GPU-based Computation of Distance Functions on Road Networks with Applications

Marta Fort
Institut d'Informàtica i Aplicacions
Universitat de Girona, Spain
mfort@ima.udg.edu

J. Antoni Sellarès
Institut d'Informàtica i Aplicacions
Universitat de Girona, Spain
sellares@ima.udg.edu

ABSTRACT

We present a GPU-based algorithm for computing discretized distance functions on road networks. As applications, we provide algorithms for computing discrete Order-k Network Voronoi diagrams and for approximately solving k-Nearest Neighbor queries and Aggregate k-Nearest Neighbor queries on road networks. Finally, we present experimental results obtained with the implementation of our algorithms.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Spatial Databases and GIS; H.4.2 [**Information Systems Applications**]: Decision Support

General Terms

Algorithms, design, experimentation

Keywords

Road networks, proximity queries, graphics hardware

1. INTRODUCTION

A Road Network Database allows to efficiently store and query objects in road networks. Static objects are represented by points of interest located on the network (hotels or gas stations) and moving objects are represented as points running along the network (cars or pedestrians). The network distance between objects on the road network (car and gas station), defined as the cost of the shortest path between them, depends on the connectivity and weights of the underlying network.

Computing shortest paths, and consequently network distances, is a fundamental combinatorial optimization problem with important applications in various domains, like Geographic Information Systems, Location-Based Services, Navigation Systems, Mobile Computing Systems, Data Clustering, etc. The computation of network distances often arises as a subroutine in the solution of many proximity problems. The Order-k Network Voronoi diagram of a set of points of interest, called *sites*, on a network associates to each subset of size k of sites the part of the network that consists of points closer to those k sites than any other k sites. Network Voronoi diagrams allow to answer proximity point queries like finding the Nearest Neighbor, the Farthest Neighbor and the k-Nearest Neighbors. Given a set of sites and a set of query points on the network, Aggregate k-Nearest Neighbor queries try to determine a subset of k sites that minimizes some aggregate distance function (max, sum, etc.) with respect to a query point set.

1.1 Preliminaries

We model a *road network* as an undirected weighted graph $N(V, E, W)$. The set V, $n = |V|$, is a set of vertices representing road intersections (with degree above 2), terminal points (with degree 1) and shape points (with degree 2). Shape points are neither intersection nor terminal points that are interpolated to create sequences of arbitrarily small linear parts of the road network. The spatial position of each vertex in V with respect to a reference coordinate system is also given. The set E, $m = |E|$, is a set of edges representing road segments, each connecting two vertices. $W : E \to \mathbb{R}^+$ associates each edge e with a positive weight $w(e)$, that may represent, for example, the Euclidean length of e, denoted $|e|$, or the time, toll, energy consumption, etc, required to travel between the two endpoints of e. A portion $\overline{e}$ of and edge e connecting to points of e is called subedge. We define the weight $w(\overline{e})$ of the subedge $\overline{e}$ as $w(\overline{e}) = (|\overline{e}|/|e|)w(e)$. In this paper we consider static road networks with a fixed weight for each edge. We denote by $\mathcal{N}$ the set of all points on the edges, including the vertices, of $N(V, E, W)$. Although in the worst case a graph could be dense ($m = O(n^2)$), in general graphs modelling networks have constant degree and consequently are sparse ($m = \Theta(n)$). We want to note that for simplicity we have modelled a road network as an undirected graph, but that our methods can be easily applied to road networks modelled as a directed graph, in which one-way roads or roads where the weight is different for the two directions are allowed.

A path $\pi(p, q)$ between two points p and q, p located on edge (v_0, v_1) and q located on edge (v_l, v_{l+1}), is a sequence $[p, v_1, \cdots, v_l, q]$, where $v_1, \cdots, v_l \in V$ and $(v_{i-1}, v_i) \in E$, $1 < i \leq l$. The cost of the path $\pi(p, q)$ is defined by $\|\pi(p, q)\| = w(\overline{pv_1}) + \sum_{1<i\leq l} w(\overline{v_{i-1}v_i}) + w(\overline{qv_{l+1}})$. The path of least cost connecting p and q is called *shortest path* between p and q. The cost of the shortest path is called the network distance between p and q, denoted $d(p, q)$. In the

following, for simplicity, we use the term distance to refer to the network distance. The distance function defined by a site p on $\mathcal{N}$ is a function d_p such that for any point $q \in \mathcal{N}$, $d_p(q)$ is the distance $d(p, q)$ between p and q.

1.2 Network Voronoi Diagrams

Let P be a set of r static sites lying on the road network $\mathcal{N}$. Given P' a subset of k sites of P, $k \in \{1, \cdots, r-1\}$, the Order-k Voronoi region $V(P')$ of P' is the set, possibly empty, of points of $\mathcal{N}$ closer to each site of P' than to any other site of P: $V(P') = \{q \in \mathcal{N} \mid d_{p'}(q) < d_p(q)\ \forall p' \in P', p \in P - P'\}$. The Order-$k$ Network Voronoi diagram of P, denoted $\mathcal{V}_k(P)$, is the set of the Order-k Voronoi regions $V(P')$ of all subsets P' of k sites. When $k = 1$ and $k = r - 1$, the Order-k Network Voronoi diagrams are called the (Closest) Network Voronoi and the Furthest Network Voronoi diagram, respectively (see Figures 4 and 5).

1.3 Nearest Neighbor Queries

Given the set P of sites and a query point q on the network, a k-Nearest Neighbor (k-NN) query finds the subset of k sites which are closest to q. Formally, $NN_k(P, q)$ retrieves a subset $P' \subset P$, $|P'| = k$, such that $d_p(q) < d_{p'}(q), \forall p \in P', p' \in P - P'$. An example of k-NN query is a query initiated by a GPS device in a car to find the five closest hotels to the car.

Continuous k-Nearest Neighbor (C-k-NN) queries are defined as the k-NNs of all points on a network (i.e., continuous with respect to the network). For example, continuously finding the five nearest hotels to a moving car. The result of this type of query is a set of intervals, bounded by split points, and their corresponding list of k-NNs. The split points specify where on the network the k-NNs of a moving query object change, and the intervals specify the locations where the k-NNs of a moving object remain the same. The Order-k Network Voronoi diagram subdivides the network into intervals according to the k nearest neighbors and consequently allows to answer Continuous k-NNs queries.

1.4 Aggregate Nearest Neighbor Queries

An Aggregate k-Nearest Neighbor (A-k-NN) query returns the k sites with the better score according to an aggregate distance function with respect to a set of query points. For example, several users at specific locations (query points) that want to find the three restaurants (sites), which leads to the minimum sum of distances that they have to travel in order to meet. Formally, given a set P of sites, a set Q of query points, P and Q located on $\mathcal{N}$, and an integer $1 \leq k < |P|$, several discrete aggregate nearest neighbor queries depending on P, Q and k are defined as follows:

MinMax$_k$(P,Q): retrieves a subset $P' \subset P$, $|P'| = k$, such that $max_{q \in Q} d_{p'}(q) \leq max_{q \in Q} d_p(q), \forall p' \in P', p \in P - P'$.

MinSum$_k$(P,Q): retrieves a subset $P' \subset P$, $|P'| = k$, such that $\sum_{q \in Q} d_{p'}(q) \leq \sum_{q \in Q} d_p(q), \forall p' \in P', p \in P - P'$.

MaxMin$_k$(P,Q): retrieves a subset $P' \subset P$, $|P'| = k$, such that $min_{q \in Q} d_{p'}(q) \geq min_{q \in Q} d_p(q), \forall p' \in P', p \in P - P'$.

MaxSum$_k$(P,Q): retrieves a subset $P' \subset P$, $|P'| = k$, such that $\sum_{q \in Q} d_{p'}(q) \geq \sum_{q \in Q} d_p(q), \forall p' \in P', p \in P - P'$.

1.5 Graphics Hardware

The increasing programmability and high computational rates of graphics processing units (GPUs) make them attractive as an alternative to CPUs for general-purpose computing. Recently, different algorithms and applications that exploit the inherent parallelism, easy programmability and vector processing capabilities of GPUs have been proposed [15]. In particular, in the field of data engineering there exist several algorithms that have a fast hardware-based implementation [5, 12].

The graphics pipeline [18] is divided into several stages. The input is a list of 3D geometric primitives expressed as vertices defining points, lines, etc. with attributes associated. The output is an image in the frame buffer, a collection of several hardware buffers corresponding to two dimensional grids whose cells are called pixels. In the first stage of the pipeline, per-vertex operations take place, each input vertex is transformed from 3D coordinates to window coordinates. Next stage is rasterization, when it finishes we obtain a fragment, with its associated attributes, for each pixel location covered by a geometric primitive. Fragment attributes are obtained from the attributes associated to the vertices by linear interpolation. The third stage, the fragment stage, computes the color for each pixel in the frame buffer, according to the fragments corresponding to it taking into account a series of tests such as the depth test and per-fragment operations. This output can be transferred to the CPU or stored as a texture, a 2D array of values, and then used in additional computations. The unique programmable parts of the graphics pipeline are the vertex and fragment shaders which are executed on a per-vertex and per-fragment basis, respectively. They are used to change the vertex or fragment attributes.

1.6 Previous Work

Practical and robust algorithms for computing the exact Voronoi diagram of a set of point sites in 2D and 3D have been extensively developed in computational geometry [14]. Different algorithms have been proposed to efficiently compute 2D and 3D approximated Voronoi diagrams of a set of generalized sites (points, segments, etc) along a grid using graphics hardware [7, 4]. Voronoi diagrams in networks and their applications have been studied in [6, 13].

Nearest Neighbor queries and Continuous Nearest Neighbor queries in road networks have been investigated extensively [16, 9, 10, 8, 11, 1, 2]. The majority of these works present solutions mainly focused on spatial index structures and query processing techniques. Aggregate Nearest Neighbor queries in road networks have been studied in [20].

1.7 Our Contribution

We present an algorithm, that exploits graphics hardware capabilities, to compute discretized distance functions on road networks. As applications, we provide algorithms to compute discrete Order-k Network Voronoi diagrams and for approximately solving k-Nearest Neighbor queries and Aggregate k-Nearest Neighbor queries on road networks. Our algorithms aim to optimize CPU/GPU processing cost while other techniques focus on reducing the communication cost overhead caused by frequent updates. Finally, we present experimental results obtained with the implementation of our algorithms.

2. DISTANCE FUNCTION COMPUTATION

We compute the distance from a given site p on $\mathcal{N}$ to all vertices of V by using the classical Dijkstra's algorithm [3]. Dijkstra's algorithm computes the shortest path from site p to all vertices of V by incrementally growing a tree of shortest paths from p out to the most distant vertices. Dijkstra's algorithm for sparse graphs, like road networks, has $O(n \log n)$ asymptotic time complexity for an undirected weighted graph with n vertices [19]. In order to reduce computation times several speed-up techniques have been developed during the last years [19, 17]. However, since most proximity queries are interested in local areas, Dijkstra's algorithm is efficient enough for our purposes.

Now we face the most general problem of computing the distance from a given site p to any point on $\mathcal{N}$. Since the continuous nature of the problem makes it difficult to efficiently compute this distance, we propose an alternative solution based on a discretization (approximation) process that allows us to explicitly store the distance function on a compact way.

2.1 Road Network Parameterization

Let $\mathcal{R}$ be a rectangular grid of the xy-plane of size $W \times H$. We map each edge e of the road network $\mathcal{N}$ to a row segment of $\mathcal{R}$ represented by the centers of its cells (see Figure 1). The mapping optimally packs the mapped edges into $\mathcal{R}$ so that: a) the mapping of two different edges of $\mathcal{N}$ do not overlap in $\mathcal{R}$; b) the mapping of all edges of $\mathcal{N}$ covers "almost" all grid cells of $\mathcal{R}$, it is to say, the number of row cells covered by the mapping of e "approximately" equals $WH|e|/\sum_{e' \in E} |e'|$. In this way we obtain a discrete parameterization of the road network $\mathcal{N}$ in the sense that each cell in the grid map represents a unique location on the road network.

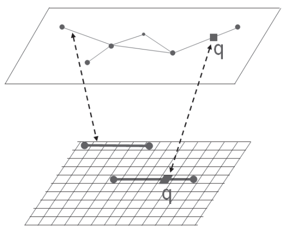

Figure 1: Road Network Parameterization.

2.2 Distance Function Discretization

To compute explicitly the distance function d_p defined by a site p on $\mathcal{N}$ we represent the discretized rectangular region $\mathcal{R}$ into a grid of $W \times H$ pixels by the depth buffer where distance values will be stored (see Figure 2).

During the initialization process of Dijkstra's algorithm, we initialize the depth buffer to the maximal depth value (1). When a new fragment is processed, the depth buffer is updated if the depth value of the current fragment is smaller than the current value in the depth buffer. At the end of the process the value stored in the depth buffer is the minimum depth (distance) defined by all the processed fragments. The discrete representation of the distance function is obtained during the distance function propagation process.

The OpenGL pipeline processes the edge vertices and rasterizes the edges into fragments by interpolation. The planar mapping is used to map the points on the edges of $\mathcal{N}$ to points onto the discretized rectangular region $\mathcal{R}$ of the xy-plane. Within the rasterization step all the parameters associated to vertices such as texture coordinates, color, etc, are linearly interpolated across the segments interior from those associated to the edge vertices. Consequently the value obtained in these channels in a fragment is the interpolation of the values associated to the edge vertices. Then the fragment shader computes the distance defined by the source at each point and sets this distance normalized into [0,1] as the depth value of the rasterized fragments. Finally, the depth test stores the minimum distance obtained at each position.

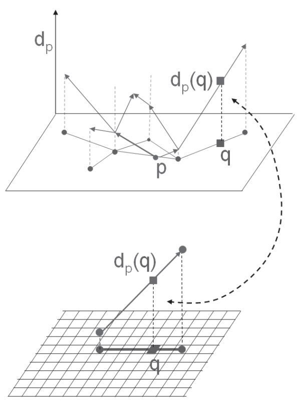

Figure 2: Discrete distance function.

3. APPLICATIONS

In the following, we assume that each site in P has an identifying associated color, and that the discrete distance functions of all the sites have already been computed and stored.

3.1 Order-k Network Nearest Diagram

The k-Nearest region associated with a given site in P is the set of points on the road network such that the site ranks number k in the ordering of the sites by distance from the point, and is a collection of one or more unconnected intervals. Each of these regions forms the so-called Order-k Network Nearest Diagram, that coincides with the k-level of the arrangement defined by the distance functions of the site in P.

The Order-k Network Nearest Diagram can be obtained with a multi-pass algorithm, that at every pass "peels" off one level of the arrangement [4]. At each pass all the distance functions are painted in their corresponding color by defining a rectangle covering $\mathcal{R}$, and the minimal depth value is stored in the depth buffer. In the first pass the closest Voronoi diagram is obtained. When it finishes, the depth buffer is transferred to a texture and send to the fragment shader. In the second pass all the distance functions are again painted. In the fragment shader the distance function that is being painted is compared with the distance obtained in the previous pass at the current fragment. Only the fragments with distance bigger than the distance obtained in the previous pass are painted, the others are discarded. Therefore, the values stored in the depth buffer in the second step are the second minimal distance. When this process is repeated k times, the kth-nearest diagram represented on $\mathcal{R}$ is obtained. By using typical texturing methods and the parameterization the Order-k Network Nearest Diagram can be represented on the network (see Figure 3).

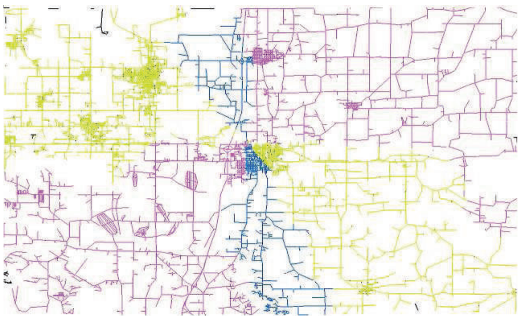

Figure 3: Order-2 Nearest Network Voronoi Diagram of three sites.

3.2 Order-k Network Voronoi Diagrams

The k-order Voronoi diagram can be obtained by overlaying the Order-i Network Nearest diagrams, $i = 1 \dots k$, with transparency $1/k$.

The particular cases of the Closest and Furthest Network Voronoi Diagrams can be obtained by painting one after the other all the distance functions in their associated colors and storing, in the depth buffer, the minimal or the maximal depth value, respectively. Figures 4 and 5 show Network Voronoi diagrams with three and 100 facilities, respectively.

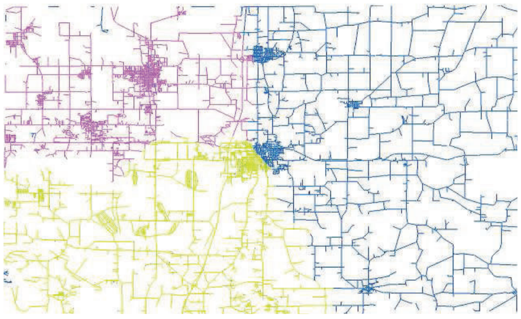

Figure 4: Network Voronoi Diagram of three sites.

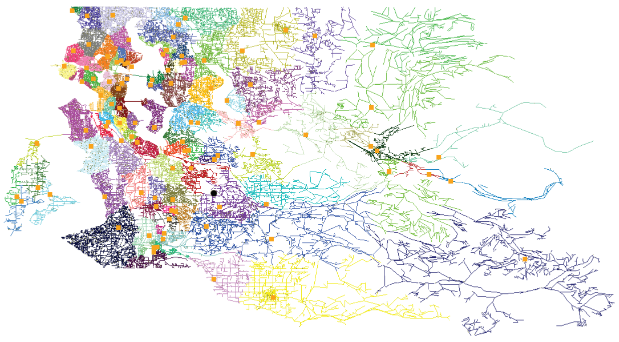

Figure 5: Network Voronoi Diagram of 100 sites.

3.3 k-Nearest Neighbor queries

We propose an algorithm for queries involving an arbitrary number k of neighbors. To answer a k-Nearest Neighbor query on a point q we sample each of the Order-i Network Nearest diagrams at q, for $i = 1, .., k$, so that the colors of the k samples identify the sites. Furthermore, the sites may be reported sorted by the distance to the query point.

3.4 Aggregate k-Nearest Neighbor queries

We can also answer discrete Aggregate k-NNs queries. Now we are given the set P of r sites and a set Q of l query points. We are interested in obtaining the set P' of k sites of P that fulfill different properties depending on the problem we are solving. We have to compute, for each $p \in P$: a) $max_{q \in Q} d_p(q)$; b) $\sum_{q \in Q} d_p(q)$; c) $min_{q \in Q} d_p(q)$. Notice that we have to find the maximum, sum or minimum of l real numbers. It can be done with graphics hardware by using a single pixel to obtain the minimum, sum or maximum of these values. Since it has to be done for each site $p \in P$, we associate a different pixel to each site. The pixel representing site p is painted l times with different depth values corresponding to $d_p(q)$ for each $q \in Q$. Then the maximum, sum or minimum is obtained by properly using the depth test or the explained technique to obtain the sum of all the values. Discrete Aggregate k-NN queries are solved as follows:

MinMax$_k$(P,Q): we compute $max_{q \in Q} d_p(q)$ for each $p \in P$ and next find the k pixels with minimum values. It can be done by using a reduction type algorithm to find the minimum together with a peeling-technique to obtain the k smaller values or by reading the pixels in the CPU and finding the k-minimum values in the CPU.

MinSum$_k$(P,Q): we compute $\sum_{q \in Q} d_p(q)$ for each $p \in P$ and next find the k pixels with minimum values, as it is explained for the MinMax$_k$(P,Q) case.

MaxMin$_k$(P,Q): we compute $min_{q \in Q} d_p(q)$ for each $p \in P$ and next find the k pixels with maximum values, as it is explained for the previous cases but storing the maximal values instead of the minimal ones.

MaxSum$_k$(P,Q): we compute $\sum_{q \in Q} d_p(q)$ for each $p \in P$ and next find the k pixels with maximum values, as it is explained for the $MaxMin_k$ case.

3.5 Error Analysis

In the results, we can distinguish among two different types of error: a) The discretization error that depends on the discretization size, the biggest the grid size the smallest the error. b) The floating errors, which are specially related to the depth buffer and depth texture precision. The 32-bit precision is sufficient to store the normalized distances which take values in the interval $[0, 1]$.

4. EXPERIMENTAL EVALUATION

We have implemented the proposed methods using C++, OpenGL and Cg. All the experiments have been carried out on a Intel Core2 Duo at 2.40GHz with 2GB of RAM and a NVIDIA GeForce 8600 GT graphics board.

We have used four real road networks: 1) Rock Island defined by 13,442 nodes and 12,865 edges; 2) Washington with 81,389 nodes and 100,703 edges; 3) Nevada with 280,942 nodes and 338,205 edges; 4) New York with 724,366 nodes and 910,310 edges.

In Table 1 we provide the time, in seconds, needed during: the parameterization computation, Dijkstra's algorithm execution and the distance function discretization process. The time needed by Dijkstra's algorithm execution and the distance function discretization process is the average time of 100 executions realized considering sites randomly placed on the road network.

Table 1: Computation times.

n	Parametrization	Dijkstra	Distance Func.
10,442	0.030	0.006	0.04
81,389	0.032	0.05	0.07
280,942	0.104	0.20	0.16
724,366	0.28	0.55	0.34

In Table 2 we present, in seconds, the time needed to compute, from already computed distance functions, Order-k Voronoi diagrams. Times provided in this table only depend on $|P|$, k and the discretization size, which is 1280×882.

Table 2: Order-k Voronoi diagram computation.

$\|P\|$	$k = 1$	$k = 10$	$k = 25$	$k = 50$	$k = 100$
50	0.81	7.16	17.49	-	-
100	1.51	14.05	14.86	69.75	-
200	2.89	27.97	69.57	109.41	278.24

k-NN queries are answered by using the already computed and stored Order-i Network Nearest diagrams, $i \leq k$. The time needed to obtain the Order-i Network Nearest diagram is that needed to compute the Order-k Voronoi diagram with extra 0.06 seconds for each stored diagram. Then answering a k-NN query takes 0.001 seconds for $k = 10$ and 0.0011 seconds for $k = 50$.

The time needed to answer an Aggregate k-NN query, $k = 5$, when 10 sites and 10 point queries are considered is of 0.06 seconds for MaxMin and MinMax queries and of 0.09 seconds for MaxSum and MinSum queries. If 50 sites and 30 point queries are considered, MinMax and MaxMin queries are answered again in 0.06 seconds, and MaxSum and MinSum in 0.013 seconds. Times do not change if we consider $k = 15$ or $k = 20$.

We want to remark that, according to the presented algorithms and experimental results, the number of nodes and edges of the road network only affects the time needed to compute distance functions.

5. FUTURE WORK

We plan to extend our GPU-based approach to solve Reverse k-Nearest Neighbor queries and some Facility Location problems such as the 1-Center and 1-Median, and the Obnoxious 1-Center and 1-Median.

6. ACKNOWLEDGMENTS

We wish to thank Anna Casas and Ivan Roset for helping us in the implementation of our algorithms. M. Fort and J. A. Sellarès are partially supported by the Spanish MEC grant TIN2007-67982-C02-02.

7. REFERENCES

[1] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to CNN queries in a road network. In *VLDB'05*, pp 865–876, 2005.

[2] V. T. de Almeida and R. H. Güting. Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In *SAC'06*, pp 58–62, 2006.

[3] E. Dijkstra. A note on two problems in connection with graphs. *Numeriche Mathematik*, 1:269–271, 1959.

[4] I. Fischer and C. Gotsman. Fast approximation of high-order Voronoi diagrams and distance transforms on the GPU. *J. of Graphics Tools*, 11(4):39–60, 2006.

[5] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin and D. Manocha. Fast computation of database operations using graphics processors. In *ACM SIGMOD'04*, pp 215–226, 2004.

[6] S. L. Hakimi, M. Labbé and E. Schmeichel. The Voronoi partition of a network and its implications in location theory. *ORSA Journal on Computing*, 4:412–417, 1992.

[7] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33:277–286, 1999.

[8] X. Huang, C. S. Jensen and S. Saltenis. The islands approach to nearest neighbor querying in spatial networks. In *SSTD'05*, pp 73–90, 2005.

[9] C. S. Jensen, J. Kolářvr, T. B. Pedersen and I. Timko. Nearest neighbor queries in road networks. In *ACM GIS'03*, pp 1–8, 2003.

[10] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB'04*, pp 840–851, 2004.

[11] M. R. Kolahdouzan and C. Shahabi. Alternative solutions for continuous k nearest neighbor queries in spatial network databases. *Geoinformatica*, 9(4):321–341, 2005.

[12] M. D. Lieberman, J. Sankaranarayanan and H. Samet. A fast similarity join algorithm using graphics processing units. In *ICDE'08*, pp 1111–1120, 2008.

[13] E. Martin. The graph voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[14] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[16] D. Papadias, J. Zhang, N. Mamoulis and Y. Tao. Query processing in spatial network databases. In *VLDB'03*, pp 802–813, 2003.

[17] P. Sanders and D. Schultes. Engineering fast route planning algorithms. In *WEA'07*, pp 23–36, 2007.

[18] M. Segal and K. Akeley. The design of the OpenGL graphics interface, 1994.

[19] D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *STACS'07*, pp 23–36, 2007.

[20] M. L. Yiu, N. Mamoulis and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):820–833, 2005.