a) (b)

Figure 7: Illustrating the flipping operation. In figure (a), point r has just been inserted and the orange edges are have been marked to be checked for flipping. Two of these end edges end up beigh flipped in (b). The edges insidewould not quaify for flipping as any quadrilateral would not form a convex region.

Algorithm 1: Flipping

```
Initialize T as any trianulation of P
while T has an illegal edge
    for each illegal edge e
        flip e
return T
```

Algorithm 2: Randomized incremental point insertion [4]

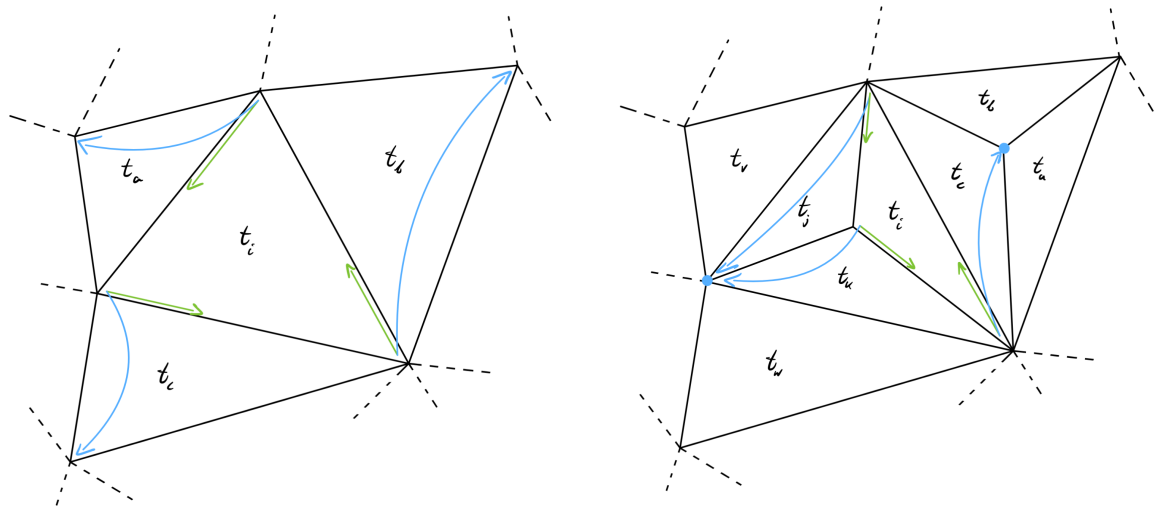
```
Initialize T with a triangle enclosing all points in P
Compute a random permutation of P
for p ∈ P
    Insert p into the triangle t containing it
    Legalize each edge in t recursively
return T
```

3.2. Parallel

Algorithm 3: Parallel point insertion and flipping [7]

```
Data: A point set P
Initialize T with a triangle t enclosing all points in P
while there are p ∈ P to insert
    for each p ∈ P do in parallel
        choose p_t ∈ P to insert
    for each t ∈ T with p_t to insert do in parallel
        split t
    while there are configurations to flip
        for each base triangle t ∈ T in a configuration marked to flip
            flip t
    update locations of p ∈ P
return T
```

3.2.1. Insertion



(a) An image of the andromeda galaxy. (b) A sunset illuminating the sky abov.

Figure 8: A figure composed of two sub figures.

3.2.2. Flipping

3.3. Data Structures

3.3.1. Triangles

The core data structure that is needed in this algorithm is one to represent a the triangulation itself. There are a handful of different approaches to this problem inculding representing edges by the qaud edge data structure [8] however we choose to represent the triangles in our triangulation by explicit triangle structures [9] which hold neccesary information about their neighbours for the construction of the trianulation and for performing point insertion and flipping operations.

```
struct Tri {
  int p[3]; // points
  int n[3]; // neighours
  int o[3]; // opposite points
};
```

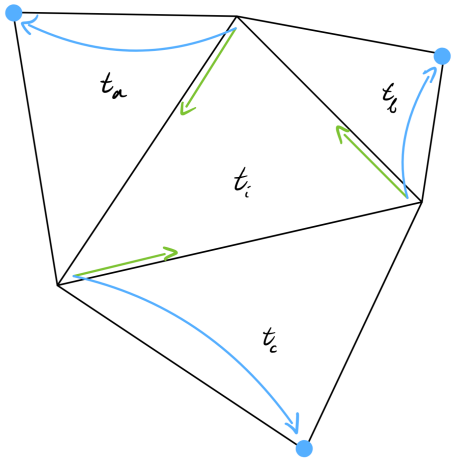


Figure 9: An illustration of the *Tri* data structures main features. We describe the triangle t_i int the figure. Oriented counter clockwise points are stored as indexes an array containing two dimensional coordinate represeting the point. The neighbours are assigned by using the right hand side of each edge using and index of the point as the start of the edge and following the edge in the CCW direction. The neighbours index will by written in the corresponding entry in the structure.

Bibliography

[1] L. Chen, “Mesh Smoothing Schemes Based on Optimal Delaunay Triangulations.,” 2004, pp. 109–120.

[2] C. L. Lawson, “Transforming triangulations,” *Discrete Math.*, vol. 3, no. 4, pp. 365–372, Jan. 1972, doi: 10.1016/0012-365X(72)90093-3.

[3] S. L. Devadoss and J. O'Rourke, *Discrete and Computational Geometry, 1st Edition*. Princeton University Press, 2011.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational geometry: algorithms and applications*. Berlin, Heidelberg: Springer-Verlag, 1997.

[5] M. J. Flynn, “Some computer organizations and their effectiveness,” *IEEE Trans. Comput.*, vol. 21, no. 9, pp. 948–960, Sep. 1972, doi: 10.1109/TC.1972.5009071.

[6] “CUDA C++ Programming Guide.” [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide

[7] T.-T. Cao, A. Nanjappa, M. Gao, and T.-S. Tan, “A GPU accelerated algorithm for 3D Delaunay triangulation,” in *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, in I3D '14. San Francisco, California: Association for Computing Machinery, 2014, pp. 47–54. doi: 10.1145/2556700.2556710.

[8] L. Guibas and J. Stolfi, “Primitives for the manipulation of general subdivisions and the computation of Voronoi,” *ACM Trans. Graph.*, vol. 4, no. 2, pp. 74–123, Apr. 1985, doi: 10.1145/282918.282923.

[9] A. Nanjappa, “Delaunay triangulation in R3 on the GPU,” 2012.