Fast Obstacle Spatial Query on Navigation Mesh

by

Shizhe Zhao

Minor Thesis

Submitted by Shizhe Zhao

in partial fulfillment of the Requirements for the Degree of

Master of Information Technology (Minor Thesis) (3316)

Supervisor: Assoc. Prof. David Taniar

Caulfield School of Information Technology
Monash University

June, 2018

Contents

Fast Obstacle Spatial Query on Navigation Mesh

Shizhe Zhao
szha414@student.monash.edu
Monash University, 2018

Supervisor: Assoc. Prof. David Taniar

Abstract

Obstacle k-Nearest Neighbours problem is the k-Nearest Neighbour problem in a two-dimensional Euclidean plane with obstacles (*OkNN*). Existing and state of the art algorithms for OkNN are based on incremental visibility graphs and as such suffer from a well known disadvantage: costly and online visibility checking with quadratic worst-case running times. In this research we develop a new OkNN algorithm which avoids these disadvantages by representing the traversable space as a collection of convex polygons; i.e. a Navigation Mesh. We then adapt an recent and optimal navigation mesh algorithm, *Polyanya*, from the single-source single-target setting to the the multi-target case. We also give two new and online heuristics for OkNN. In a range of empirical comparisons we show that our approach can be orders of magnitude faster than competing methods that rely on visibility graphs.

Keywords: Obstacle Nearest Neighbor, kNN, Navigation Mesh, Spatial Search, Obstacle Distance, Obstacle Navigation

Chapter 1

Introduction

1.1 Overview

Obstacle k-Nearest Neighbor (OkNN) is a common type of spatial analysis query which can be described as follows: given a set of target points and a collection of polygonal obstacles, all in two dimensions, find the k closest targets to an a priori unknown query point q. Such problems appear in a myriad of practical contexts. For example, in an industrial warehouse setting a machine operator may be interested to know the k closest storage locations where a specific inventory item can be found. OkNN also appears in AI path planning field, for example, in competitive computer games, agent AIs often rely on nearest-neighbour information to make strategic decisions such as during navigation, combat or resource gathering.

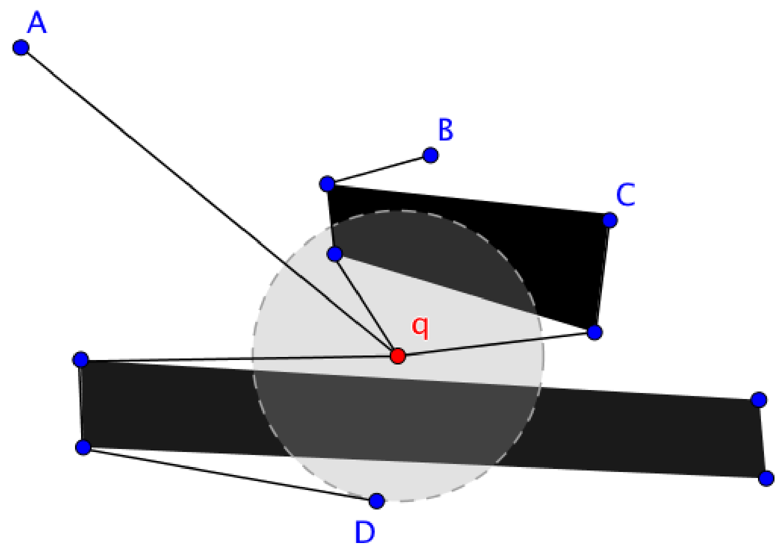


Figure 1.1: We aim to find the nearest neighbour of point q from among the set of target points A, B, C, D. Black lines indicate the Euclidean shortest paths from q. Notice D is the nearest neighbor of q under the Euclidean metric but also the furthest neighbor of q when obstacles are considered.

Tradional kNN queries in the plane (i.e. no obstacles) is a well studied problem that can be handled by algorithms based on spatial index. A key ingredient to the success of these algorithms is the Euclidean metric which provides perfect distance information between any pair of points. When obstacles are introduced however the Euclidean metric becomes an often misleading lower-bound, and the metric becomes obstacle distance. Figure 1.1 shows such an example.

1.2 Major Challenges

Two popular algorithms for OkNN, which can deal with obstacles, are *local visibility graphs* [21] and *fast filter* [20]. Though different in details, both of these methods are similar in that they depend on the incremental and online construction of a graph of co-visible points, and use *Dijkstra* to compute shortest path. Algorithms of this type are simple to understand, provide optimality guarantees and the promise of fast performance. Such advantages make incremental visibility graphs attractive to researchers and, despite more than a decade since their introduction, they continue to appear as ingredients in a variety of kNN studies from the literature; e.g. [9–11]. However, incremental visibility graphs also suffer from a number of notable disadvantages including:

1. online visibility checks;
2. an incremental construction process that has up to quadratic space and time complexity for the worst case;
3. duplicated effort, since the graph is discarded each time the query point changes.

In section 2.4, we will introduce these two algorithms with detail, and discuss why they have such disadvantages.

1.3 Major Objectives

In this research, we develop a new method for computing *OkNN* which avoids same disadvantages in existing works. Our research extends an existing very fast point-to-point pathfinding algorithm *Polyanya* to multi-target case.

1.4 Thesis Organisation

The rest of the thesis is organised as follows:

- In chapter 2, we review related works in different area, includes: AI searching, spatial index and spatial query processing.
- In chapter 3, we introduce the proposed algorithms and discuss their behaviors, formal proof for correctness will be proveded.
- In chapter 4, we provide experiment results to demonstrate the performance of propsed algorithms.
- In chapter 5, we summarize our contributions and discuss future works.

Chapter 2

Literature Review

2.1 Overview

As we've mentioned in previous chapter, OkNN problem appears in both AI path planning and Spatial query processing. Therefore, this literature review includes related works in these two fields.

In section 2.2, we introduce two classic pathfinding algorithms: *Dijkstra* and *A**, as the historic background. In section 2.3, we introduce a spatial index *R-tree*, and discuss how it solves traditional kNN problem. In section 2.4, we focus on existing works on OkNN, two algorithms based on *Local Visibility Graph* will be discussed. In section 2.5, we introduce a very fast point-to-point algorithm in AI path planning field which shows a new direction to solve OkNN problem. In section 2.6, we briefly discuss other related spatial queries which can be improved by our research.

2.2 Classic pathfinding

The most widely used pathfinding algorithm is *Dijkstra* [8]. The algorithm works on a nonnegative weighted graph, it requires a priority queue and regards the length of shortest path as key, and it visit vertices in the order of lenght of shortest path until requirements be satisfied, e.g. the target has been found. When the target is the furthest vertex to the start vertex, *Dijkstra* has to explore the entire map. Based on such consideration, researchers generalized *Dijkstra* algorithm to *best-first search* which explores a graph by expanding the most promising node chosen acording to a specified rule. *A** [14] is known as a famous *best-first search*, it select the path that minimizes:

$$f(n) = \textit{g-value}(n) + \textit{h-value}(n)$$

where n is the last node on the path, *g-value* is the length of shortest path from start to n, *h-value* is a estimation of shortest path from n to the goal which is problem-specific. One important propert of *h-value* is admissibility, meaning that it never overestimates the actual cost to the target. For example, in an Euclidean plane with obstacles, *h-value* can be the Euclidean distance.

In following sections and the chapter 3, we will show how *Dijkstra* and *A** algorithms be applied on the OkNN problem.

2.3 Spatial Index

2.3.1 *R-tree*

R-tree has many variations [2, 13, 16, 19], they improve efficiency in different aspects, but they still provide same functionality, so we only introduce the classic *R-tree* in this section.

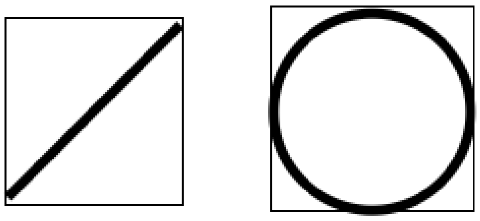
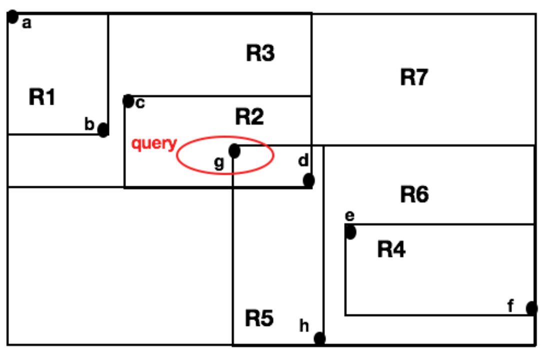
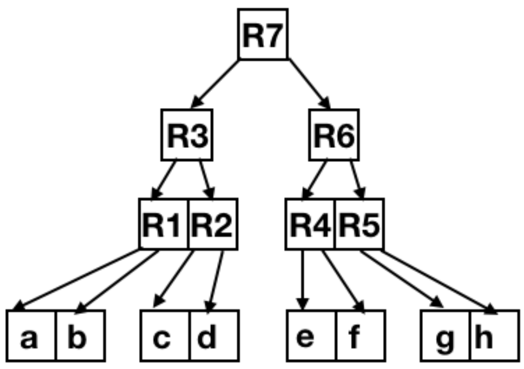


Figure 2.1: Both segments, circle and irregular shape can be represented by their MBR

R-tree is a heighb-balanced tree [13], all objects are stored in leaf node. In leaf node, if an object is not a point, it would be represented by its *Minimal Bounding Rectangle* (MBR), figure 2.1 shows example of MBRs. Each interior node is also represented by a MBR which contains either leaf nodes or descendant interior nodes. To guarantee efficiency, each non-root node of *R-tree* can contain at least m entries and at most M entries, where m, M are specified constant when *R-tree* is built, and *R-tree*'s root alway has two entries. Usually, objects retrieval start from the root, then narrow down to children nodes based on spatial information in their MBRs, and finally retrieve objects from leaf nodes. Figure 2.2 shows how to store and retrieve objects.



(a) Hierachy of MBRs



(b) Corresponding tree structure

Figure 2.2: $\{a, b, c, d, e, f, g, h\}$ is the object set, $R1, R2, R4, R5$ are leaf nodes, $R3, R6$ are interior nodes, and $R7$ is the root. The red oval is a range query, starting with $R7$, since $R6$'s MBR overlapped with query area, we narrow down to $R6$, then to $R5$, and finally retrieve g. Notice that $R3, R2$ also overlap with the query, so they will also be visited, but nothing retrieved.

From the example in figure 2.2, we can see that overlapping area will be explored multiplel times in retrieval progress, which duplicated efforts. Thus, some variants use strict non-overlapping interior node (e.g. R^+-*tree* [19]), and non-overlapping *R-tree* is a wide topic in spatial index which beyond the scope of this thesis.

2.3.2 Nearest Neighbor Query

In the *R-tree*, all nodes are organized by their spatial information, so that the nearest neighbor of a point can be retrieved by exploring tree nodes in some order. To introduce the algorithm, we need to discuss two metrics: given query point q and the MBR of a tree node

- ***mindist*** is the minimal distance from q to the MBR, it estimates the distance from q to inside object, so this metric is the priority of the tree node;
- ***minmaxdist*** is the uppber bound of the NN distance of any object inside the MBR, if the *mindist* of any MBR large than this value, then such MBR cannot contains the nearest object, so this metric is for pruning.

The algorithm starts with root node and proceeds down the tree. When it visits a leaf node, current nearest neighbor will be updated; When it visits a non-leaf node, the children of such node is sorted by *mindist*, and pruned by *minmaxdist*, then algorithm does *depth-first-search* on ordered and pruned children nodes; when algorithm finished, the updated nearest neighbor is the answer, and it can be easily generalized to finding kNN by changes below:

- Tracking k current nearest neighbors, instead of one.
- The pruning is according to the current k-th nearest neighbor.

This kind of algorithm is called *branch-and-bound* traversal, which has been well studied and widely used in other artifical intelligence areas [19], and existing NN queries are based on it with different ordering and pruning strategies, more details are in [3, 18].

2.4 Obstacle k-Nearest Neighbor

In OkNN problem, the metric is obstacle distance, so all existing OkNN algorithms are actually Obstacle Distance Computation (ODC). In following subsections, we introduce these ODC algorithms and show how to apply them on OkNN.

2.4.1 In-main-memory OkNN

Solving obstacle path problems in-main-memory has been well studied [6], these works need to compute visibility graph which any pair of co-visible vertices has en edge, figure 2.3 shows an example.

Since all edges of the shortest path belong to visibility graph [17], once precomputed it and include visible edges from query point, we can run shortest path algorithm (e.g. *Dijkstra*) to find k-nearest neighbor. However, precomputing visibility graph(VG) is costly: even the best algorithm [12] has $O(m + nlogn)$ runtime, where n is the number of vertex and m is the number edges, and in practice m can reach to $O(n^2)$. In spatial database scenario, n can be more than $10, 000$, so in-main-memory approach is not suitable for spatial database scenario.

2.4.2 Local Visibility Graph

Since building global VG is infeasible, researchers in spatial database field are motivated to design an algorithm that only consider query related area.

In 2004, Zhang proposed the *Local Visibility Graph* (LVG) algorithm [21] to compute obstacle distance. Assume obstacles are stored in *R-tree*, given query point q, and a target t, the algorithm runs in following way:

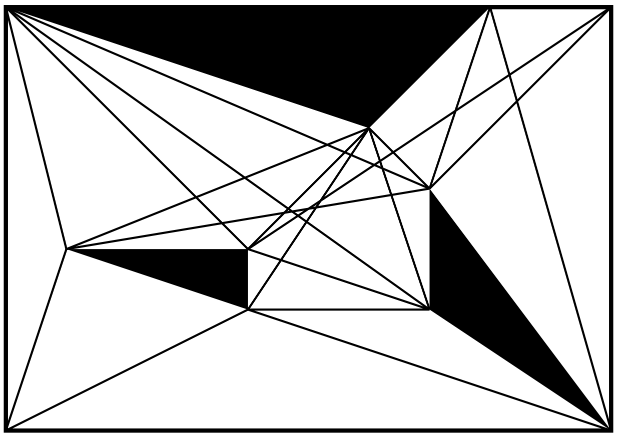
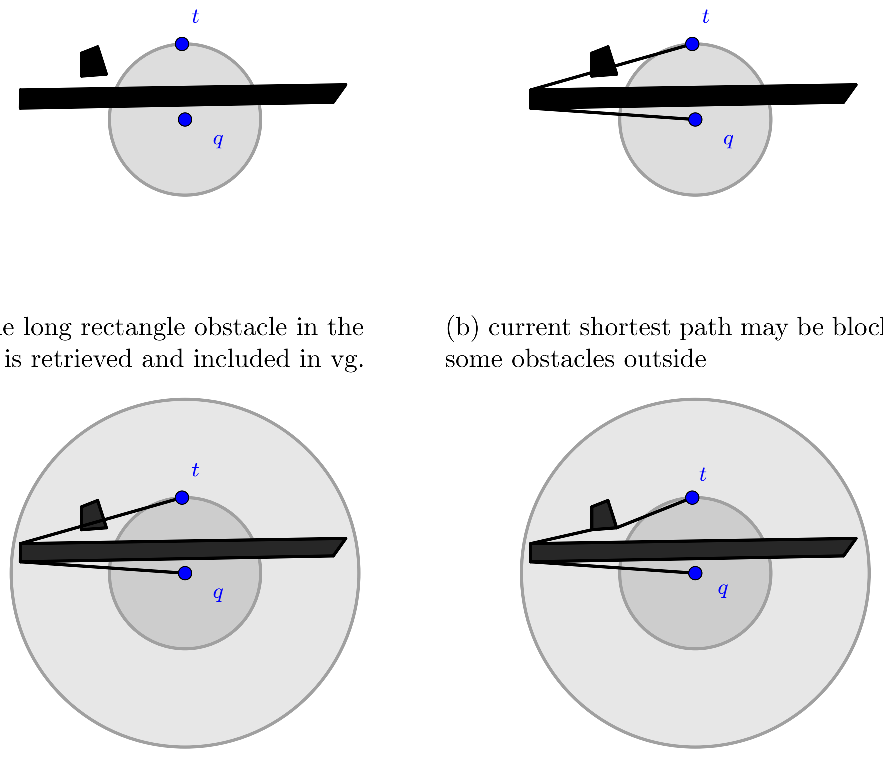


Figure 2.3: The rectangle is the boundary of the map, black polyogns are obstacles, and all black lines are edge in the visibility graph.

1. It starts with a small VG centered on q with radius $r = d_e(q, t)$ (figure 2.4a);
2. Then compute shortest path on current VG (figure 2.4b);
3. Enlarge the circle to current obstacle distance $r = d_o(q, t)$, update the VG incrementally (figure 2.4b) and recompute the shortest path (figure 2.4d);
4. Repeat the previous step until $r >= d_o(q, t)$.



(a) the long rectangle obstacle in the circle is retrieved and included in vg.

(b) current shortest path may be blocked by some obstacles outside

(c) enlarges the circle and updates the vg

(d) recomputes the shortest path

Figure 2.4: LVG algorithm

Since $r >= d_o(q,t)$, when algorithm finish, it guarantees that no obstacle on current shortest path, and thus proof the correctness. This algorithm can be extended to multi-target scenario to solve OkNN problem:

- Initially, t is the *k-th* nearest neighbor in Euclidean space, so that it guarantees that the VG always contains at least k targets;
- Terminate when r not less than the current *k-th* nearest distance;

2.4.3 Fast filter

There's another similar work proposed by Xia [20], the difference is, instead of considering obstacles in a circle area, it only retrieves obstacles on current shortest path, updates the LVG and recomputes the shortest path. Since less obstacles are involved in VG, the algorithm is called *Fast Filter*. Figure 2.5 shows an example.

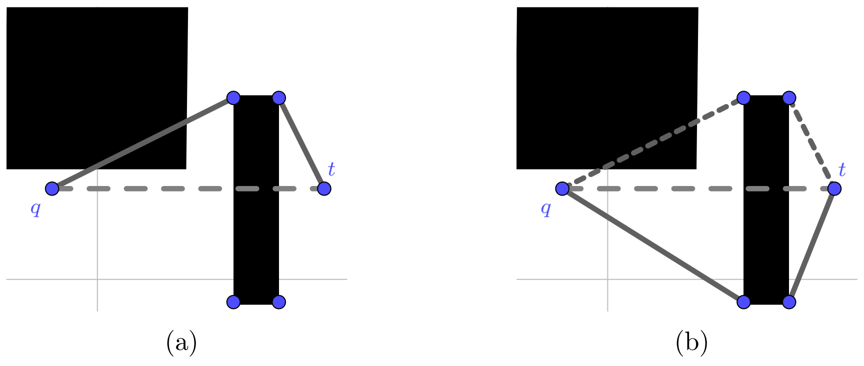


Figure 2.5: Black polygons are obstacle, q is query point, t is the target. Initially, the VG only contains the rectangle obstacle between the qt, and the corresponding shortest path is computed (fig 2.5a). Then retrieves the square obstacle on the current path, updates VG and recomputes shortest path (fig 2.5b).

To solve OkNN, we need following changes:

- initially compute obstacle distance for k nearest neighbors, and store results in *max-heap* with size k;
- keep retrieving next NN and compute the obstacle distance d_o;
- when d_o not large than the top value of heap, pop top and insert the current d_o;
- terminate when the Euclidean distance to current NN is large than the Obstacle distance on top of the heap;

2.4.4 Discussion

LVG [21] and *Fast filter* [20] are simple to understand, provide optimality guarantees and the promise of fast performance. Such advantages make them attractive to researchers and, despite more than a decade since their introduction, they continue to appear as ingredients in a variety of kNN studies from the literature; e.g. [9,10]. However, these visibility graph based algorithms also suffer from a number of notable disadvantages including:

(i) costly online visibility checks;

(ii) an incremental construction process that has up to quadratic space and time complexity for the worst case;

(iii) duplicated effort, since the graph is discarded each time the query point changes.

2.5 Pathfinding on Navigation Mesh

2.5.1 Historic background

Because of the limitation of VG, *Navigation Mesh* comes to our sight. Ronald first proposed this concept in 1986 [1], then it is applied on robotics and game pathfinding.

Navigation mesh divides traversable space into convex polygons, figure 2.6 shows an example. Compare to VG (fig 2.3), it can be generated by*Constrained Delaunay Triangulation* [4] in $O(nlogn)$ times, so that it is feasible to preprocess the entire map. Additionally, adding or removing obstacles only need to modify a few number of mesh polygons, which is very flexible. It seems like a perfect framework for OkNN, the only problem is: how to compute obstacle distance on it?

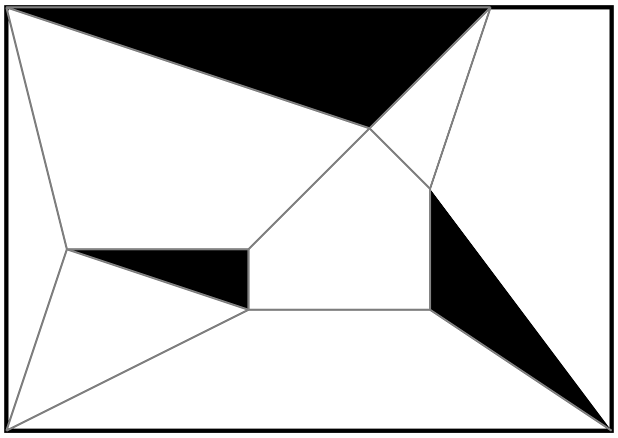


Figure 2.6: Navigation Mesh: The rectangle is the boundary of the map, black polyogns are obstacles, gray lines are edges of mesh polygons.

Previous pathfinding algorithms on navigation mesh are not suitable for spatial query processing, and thus it's not attractive to researchers in this fields. Three widly used algorithms are:

- Channel Search [15]: first find an abstract path from start to target composed by polygons, then refine the abstract path to a sequence of concrete points. This algorithm only generate an approximately shortest path, the lack of optimality makes it not suitable for OkNN query.
- TA* [7]: similar to Channel Search, but it can repeat the search process until finding an optimal shortest path. The repeating is time-consuming, so it is not suitable for query processing.
- TRA* [7]: similar to TA*, but TRA* can utilize preprocessing to speed up pathfinding. The problem is that the precomputed information will be invalid when environment change, so it is not suitable for database scenario.

However, this fact has been changed by a recent work called *Polyanya* [5]. It is a fast, optimal and fexible pathfinding algorithm which is perfect for query processing in spatial database.

2.5.2 Polyanya

2.6 Related Spatial Queries

Chapter 3

Proposed Algorithms

3.1 Overview

3.2 Interval Heuristic

3.3 Target Heuristic

3.4 Summary

Chapter 4

Empirical Analysis

4.1 Overview

4.2 Benchmark

4.3 Competitors

4.4 Experiment 1: lower bounds on performance

4.5 Experiment 2: computing more nearest neighbor

4.6 Experiment 3: changing number of targets

Chapter 5

Conclusion and Future Work

5.1 Research Contributions

5.2 Future Works

Bibliography

[1] Ronald C. Arkin. Path planning for a vision-based autonomous robot. volume 0727, pages 0727 – 0727 – 11, 1987.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, volume 19, pages 322–331. ACM, 1990.

[3] King Lum Cheung and Ada Wai-Chee Fu. Enhanced nearest neighbour search on the r-tree. *ACM SIGMOD Record*, 27(3):16–21, 1998.

[4] L Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[5] Michael L Cui, Daniel D Harabor, Alban Grastien, and Canberra Data61. Compromise-free pathfinding on a navigation mesh. *IJCAI*, 2017.

[6] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[7] Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Aaai*, volume 6, pages 942–947, 2006.

[8] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[9] Yunjun Gao, Qing Liu, Xiaoye Miao, and Jiacheng Yang. Reverse k-nearest neighbor search in the presence of obstacles. *Information Sciences*, 330:274–292, 2016.

[10] Yunjun Gao, Jiacheng Yang, Gang Chen, Baihua Zheng, and Chun Chen. On efficient obstructed reverse nearest neighbor query processing. In *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in Geographic Information Systems*, pages 191–200. ACM, 2011.

[11] Yunjun Gao and Baihua Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 577–590. ACM, 2009.

[12] Subir Kumar Ghosh and David M Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[13] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[14] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[15] Marcelo Kallmann. Path planning in triangulations. In *Proceedings of the IJCAI workshop on reasoning, representation, and learning in computer games*, pages 49–54, 2005.

[16] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. Technical report, 1993.

[17] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[18] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79. ACM, 1995.

[19] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. Technical report, 1987.

[20] Chenyi Xia, David Hsu, and Anthony KH Tung. A fast filter for obstructed nearest neighbor queries. In *British National Conference on Databases*, pages 203–215. Springer, 2004.

[21] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Spatial queries in the presence of obstacles. *Advances in Database Technology-EDBT 2004*, pages 567–568, 2004.