Parallel Voronoi Computation for Physics-Based Simulations

Julio Toss, João Comba, Bruno Raffin

Editors: Cláudio T. Silva, csilva@nyu.edu I Daniel Weiskopf, weiskopf@visus.uni-stuttgart.de

Parallel Voronoi Computation for Physics-Based Simulations

Julio Toss and João Comba
Universidade Federal do Rio Grande do Sul (UFRGS)
Bruno Raffin
Université Grenoble Alpes, INRIA

Voronoi diagrams are fundamental data structures in computational geometry, with applications in such areas as physics-based simulations. For non-Euclidean distances, the Voronoi diagram must be performed over a grid-graph, where the edges encode the required distance information. The major bottleneck in this case is a shortest path algorithm that must be computed multiple times during the simulation.

We present a GPU algorithm for solving the shortest path problem from multiple sources using a generalized distance function. Our algorithm was designed to leverage the grid-based nature of the underlying graph that represents the deformable objects. Experimental results report speed-ups up to 65× over a current reference sequential method.

Voronoi Diagrams

The Voronoi diagram is a classical partitioning of a space into closest-point regions (see Figure 1). It has a vast application domain, usually employed for answering proximity queries such as finding nearest site, facility location, motion planning, and coverage in sensor networks.

The Voronoi diagram is also a key for the Sibson's natural neighbor interpolation (NNI) method.[1] NNI is a well-known method used for interpolating irregularly spaced

 1521-9615/16/$33.00 © 2016 IEEE Copublished by the IEEE CS and the AIP

data, with applications in several different fields such as medical imaging, meteorological or geological modeling,[2] flow map reconstruction,[3] and scattered data visualization.[4] Natural neighbor-based interpolations have also been applied to the field of solid mechanics via the natural element method (NEM), which uses Sibson and non-Sibsonian (Laplace) interpolators to perform crack simulations.[5]

Voronoi-based interpolations have also been applied to meshless simulation of complex deformable bodies.[6] In that recent study, researchers computed a Voronoi tessellation on a non-Euclidean space by using a discrete distance map that encodes material-aware distances biased according to the local rigidity inside the simulated body (see Figure 2). Although the idea of a Voronoi partitioning of the space remains the same as in other classical applications, its computation here is fundamentally different. The distances aren't defined on the Euclidean space—instead, they rely on shortest paths computation over an implicit grid-graph. Computing this variant of Voronoi diagrams is therefore significantly more costly than the classical discrete Voronoi case. Using Sibson's NNI method on a graph space actually requires computing a shortest path tree for each interpolated value queried.

A lot of effort has already been dedicated to improving NNI's performance, particularly when interpolating on a discrete grid in the Euclidean space (discrete Sibson interpolation). A popular approach relies on GPU parallelization of the discrete Voronoi diagram, such as in the DEM construction.[2]

Early studies on parallelizing the discrete Voronoi diagram already exploited GPUs' parallel processing capabilities.[7] With the popularization of these architectures and the evolution of programming tools such as CUDA and OpenCL, other algorithms for Voronoi computation emerged to allow a better utilization of their computing power.[8,9]

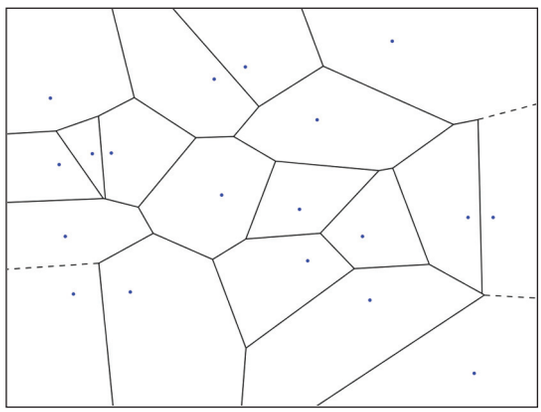

Figure 1. Voronoi diagram of a set of data points (in blue) in 2D Euclidean space. The space is partitioned into closest-point regions called Voronoi cells.

The graph variant of the Voronoi diagram (called the *graph Voronoi diagram*)[10] defines a vertex partitioning in a connected graph $G(V, E)$. Given a subset $S \subset V$ of source vertices, each vertex $v \in V$ is assigned to the partition P_i of the source vertex $s_i \in S$ with the shortest path distance. Applications of this kind of Voronoi diagram arises in several network problems and social data analysis, such as in community detection algorithms.[11]

Parallel solutions for computing the graph Voronoi have to deal with the shortest path problem. Dijkstra's algorithm implemented with a

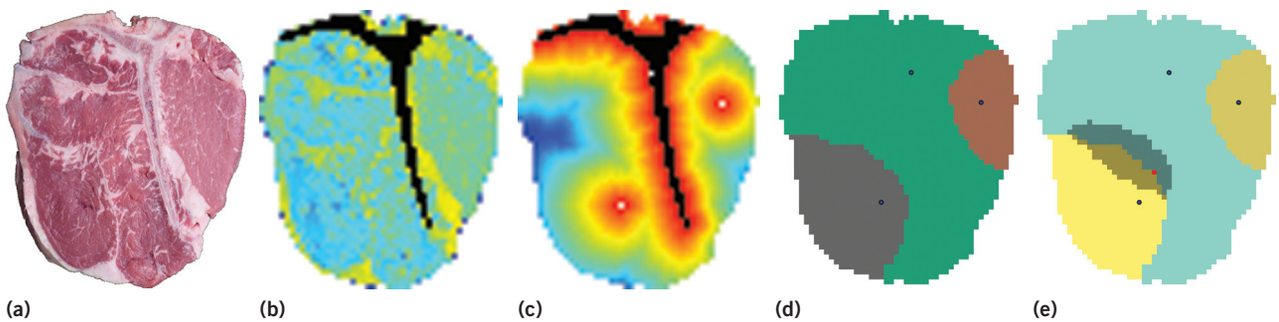

Figure 2. Use case example. (a) A T-bone steak contains a mixture of flexible meat, softer grease, and a rigid bone. As input, we take (b) the voxelized material map of stiffness values and the coordinates of the simulation nodes. (c) Distances to the nodes inside the object are biased according to the stiffness values and used to compute (d) the Voronoi diagram of the nodes. This diagram will be used to compute (e) the natural neighbors interpolation on the other voxels of the domain.

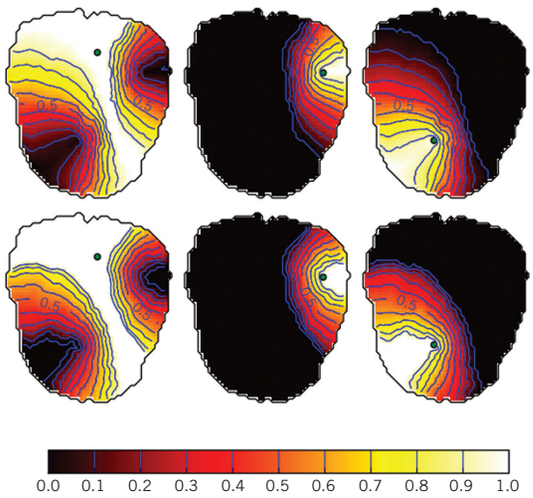

Figure 3. Shape functions' weights for three simulation nodes (from Figure 2) computed with two different interpolation methods: Sibson (top) and distance ratio (bottom). Weights are normalized starting at 1 in the node locations (Voronoi cell center) and decrease until vanishing outside of the support.

priority queue provides an efficient solution to this problem but is inherently sequential with lots of synchronizations. For graphs with grid topology, our previous work created a first parallel algorithm to compute the Voronoi diagram.[12] The algorithm creates a distance map over a discrete image by computing the shortest paths from each pixel to the closest Voronoi seed (Figure 2d). The implementation is done on GPUs and takes advantage of grid connectivity to leverage parallelism.

Discrete Voronoi Diagrams in SOFA

The discrete Voronoi diagram is used in a real-time simulation framework called SOFA (www.sofa-framework.org), a modular and extendable architecture that allows researchers of different fields related to physics-based simulation to implement and compare their own algorithms.

Deformation Using Shape Functions

Researchers have proposed a method for simulating complex objects that are composed of mixed types of materials with different stiffness.[6] This method relies on meshless models using sparse samples to capture the displacements at the simulation nodes, which are then interpolated within the object using a novel *material-aware shape function*. This shape function uses a special distance metric scaled according to the local material rigidity of the simulated object (Figures 2b and 2c), a technique that lets us easily take into account material heterogeneity during simulation. The object's material properties, such as stiffness, are usually represented as a volumetric image of voxels containing property values. To determine the material-aware distance, the shortest path is computed over this 3D grid of voxels, where each voxel represents a vertex of a grid-graph with 26 neighbors, and the edges' weights are defined as a function of the stiffness of the adjacent voxels.

To understand the role of shape functions in numerical simulations consider the following steps:

1. The displacement of a deformable object is sampled at discrete locations called degrees of freedom (DoFs). Each DoF has a shape function associated that defines where and how it will influence other points in the object. The area of influence is often referred as the *support* of the shape function (Figure 3).
2. The goal is then to interpolate these sampled displacements within the rest of the object.
3. The displacement at a given point is interpolated as a weighted sum of the node's displacements, where the weights are the values of the shape function for each DoF influencing this point.

Finally, the problem of defining how weights should be computed, which informs how shape functions from different DoFs will be blended in the rest of the domain. Voronoi diagrams have been used in the so-called *Voronoi shape functions* to compute these weights.

Voronoi-Based Interpolations

Consider the problem of finding neighbors in a set of nonuniform distributed data points. The Voronoi tessellation generated from this point provides us with the notion of *natural neighbors*, which are those data points whose Voronoi cells share a common frontier. Two interpolation schemes are based on this notion of neighborhood. The first one, the Sibson interpolation, is defined as a ratio of areas in 2D (volumes in 3D). It is computed by inserting the query point q in the initial Voronoi tessellation of sample points. The interpolating weight of each data point is then given by the ratio between the area stolen from the neighbor Voronoi cell and the area of the newly inserted Voronoi cell (Figure 2e). The second method, the Laplace (or non-Sibsonian) interpolation,[5,13] uses the same

notion of natural neighbor, but it computes the ratio between segments in 2D (areas in 3D). Instead of taking the area of the neighbor cells, it uses the ratio between the length of the Voronoi frontier (line in 2D; facet in 3D) and the distance from q to its natural neighbor nodes.

Both of these NNI methods require the computation of a new Voronoi diagram for each query point $q \in Q$ added to the input diagram of the data samples. This results in $|Q|$ executions of the Voronoi diagram, where Q is the interpolation resolution desired.

To reduce the number of Voronoi diagrams computed, an alternative interpolation method, called distance ratio,[6] only needs to compute a constant number of Voronoi diagrams per data sample $s \in S$, where $|S| \ll |Q|$. This method applies a particular scheme that computes the ratio between the distance from the point to the Voronoi border and the distance to the node (center of the Voronoi cell). Although less formal guaranties on the properties were presented for this interpolant, this algorithm is implemented on SOFA and shows good practical results, with the advantage of being more computationally efficient.

Parallel Voronoi Diagram Computation

As shown previously, many other algorithms, particularly those in physics-based simulations, build on Voronoi diagrams. In the case of real-time and interactive simulations, the computation of the Voronoi shape functions must meet strict performance requirements. Parallel processing is one popular strategy to meet this goal.

Parallel Voronoi computation on Euclidean distance have been extensively studied in previous work.[7–9] However, such approaches can't be directly applied in the physics simulation use cases described here, where distance measures are actually shortest paths computed on a graph.

Geodesic distance is closely related to the well-studied single-source shortest path (SSSP) problem from the graph theory domain. Parallel algorithms for solving the SSSP problem on general graphs have been proposed,[14–17] usually based either on the Dijkstra or Bellman-Ford algorithms. These algorithms assume a general graph without having any prior knowledge about its structure. They rely on a generic graph representation (such as adjacency matrix or list) where a vertex can have any number of edges with no structured neighborhood. In physics-based simulation, the Voronoi-shape functions are built on top of a much more regular-structured graph, which allows us to use a less general but more optimized algorithm. Targeting efficiency, we presented[12] a parallel algorithm using GPUs to compute the graph Voronoi diagram on a voxelized 3D grid. This solution is well suited for the physics simulations used in SOFA as it exploits the grid topology that implicitly represent edges. The solution is based on parallel wavefront expansions starting at each Voronoi seed. By using the massive amount of parallel threads in a GPU, we can compute each Voronoi cell concurrently.

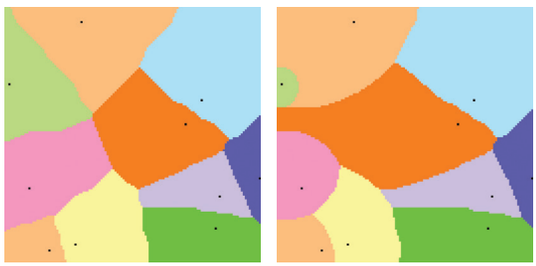

Figure 4. Comparison of Voronoi diagrams generated with the same set of seeds on two different material maps: (a) uniform stiffness and (b) a stiffness gradient.

We present here some experimental results of speed-up of the parallel Voronoi computation. We conducted experiments on an Nvidia GPU GTX480 with 1.5 Gbytes of global memory and 15 multiprocessors with 32 cores each, totaling 480 CUDA cores. The speed-up presented is related to the base sequential version from SOFA and executed on an Intel Core i7 CPU model 930 with 4 cores running at 2.89 GHz and 12 Gbytes memory.

The benchmark consists of computing the Voronoi diagram on a 3D volume for a given set of randomly distributed data points. The dataset used varies in volume dimensions, distance map distribution, and number of seeds. Figure 4 shows an example of two Voronoi diagrams computed with the same dimensions and seeds but using different distance distribution. Figure 4b was computed from an image with a gradient of stiffness increasing from right to left, whereas Figure 4a has constant stiffness. These two distances distributions are referred as gradient and constant on the bar plots of Figure 5. Note that the distance between two neighbor voxels is given by a function of the stiffness between them. As seen in Figure 4 the shape of the Voronoi diagram greatly depends on the stiffness properties of material map of the object simulated.

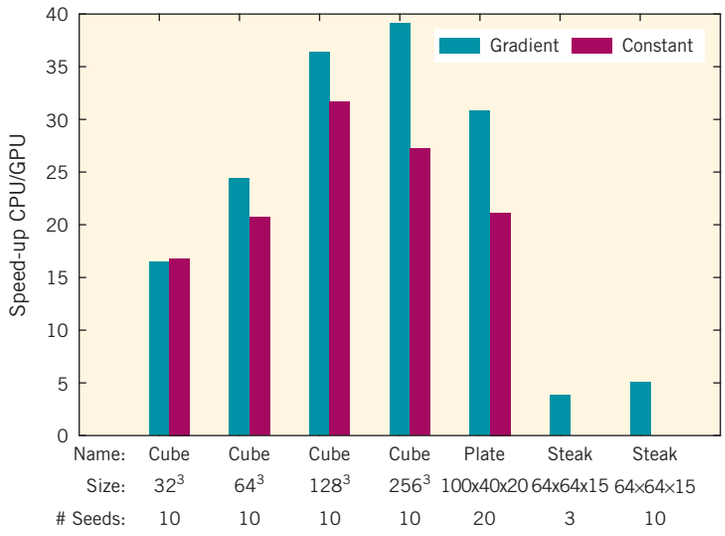

Figure 5. Speed-up for different input sizes. Gradient and constant topologies are presented for synthetic benchmarks only. The steak's topology corresponds to the dataset shown in the use case of Figure 2.

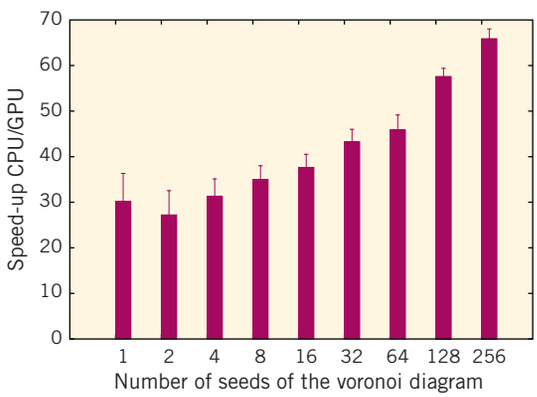

Figure 6. Average speed-up when increasing the number of seeds of the Voronoi diagram. The instance used has a volume size of 128^3 and a gradient material map.

In the parallel algorithm implemented, each CUDA thread processes a single voxel. With larger input volumes, more parallelism is exposed, therefore we note an increase in speed-up (Figure 5). At 256^3, the speed-up for the constant topology slightly decreases. We attribute this to an overhead of scheduling an excessive number of idle threads. This happens because with a volume of 256^3 and only 10 seeds, the Voronoi diagram becomes overly sparse.

The amount of parallel work available also increases with the number of seeds in the Voronoi diagram (Figure 6). When many Voronoi cells are being computed concurrently, more threads are active at the same time.

Parallel Natural Neighbor Interpolations

In the classical NNI, each queried point is inserted, one at a time, to the Voronoi diagram of initial data samples. For each seed added, the initial diagram is updated to generate the new Voronoi cell, which will then be used to compute the interpolation.

As seen in the experimental results, the number of seeds computed in the Voronoi diagram has a big impact on the amount of parallelism that will be exposed. Computing a single Voronoi cell in parallel reduces the possibilities of parallelization. This situation is even worse when we consider updating an existing Voronoi diagram with the addition of a new seed (the *query seed*). In this case, only a limited region (around the query seed) would be recomputed (Figure 2e). Performing NNI simply as sequence of (parallel) Voronoi computations on the GPU doesn't pay off the overhead of memory transfer and thread scheduling inherent to this architecture.

One strategy to generate interpolated values over a grid would be to perform multiple queries concurrently in parallel. This approach has already been used[2] to generate an interpolation of a regular grid using the assumption that every sample has a limited radius of influence, thereby allowing the decomposition of the domain in independent blocks where queries can be answered in parallel batches. One study[4] proposed a more efficient implementation of Sibson's interpolation on raster images. The method avoids the explicit construction of a new Voronoi diagram for each query point, favoring instead a *Kd-tree* structure to find the closest seed to the current query point and using this distance as a radius of influence to increment the interpolation weight. Again, these techniques make assumptions that are valid for Euclidean space but not trivially generalized for geodesic (graph) distances.

Parallelization can still be implemented for NNI over graph spaces if we duplicate some data structures. More precisely, for each NNI query, we can copy the input Voronoi diagram of the data sample and update it with the addition of a new seed at the coordinates of this query.

Table 1 shows the computation time spent for the parallel Sibson algorithm on an Nvidia Tesla K40 GPU. The interpolation is performed over a uniform grid of dimensions $100 \times 40 \times 20$ (80,000 voxels) and 20 data points. The GPU algorithm performs batches of parallel NNI queries in sequence iteratively until the entire grid is comput-

Table 1. Computation time for parallel NNI queries. Average Sibson query corresponds to time spent for querying a whole batch.

Batch size	Average Sibson query (ms)	Total time (ms)
CPU 1 (seq)	**1.003**	**81,057.226**
GPU 1	**9.745**	**779,657.756**
GPU 10	**12.413**	**99,309.125**
GPU 100	**29.537**	**23,629.870**
GPU 150	**34.807**	**18,587.130**

ed. We show the average amount of time spent by each batch of parallel queries and the total time for interpolating the whole grid. Note that the parallel version manages to amortize the overhead when more than 10 NNI queries are done in parallel.

The parallel Voronoi implementation presented here has a valuable application in soft object simulation methods. It provides a performance solution to those in the physics-based simulation community who wants to employ Voronoi shape functions in their meshless simulations.

Recent work has proposed using Voronoi shape functions on grids with extended connectivity, called *non-manifold grids*,[18] which would let us represent objects with more complex topologies in meshless frameworks. Possible extensions of this work will consider the application of our parallel algorithm in these new domains. ■

Acknowledgments

We thank the projects Capes/Cofecub 764/13 and CNPq 308851/2015-3 for financial support.

References

1. R. Sibson, "A Brief Description of Natural Neighbor Interpolation," *Interpreting Multivariate Data*, vol. 21, 1981, pp. 21–36.
2. A. Beutel, T. Mølhave, and P.K. Agarwal, "Natural Neighbor Interpolation Based Grid DEM Construction Using a GPU," *Proc. 18th Int'l Conf. Advances in Geographic Information Systems*, 2010, p. 172.
3. S.S. Barakat and X. Tricoche, "Adaptive Refinement of the Flow Map Using Sparse Samples," *IEEE Trans. Visualization and Computer Graphics*, vol. 19, no. 12, 2013, pp. 2753–2762.
4. S.W. Park et al., "Discrete Sibson Interpolation," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 2, 2006, pp. 243–252.
5. N. Sukumar, "Voronoi Cell Finite Difference Method for the Diffusion Operator on Arbitrary Unstructured Grids," *Int'l J. Numerical Methods in Eng.*, vol. 57, no. 1, 2003, pp. 1–34.
6. F. Faure et al., "Sparse Meshless Models of Complex Deformable Solids," *Proc. ACM SIGGRAPH*, 2011, p. 1.
7. K.E. Hoff III et al., "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware," *Proc. 26th Ann. Conf. Computer Graphics and Interactive Techniques*, 1999, pp. 277–286.
8. O.Weber et al., "Parallel Algorithms for Approximation of Distance Maps on Parametric Surfaces," *ACM Trans. Graphics*, vol. 27, no. 4, 2008, pp. 1–16.
9. G. Rong and T.S. Tan, "Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform," *Proc. Symp. Interactive 3D*, 2006, p. 109.
10. M. Erwig, "The Graph Voronoi Diagram with Applications," *Networks*, vol. 36, no. 3, 2000, pp. 156–163.
11. D. Deritei et al., "Community Detection by Graph Voronoi Diagrams," *New J. Physics*, vol. 16, no. 6, 2014, article no. 063007.
12. J. Toss, J.L.D. Comba, and B. Raffin, "Parallel Shortest Path Algorithm for Voronoi Diagrams with Generalized Distance Functions," *Proc. 27th Conf. Graphics, Patterns and Images*, 2014, pp. 212–219.
13. V.V. Belikov et al., "The Non-Sibsonian Interpolation: A New Method of Interpolation of the Values of a Function on an Arbitrary Set of Points," *Computational Mathematics and Mathematical Physics*, vol. 37, no. 1, 1997, pp. 9–15.
14. S. Kumar, A. Misra, and R.S. Tomar, "A Modified Parallel Approach to Single Source Shortest Path Problem for Massively Dense Graphs Using CUDA," *Proc. 2nd Int'l Conf. Computer and Communication Tech.*, 2011, pp. 635–639.
15. P. Harish, V. Vineet, and P.J. Narayanan, "Large Graph Algorithms for Massively Multithreaded Architectures," tech. report IIIT/TR/2009/74, Int'l Inst. Information Tech. Hyderabad, 2009.
16. K. Madduri et al., "Parallel Shortest Path Algorithms for Solving Large-Scale Instances," *Proc. 9th DIMACS Implementation Challenge: The Shortest Path Problem*, 2006, pp. 1–39.

17. H. Ortega-Arranz et al., "A New GPU-Based Approach to the Shortest Path Problem," *Proc. Int'l Conf. High Performance Computing & Simulation*, 2013, pp. 505–511.
18. P.-L. Manteaux et al., "Interactive Detailed Cutting of Thin Sheets," *Proc. ACM SIGGRAPH Motion in Games*, 2015, pp. 1–8.

Julio Toss is a joint PhD student at Universidade Federal do Rio Grande do Sul and Université Grenoble Alpes. Contact him at jtoss@inf.ufrgs.br.

João Comba is an associate professor at Universidade Federal do Rio Grande do Sul. Contact him at comba@inf.ufrgs.br.

Bruno Raffin is a director of research at INRIA, leader of the DataMove joint research team between INRIA and the Université Grenoble Alpes. Contact him at bruno.raffin@inria.fr.