High Performance Graphics (2012)
C. Dachsbacher, J. Munkberg, and J. Pantaleoni (Editors)

A Hybrid 2-D Delaunay Triangulation Algorithm Exploiting the GPU Parallelism

Petros G. Vasileiou and Emmanouil Z. Psarakis

University of Patras, 26500 Rion-Patras, Greece

Extended Abstract

A Fast solver of Delaunay Triangulation (DT) problem constitutes one of the basic ingredients in many practical and sientific applications. Existing Graphics Processing Units (GPU) based implementations of DT algorithms suffer from two serious drawbacks. The first is related to the dependency of the CPU guidance algorithm on GPU calculations. Albeit the modern GPUs have high computational throughput, if the feedback from CPU is necessary for the algorithmic evolution, the overhead caused by CPU-GPU communication can seriously degrade the performance. The second most serious drawback is their dependency on the distribution of the given point-set. Most of the GPU-based implementations can optimally run only on uniformly distributed point-sets, however, in many practical applications this is not the case.

In this paper we propose a new GPU-based algorithm that:

- exploits the high parallelism offered by GPU while simultaneously runs exclusively on GPU under the guidance of CPU, but without the need of any feedback from GPU.
- enjoys performance which is almost independent from the distribution of the point-set.
- reduces the necessary memory footprint by using the classical geometric data structure "half-edges", which is widely used in CPU-based implementations of many computational geometry algorithms.

Problem Formulation

Let $\mathcal{T}(\mathcal{P})$ be a triangulation of a given 2-D point-set $\mathcal{P}$ having $3|\mathcal{T}(\mathcal{P})|$ angles contained in set $\mathcal{A}(\mathcal{T}(\mathcal{P})) = \{\alpha_i,\ i = 1,2,\cdots,\ 3|\mathcal{T}(\mathcal{P})|\Big|\mathcal{T}(\mathcal{P})\}$. Let $\mathcal{L}_{\mathcal{A}(\mathcal{T}(\mathcal{P}))}$ be the sorted, in ascending order, angle list of $\mathcal{T}(\mathcal{P})$ triangulation. Then, the Delaunay Triangulation $\mathcal{T}_{\mathcal{D}}(\mathcal{P})$ can result from the solution of the following optimization problem:

$$\mathcal{T}_{\mathcal{D}}(\mathcal{P}) = \text{lex} \inf_{\mathcal{T}(\mathcal{P})} \mathcal{L}_{\mathcal{A}(\mathcal{T}(\mathcal{P}))},$$

i.e. DT constitutes the lexicographic infimum over all angle lists formed from the triangulations of the given point-set $\mathcal{P}$.

Algorithm Overview

The proposed algorithm is based on the *divide* and *conquer* method (presented by Guibas and Stolfi) in combination with the *incremental insertion* strategy. Both strategies have been followed in a specific way that harnesses the high parallelism offered by the architecture of the modern GPUs. In order to be able to increase the utilization of the GPU we split the given point-set in a number of regions that is closely related to the number of GPU cores.
Our algorithm is composed by the following four discrete steps which are entirely executed on GPU:

$\mathcal{S}_1$: Partitioning of the given 2-D point-set $\mathcal{P}$ in a specified number of subsets. Each subset is sorted according to the x-coordinates of its points.

$\mathcal{S}_2$: Delaunay Triangulation of each subset by using an *order recursive* insertion algorithm tailored to exploit both GPU architecture and the special form of the sequence of the insertion problems imposed by Step $\mathcal{S}_1$ of the algorithm. Note that the complexity of this step is an increasing function of the cardinality of each subset.

$\mathcal{S}_3$: Merging of the vertical subsets in each horizontal zone. This step is implemented by mapping all the vertical subsets of each horizontal zone onto a corresponding binary tree whose depth specifies the complexity of this specific step. As it is clear, this complexity is an increasing function of the depth of binary tree which, in turn, is a decreasing function of the cardinality of subsets.

$\mathcal{S}_4$: Merging of the horizontal zones. This step is similar to the previous one, but it is, instead, applied onto the horizontal zones.

Finally, by using the fact that the complexity of step $\mathcal{S}_2$, is inversely proportional to the complexities of steps $\mathcal{S}_3$ and $\mathcal{S}_4$, we can specify almost in an optimal way, the number of subsets needed in Step $\mathcal{S}_1$ in order to partition the given point-set $\mathcal{P}$.

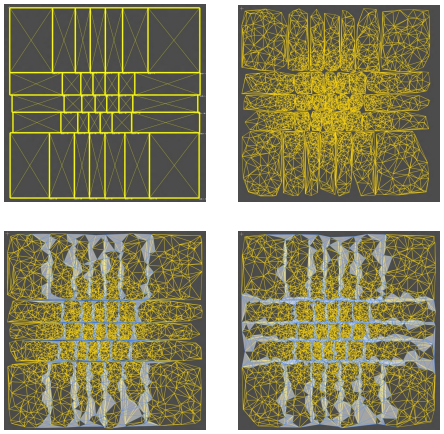

Figure 1: Results obtained from the application of each step ($\mathcal{S}_1 - \mathcal{S}_4$) of the proposed algorithm on a *Gaussian* distributed point-set.

Experimental Results

We have applied the proposed algorithm, and its rivals, in a number of uniformly and non-uniformly distributed point-sets with their cardinalities ranging from $50K$ up to $1M$ points. The performance of the proposed algorithm in terms of running time needed for the formation of the optimum set $\mathcal{T}_{\mathcal{D}}(\mathcal{P})$, clearly outperforms CPU-based, as well as, state of the art GPU-based implementations of DT algorithms. Specifically, depending on the cardinality of the point set, the proposed algorithm is 4 to 30 times faster than the well known CPU-based *Triange* DT algorithm, while its relative gain against the GPU-based implementation $GPU - DT$ varies from 10% up to 76% .