Visibility Color Map for a Fixed or Moving Target in Spatial Databases

Ishat E. Rabban[1(✉)], Kaysar Abdullah[1], Mohammed Eunus Ali[1], and Muhammad Aamir Cheema[2]

[1] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
{ieranik,kaysar,eunus}@cse.buet.ac.bd
[2] Faculty of Information Technology, Monash University, Clayton, Australia
aamir.cheema@monash.edu

Abstract. The widespread availability of 3D city models enables us to answer a wide range of spatial visibility queries in the presence of obstacles (e.g., buildings). Example queries include *"what is the best position for placing a billboard in a city?"* or *"which hotel gives the best view of the city skyline?"*. These queries require computing and differentiating the visibility of a target object from each viewpoint of the surrounding spe. A recent approach models the visibility of a *fixed* target object from the surrounding area with a visibility color map (*VCM*), where each point in the space is assigned a color value denoting the visibility measure of the target. In the proposed *VCM*, a viewpoint is simply discarded (i.e., considered as non-visible) if an obstacle even slightly blocks the view of the target from the viewpoint, which restricts its applicability for a wide range of applications. To alleviate this limitation, in this paper, we propose a scalable, efficient and comprehensive solution to construct a *VCM* for a *fixed* target that considers the *partial* visibility of the target from viewpoints. More importantly, our proposed data structures for the fixed target support incremental updates of the *VCM* if the target moves to near-by positions. Our experimental results show that our approach is orders of magnitude faster than the straightforward approach.

1 Introduction

3D city models are increasingly available through popular mapping services such as Google Maps, Google Earth and OpenStreetMap. We envision that these 3D datasets will provide a new platform for answering many real-life user queries, e.g., visibility queries in the presence of 3D obstacles, that form the basis of a large class of location based applications. For example, an advertisement company may want to check the visibility of its billboard from the surrounding areas before deciding on billboard's position; a tourist may want to check visibility of beautiful city skylines from available apartments; and a security company may want to find the suitable positions for surveillance cameras.

All of the above applications require computing and differentiating the visibility of (from) a target object from (of) the surrounding area. For example,

C. Claramunt et al. (Eds.): SSTD 2015, LNCS 9239, pp. 197–215, 2015.
DOI: 10.1007/978-3-319-22363-6_11

a target billboard may be more visible from one location than another due to different factors such as distance, viewing angle and obstacles. Also, a billboard may be seen from many viewpoints but is readable only from viewpoints closer to the target. Thus, our target applications require modeling the visibility as a continuous notion, i.e., one needs to compute the visibility of the target for every viewpoint in the space. In this paper, we propose efficient techniques to compute the visibility of a target object from the surrounding continuous space, which we call a visibility color map (*VCM*).

A *VCM* is a surface color map, where every viewpoint in a 3D space is assigned a color value denoting the *visibility measure* of the target from that viewpoint. In a recent work [1], Choudhury et al. proposed a technique to compute a *VCM* for a fixed target. Two major limitations of this technique are as follows: (i) They do not take the partial visibility into account, i.e., a viewpoint is simply discarded (i.e., considered as non-visible) if an obstacle even slightly blocks the view of the target from that viewpoint. For example, if only a small part of a billboard cannot be seen from a viewpoint, the viewpoint is declared as non-visible, though a major part of the billboard is readable from the viewpoint. Moreover, in a real 3D city environment, since many viewpoints are partially obstructed due to huge number of obstacles, only a small portion of the viewpoints surrounding the target constitute the *VCM*, which is not desirable for real life applications. (ii) They do not consider the case of a moving target, and thus a slight change of the target's position invalidates the entire *VCM*.

There is no straightforward way to incorporate the partial visibility and moving target into the existing technique due to the following reasons. First, since the proposed technique in [1] uses simple tangents between extreme points of an obstacle and the target, it cannot be converted to assess the partial visibility while computing the *VCM*. In this paper, we take the partial visibility into account, which is the correct form of the *VCM* for a fixed target and is a much harder problem with wider acceptability than [1]. Second, if the position of the target changes, the entire *VCM* needs to be reconstructed as the proposed data structure in [1] does not support incremental updates of the *VCM*.

To alleviate the above limitations, in this paper, we propose an efficient technique to construct the *VCM* for both fixed and moving target using real datasets comprising a large number of obstacles. One key idea of our approach is to identify the *potentially visible set* (*PVS*) of obstacles from the large obstacle set, by removing obstacles that cannot affect the construction of the *VCM*. To find the *PVS*, we adopt the concept of projection from computer graphics [2] and make it scalable and workable for a large number of obstacles indexed using an R-tree [3] in the database. After finding the *PVS*, we determine the *visibility state*s of several *boundary point*s on the target by considering the occlusion effect of the obstacles. The visibility state of a boundary point on the target represents which *cell*s are (not) visible. Finally, we add the effects of distance and angle between the target and each cell to compute the visibility of every cell.

To extend the above approach for a moving target, we rely on a precomputation based idea that assumes an extended buffer area around the target and computes the *PVS* and visibility states for the extended region. Once the

target moves to a near-by position, our proposed data structure is incrementally updated to generate the *VCM* for the new target position.

We have evaluated the performance of our solution with both real and synthetic 3D datasets. The experimental results show that our approach is on average 10^6 times faster than the straightforward approach.

In summary, we make the following contributions:

- We devise an effective algorithm to construct the *VCM* for a fixed target in the presence of a large set of obstacles considering the partial visibility of the target.
- We propose an efficient way to reconstruct the *VCM* for a moving target.
- We conduct experiments with real 3D datasets to demonstrate the effectiveness and efficiency of our solution.

2 Related Works

The notion of visibility is fundamental to various fields including computational geometry, computer graphics, urban planning, architecture and spatial databases. In this section we briefly discuss existing works on visibility.

2.1 Visibility in Computational Geometry and Computer Graphics

Visibility computation in computational geometry involves determining visibility graphs [4] and visibility polygons [5,6]. In computer graphics, a visibility map is a graph describing a view of the scene including its topology. Various methods for constructing a visibility map for a fixed [7–9] and moving [10] viewpoint have been developed. In all the above approaches, visibility is defined from a point source and consequently a binary notion, i.e. a point in the space is declared as either visible or non-visible from the viewpoint.

Visibility of/from an extended region, i.e. from-region visibility, was studied by Kim et al. [11]. His method determines the subset of the whole space which is completely visible from a region, but does not handle the case of partial visibility. Works done by Durand et al. [2] and Koltun et al. [12,13] focus on determining the set of obstacles visible from an extended region. However, they do not provide any measure of the visibility of the region from the space.

2.2 Visibility in Urban Planning and Architecture

Visibility related problems are actively studied in the fields of urban planning and architecture. Relevant contributions include [14,15]. These approaches treat visibility as a binary notion and are only applicable to cases where the number of obstacles is small enough to fit into the main memory. Urban planners and architects make use of software systems to visualize and render 3D data, such as Google Sketchup [16], AutoCAD [17] and Maya [18]. These softwares do not provide any functionality for quantifying visibility of a 3D object. By incorporating our techniques to construct the *VCM*, these applications can be equipped to answer many realistic visibility queries which require quantification of the visibility of an extended target object.

2.3 Visibility in Spatial Queries

Visibility problems studied in context of spatial databases include nearest neighbor queries [19–21] and maximum visibility queries [22,23]. The variants of nearest neighbor queries find the nearest object in an obstructed scene from a single query point or all points on a line segment, where results are ranked according to the distances from the query point. Maximum visibility query finds a subset of query points that provides the best view of an extended target.

Construction of the *VCM* for the entire space for a fixed position of the target object is studied by Choudhury et. al. [1]. But they do not handle the case of partial visibility and their solution is not applicable for a moving target as we outline in the introduction. In this paper, we devise a method to construct the *VCM* of the entire dataspace for a moving target, which also handles the case of partial visibility in the presence of a large set of obstacles.

3 Problem Formulation

To construct a *VCM*, we need to produce a color map of the dataspace where each point in the space is assigned a value that corresponds to the visibility measure of the target from that point. We formally define the *VCM* as follows:

Definition 1 *VCM. Given a d-dimensional dataspace R^d (d=2 or 3) and a set O of obstacles in the dataspace, the VCM is a color map, where for each point p in R^d, there exists a visibility color v_p in [0,1]. The color v_p corresponds to the visibility of a given target object T from p. Here, higher value of v_p corresponds to higher visibility of T from p and vice versa.*

In an earlier attempt [1] to construct the *VCM* for a fixed target, a point gets a nonzero color if the target is *entirely* visible from that point. However, the target can be partially visible from a point because of obstruction by the obstacles. In this paper, while assigning visibility color to a point, we consider the case of partial visibility by determining what portion of the target is visible from that point.

We also address the problem of reconstructing the *VCM* efficiently as the position of the target changes, i.e., for a moving target. Let VCM_p denote the *VCM* when the target is at position p. We formulate a method to incrementally reconstruct $VCM_{p'}$ efficiently for all $p' \in P$, where P is the set of all candidate positions.

4 Preliminaries

In this section we discuss some basic ideas on which our solution is built upon. First we describe the factors affecting the visibility and then we discuss the partitioning scheme of the dataspace that we have used in our solution. The examples illustrated in this section are in 2D, which can be easily extended for a 3D scenario.

4.1 Factors Affecting Visibility Color

For a particular position of the target, the visibility measure (or, color), v_p, of a viewpoint p is obtained by considering two different visibility measures: *orientation based visibility*, v_p^{or}, and *obstruction based visibility*, v_p^{ob}. Both of these values are in the range of *[0,1]*. After computing v_p^{or} and v_p^{ob}, we can estimate the visibility measure of a viewpoint p as $v_p = v_p^{or} * v_p^{ob}$.

Orientation based visibility captures the effect of both distance and the angle between the target T and the viewpoint, and is measured as the visual angle [24]. The visual angle, α_p, is the imposed angle by T at p as shown in Fig. 1(a).

To compute the obstruction based visibility measure, v_p^{ob}, for a viewpoint p, we consider a number of equally spaced boundary points on the surface of T. Let B_p^T be the set of boundary points which are visible from a point p in the absence of obstacles and B_p^O be the set of boundary points which are visible from p in the presence of all obstacles. Then $v_p^{ob} = |B_p^O|/|B_p^T|$. Here, the notation $|.|$ stands for cardinality of a set.

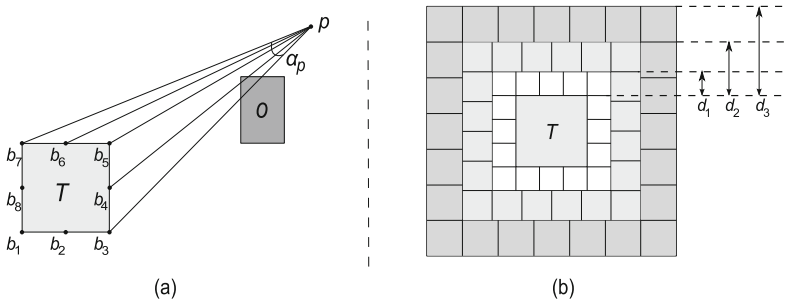

Fig. 1. (a) Factors affecting visibility color of point p around target T. (b) A sample partitioning of space. First 3 equidistant stripes are shown in different shades of gray.

In the scenario described in Fig. 1(a), there are 8 boundary points $b_1, b_2, \ldots, b_8$ on target T in a 2D space and the obstacle set O contains one obstacle, o. Here $B_p^T = \{b_3, b_4, b_5, b_6, b_7\}$, $B_p^O = \{b_5, b_6, b_7\}$ and $v_p^{ob} = 3/5 = 0.6$. Thus, v_p^{ob} measures what portion of the target T is visible from p.

4.2 Partitioning into Cells

The d-dimensional data-space R^d consists of infinitely many points and assigning a visibility color, v_p, to each viewpoint p in R^d is a prohibitively expensive task. To address this problem, we partition the whole space into a finite number of *equi-visible cell*s as proposed by Choudhury et al. [1], and assign a single visibility color to each cell. The visibility color of cell c is denoted by v_c. Each cell is constructed in a way so that the deviation in visibility of T from the viewpoints inside a cell, measured as visual angle, is not visually perceivable.

To find equi-visible cells, we first partition the space based on distance and then based on the angle between the target and the viewpoints. All the viewpoints having a distance between d_{i-1} and d_i from the target, where $0 < i \leq k$ and k is a positive integer, constitute the i^{th} *equidistant stripe*, which is denoted by S_i. Here k is the number of equidistant stripes. Then each equidistant stripe is further divided into cells based on the angle between the viewpoints and the target. The detailed process of the partitioning is described in [1]. For simplicity, we consider each cell as a rectangular region as shown in Fig. 1(b).

5 A Straightforward Approach

In this section, we present a straightforward approach to construct the *VCM* for a fixed target. Let us consider a cell c with midpoint p. To determine visibility color, v_c, of c, we need to know the number of boundary points, which are visible from p in the presence of obstacles. To determine the visibility between p and a boundary point b, we check the line segment joining b and p against all obstacles. If no obstacle intersects that line segment, b is visible from p. Otherwise b is not visible from p. Finally we incorporate the effect of the distance and angle between p and the target. This straightforward process is expensive both in terms of I/O and computation, and not suitable for a moving target.

6 Our Approach

To construct the *VCM* for a target, we have to assess the occlusion effect of the obstacles. One of the main challenges to construct the *VCM* for a target is to deal with the huge obstacle set. Our strategy is to significantly reduce the number of obstacles by discarding those obstacles that do not affect the calculation of the *VCM*. This reduced obstacle set is called the *potentially visible set*, (*PVS*). To determine the *PVS*, we adopt a projection based idea of computer graphics [2] and propose additional adjustments to determine the occlusion effect of a large set of obstacles. We store the obstacles in an R-tree and perform a plane sweep algorithm to determine the combined occlusion effect of multiple obstacles. The process of determining the *PVS* is described in Sect. 6.1.

After determining the *PVS*, the next challenge is to efficiently compute the view of the target from each cell of the partitioned of dataspace. The visibility measure of each cell has two components: (i) orientation based visibility measure, which can be computed using simple equations as described in Sect. 4.1, (ii) obstruction based visibility measure, which needs to consider *what portion* of the target is visible from the cell in the presence of obstacles. As the next step of our algorithm, we determine the *visibility states*, indicating which cells are visible from a particular boundary point of the target (Sect. 6.2). Visibility states of all boundary points are then combined to measure the obstruction based visibility measure of each cell (Sect. 6.3). To compute the *VCM* for a fixed target, orientation and obstruction based visibility values are combined to obtain the visibility color value for every cell (Sect. 6.4). In Sect. 6.5, we discuss the construction of the *VCM* for a moving target.

6.1 Determining PVS

In this section, we formulate a methodology to prune out those obstacles which do not affect the calculation of the VCM. Thus we obtain a reduced set of obstacles, which we call the potentially visible set (PVS). To efficiently determine the PVS, we index all obstacles in an R-tree [3]. 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 (obstacles, in our case) are stored in leaf nodes. Before going to the details of determining PVS, we first discuss some terminologies related to visibility and projection.

Definition 2 ***Target-Obstacle Visibility.*** *Given a target T and a set of obstacles O, T and an obstacle $o \in O$ are defined to be visible to each other if and only if there exists a pair of points p_T on T and p_o on o such that no obstacle $o' \in O$ and $o' \neq o$, intersects or touches the line segment joining p_T and p_o.*

Visibility between the target and an obstacle is bidirectional, i.e., if the target is visible from a particular obstacle o, then o is visible from the target and vice versa. An obstacle cannot affect the computation of the VCM if it is not visible from the target. So we can ignore the obstacles which are not visible from the target and obtain a reduced set of obstacles, the PVS, which we denote by O_v.

To determine the PVS, we adopt a projection based idea proposed by Durand et al. [2]. This work employs a conservative occlusion culling technique combining occlusion effects of multiple obstacles on the target visibility. A plane sweep in each principal axis direction is performed to identify obstacles that are not visible from the target. We add further modifications to this approach so that it fits our purpose of dealing with a large obstacle set indexed in an *R-tree*.

Preliminaries of Projection. In this section we describe several key concepts regarding projection. For ease of explanation, we have assumed 2D scenario with axis aligned rectangular target and obstacles. However, our approach is applicable to any convex target and obstacles in 2D and 3D spaces. We also assume that the field of view (FOV) is 90 degree centering the positive X direction. Other directions along the principal axes can be treated similarly. In subsequent sections, we treat an R-tree node (i.e., an MBR) or an obstacle as an *object*.

First, we define the *near distance* and the *far distance*. The *near distance* and the *far distance* of an object are respectively the smallest and the largest of the x ordinate values of all points of the object. We denote the near distance and the far distance of an object o, by o_n and o_f respectively. Now we discuss the idea of projection of an object [2]. The *projection* of an object is computed on a projection plane in 3D (or, projection line in 2D) with respect to the target. If the projection plane is in between the target and the object, then the projection of the object is the union of all views from any point of the target. If the projection plane is in the opposite side of the object from the target then the projection of the object is the intersection of all views from any point of the target. The projection of an object o, on the projection line $x = l$, is denoted by P_o^l.

Now we describe the concepts of *aggregated projection* and *re-projection*. We denote the aggregated projection at the projection line $x = l$, by A_l. A_l reflects the combined occluding effect of all obstacles with far distances less than or equal to l, i.e., obstacles which are entirely in front of the sweep line $x = l$. As a result, A_l consists of several disjoint projections on $x = l$.

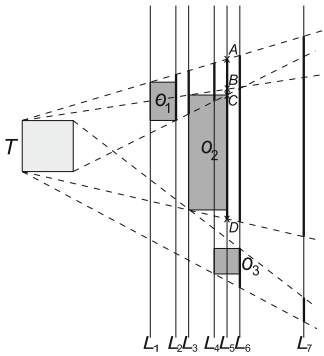

Fig. 2. Bold segments stand for aggregated projections.

The process of computing the aggregated projection is as follows. Initially, we set the aggregated projection as null, $A_{init} = \emptyset$. Then we incrementally update the aggregated projection as we encounter obstacles in the increasing order of their far distance.

Suppose we know the aggregated projection on line $x = l_p$, A_{l_p}, and we want to calculate A_{l_n}, where $l_p < l_n$. (Here subscripts p and n stand for *previous* and *next* respectively.) If no obstacle has far distance between l_p and l_n, then we obtain A_{l_n} by *re-projecting* A_{l_p} on the projection line $x = l_n$. Re-projecting an aggregated projection onto another projection plane involves projecting each disjoint projection of the aggregated projection separately onto the destination plane. Now assume that there is an obstacle o, with far distance l, where $l_p < l \leq l_n$. In such cases, first we calculate the union of P_o^l and the re-projection of A_{l_p} on $x = l$ and then re-project this combined projection on $x = l_n$. This method demonstrates that the aggregated projection needs to be recalculated only at the far distances of the obstacles.

The process of calculating aggregated projections is simulated in Fig. 2. Let, l_1, l_3 and l_4 be the near distances of o_1, o_2 and o_3 respectively, and l_2, l_5 and l_6 be the far distances of o_1, o_2 and o_3 respectively. In the figure, projection line $x = l_i$ is marked as L_i. Initially, $A_{init} = \emptyset$. We encounter the first far distance at $x = l_2$. As a result, A_{l_2} is $P_{o_1}^{l_2}$. On $x = l_5$, the re-projection of A_{l_2} on $x = l_5$ is AC and $P_{o_2}^{l_5}$ is BD. The union of AC and BD is AD. Thus, A_{l_5} is $\{AD\}$. At $x = l_6$, the re-projection of A_{l_5} on $x = l_6$ and $P_{o_3}^{l_6}$ are disjoint. Thus, A_{l_6} consists of two disjoint segments as shown in Fig. 2.

Algorithm 1. determinePVS(R,T)

```
1  begin
2      O_v, A_init, Q ⟵ ∅;
3      Q.push(R.root);
4      while Q ≠ ∅ do
5          o ⟵ Q.pop();
6          if o is entirely outside FOV then
7              continue;
8          l ⟵ o_n;
9          determine A_l;
10         if A_l completely spans FOV then
11             break;
12         if P_o^l ⊆ A_l then
13             continue;
14         if o is an MBR then
15             for c ∈ o.children() do
16                 Q.push(c);
17         else
18             O_v.push(o);
19     return O_v
```

The Algorithm. In this section, we formally describe the process of determining the *PVS* in the algorithm *determinePVS*. We retrieve objects from the R-tree in non-decreasing order of near distance (Fig. 3 shows an example R-tree for 8 obstacles in a 2D space). This retrieval process can be visualized as a line sweep over the 2D plane. A projection line (sweep line) perpendicular to the X axis is moved towards the positive X direction. Suppose it enters an object, o, at position $x = l$, i.e., $o_n = l$. Then the aggregated projection on the projection line $x = l$, A_l, is calculated, which reflects the combined occlusion effect of all obstacles entirely in front of the sweep line. If the projection of o on the sweep line, P_o^l, is a subset of A_l, then o is occluded by the obstacles in front of o, and consequently o is discarded (Lines 12–13). o is also discarded if o is entirely outside the field of view (Lines 6–7). If o cannot be discarded, we either declare o as a potentially visible obstacle (in case o is an obstacle, Lines 17–18) or mark o's children for later consideration (in case o is an *MBR*, Lines 14–16). The algorithm terminates when there are no more objects to process or the aggregated projection completely spans the *FOV* (Lines 10–11). Note that an obstacle, o, is discarded only if o is certainly not visible from the target, otherwise, o is considered as potentially visible.

In the algorithm *determinePVS*, movement of the sweep line is implemented by a priority queue, Q, which holds objects, i.e. *MBR*s and obstacles, in non-decreasing order of near distance. Other variables hold their usual meaning.

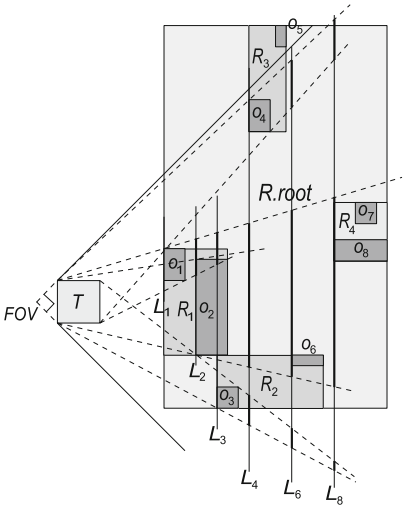

Fig. 3. The PVS computation with an R-tree. Near distance of obstacle o_i is l_i and the sweep line $x = l_i$ is represented by L_i. The bold segments on the L_is identify A_{l_i}.

In Line 9, we determine the aggregated projection at the current sweep line position, which can be calculated efficiently by considering only those obstacles that have far distances between the previous and current sweep line positions.

In Fig. 3, we present a simulation of the algorithm *determinePVS*. At projection line L_1, A_{l_1} is null. As a result, o_1 is added to O_v. On L_2, $P^{l_2}_{o_2}$ is not a subset of A_{l_2} and o_2 is declared as potentially visible. Similar is the case with o_3 and o_4. o_5 is entirely outside the field of view, and consequently discarded. o_6 is rejected because, at L_6, $P^{l_6}_{o_6}$ is a subset of A_{l_6}. The projection of the *MBR*, R_4 on L_8 is a subset of A_{d_8}. As a result, R_4, along with o_7 and o_8, is discarded. Finally, the algorithm returns the set $\{o_1, o_2, o_3, o_4\}$ as the *PVS*.

6.2 Determining Visibility State of a Point

The *visibility State* of a point, b, on the target indicates which cells in the partition are visible from b and which cells are not. The process of determining visibility state of b is equivalent to assigning a boolean value to each cell in the partition indicating whether or not the cell is visible from b. As mentioned before, to construct the *VCM*, we need to determine the visibility states of all the boundary points. In this section, we describe how to determine visibility state of a particular boundary point, b, assuming that the field of view along the positive X direction.

A cell c and a point b is *visible* to each other if and only if the line segment joining b and the midpoint of c is intersected or touched by no obstacle. We observe that the midpoints of cells in a particular equidistant stripe are situated on a straight line perpendicular to the X axis. The line joining the midpoints of the cells of S_i is called the i^{th} *midway line*, and denoted by M_i. Recall from

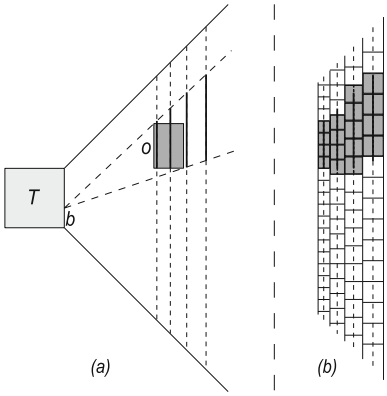

Fig. 4. (a) Aggregated near projections (bold segments) on successive mid-way lines (dotted lines). (b) Visibility state of b. Non-visible cells are darkened.

Sect. 4.2 that S_i stands for the i^{th} equidistant stripe. Let the equation of the line M_i be $x = m_i$.

As the midway lines are vertically aligned, we can reuse the idea of the projection based strategy, constructed in Sect. 6.1, for determining the occlusion effect of obstacles around the point b. Let us first consider the point b as the target (i.e., a point target) and the PVS, returned by the algorithm *determinePVS*, as the set of obstacles. For each midway line M_i, where $i = 1, 2, \ldots, k$, we calculate the occlusion effect of all obstacles situated partially (as we explain shortly) or entirely in front of M_i. We declare a cell as not visible from b, if the cell's midpoint is occluded and vice versa.

To accurately measure the visibility state of a point, we consider the occlusion effect of an obstacle at projection lines after the near distance of the obstacle. Thus we ensure that when the projection line is between the near distance and far distance of an obstacle, the occlusion effect of the portion of the obstacle in front of the projection line is taken into account. If the occlusion effect of all obstacles situated partially or entirely in front of the projection line is taken into account, the combined projection is defined as *aggregated near projection*. Aggregated near projection at projection line $x = l$ is denoted by A_l^n.

To determine the visibility state of a boundary point, aggregated near projections are determined on successive midway lines considering O_v as the set of obstacles. In Fig. 4, a sample scenario is illustrated, where visibility state of a boundary point b is determined in the presence of an obstacle, o. Figure 4(a) shows the aggregated near projections on consecutive midway lines and Fig. 4(b) marks the cells which are declared as non-visible from b, i.e., those cells which have midpoints on the aggregated near projections.

6.3 Visibility State of a Target

In the previous section, we compute the visibility state of a point on the target. In this section, we will combine the visibility states of all boundary points on the target to find the obstruction based visibility measure of the target from all equi-visible cells of the partitioned dataspace.

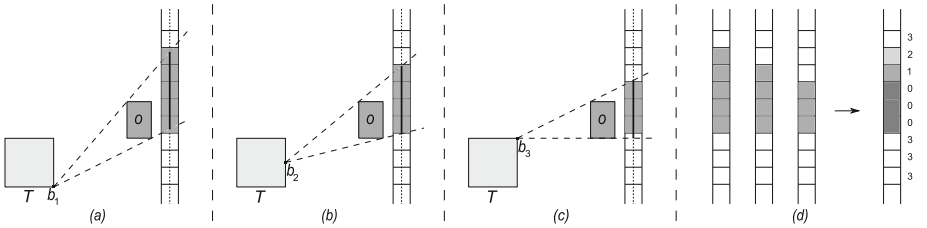

Fig. 5. Combining the visibility states of a particular equidistant stripe of 3 boundary points. Lighter shade of gray on the leftmost stripe corresponds to better view of T.

Let us consider three boundary points b_1, b_2, and b_3 on the positive X direction of the target T as shown in Fig. 5. We need to compute the visibility states of all cells on the positive X direction in the presence of an obstacle o. Figure 5(a), (b) and (c) show visibility states of a particular equidistant stripe for boundary points b_1, b_2 and b_3, respectively. We can see that 5, 4, and 3 cells of the equidistant stripe are not visible from boundary points b_1 (Fig. 5(a)), b_2 (Fig. 5(b)), and b_3 (Fig. 5(c)), respectively. If we combine these three visibility states, we obtain the obstruction based visibility measures of all cells on the equidistant stripe. Figure 5(d) shows the cells and corresponding numbers indicating how many boundary points are visible from those cells, which indicate the obstruction based visibility measure for the *VCM*.

6.4 VCM for a Fixed Target

Based on the above constructs, we summarize the process of *VCM* construction for a fixed target as follows. First, we divide the dataspace into a set of equi-visible cells. Second, for each cell c, we compute the orientation based visibility measure v_c^{or} based on the distance and angular placement of c's midpoint with respect to the target. Third, we compute the potentially visible set (PVS), O_v, by the projection based line-sweep along every axis-parallel direction. Fourth, we compute the visibility states of all the boundary points on the target by considering the occlusion effect of O_v on target visibility. Fifth, for each cell c, we compute the obstruction based visibility measure, v_c^{ob}, as $|B_c^O|/|B_c^T|$, where B_c^T is the set of boundary points that are visible from c in the absence of obstacles and B_c^O is the set of boundary points that are visible from c in the presence of obstacles. Finally, we estimate the visibility measure or the color value v_c of each cell c as $v_c^{or} * v_c^{ob}$, which constitutes the *VCM* for the target.

6.5 VCM for a Moving Target

In this section, we discuss our solution to construct the *VCM* for a moving target. The key concept of extending our method for a moving target is to pre-compute the *PVS* and visibility states for an extended buffer area around the target, and then incrementally update the *VCM* when the target moves to a near-by position. Thus we can avoid a large amount of repetitive computation and obtain the desired efficiency to construct the *VCM* for a moving target. Note that performing the pre-computation for an extended region around the target is consistent with realistic application scenarios, e.g., while placing a billboard, a user may want to check its visibility from the surrounding space. Our proposed method enables him to choose the right placement by constructing the *VCM* on-the-fly for different near-by positions of the billboard. In this paper, we consider a rectangular region that encloses the target as the buffer area. Defining the buffer region more meaningfully is in the scope of our future work.

We name the extended buffer region around the target as a *super target*, T_s and consider a number of equally spaced points in T_s as *candidate points*. We follow two pre-computation steps involving the super target and candidate points: (i) determine the *PVS* for T_s, and (ii) compute visibility states of all the candidate points by considering the *PVS* as the obstacle set.

After the pre-computation steps, we can construct the *VCM* by combining the visibility states of those candidate points that lie on the boundary of the target. When the target moves to a position inside the super target, we add the effects of visibility states of the new candidate points and remove the effects of candidate points that do not belong to the new target. We assume that the target is moved to a discrete position defined by a set of candidate points. However, if the target moves to a position where its boundary points do not superimpose with the candidate points, we approximate the given position of the target to the nearest discrete position defined by the candidate points, and construct the *VCM* accordingly. If a target moves outside the super target area, we need to calculate the new *PVS* and visibility states of the new candidate points.

The speedup in the construction of the *VCM* for a moving target comes from two sources. First, we calculate the *PVS* only once. If the target is entirely inside the super target, we can use the reduced obstacle set determined in Step (i) as the *PVS*. Second, one candidate point can be used to approximate targets at many positions. Consequently, we do not need to repeat the process of determining the *PVS* and the visibility states for any near-by positions of the target. Thus the cost of the preprocessing steps is amortized over all subsequent runs of the process for different positions of the target inside the super target.

7 Experimental Evaluation

We evaluate the performance of our proposed algorithm for constructing the *VCM* with real and synthetic datasets. At first, we compare our approach to construct the *VCM* for a fixed target with the exact straightforward approach

presented in Sect. 5. Then we evaluate the efficiency of our approach for constructing the *VCM* for a moving target. The algorithms are implemented in C++ and the experiments are conducted on a core i5 2.40 GHz PC with 3GB RAM, running Microsoft Windows 7.

7.1 Experimental Setup

We conduct experiments for two real 3D datasets: (1) British[1] dataset, representing 5985 data objects obtained from British ordnance survey[2] and (2) Boston[3] dataset, representing 130,043 data objects in Boston downtown. We also conduct experiments using synthetic datasets. We vary the synthetic dataset size using both *Uniform* and *Zipf* distributions of the obstacles. In all datasets, objects are represented as 3D rectangles that are used as obstacles in our experiments. All obstacles are indexed by an R-tree, with the disk page size fixed at 1KB. We vary several parameters to evaluate our solution. The range and default value of each parameter are listed in Table 1.

Table 1. Parameters

Parameter	Range	Default
Dataset	Real, Synthetic	Real
Angular Resolution	0.5, 1, 2, 4, 8	2
Number of Boundary Points	3^2, 4^2, 5^2, 6^2, 7^2	4^2
Length of Target	15, 30, 60, 120, 240	60
Dataset Size (Synthetic)	5K, 10K, 15K, 20K, 25K	

7.2 Performance Evaluation

We compare our approach to construct the *VCM* with the exact straightforward approach described in Sect. 5. The results of our approach deviate slightly from the results of the exact method, because we have used an approximation for ease of implementation. Recall from Sect. 6.2 that while determining the visibility states, we calculated the aggregated near projections to correctly assess the occlusion effect of an obstacle, i.e. we consider the portion of the obstacles partially or completely in front of the projection lines. But in our implementation, we have calculated the projection of each obstacle at its' near distance. We have treated this projection as the occlusion effect of the obstacle and removed the obstacle from further consideration. To formulate the error incurred by this approximation, let v_c^e denote the visibility color of cell c determined by the exact

[1] http://www.citygml.org/index.php?id=1539.
[2] http://www.ordnancesurvey.co.uk/oswebsite/indexA.html.
[3] http://www.bostonredevelopmentauthority.org.

approach and v_c^a denote the visibility color of cell c determined by our implementation. Then the error is given by the following formula:

$$error = \frac{\sum_{c \in C} |v_c^e - v_c^a|}{|C|}$$

The performance metrics used in our experiments include: (i) total processing time, (ii) I/O cost, i.e. the number of nodes retrieved from the R-tree, and (iii) error incurred by the approximation. For each experiment, we have evaluated the results for 20 random positions of the target and reported the average.

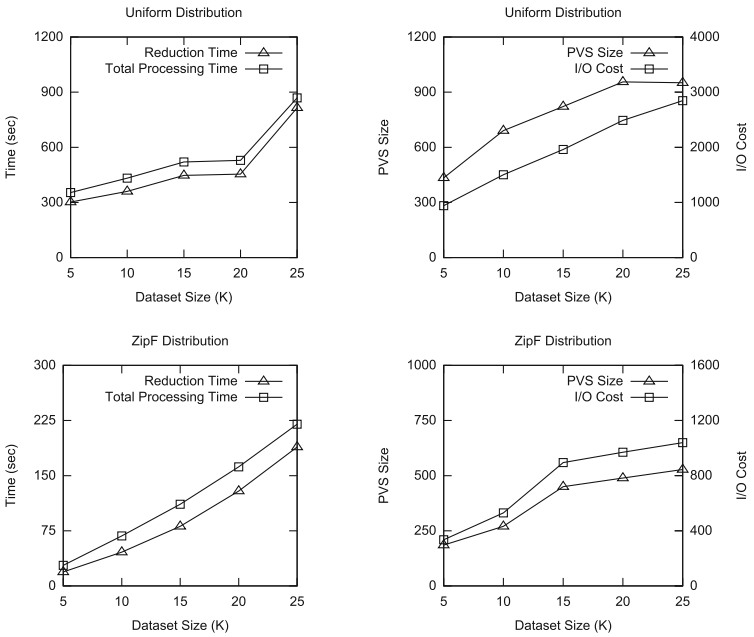

Fig. 6. Effect of dataset size on I/O performance.

7.3 Performance for Fixed Target

To evaluate the performance of our approach to construct the *VCM* for a fixed target, we vary the size of the target, dataset size, angular resolution and the number of boundary points independently and determine the total processing time, I/O cost and the incurred error. When one parameter is varied, the other parameters are kept at their default values.

Effect of Dataset Size. In this experiment we vary the dataset size from 5 K to 25 K using both Uniform and ZipF distributions and compare the performance of our solution with the straightforward exact approach. Table 2 shows the effect of

dataset size on total processing time and error. Our approach runs approximately 10^5 to 10^6 times faster than the straightforward approach. The rate of error is quite low and does not change significantly with varying dataset size. The total processing time, reduction time, *PVS* size and I/O cost of our approach is shown graphically in Fig. 6 to analyze the I/O performance of our solution. We see that reduction time is dominating the total processing time in both datasets.

Table 2. Effect of dataset size on total processing time (in seconds) and Error

Distribution	Uniform			ZipF		
Approach	Our	Naive	Error(%)	Our	Naive	Error(%)
5K	354.3	3.4e+7	2.66	28.3	4.1e+7	2.16
10K	431.5	9.6e+7	4.98	67.9	1.6e+8	3.21
15K	519.7	1.9e+8	3.50	111.1	2.9e+8	4.46
20K	528.5	2.8e+8	4.37	161.9	3.6e+8	4.23
25K	869.4	3.4e+8	4.40	219.5	4.2e+8	4.26

Increasing the dataset size results in an increase in reduction time and the size of the *PVS*, which in turn increase the cost of determining the visibility states. As a result, the total processing time and I/O cost rise with increasing dataset sizes. The experimental results are in accordance with the above reasoning.

Effect of Target Size, Angular Resolution and Number of Boundary Points. According to our experiments, as the size of the target increases, the total processing time and I/O cost of our approach increase. The reason is, as the size of the target increases, the number of obstacles visible from the target also increases. As a result, the size of *PVS* grows with an increase in target size. Consequently, the total processing time and the I/O cost increase.

We find that the total processing time is inversely proportional to angular resolution or cell size. The reduction time does not depend on angular resolution. But the computational overhead of determining visibility states and combining the results decrease with increasing angular resolution. This is because the number of equidistant stripes decreases as the angular resolution increases.

Our experiments reveal that with an increase in number of boundary points, total processing time increases and the error decreases. Total processing time increases, because the cost of determining the visibility states is proportional to the number of boundary points. The decrease in error occurs, because with higher number of boundary points, we can obtain a more accurate measure of what portion of the target is visible from each cell.

In the above cases, our solution runs 10^5 to 10^7 times faster than the exact straightforward approach and the error rate varies from 3 % to 7 %, which is negligible. The experimental results are not shown in details for brevity.

7.4 Performance for Moving Target

To assess the efficiency of our solution for constructing the *VCM* for a moving target, we vary the ratio of the lengths of an edge of the super target and the target from 2 to 5. We determine the average total processing time for all discrete positions of the target inside the super target in two ways. First we separately run the process of constructing the *VCM* for a fixed target as described in Sect. 6.4 for each discrete position of the target inside the super target and take the average processing time, which we call *fixed target average cost*. Then we apply the method described in Sect. 6.5 to construct the *VCM* for all discrete positions inside the super target and determine the average processing time by amortizing the cost of the preprocessing steps over all discrete positions. This is called *moving target average cost*.

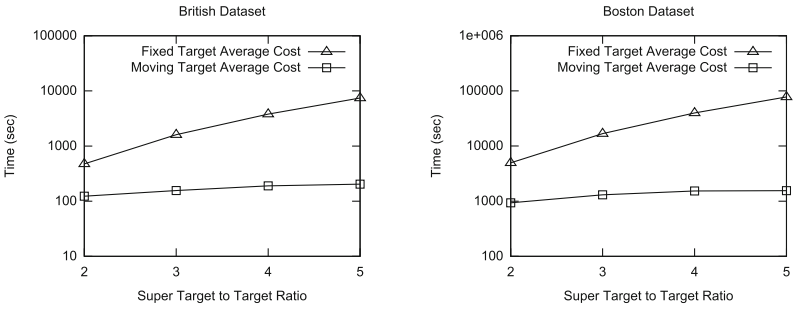

Fig. 7. Performance for moving target.

The experimental results in Fig. 7 illustrate that our pre-computation based solution for constructing the *VCM* for a moving target runs 10 to 100 times faster on average than the approach described in Sect. 6.4. As the size of the target with respect to the super target decreases, the number of discrete positions for the target inside the super target increases. As a result, the cost of the preprocessing steps becomes negligible in average and the moving target average cost becomes much smaller than the fixed target average cost.

8 Conclusion

In this paper, we have proposed an efficient and scalable technique to compute the visibility color map (*VCM*) that forms the basis of many real-life visibility queries in 2D and 3D spaces. The *VCM* quantifies the visibility of (from) a target object from (of) each viewpoint of the surrounding space and assigns colors accordingly in the presence of obstacles. Our plane-sweep based solution finds the *VCM* in three phases: finding the potentially visible obstacle set (*PVS*) from a large set of obstacles, determining the occlusion effects of obstacles in the *PVS*, and finally adding the effects of distance and angle between the target and

each cell of the partitioned dataspace. Our solution works for both fixed and moving target, and handles the partial visibility of the target. When the target moves to a near-by position, our proposed data structure can be incrementally updated to generate the *VCM* on-the-fly. Experiments with real and synthetic 3D datasets demonstrate that for a fixed target, our approach outperforms the straightforward approach by 5–6 orders of magnitude in terms of total processing time. Our solution to calculate the *VCM* for a moving target runs 10 to 100 times faster than our solution for a fixed target.

Acknowledgements. This research is supported by the ICT ministry - Bangladesh innovation fund for the project "Visibility Queries in 3D Spatial Databases". Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405.

References

1. Choudhury, F.M., Ali, M.E., Masud, S., Nath, S., Rabban, I.E.: Scalable visibility color map construction in spatial databases. Inf. Syst. **42**, 89–106 (2014)
2. Durand, F., Drettakis, G., Thollot, J., Puech, C.: Conservative visibility preprocessing using extended projections. In: SIGGRAPH, pp. 239–248 (2000)
3. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
4. Ben-Moshe, B., Hall-Holt, O., Katz, M.J., Mitchell, J.S.B.: Computing the visibility graph of points within a polygon. In: SCG, pp. 27–35 (2004)
5. Suri, S., O'Rourke, J.: Worst-case optimal algorithms for constructing visibility polygons with holes. In: SCG, pp. 14–23 (1986)
6. Asano, T., Asano, T., Guibas, L., Hershberger, J., Imai, H.: Visibility-polygon search and euclidean shortest paths. In: SFCS, pp. 155–164 (1985)
7. James Stewart, A., Karkanis, T.: Computing the approximate visibility map, with applications to form factors and discontinuity meshing. In: Drettakis, G., Max, N. (eds.) Rendering Techniques 1998. Eurographics, pp. 57–68. Springer, Vienna (1998)
8. Grasset, J., Terraz, O., Hasenfratz, J.M., Plemenos, D.: Accurate scene display by using visibility maps. In: SCCG, pp. 180–186 (1999)
9. Bittner, J.: Efficient construction of visibility maps using approximate occlusion sweep. In: SCCG, pp. 167–175 (2002)
10. Tsai, Y.H.R., Cheng, L.T., Osher, S., Burchard, P., Sapiro, G.: Visibility and its dynamics in a PDE based implicit framework. J. Comput. Phys. **199**(1), 260–290 (2004)
11. Kim, D.S., Yoo, K.H., Chwa, K.Y., Shin, S.Y.: Efficient algorithms for computing a complete visibility region in three-dimensional space. Algorithmica **20**(2), 201–225 (1998)
12. Koltun, V., Chrysanthou, Y., Cohen-Or, D.: Hardware-accelerated from-region visibility using a dual ray space. In: Gortler, S.J., Myszkowski, K. (eds.) Rendering Techniques 2001. Eurographics, pp. 205–215. Springer, Vienna (2001)
13. Koltun, V., Chrysanthou, Y., Cohen-Or, D.: Virtual occluders: an efficient intermediate pvs representation. In: Péroche, B., Rushmeier, H. (eds.) Rendering Techniques 2000. Eurographics, pp. 59–70. Springer, Vienna (2000)

14. Hernndez, J., Garca, L., Ayuga, F.: Assessment of the visual impact made on the landscape by new buildings: a methodology for site selection. Landsc. Urb. Plan. **68**(1), 15–28 (2004)
15. Bartie, P., Reitsma, F., Kingham, S., Mills, S.: Advancing visibility modelling algorithms for urban environments. Comput. Environ. Urb. Syst. **34**(6), 518–531 (2010)
16. Google sketchup. http://www.sketchup.com
17. Autocad. http://www.autodesk.com/products/autodesk-autocad/overview
18. Maya. http://www.autodesk.com/products/autodesk-maya/overview
19. Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. IEEE Trans. Knowl. Data Eng. **22**, 665–681 (2010)
20. Gao, Y., Zheng, B.: Continuous obstructed nearest neighbor queries in spatial databases. In: SIGMOD, pp. 577–590 (2009)
21. Gao, Y., Zheng, B., Lee, W.C., Chen, G.: Continuous visible nearest neighbor queries. In: EDBT, pp. 144–155 (2009)
22. Masud, S., Choudhury, F.M., Ali, M.E., Nutanong, S.: Maximum visibility queries in spatial databases. In: ICDE, pp. 637–648 (2013)
23. Ali, M.E., Tanin, E., Zhang, R., Kulik, L.: A motion-aware approach for efficient evaluation of continuous queries on 3d object databases. VLDB J. **19**, 603–632 (2010)
24. Kaiser, P.: The Joy of Visual Perception. York University, Toronto (1996)