Maximum Visibility Queries in Spatial Databases

Sarah Masud #1, Farhana Murtaza Choudhury #2, Mohammed Eunus Ali #3, Sarana Nutanong *4

#*Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology*
Dhaka 1000, Bangladesh
1 sarahmasud.sm@gmail.com
2 farhanamc@gmail.com
3 eunus@cse.buet.ac.bd

**University of Maryland*
College Park, Maryland, USA
4 nutanong@umiacs.umd.edu

Abstract— **Many real-world problems, such as placement of surveillance cameras and pricing of hotel rooms with a view, require the ability to determine the visibility of a given target object from different locations. Advances in large-scale 3D modeling (e.g., 3D virtual cities) provide us with data that can be used to solve these problems with high accuracy. In this paper, we investigate the problem of finding the location which provides the best view of a target object with visual obstacles in 2D or 3D space, for example, finding the location that provides the best view of fireworks in a city with tall buildings. To solve this problem, we first define the quality measure of a view (i.e., visibility measure) as the visible angular size of the target object. Then, we propose a new query type called the *k*-Maximum Visibility (*k*MV) query, which finds *k* locations from a set of locations that maximize the visibility of the target object. Our objective in this paper is to design a query solution which is capable of handling large-scale city models. This objective precludes the use of approaches that rely on constructing a visibility graph of the entire data space. As a result, we propose three approaches that incrementally consider relevant obstacles in order to determine the visibility of a target object from a given set of locations. These approaches differ in the order of obstacle retrieval, namely: query centric distance based, query centric visible region based, and target centric distance based approaches. We have conducted an extensive experimental study on real 2D and 3D datasets to demonstrate the efficiency and effectiveness of our solutions.**

I. INTRODUCTION

Visibility computation has been an active research topic in the field of computational geometry, computer graphics, and spatial databases. In particular, the development of location based services has introduced a new platform for applications tailoring the concept of visibility for spatial queries. Our research is motivated by an observation that many real-life problems (e.g., placement of surveillance cameras, pricing of hotel rooms with a view) involve determining how much of a given target object is visible to us. More concrete examples can be given as finding a suitable location to enjoy maximum visibility of the new year firework above a city tower where the high-rise buildings can obstruct the firework's view. Alternatively, in a safari park the tourists may wish to find a particular location that provides the maximum view of an interesting spot. A similar scenario from medical context arises, where a possible set of locations inside the abdomen is given to insert a camera for operating on the liver, whose view is obstructed by the presence of other internal organs.

In each of the aforementioned cases, it is desired to select a location from a set of candidate locations, that maximizes the visibility of a target object in the presence of obstacles. Moreover, one may want to find k such locations to have more flexibility. For example, a tourist may want to find the best three locations according to their provided view of a tourist spot, so that he/she can select the most convenient location among them.

In this paper, we model this class of visibility problems as finding and ranking locations according to the visibility of a target object measured as its visible angular size from a particular location. We propose a new query type called the *Maximum Visibility* (MV) query that considers the impact of obstacles on the visibility of a target object. Given a target object T, a set O of m obstacles, and a set Q of n candidate locations (termed as query points), the MV query finds the query point in Q that provides the maximum visibility of T. We also introduce *k-Maximum Visibility* (*k*MV) query, a generalization of the MV query to find the k query points that maximize the visibility of T.

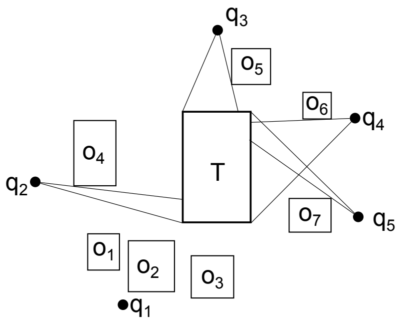


Fig. 1: A target object T, a given set of query points $\{q_1, q_2, \ldots, q_5\}$, and a set of obstacles $\{o_1, o_2, \ldots, o_7\}$

An example is illustrated in Fig. 1 that shows a target object T, a set of seven obstacles $\{o_1, o_2, \ldots, o_7\}$, and a set of five query points $\{q_1, q_2, \ldots, q_5\}$. We need to find the query point that provides the maximum visibility of T. Without considering any obstacle, the query point q_1 provides the maximum visibility of T. Now, visibility of T from a query point q is reduced due to the presence of an obstacle if that obstacle intersects any line connecting q and T. For example, the visibility of T from the query point q_2 is reduced due to the obstacle o_4. Thus when all the obstacles are considered, q_4 is found to be the best query point that provides the maximum visibility of T. Then q_3 is the next best point and so on.

To answer this query we propose solutions which can be applied to two-dimensional (2D) and three-dimensional (3D) cases. As an approximation of the 3D case, the 2D case is useful when obstacles can be simplified as polygons using their footprints, and the elevations of query points are ignored, e.g., placement of street cameras with buildings as obstacles. The 3D case, on the other hand, is suitable when we want to take into account the heights of obstacles and the elevations of query points, e.g., choosing the hotel room with the best river view in a city with buildings of different heights.

Algorithms used in computational geometry and computer graphics efficiently solve the visibility problem by constructing visibility polygon and visibility graph. But they must rely on preprocessing and/or accessing all obstacles [1], [2], [3], [4]. Unlike computational geometry, spatial applications involve dynamic data as location of query point, target, or obstacles. Hence the traditional visibility computation algorithms are not suitable for many spatial database applications as any update will invalidate the preprocessed data. Moreover, accessing all obstacles for each query point is expensive. For example, location based applications involving visual simulation of walk-through, driving simulators, or virtual reality programs require high update rates for interactive feedback. So, visibility calculation for a single query point has to be carried out in realtime without inferring significant runtime overhead. For very large databases, to achieve this kind of performance, hardware acceleration must be assisted by algorithms that reduce the geometry to be rendered to a tolerable amount.

The novelty of our work is a query processing method that avoids computing the exact visible angular size (or the visibility) of the target object from each query point. Specifically, we apply the concept of incremental branch-and-bound search to rule out those query points that will certainly yield *bad views*. This is achieved by incrementally including obstacles around the query points and the target to refine the visible angular size of the target from each query point as the search progresses. In this way, a query point that is largely obstructed by a subset of obstacles can be ruled out from the search process without having to compute the exact visible angular size of the target.

Since the performance of the described query processing method greatly depends on the order in which obstacles are considered, we focus our investigation on different approaches of ordering obstacles. Specifically, we propose three different approaches to answer an MV query on a large dataset, which differ in the order we consider obstacles. We refer to them as: *query centric distance based approach* (QD), *query centric visible region based approach* (QV), and *target centric distance based approach* (TD). All these approaches compute the MV query by accessing reduced number of obstacles. The QD approach processes obstacles based on their distances from the query point under consideration. The intuition behind this approach is that the obstacles that fall completely within the shadow of another obstacle will not affect the visibility of the target from that query point. So they can be discarded safely. On the other hand the QV approach considers obstacles based on their impact on the visibility of the target instead of their distances from a query point. The key concept behind this approach is that, the more an obstacle obstructs the visible portion of the target, the more the other obstacles are likely to be pruned (i.e., not accessed from disk or not considered for visibility calculation) from the dataset for that query point. Unlike the first two approaches, the TD approach processes obstacles based on their distances from the target. This approach can be suitable when the number of obstacles is relatively small compared to the number of query points and the obstacles are closely oriented towards the target.

In summary, the contributions of this paper are as follows:

- We introduce the *Maximum Visibility* (MV) query in a d-dimensional space.
- We formulate the MV query in spatial databases for the $2D$ and $3D$ spaces.
- We propose three different approaches: *query centric distance based approach*, *query centric visible region based approach*, and *target centric distance based approach* to answer the MV query.
- We conduct an experimental study on two real datasets that shows that our approaches reduce the number of obstacles retrieved and the total processing time.

The rest of the paper is organized as follows. Section II reviews the related works. In Section III, we formally define our problem, and successively in Section IV we present an overview of our proposed solution. Section V presents the experimental results. Finally in Section VI we conclude and discuss the future directions.

II. Related Works

Visibility computation has been actively studied in computational geometry, computer graphics, and spatial databases. In this section, we first briefly discuss the existing works on visibility in computational geometry and computer graphics. Then we discuss the visibility computation in spatial queries in details as it is our main focus in this paper.

A. Visibility in Computational Geometry and Graphics

For visibility computation in computational geometry, several approaches have been proposed to construct visibility graph and visibility polygon ([1], [2], [3], [4], [5], [6]). A visibility graph is defined by a set P of n points inside a polygon Q where two points $p, q \in P$ are joined by an edge if the segment $pq \subset Q$ [5]. Ben et al. [5] have constructed the visibility graph by examining the general case in which P is an arbitrary set of points, interior or on the boundary of Q. They have given a nearly optimal algorithm for simple polygons. For polygons with holes (non-simple polygons) they have introduced a notion of *robust* visibility and given a nearly optimal algorithm to construct the visibility graph.

Two points inside a polygon are visible to each other if their connecting segment remains completely inside the polygon. The visibility polygon of a point q, called $V(q)$ in a polygon P is defined as the set of points in P that are visible from q [1]. Zarei and Ghodsi consider the problem of computing the visibility polygon of a query point inside a non-simple polygon [1]. They add some new edges and vertices to the non-simple polygon to unfold it along those edges and convert it into a simple polygon. They utilize some existing methods and improve the time to compute visibility polygon in exchange of costly preprocessing of given input and an expensive data structure of size $O(n^3)$. Later Asano et al. have improved the preprocessing time and reduced the space requirement [2], [6]. Similar studies are also conducted in [3], [4].

Though the algorithms above efficiently solve the problem of constructing visibility polygon, they must rely on preprocessing and/or accessing all obstacles. Consequently, they are inappropriate for many spatial database applications as any update will invalidate the preprocessed data and accessing all objects for each query point is not feasible in such cases.

Visibility computation is also a well studied topic in computer graphics. The existing methods utilize various indexing structures such as LoD-R-tree, HDoV-tree to deal with visibility queries in visualization systems ([7], [8], [9]). They use various acceleration techniques to render complex models at interactive frame rates. However their main focus is rendering a scene while we focus on calculating and ranking the visibility of a specific target object from various query points. Besides they rely on preprocessing and consume significant amount of main memory space, which is not feasible for the MV query.

B. *Visibility in Spatial Queries*

The concept of finding k nearest neighbors (kNN) is a powerful and versatile concept which can be applied to many spatial problems. In general, the kNN query finds the k nearest data points with respect to a query point based on a given distance measure. In this subsection, we discuss variants of the kNN query where the distance between two objects can be affected by obstacles, which include Obstructed NN (ONN) query [10], [11], Visible NN (VNN) query [12], [13], Continuous Obstructed NN (CONN) query [14], and Continuous Visible NN (CVNN) query [15].

Given a set P of data points and a query point q in a multidimensional space, ONN query returns k points in P that have the smallest obstructed distances to q. Obstructed distance is the length of the shortest path that connects any two points without crossing any obstacle. Zhang et al. [10] propose several algorithms for various spatial queries such as, range search, nearest neighbors, e-distance joins and closest pairs in an obstructed space. They maintain local visibility graphs only for the obstacles that are around the point q and may influence visibility. Xia et al. [11] focus on finding the kNNs of a given query point according to the obstructed distance. The algorithm starts with a *local workspace* and introduces obstacles relevant to the query point incrementally to filter out irrelevant query points and obstacles.

Nutanong et al. introduce VNN search to find the NN that is visible to a query point [12], [13] based on the fact that a farther object cannot affect the visibility of a nearer object. The basic idea is to perform NN search and check its visibility condition incrementally. Another variant is Aggregate VkNN (AVkNN) query, which finds the visible k nearest objects to a set of query points based on an aggregate distance function [13]. Their proposed approach accesses the database via multiple VkNN queries. To improve its performance, another approach is proposed that issues an aggregate k nearest neighbor query to retrieve objects from the database and then re-rank the results based on the aggregate visible distance.

The Continuous Nearest Neighbor (CNN) query finds the kNN results for a moving query point [16]. Gao et al. studied a variation of CNN namely Continuous Obstructed Nearest Neighbor (CONN) query [14]. Given a data set P, an obstacle set O, and a query line segment q in a 2D space, a CONN query retrieves the nearest neighbor of each point on q according to the obstructed distance.

The Continuous Visible Nearest Neighbor (CVNN) query is proposed in [15]. CVNN retrieves visible nearest neighbor along a query line segment in the presence of obstacles. Given a data set P, an obstacle set O, and a query line segment q, a CVNN query returns a set of (p,R) tuples such that $p \in P$ is the NN to every point r along the interval $R \subset q$ and p is visible to r. CVNN extends the CNN [16], and CONN query [14] by taking visibility into consideration.

The aforementioned spatial queries find the nearest object in an obstructed space from a given query point where query results are ranked according to the distances or visible distances from the query point. Our Maximum Visibility (MV) query, on the other hand, ranks locations based on how much a given target object is visible to each of them. This requires a visibility computation rather than a distance calculation. Therefore, the approaches used in various NN queries are not applicable in this context. So we introduce the MV query and provide efficient solutions to answer this query.

III. Problem Formulation

In this section, we formulate the *k-Maximum Visibility* (kMV) query. Given a set Q of n query points $\{q_1, q_2, \ldots, q_n\}$, a set O of m obstacles $\{o_1, o_2, \ldots, o_m\}$, and a target object T in a d-dimensional space, the *k-Maximum Visibility* (kMV) finds k query points $Q'\{q'_1, q'_2, \ldots, q'_k\}$, $Q' \subseteq Q$, where *visibility*$(q'_i) \geq$ *visibility*(q'_j), $1 \leq i < j \leq k$, and *visibility*$(q') \geq$ *visibility*(q'') for each $q' \in Q'$, $q'' \in Q - Q'$.

To formally define the *visibility* of a query point q, *visibility*(q), at first we need to define the point to point visibility.

Definition 3.1: Point to point visibility. Given two points p, p' and a set O of obstacles in a space, p and p' are visible to each other if and only if the straight line connecting them, $\overline{pp'}$ does not cut through any obstacle, i.e., $\forall o \in O$, $\overline{pp'} \cap o = \emptyset$.

Hence, the visibility of T from a point q is the set P of points over T, where $\forall p \in P$, p is point to point visible to q. Based on this definition we formulate the visibility metric that is used to rank query points. Without loss of generality in the subsequent sections we limit our discussion in 2D and 3D spaces. However, from a theoretical standpoint, our approach is also applicable to higher dimensions.

A. *Visibility Metric*

The perceived size of an object depends mainly on its visual angle [17] and the distance from the viewing point [18]. So the visibility of the object varies with these metrics. For simplicity we measure the visibility of an object as the angular size of the visible part of the object in this paper. However, the distance based metric can also be incorporated in our proposed query processing solution. This subsection depicts how the visibility metric, i.e., the angular size, is used to find the visibility of a target in 2D and 3D spaces.

Visibility in 2D: In 2D, if a face of an object is viewed from an oblique angle, the oblique projection of the length of the face becomes smaller than the original length. Let a line $\overline{qm}$, which connects a point q and the midpoint m of line l, creates an angle θ with l. According to the convention used in engineering drawing of oblique projection, the oblique visibility of the line l from q is reduced by the factor $\frac{\theta}{90^o}$ [17]. This is depicted in Fig. 2 using a rectangle $abcd$ and a point q in a 2D space. The midpoint of the line $\overline{ab}$ is M_w and the line $\overline{qM_w}$ creates an angle of 30^o with $\overline{ab}$. So the visibility of $\overline{ab}$, measured as length, reduces from $300m$ to $\frac{30}{90} \times 300 = 100m$.

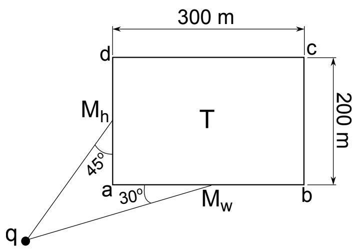


Fig. 2: Visibility metric in 2D

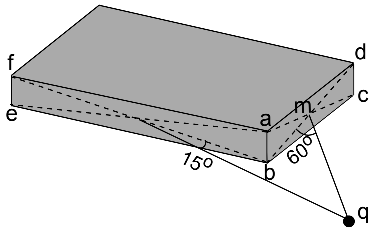


Fig. 3: Visibility metric in 3D

Visibility in 3D: The calculation of visibility in 3D is a generalization of the 2D counterpart. In the 2D case, the visibility of T is measured as length, while in the 3D case, the visibility of T is measured as area. For the ease of explanation, we consider 3D objects as rectangular cuboids. However, the core concept can be extended to support any 3D shape. In a 3D space, at most three faces of the target can be visible from a query point, as each of the visible faces occludes its opposite face. Hence the invisible faces can be discarded from further consideration for that query point. Thus, the visibility of the target with respect to a query point is the union of the visibility of the three visible faces of the target.

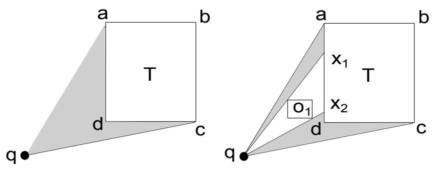


Fig. 4: Visible region in 2D

Let a line $\overline{qm}$ connects a point q and the center point m (i.e., the point where the diagonals of the plane intersect) of a bounded plane p and creates an angle θ with p. The oblique visibility of p from the point q is reduced by the factor $\frac{\theta}{90^o}$. In Fig. 3, the center point of the plane $abcd$ is m. The line $\overline{qm}$ creates an angle of 60^o with the plane $abcd$. So the reduced area viewed from q is $v = \frac{60}{90} \times area(abcd)$. Thus the visibility of face $abcd$ from q is v.

To answer a kMV query we need to find the query point for which our visibility metric (i.e., angular size) is maximized. As stated earlier, calculating the visibility from each query point is prohobitively expensive. Hence, our proposed branch-and-bound solution relies on incremental maintenance of visible regions of different query points. We discuss the concept of visible region in the next subsection.

B. *Visible Region*

The visibility of a target object T with respect to a query point q can be affected by a small subset of obstacles from a large number of obstacles in the database. So an efficient method to determine relevant obstacles is very crucial to the performance of our algorithm. In general, an obstacle can affect the visibility of T with respect to q only if it obstructs the line of sight from q to any point on T. We can prune irrelevant obstacles by maintaining a region that may be affected by an obstacle. We call this region the *visible region* and formally define it as follows.

Definition 3.2: VISIBLE REGION. Given a set O of obstacles, the visible region of a target T with respect to a point q is the set of points p such that (i) p is between q and T; (ii) p is point to point visible to q; (iii) p is point to point visible to a location on T.

In our proposed solution obstacles are included into consideration one by one. The visible region of a target T with respect to a query point q is used to (i) determine which obstacles could possibly affect the visibility of T; (ii) keep track of the visible parts of the target T to estimate its visibility for search ordering purposes. As we include more obstacles into consideration, the visible region from a point q reduces, and hence the visibility estimate also reduces.

Visible region in 2D: Let q be a point and T be the target object in a 2D space. The function *visible_region(q,T)* returns the set of visible points from q, that lies within the same 2D plane as T and are bounded by q and the visible points over T. For example, in Fig. 4, *visible_region(q,T)* returns $\{adcq\}$ as the visible region of T from q without any obstacle. Visible region of T from q is reduced by the presence of an obstacle in the visible region. So, due to o_1, *visible_region(q,T)* returns the union of the regions $\{ax_1q\}$ and $\{x_2dcq\}$.

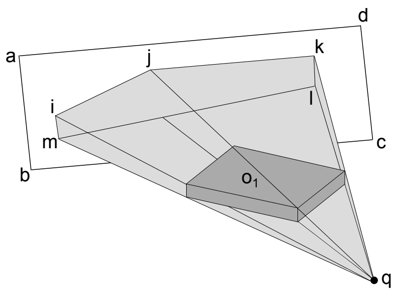


Fig. 5: Visible region in 3D

Visible region in 3D: Like visibility, the visible region of the target with respect to a query point in 3D is the union of the three visible regions of the visible faces of the target. In Fig. 5, the plane *abcd* resembles a face of the target T. Without any obstacle, the visible region from q for plane *abcd* is the polyhedron bounded by the points q, a, b, c, and d. Due to the presence of o_1, the visible region of T for q is reduced. Hence the polyhedron bounded by the points q, i, j, k, l, and m (lightly shaded region) is subtracted from the visible region of T for q.

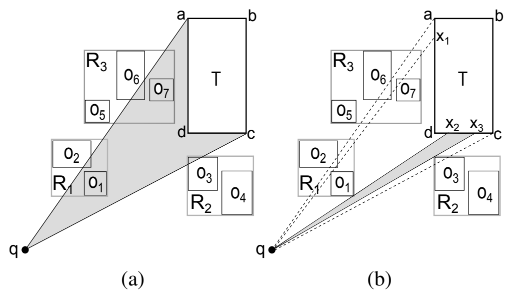


Fig. 6: Visible region computation (a) initial visible region and (b) final visible region

Generally, when an obstacle is projected upon a face of the target, it renders a polygon shaped shadow that depends on the relative positions of the query point, the obstacle, and the target. For calculating the visibility of T, the area of the shadow rendered by the obstacle is discarded from the current visible area of T. As the area calculation of a polygon is an expensive operation [19], we regard this shadow as a quadrilateral. This assumption reduces the computational cost.

To sum up, visibility of the target from a query point is the visible portion of the target measured according to our defined visibility metric while visible region covers the region bounded by a query point and a visible face of the target. The concept of visible region is used to prune obstacles that fall completely in the shadow of another obstacle.

IV. Our Approach

We assume all obstacles are indexed with an R*-tree [20], a variation of the R-tree [21]. However, our solutions also hold for other hierarchical structures. An R-tree consists of a hierarchy of *minimum bounding rectangles (MBRs)*, where each MBR corresponds to a tree node and bounds all the MBRs in its sub-tree. Data objects are stored in leaf nodes. As shown in Fig. 6, the obstacles $\{o_1, o_2, \ldots, o_7\}$ are indexed in three MBRs of an R*-tree, $\{R_1, R_2, R_3\}$ according to their spatial orientations. These three MBRs are hierarchically organized to form the root node.

To answer a kMV query, we need to calculate the visibility of the target T from different query points according to our defined visibility metric and find the first k query points that maximize the visibility of T. A *straightforward* approach to calculate the visibility is to retrieve all obstacles from the visible region of each query point. For each obstacle o, the value of the visibility and the visible region of the target from each query point (that has o in its visible region) are updated. An example of computing the visible region from a query point in straightforward approach is illustrated in Fig. 6. Initially without considering any obstacle, the visible region of a target T with respect to a query point q is the polygon *adcq* (Fig. 6(a)). According to the straightforward approach all the MBRs that overlap with the initial visible region (i.e., R_1, R_2, R_3) are retrieved from the R*-tree. These MBRs are further explored and their elements are considered to update the visible region. Let o_1 is considered first and it splits the initial visible region into ax_1q and x_2cq (shown with dotted lines in Fig. 6(b)). Thus when the effects of all obstacles are considered, the visible region is reduced to x_2x_3q.

The above process continues until the visible region from each query point is calculated. The query points are then sorted according to the values of their provided visibility of the target. Thus the first k points with the highest values are selected as the result of a kMV query. This straightforward method involves a large number of object retrieval resulting in high I/O overhead. Moreover, this approach requires calculating visibility of the target for each query point and causes high computational overhead.

To overcome the limitations of this approach, we propose three different solutions where only the obstacles that are necessary (e.g., affect the current visible region) for calculation, are retrieved incrementally from the R*-tree. These solutions differ in the retrieval order of obstacles. These approaches avoid considering all query points in each iteration. The details of these approaches are described below.

A. Query Centric Distance Based Approach (QD)

In this approach, the obstacles are incrementally retrieved from the R*-tree according to their non-decreasing distances from the query point currently being considered. As the obstacles are retrieved according to their distances from query points, we term this approach as *query centric distance based approach* (QD).

In practice, there are many query points. Considering all of them in each iteration can be prohibitively costly. Based on the best-first search principle, we propose an algorithm that is incremental in two aspects: (i) the way in which we consider query points, and (ii) the way in which we consider obstacles. Specifically, we use a priority queue VQ to maintain the search order of the query points according to the current visibility estimate of each of them. A visibility estimate is the visibility of the target T computed from a subset of relevant obstacles, and is incrementally computed by considering one

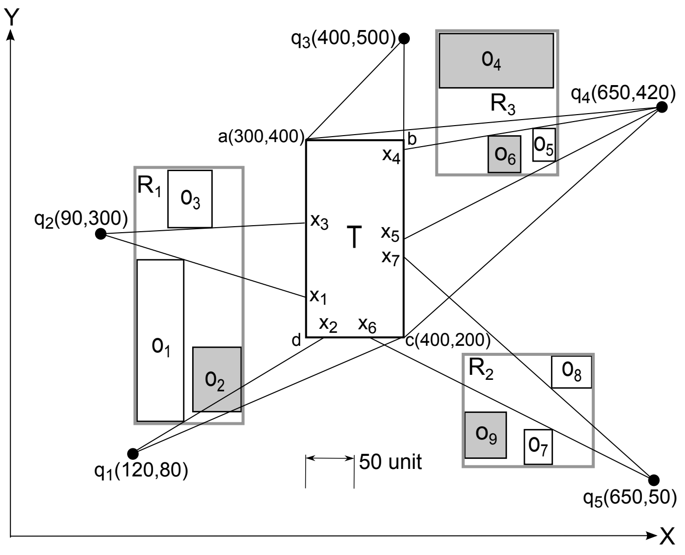


Fig. 7: Processing the MV query using the QD approach

TABLE I: Notations used and their meanings

Notation	Meaning
T	The target object.
Q	A set of n query points $Q=\{q_1, q_2, \ldots, q_n\}$.
O	A set of m obstacles $O=\{o_1, o_2, \ldots, o_m\}$.
$MinDist(q, p)$	The minimum distance between a point q and a point or an extended (e.g., rectangle) object p.
VQ	Priority queue of query points based on their non-increasing visibility of the target.
CO	List of R*-tree nodes that have already been considered.
LO	List of priority queues where a queue is maintained for each query point q. An element of the list, LO_i is a priority queue of obstacles that affect the visibility of q_i based on their non-decreasing value of minimum distances from q_i.
L	Ordered list of k query points according to their provided visibility.

obstacle at a time. For the QD method, obstacles are ordered according to their distances from a query point.

Algorithm 1 shows the steps of the QD approach for finding the kMV query result. The algorithm takes the following parameters as input: a target object T, a set Q of n query points $\{q_1, q_2, \ldots, q_n\}$, and a number k representing the number of query points to be returned. We assume all the obstacles and the target are indexed in an R*-tree. The output is L, an ordered list of k query points where the elements are sorted in order of their non-increasing visibility of the target. We summarize the notations used in this section in Table I.

In the initialization step (Lines 1.1-1.10), no obstacle is considered. Hence the unobstructed visibility of the target from each query point is calculated, the corresponding visible regions are constructed and the points are placed in a priority queue VQ, according to the non-increasing value of their provided visibility of the target. For each query point q_i, a priority queue LO_i of obstacles is maintained, based on their distances from q_i.

Algorithm 1: *k-MaximumVisibility(T, Q, k)*

```
1.1  Initialize $L$ to an empty list
1.2  Initialize a priority queue $VQ$
1.3  Initialize a list of priority queues $LO$ to an empty list
1.4  Initialize $CO$ to an empty list
1.5  $end \leftarrow$ false
1.6  $continue \leftarrow$ true
1.7  $node \leftarrow$ root
1.8  for each query point $q_i$ of $Q$ do
1.9      $UpdateVisibility(q_i, NULL)$
1.10     $Enqueue(VQ, q_i, visibility(q_i))$
1.11 while $node \neq$ empty and $end$ = false do
1.12     if $node$ is not an element of $CO$ then
1.13         for each element $n_e$ of $node$ do
1.14             for each query point $q_i$ of $Q$ do
1.15                 if $InsideVisibleRegion(q_i, n_e)$ = true then
1.16                     $Enqueue(LO[i], n_e, MinDist(q_i, n_e))$
1.17
1.18         $CO \leftarrow$ insert($node$)
1.19     $continue \leftarrow$ true
1.20     while $continue$ = true do
1.21         $current\_best \leftarrow$ Dequeue($VQ$)
1.22         $node \leftarrow$ Dequeue($LO[current\_best]$)
1.23         if $node$ = empty or $node$ = $T$ then
1.24             $L \leftarrow current\_best$
1.25             decrement $k$
1.26             if $k$ = 0 then
1.27                 $end \leftarrow$ true
1.28                 $continue \leftarrow$ false
1.29
1.30         else if $node$ is a data object and $node$ is not an
             element of $CO$ then
1.31             if $InsideVisibleRegion(current\_best, node)$ = true
                 then
1.32                 $UpdateVisibility(current\_best, node)$
1.33                 for each element $q_i$ of $VQ$ do
1.34                     if $InsideVisibleRegion(q_i, node)$ = true
                         then
1.35                         $UpdateVisibility(q_i, node)$
1.36
1.37             $Enqueue(VQ, current\_best, visibility(current\_best))$
1.38             $CO \leftarrow$ insert($node$)
1.39         else
1.40             $Enqueue(VQ, current\_best, visibility(current\_best))$
1.41             $node \leftarrow$ child($node$)
1.42             $continue \leftarrow$ false
1.43 return $L$;
```

In our approach, accessing the obstacles from the R*-tree starts from the root node. For each query point q_i, the priority queue LO_i is updated if an element of the R*-tree node intersects the current visible region of q_i. The accessed R*-tree node is placed in a list CO. Lines 1.12-1.18 of the algorithm show these steps. We can safely avoid accessing the R*-tree nodes that are already in the CO list as the queue of obstacles for each query point is updated for these nodes. This ensures that the same node is not accessed multiple times and hence reduces the I/O cost.

In each iteration, the query point q, from the top of the queue VQ is considered. From the priority queue of obstacles of this query point, the top element o, i.e., the nearest object affecting the visible region of q, is retrieved (Lines 1.21-1.22) first. If the retrieved object o is an MBR, its elements are further discovered from the R*-tree in the next iteration (Lines 1.40-1.42) as described in the previous paragraph. Otherwise, o is an obstacle; so the visibility of T from each query point that contains o in its visible region, is calculated using the function *UpdateVisibility* and VQ is updated accordingly. The corresponding visible regions from those query points are also updated. The retrieved obstacle o is then placed in the list CO. Lines 1.30-1.38 of the algorithm show these steps. We can safely avoid the calculation for an obstacle that is in CO and hence reduce the computational cost.

When a query point q is at the top of VQ and its priority queue of obstacles is empty or the target T is reached, it means that we have considered all obstacles that may affect the visibility of T with respect to q and hence the current visibility is final. The best-first search order guarantees that no other query point in the priority queue may produce a higher visibility than q. Hence, we can safely regard q as our next query point which maximizes the visibility of T. This process continues until k query points are found (Lines 1.23-1.28). Finally these k query points are returned as the result of the kMV query in the non-increasing order of their provided visibility of T.

Fig. 7 provides an example of a kMV query with 5 query points $\{q_1, q_2, ..., q_5\}$, a target object T, a set of 9 obstacles $\{o_1, o_2, ..., o_9\}$ and the value of k as 1. Table II shows the steps of visibility calculation of the QD approach according to the object retrieval order. After calculating the visibility for six retrieved obstacles, q_4 is finally returned as the query answer.

The intuition behind the QD approach, which orders obstacles according to their distances from the query points, is that obstacles closer to the query point have a greater tendency to obstruct the target and can subtract a greater portion from the visible region of T than the farther ones. Recall that the visible region is used to derive an intermediate estimate of the visibility of a target T with respect to a query point, and these intermediate estimates from different query points are used to provide the search order. Hence, the faster we reduce the visible region (to get it closer to the final result), the more accurate these intermediate estimates are.

B. Query Centric Visible Region Based Approach (QV)

The QV approach can be considered as a variant of the QD approach. The main difference between these two approaches is that QD uses the distances from query points to determine the order in which obstacles are considered, while the obstacle retrieval order in QV is directly derived from the ability to obstruct the visible region. In each iteration, we add the obstacle that has the greatest effect on the visibility of T, i.e., the one that obstructs the angular size of T the most. Hence, the visible region from a query point q reduces more rapidly in QV than in QD. As QV retrieves obstacle by computing visible region, it retrieves less obstacles than QD but results in higher computational cost than QD. As in Fig. 7, at the first step, the query point of the maximum visibility is q_2. Obstacle o_1 is the nearest obstacle of q_2 and o_3 affects the visible region of q_2 the most. Here o_1 is retrieved as the first obstacle according to the QD approach, whereas o_3 is retrieved as the first obstacle according to the QV approach.

TABLE II: The steps in QD approach

Obstacle	VQ	LO	**Visible region**
-	$(q_2, 155)$	$\{R_1\}$	$\{aq_2d\}$
	$(q_1, 150)$	$\{R_1\}$	$\{aq_1c\}$
	$(q_4, 145)$	$\{R_3\}$	$\{aq_4c\}$
	$(q_5, 130)$	$\{R_2, R_3\}$	$\{bq_5d\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_1	$(q_2, 147)$	$\{o_3\}$	$\{aq_2x_1\}$
	$(q_4, 145)$	$\{R_3\}$	$\{aq_4c\}$
	$(q_5, 130)$	$\{R_2, R_3\}$	$\{bq_5d\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_3	$(q_4, 145)$	$\{o_5, o_6\}$	$\{aq_4c\}$
	$(q_5, 130)$	$\{R_2\}$	$\{bq_5d\}$
	$(q_2, 83)$	$\{\}$	$\{x_3q_2x_1\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_5	$(q_5, 130)$	$\{o_7, o_8, o_9\}$	$\{bq_5d\}$
	$(q_4, 117)$	$\{o_6\}$	$\{aq_4c\}$
	$(q_2, 83)$	$\{\}$	$\{x_3q_2x_1\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_7	$(q_5, 117)$	$\{o_8, o_9\}$	$\{bq_5x_6\}$
	$(q_4, 117)$	$\{o_6\}$	$\{aq_4x_4, x_5q_4c\}$
	$(q_2, 83)$	$\{\}$	$\{x_3q_2x_1\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_8	$(q_4, 117)$	$\{o_6\}$	$\{aq_4x_4, x_5q_4c\}$
	$(q_2, 83)$	$\{\}$	$\{x_3q_2x_1\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_5, 70)$	$\{o_9\}$	$\{x_7q_5x_6\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$
o_6	$(q_4, 117)$	$\{\}$	$\{aq_4x_4, x_5q_4c\}$
	$(q_2, 83)$	$\{\}$	$\{x_3q_2x_1\}$
	$(q_1, 77)$	$\{o_2\}$	$\{x_2q_1c\}$
	$(q_5, 70)$	$\{o_9, R_3\}$	$\{x_7q_5x_6\}$
	$(q_3, 70)$	$\{\}$	$\{aq_3b\}$

C. Target Centric Distance Based Approach (TD)

Both of the aforementioned approaches find the result by expanding the search space from the current query point. An immediate alternative can be incrementally expanding the search space from T (i.e., retrieving obstacles in order of their increasing distances from T). We call this approach *target centric distance based approach* (TD). This approach is efficient when the number of obstacles is relatively small compared to the number of query points and the obstacles are closely oriented to T. In this approach, at first the visibility of the target from each query point is calculated considering no obstacle. A priority queue of query points is maintained according to the visibility metric. Another priority queue of obstacles is maintained according to their non-decreasing distances from T. At first o_2, the nearest obstacle from T is retrieved (Fig. 8). Then each query point q_i whose visible region is affected by o_2, that is $o_2 \in LO_i$, is considered for evaluation. The priority queue is adjusted accordingly and o_2 is removed from the corresponding LO_i. Similarly, other obstacles are retrieved and the visibility is updated.

When the priority queues of obstacles of top k query points are empty, visibility from these points will not decrease for any other obstacle. Moreover, no other point can be candidate for the query result, because for each such point, its current visibility is already less than or equal to the visibility of the top k points. Hence, this approach terminates at this stage and returns the top k points as the query result.

Fig. 8 demonstrates the TD approach using the previous example. After o_8 is retrieved in step 8, the query point q_4 is the top element in the priority queue. Moreover, the priority queue of obstacles of q_4 is empty at this stage. So TD returns q_4 as the query result.

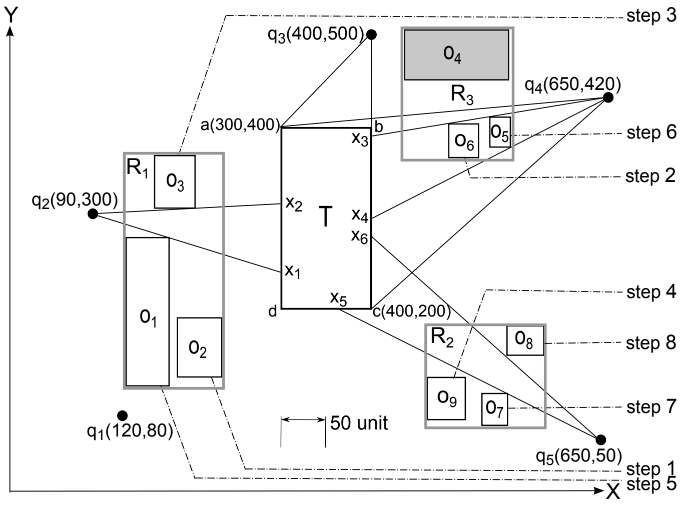


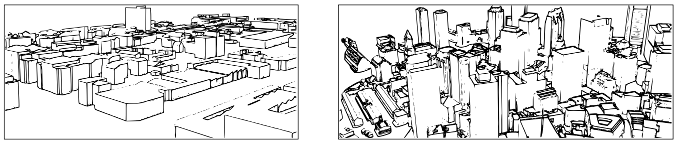
Fig. 8: The steps of the TD approach

V. Experimental Evaluation

In this section, we have evaluated the performances of our proposed algorithms for processing *k-Maximum Visibility* (*k*MV) queries with two real data sets. Specifically, we have considered our three proposed solutions: *query centric distance based approach* (QD), *query centric visible region based approach* (QV), and *target centric distance based approach* (TD). We have also compared these three approaches with the baseline approach, which is described as the *straightforward* approach in the previous section. Specifically, the straightforward approach calculates visibility for each query point by considering all obstacles in the corresponding visible region and sort these points to obtain the *k*MV results. The algorithms are implemented in C++ and the experiments are conducted on a core i5 2.53 GHz PC with 6GB RAM, running Microsoft Windows 7.

A. Experimental Setup:

Our experiments are based on two real data sets: (1) British[1] represents 5985 data objects obtained from British ordnance survey[2] and (2) Boston[3] represents 130,043 data objects in Boston downtown. In both datasets, objects are represented as 3D rectangles which are used as obstacles in our experiments. They are depicted in Table III. For both data sets, we normalize the data space into a span of $10,000 \times 10,000$ square units. For the 2D case, the datasets are normalized by considering the z-axis value as 0. The query points are generated based on *Uniform* (U) distribution and *Zipf* (Z) distribution over the search space. The coordinate of each query point in U is generated uniformly along each dimension, and that of each point in Z is generated according to *Zipf* distribution with skew coefficient $\alpha = 1$. A random object from the data object set is selected as the target.



(a) British ordnance survey (b) Boston downtown

Fig. 9: Real datasets used in the experiments

TABLE III: Description of datasets

Dataset	Cardinality	Description
British	5985	British Ordnance Survey
Boston	130043	Boston Downtown

All obstacles are indexed by an R*-tree, with the disk page size fixed at 1KB. For each query, n_q query points are generated and distributed into a query space of a given area A_Q. The experiments investigate the performance of the proposed solutions by varying three parameters: number n_q of candidate locations or query points, number k of query points to be returned as the answer, and the query space area A_Q as the percentage of total area. The default value and range of each parameter are listed in Table IV. In each experiment we have varied only one parameter while the other parameters are fixed at their default values. The performance metrics used in our study are (i) the I/O cost (number of pages accessed), (ii) the total processing time (execution time of the algorithm), and (iii) the number of pruned obstacles (the number of obstacles that are not accessed from the R*-tree or not considered for visibility calculation during execution). For each experiment we have evaluated our three proposed algorithms for 100 groups of query points keeping the target fixed and reported their average performance. In Section V-D we vary the target for each group of query points and measure the effect of varying the parameters. Finally in Section V-E we compare our proposed methods with the straightforward approach.

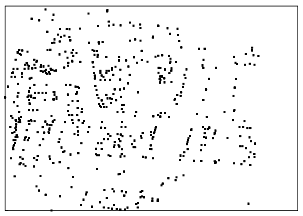
B. Performance in 2D:

1) Effect of k: In this experiment, we vary the value of k as 1, 2, 4, 8, and 16 and measure the I/O cost, the total processing

[1]http://www.citygml.org/index.php?id=1539

[2]http://www.ordnancesurvey.co.uk/oswebsite/indexA.html

[3]http://www.bostonredevelopmentauthority.org/BRA_3D_Models/3D-download.html


(a) British ordnance survey

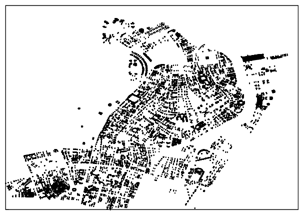

(b) Boston downtown

Fig. 10: Dataset distribution

TABLE IV: Parameters

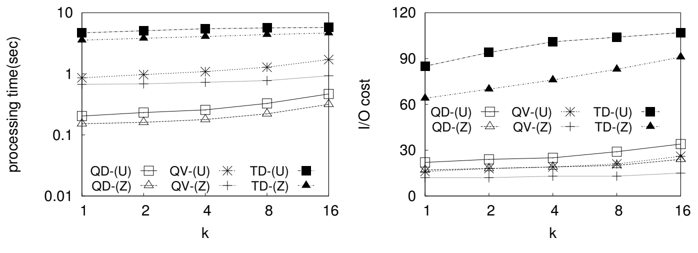
Parameter	Range	Default
k	1, 2, 4, 8, 16	4
n_q	8, 16, 32, 64, 128	32
A_Q	0.05, 0.10, 0.15, 0.2, 0.25	0.15

time, and the number of pruned obstacles. Fig. 11(a)-(c) and Fig. 11(d)-(f) show the results for British and Boston datasets, respectively. As the value of k increases the required number of visibility computation increases too. Thus the CPU cost and the I/O cost increase for all three approaches in both datasets. As mentioned earlier, QD and QV retrieve obstacles based on their increasing order of distances and their effect on current visible region respectively. But a closer obstacle is more likely to affect the visible region than a farther obstacle. Hence the obstacle sets affecting the visibility of target overlap in QV and QD and no large variation is observed in the I/O cost and the number of pruned obstacles between these two approaches.

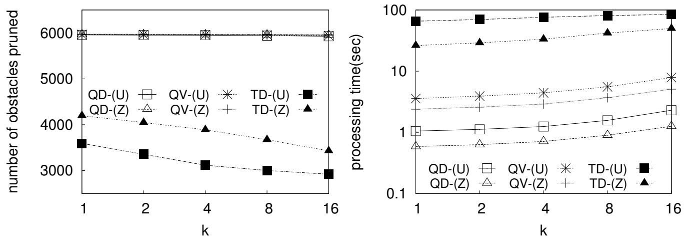
For British dataset, on average QD is approximately 4 times faster than QV and 19 times faster than TD for the *Uniform* distribution of query points. The average I/O cost of TD is about 5 times higher than the average I/O cost of both QV and QD. With the increase of k, the I/O costs for QD and QV increase slowly. Thus the number of the pruned obstacles in both QD and QV slightly vary with the increase of k and are approximately twice than that of TD. In case of TD approach, for a higher value of k we need to consider more obstacles from the target. So, with the increase of k, the number of accessed obstacles increases, thereby the number of pruned obstacles decreases. We can observe similar results for the *Zipf* distribution of query points.

In case of Boston dataset, on average QD is 4 times faster than QV and 55 times faster than TD for the *Uniform* distribution of query points. The average I/O cost of TD is approximately 10 times higher than the average I/O cost of QD and QV. Like the British dataset, the number of pruned obstacles in QD and QV vary slightly with the increase of k, and both of them prune 50% more obstacles than TD on average. The *Zipf* distribution of query points yields similar results.

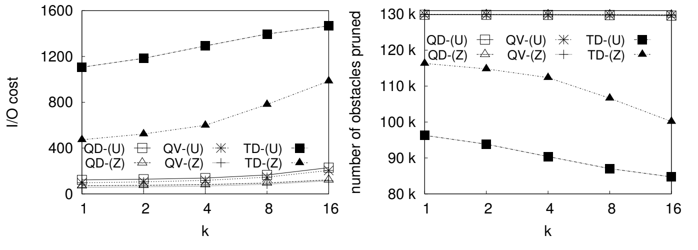
Although *pruned obstacles* refer to either the obstacles that are not accessed from the R*-tree or the obstacles that are not considered for visibility calculation, while we vary k, the number of pruned obstacles are found to be inversely proportional to the I/O cost. This holds for other cases too where we vary the number n_q of query points or query area



(a) (b)



(c) (d)



(e) (f)

Fig. 11: The affect of varying k for 2D British (a-c) and Boston (d-f) datasets

A_Q. As a result, for the brevity of the presentation we omit the graphs in the rest of the paper describing effects of varying k, n_q, and A_Q on the number of pruned obstacles.

2) Effect of n_q: In this set of experiments, we vary the value of n_q as 8, 16, 32, 64, and 128. Fig. 12(a)-(b) and Fig. 12(c)-(d) show the results for British and Boston datasets, respectively. In general, as n_q increases the total processing time and the I/O cost increase for all methods in both datasets. For both QD and QV the I/O cost slightly varies for different values of n_q, which is also the case while we vary k. In case of British dataset, on average QD is 4 times faster than QV and 19 times faster than TD for the *Uniform* distribution of query points. The average I/O cost of TD is approximately 5 times higher than that of QD and QV. *Zipf* distribution of query points yields similar results. Results obtained from Boston dataset are similar to the results of British dataset. The I/O cost remains almost constant with the increase of n_q, as the set of obstacles obstructing the target do not vary with the variation of n_q. However as the total visibility calculation increases for the increase of n_q, the processing time increases.

3) Effect of A_Q: We vary the query area A_Q as 5%, 10%, 15%, 20%, and 25% of the total data space. The results are presented in Fig. 13(a)-(b) and Fig. 13(c)-(d) for British and Boston datasets respectively. For both QD and QV with the increase of A_Q, the I/O cost increases slowly. Although QD considers obstacles based on a distance metric and QV con-

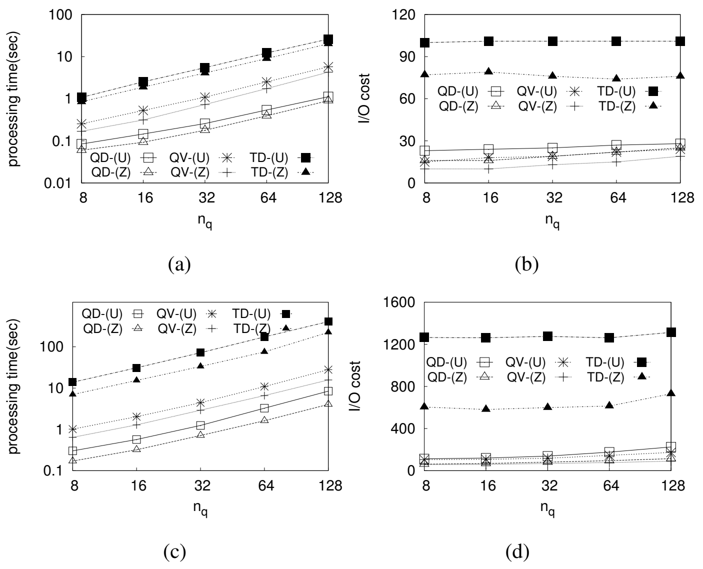


Fig. 12: The effect of varying n_q for 2D British (a-b) and Boston (c-d) datasets

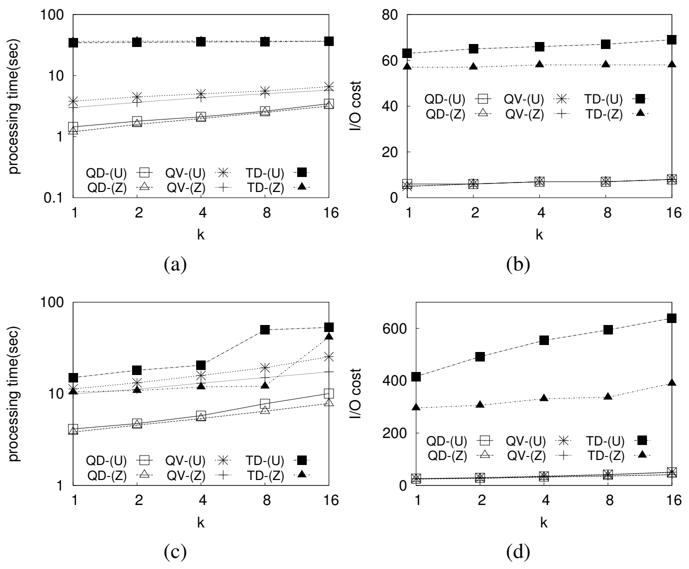


Fig. 14: The effect of varying k for 3D British (a-b) and Boston (c-d) datasets

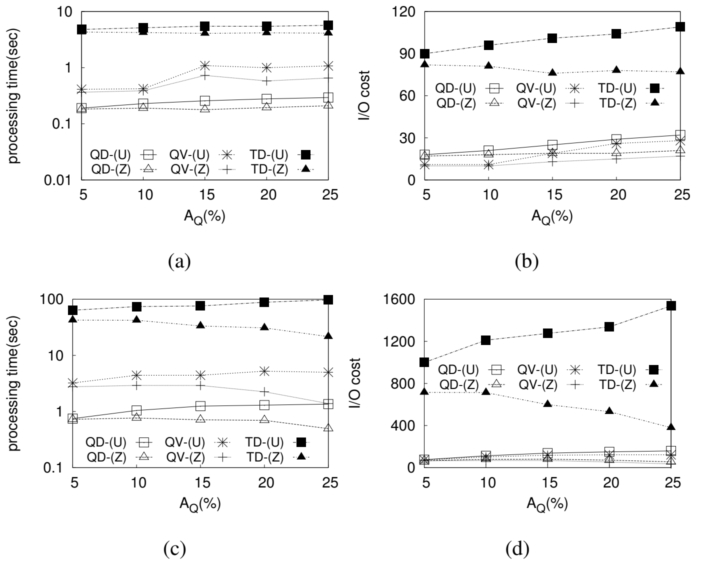


Fig. 13: The effect of varying A_Q for 2D British (a-b) and Boston (c-d) datasets

siders obstacles based on a visibility metric, both approaches perform similarly in terms of the I/O cost.

For British dataset, as A_Q increases the total processing time and the I/O cost increase in each of three methods for the *Uniform* distribution of query points. On average QD is 3 times faster than QV and 22 times faster than TD. The average I/O cost of TD is 6 times higher than that of QD and QV approaches. Similar results are found for QD and QV approaches using the *Zipf* distribution of the query points. In case of TD, we find that the I/O cost for *Zipf* distribution decreases with the increase of A_Q. This is because, in the *Zipf* distribution query points are clustered and thus the visibility of the target from these points are affected by a set of closely oriented obstacles. So when compared to *Uniform* distribution, *Zipf* distribution accesses fewer number of obstacles.

For Boston dataset, QD is 4 times faster than QV and 71 times faster than TD for the *Uniform* distribution of query points. In terms of the I/O cost, both QD and QV are 12 times faster than TD. The results vary in case of the *Zipf* distribution. As shown in Fig. 10, the objects in Boston dataset are clustered. With the increase of A_Q, the query points get clustered too and are obstructed by closely oriented obstacles. So a large portion of the obstacle set can be discarded from consideration. Thus the required number of visibility computation decreases and the number of pruned obstacles increases. Hence, the processing time and the I/O cost reduce with the expansion of query space for all three approaches.

C. Performance in 3D:

1) Effect of k: In this experiment, we vary the value of k to measure the effect using two real 3D datasets as depicted in Fig. 14. With the increase of k, the number of visibility computation increases as we need to consider more candidate query points and retrieve more obstacles. Thus, both the processing time and the I/O cost increase with the increase of k in general. As the retrieved obstacle sets overlap in QD and QV, we find that the I/O cost for both QD and QV are almost same in 3D, which is similar to its 2D counterpart.

For British dataset, QD is 2 times faster than QV and 17 times faster than TD on average. In this case, the I/O cost of TD is approximately 10 times higher than that of QV and QD. Results obtained from Boston dataset are similar to the results of British dataset. For both datasets, the *Zipf* distribution of query points yields similar results.

2) Effect of n_q: The effect of varying the value of n_q is shown in Fig. 15 for both British and Boston datasets. In general, with the increase of n_q the total processing time and

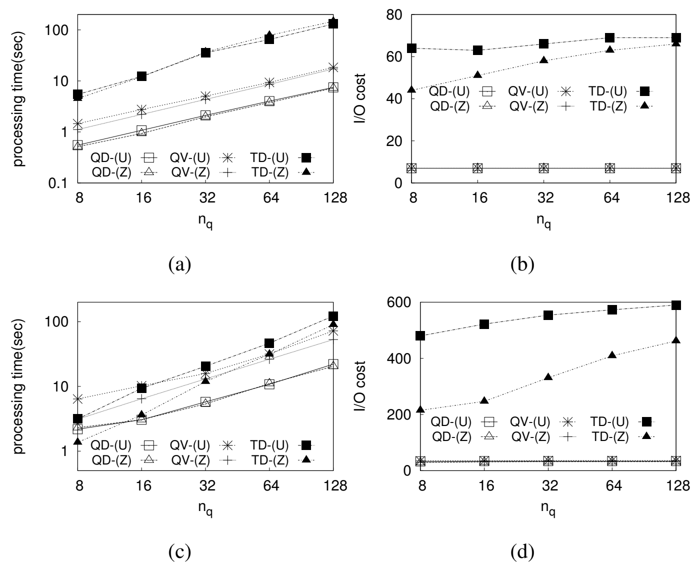


Fig. 15: The effect of varying n_q for 3D British (a-b) and Boston (c-d) datasets

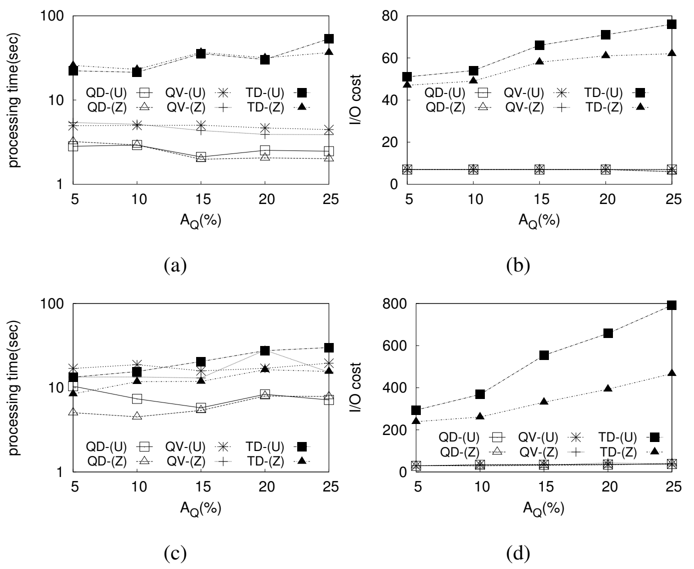


Fig. 16: The effect of varying A_Q for 3D British (a-b) and Boston (c-d) datasets

the I/O cost increase for all of the three methods. Like the 2D cases, both QD and QV perform similarly in terms of the I/O cost. Also, they both outperform TD in terms of processing time and I/O cost. In both datasets when the value of n_q increases, the processing time increases rapidly for all three approaches as the number of computation increases for larger value of n_q. On the other hand, with the increase of n_q the I/O cost increases slowly, as the set of obstacles obstructing the target remains almost constant in spite of the increase in n_q.

3) Effect of A_Q: In this experiment, the query area A_Q is varied between 5% and 25% of the total data space. Fig. 16(a)-(b) and Fig. 16(c)-(d) show the results for British and Boston datasets, respectively. Like the previous two experiments for 3D datasets, QD and QV show similar performance in terms of the I/O cost.

For British dataset (Fig. 16(a)-(b)), on average QD is approximately 2 times faster than QV and 13 times faster than TD for the *Uniform* distribution of query points. The average I/O cost of TD is approximately 9 times higher than that of both QD and QV approaches.

For Boston dataset (Fig. 16(c)-(d)), on average QD is about 2.5 times faster than QV and TD for the *Uniform* distribution of query points. As TD accesses obstacles starting from the target, when the query space expands TD accesses more obstacles for visibility computation than QD and QV. Hence the I/O cost increases in TD. The average I/O cost of TD is 15 times higher than the average I/O cost of QD and QV. For both datasets, the *Zipf* distribution of query points shows similar results to *Uniform* distribution of query points.

D. Effect of Target Variation:

In all of the above experiments we have kept the target fixed to see the effect of varying k, n_Q, and A_Q. In this set of experiments we vary the target for each group of query points

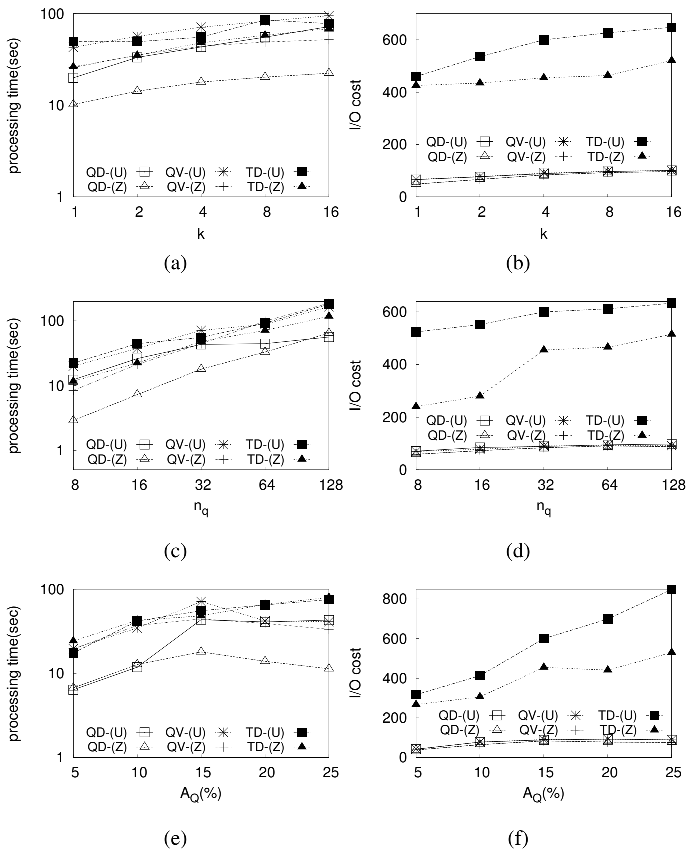


Fig. 17: The effect of random targets for 3D Boston dataset: varying k (a-b), n_q (c-d), and A_Q (e-f)

TABLE V: Comparison with the baseline approach

	British dataset		Boston dataset	
	Time(sec)	I/O	Time(sec)	I/O
Straightforward	34.600425	62	36.7561	590
QD	2.045	7	5.576	33
QV	4.6857	7	14.4613	34
TD	36.18674	62	16.1952	442

and measure the performance in Boston dataset (the larger dataset). As the value of k is varied as 1, 2, 4, 8, and 16, the processing time and I/O cost increase (Fig. 17(a)-(b)). Similar results are also obtained with the variation of n_q (Fig. 17(c)-(d)) and query area A_Q (Fig. 17(e)-(f)). The I/O cost of TD is 6 times higher than the I/O cost of both QD and QV on average. QD is approximately 2.2 times faster than QV and TD in terms of the total processing time. Thus the results obtained by varying target match with the results obtained using a fixed target.

E. Comparison with the Baseline Approach

We have compared our three proposed approaches with the *straightforward* approach, which considers every obstacle in the visible region, using the default values of k, n_q, and A_Q for 3D British and Boston datasets (see Table V).

For British dataset, QD is 17 times faster and requires 9 times less I/Os than the straightforward approach on average. QV is found to be 7 times faster and takes 9 times less I/Os than the straightforward approach on average. Note that since TD is designed for the environment where the number of obstacles is relatively small compared to the number of query points, which is not often the case in real data sets, the performance of TD is found to be similar with that of the straightforward approach for British dataset.

The results of Boston dataset show that on average QD is 6.6 times faster and takes 18 times less I/Os than the straightforward approach. We have found that on average QV is 2.5 times faster and takes 18 times less I/Os than that of the straightforward approach. TD approach is 2.3 times faster and takes 33% less I/Os than that of the straightforward approach.

VI. Conclusion

In this paper, we have introduced a novel form of spatial queries, called the *Maximum Visibility* (MV) query that has several applications including finding one or more locations that provide maximum visibility of a firework and choosing points inside abdomen to insert camera to operate on an internal organ. To efficiently process the query, we have proposed three different approaches: *query centric distance based approach* (QD), *query centric visible region based approach* (QV), and *target centric distance based approach* (TD). Starting from a query point, the QD and QV approaches retrieve obstacles incrementally based on their distances from the query point and based on their effects on visible region, respectively. On the contrary, TD approach retrieves obstacles incrementally based on their distances from the target. We have conducted an extensive experiment on 2D and 3D real datasets. The I/O efficiency of our proposed solutions is demonstrated by the pruning capability of QV and QD, which needs to consider only a small number of obstacles to produce kMV results in comparison to the baseline approach.

In case of 2D datasets, QD is approximately 3.7 times faster than QV and 37 times faster than TD on average. The average I/O cost of TD is 8 times higher than that of QV and QD. In case of 3D datasets, QD is found to be about 2.4 times faster than QV and 9.25 times faster than TD on average. The average I/O cost of TD is nearly 10.5 times higher than that of QV and QD. In summary, both QV and QD access fewer number of obstacles and yield similar I/O cost. However, QD outperforms QV in terms of total processing time, as in QV the number of visibility computation is greater than QD.

In future, we aim to extend our work for moving objects, e.g., the moving target or query points. We will also develop techniques to process the aggregate MV query that finds the combined maximum visibility for a group.

References

[1] A. R. Zarei and M. Ghodsi, "Efficient computation of query point visibility in polygons with holes," in *SCG*, 2005, pp. 6–8.

[2] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai, "Visibility of disjoint polygons," in *Algorithmica*, 1986, pp. 49–63.

[3] S. Suri and J. O'Rourke, "Worst-case optimal algorithms for constructing visibility polygons with holes." in *Symposium on Computational Geometry*, 1986, pp. 14–23.

[4] P. J. Heffernan and J. S. B. Mitchell, "An optimal algorithm for computing visibility in the plane." in *WADS*, 1991, pp. 437–448.

[5] B. Ben-Moshe, O. Hall-Holt, M. J. Katz, and J. S. B. Mitchell, "Computing the visibility graph of points within a polygon," in *SCG*, 2004, pp. 27–35.

[6] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai, "Visibility-polygon search and Euclidean shortest paths," in *FOCS*, 1985, pp. 155–164.

[7] L. Shou, Z. Huang, and K.-L. Tan, "HDoV-tree: The structure, the storage, the speed." in *ICDE*, 2003, pp. 557–568.

[8] C. Erikson, D. Manocha, and W. V. B. III, "HLODs for faster display of large static and dynamic environments." in *SI3D*, 2001, pp. 111–120.

[9] M. Guthe, P. Borodin, À. Balàzs, and R. Klein, "Real-time appearance preserving out-of-core rendering with shadows." in *Rendering Techniques*, 2004, pp. 69–80.

[10] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu, "Spatial queries in the presence of obstacles," in *EDTB*, 2004, pp. 366–384.

[11] C. Xia, D. Hsu, and A. K. H. Tung, "A fast filter for obstructed nearest neighbor queries," in *BNCOD*, 2004, pp. 203–215.

[12] S. Nutanong, E. Tanin, and R. Zhang, "Visible nearest neighbor queries." in *DASFAA*, 2007, pp. 876–883.

[13] ——, "Incremental evaluation of visible nearest neighbor queries," *TKDE*, vol. 22, pp. 665–681, 2010.

[14] Y. Gao and B. Zheng, "Continuous obstructed nearest neighbor queries in spatial databases," in *SIGMOD*, 2009, pp. 577–590.

[15] Y. Gao, B. Zheng, W.-C. Lee, and G. Chen, "Continuous visible nearest neighbor queries," in *EDBT*, 2009, pp. 144–155.

[16] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *VLDB*, 2002, pp. 287–298.

[17] K. Morling, *Geometric and Engineering Drawing*, 3rd ed. Routledge, 2012.

[18] M. A. Seeds and D. E. Backman, *Stars and Galaxies*, 7th ed. Brooks Cole., 2010.

[19] A. W. Paeth, *Graphics Gems V*. Academic Press, 1995.

[20] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331.

[21] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.