Computing Highly Occluded Paths on a Terrain*

Niel Lebeck
Duke University
nl53@duke.edu

Thomas Mølhave
Duke University
thomasm@cs.duke.edu

Pankaj K. Agarwal
Duke University
pankaj@cs.duke.edu

ABSTRACT

Understanding the locations of highly occluded paths on a terrain is an important GIS problem. In this paper we present a model and a fast algorithm for computing highly occluded paths on a terrain. It does not assume the observer locations to be known and yields a path likely to be occluded under a rational observer strategy. We present experimental results that examine several different observer strategies. The repeated visibility map computations necessary for our model is expedited using a fast algorithm for calculating approximate visibility maps that models the decrease in observational fidelity as distance increases. The algorithm computes a multiresolution approximate visibility map and makes use of a graphics processing unit (GPU) to speed up computation. We present experimental results on terrrain data sets with up to 144 million points.

Categories and Subject Descriptors: F.2.2 [Nonnumerical Algorithms and Problems]: Geometrical problems and computations; H.2.8 [Database Management]: Database Applications—*Data Mining, Image Databases, Spatial Databases and GIS*

General Terms: Performance, Algorithms

Keywords: Terrain modeling, GIS, visibility, navigation

1. INTRODUCTION

In this paper we consider the problem of computing a highly *hidden* or *occluded* path on a terrain Σ between two given points $a, b \in \Sigma$, that is, a path that can be traversed with minimal risk of exposure to a set O of observers placed on Σ. The problem of finding a highly occluded path has a wide range of applications, including military applications (e.g. planning troop movements in enemy territory), city planning (e.g. planning the location of visually unappealing construction projects such as power lines), and virtual environments (e.g. video games).

In many applications, the set O of observers is not known, but given Σ we still want to compute a path from a to b that is likely to be highly occluded. In such cases, the problem can be considered in an adversary model where the (rational) adversary chooses O to minimize the number of good hidden paths. We assume that the adversary does not know a or b in advance. Since the adversary is rational and Σ is known in advance, it is possible to exploit information about the topology and features of the terrain to find paths that are likely to be well hidden from the adversary, regardless of his choice of O.

This raises a closely related problem of guessing where the observers are placed on Σ—guessing the strategies the adversary can use to choose a good set of observers O such that for any $a, b \in \Sigma$ a good hidden path from a to b is unlikely to exist. We can then compute the path from a to b on Σ that is most hidden with respect to the guessed observers. If the set O^* of guessed observers is close to the actual set of observers, then occluded paths with respect to O^* will remain occluded with respect to O. Even if O and O^* differ, our hypothesis is that if O^* accurately captures the highly visible areas of Σ, then paths that are occluded with respect to O^* will remain so with respect to O.

A terrain Σ in a GIS is stored as a digital elevation model (DEM). Due to its simplicity the most popular type of DEM is the grid DEM, represented as a two dimensional array of points, in which elevation is specified at each grid point. With the advances in sensing and mapping technology, very large, high-resolution grid DEMs of terrains are becoming easily available. For example, modern airborne laser altimetry (LiDAR) technology can map the earth's surface at a 15-25 cm horizontal resolution. While such high-resolution DEMs provide unprecedented opportunities, their large size requires the availability of efficient algorithms for terrain modeling and analysis, and this is often the bottleneck in taking full advantage of these applications. This is particularly true for path planning and visibility analysis on terrains.

Computing visibility information on a grid DEM can be costly. The *visibility map* V_o of an observer $o \in O$ is, intuitively, the region of Σ visible to o. The visibility map for a single observer, V_o, can have linear complexity in the size of the grid DEM representation of Σ and naive algorithms for computing it are of worst case quadratic complexity. This is prohibitively expensive when O is large.

We observe that observational fidelity decreases with dis-

*Work on this paper is supported by NSF under grants CCF-09-40671, CCF-10-12254, and CCF-11-61359, by an ARO grant W911NF-08-1-0452, and by an ERDC contract W9132V-11-C-0003.

ACM SIGSPATIAL GIS '13 November 5-8, 2013. Orlando, FL, USA
 http://dx.doi.org/10.1145/2525314.2525363 ...$15.00.

tance, which implies that we can employ approximation algorithms that use an appropriate level of detail based on the features of the terrain and the distance between the terrain and the observers. This will allow us to decrease, using terrain simplification techniques, the amount of information about Σ that we need to process far away from o which will speed up the computation. Furthermore, we need not compute V_o itself at a high resolution far away from o, which will further speed up the computation.

Furthermore, over the last decade modern PCs have started to become equipped with advanced and increasingly powerful graphics processing units (GPUs). Although originally designed for rapidly transforming 3D geometric scenes into pixels on the image plane (screen) and extensively used in video games, they can be regarded as massively parallel vector processors suitable for general purpose computing. To speed up the computation of V_o we want to employ the GPU which has the potential to significantly improve performance.

Related work. There is extensive work in computational geometry, robotics, GIS, and virtual environment communities on path planning and visibility related problems. It is beyond the scope of this paper to review all of them, even the work on visibility based path planning. We therefore focus on the work that is closely related to our results. We refer the reader to the books [12, 22] for a review of visibility based planning and related problems.

Most of the research on optimal path-planning on terrains in computational geometry has focused on finding the paths of minimum length. The best known algorithm takes quadratic time [5] , and there are faster sampling based approximation algorithms [24, 3]. There is also some work on the so-called weighted-region problem, where a weighted planar subdivision is given and the goal is to find a path of the minimum weighted length. The exact algorithms are quite expensive in 2D, but faster approximation algorithms are known; see [2] and references therein. One can assign the weights of a region based on visibility of that region from a given set of observers and can formulate the problem of computing a highly occluded path as a weighted-region problem. However, this approach will be quite expensive because the number of regions will be quite large, and the algorithm will be impractical even for small-size terrains.

There are two variants of visibility-based path planning problems that are extensively studied in computational geometry. First, the so-called *shortest watchman tour* problem—find the shortest path π inside a polygonal environment P so that every point of P is visible from at least one point of π; see [18]. The other variant is the so-called visibility based pursuer-evader games, where one or multiple pursuers wish to catch an intruder inside a polygonal environment. Here the objectives are—how many pursuers are needed to catch an intruder, and how quickly can they catch them; see [10] for some recent work and for a review of earlier work. The so-called *localization* problem [20] and *art gallery* problem [19, 25] are also related.

The problem of visibility based exploration in an unknown environment is widely studied in robotics and GIS. Here the goal is to explore as much environment as possible with as little motion as possible [12]. A commonly used approach is the visibility based probabilistic road map approach, which carefully samples points in the environment, constructs a graph, and traverses the graph to explore the environment [4, 23]. Another interesting approach is based on level-set methods [11, 9]. See [16] for a comparison of different approaches.

Finally, there is some work in GIS and virtual environment communities on finding paths that are occluded from a given set of observers [8, 13, 14, 17]. All these papers sample a set of points on the terrain or polygonal environment, assign a weight to each point based on the visibility from a given set of observers, and use a steepest-descent, A* search, or Dijkstra's algorithm to compute a desired path. However, all these algorithms do not seem to scale to large terrains. For example, the largest terrains on which the algorithms in [14] were tested had size 257×257. Franklin et. al [8] propose a terrain compression algorithm and compute the paths on the compressed terrain. They present experimental results on terrains of sizes up to $160,000$ (400×400) grid points.

Our contributions. The main contribution of our paper is a scalable algorithm for computing a highly occluded path between two points on a terrain, represented as a grid DEM; our algorithm can handle large terrains by relying on approximation techniques and graphics processing units (GPUs) available on modern PCs. There are three main ingredients of our algorithm:

(i) We describe a model for highly occluded paths on an arbitrary terrain, represented as a continuous height function, with respect to a given set of observers (Section 2). Our model unifies several previous models and implicitly optimizes both the visibility and the length of the path. Although not described explicitly, it can also incorporate other features of the terrain in its cost function, such as slope. We then adapt this model for terrains represented as a grid DEM.

(ii) Next, we present an algorithm for computing the visibility map of the terrain from a fixed viewpoint (Section 3). Since our focus is on handling large terrains, we rely on approximation techniques and compute an approximate visibility map by simplifying the geometry of the terrain progressively as we move farther away from the viewpoint. Unlike some of the previous work, our simplification algorithm is adaptive to the location of the observer; similar ideas were previously used in graphics [15]. We then describe an efficient implementation of our algorithm using a GPU.

(iii) In many applications we do not know the location of observers, so we describe two approaches to guess their locations (Section 4). The first approach is based on the topological analysis of the algorithm—the idea being that observers are likely to be on high maxima or saddle points of the terrain, from which one can see most of the terrain. However, simply choosing k highest maxima and/or saddle points is not sufficient since they may be clustered and not cover the terrain well. We therefore describe an algorithm based on topological persistence [7]. The second algorithm is an efficient implementation of the so-called art gallery problem using our fast visibility-map algorithm of Section 3. We also present a third approach which is a hybrid of these two algorithms.

We have implemented our algorithm and Section 5 demonstrates its effectiveness by detailed experimental results. We show that our visibility map algorithm allows fast visibility map calculations, with a trade-off between approximation fidelity and speed. In fact, we compute a visibility map at a reasonable level of approximation of a terrain with 144 million points in about 40 seconds. We also examine the observer selection strategies described in Section 4. We show that persistence and coverage-based selection, as well as a hybrid of

the two, are significantly more effective than simple-minded selection strategies, and that occluded paths calculated using these strategies are also highly occluded when observer placements follow the trivial strategies.

2. OUR MODEL

In this section we describe our model for computing a highly occluded path on a terrain. We begin by defining the notion of visibility and hidden paths on an arbitrary terrain surface and then tailor it to terrains represented by grid DEMs.

Continous model. Let $\mathbb{M}$ be a planar region, which, for simplicity, we assume to be a square. Let $h : \mathbb{M} \to \mathbb{R}$ be a height function. The graph of the function h, denoted by Σ is called a *terrain*, i.e., $\Sigma = \{(x, h(x)) \mid x \in \mathbb{M}\}$. For a point $p \in \mathbb{M}$, we use $\hat{p} = (p, h(p))$ to denote the corresponding point on the terrain Σ. For a pair of points $\xi, \eta \in \mathbb{R}^3$, ξ is *visible* from η, and vice-versa, if no point on segment $\xi\eta$ lies below Σ, i.e., for any $(q, z) \in \xi\eta$ where $q \in \mathbb{R}^2$ and $z \in \mathbb{R}$, $z \geq h(q)$. Otherwise ξ is *occluded* from η.

Given an observer $o \in \mathbb{R}^3$ lying above Σ, we define the *visibility map* $V_o : \mathbb{M} \to \{0, 1\}$ as

$$V_o(p) = \begin{cases} 1 & \text{if } \hat{p} \text{ is visible from } o, \\ 0 & \text{otherwise.} \end{cases}$$

With a slight abuse of notation we also use V_o to denote the region $V_o = \{p \in \mathbb{M} \mid V_o(p) = 1\}$. Since an observer cannot see faraway points and the quality of visibility deteriorates with distance, we also introduce the notion of an *attenuated visibility map*. Let $\alpha : \mathbb{R}_{\geq 0} \to [0, 1]$ be a monotonically decreasing function, which models the attenuation in visibility with distance; two typical examples of α are $\alpha(x) = \max\{0, 1 - x/a\}$ or $\alpha(x) = e^{-ax^2}$ for some constant $a \in \mathbb{R}$. With a slight abuse of notation, we define the attenuated visibility map as $V_o : \mathbb{M} \to [0, 1]$, where

$$V_o(p) = \begin{cases} \alpha(||p - o||) & \text{if } \hat{p} \text{ is visible from } o, \\ 0 & \text{otherwise.} \end{cases}$$

The unattenuated visibility map is a special case of the attenuated one with $\alpha(\cdot) = 1$.

Let $O = \{o_1, \ldots, o_m\} \subseteq \mathbb{R}^3$ be a set of observers, each lying above Σ. We define the *aggregated visibility map* $V_O : \mathbb{M} \to [0, m]$ as

$$V_O(p) = \sum_{i=1}^{m} V_{o_i}(p).$$

That is, how well the observers in O collectively can see the point p. Instead of using a simple sum, one can use a more sophisticated aggregation function.

Given O, we can use V_O to model how visible each point in $\mathbb{M}$ is from O. However, once the "visibility" of a point p reaches a certain threshold, the increase in $V_O(p)$ does not matter. Put differently, a change in $V_O(p)$ from 0 to 1 has a much greater impact on the visibility of p than an increase from 10 to 12. So we define another map, called the *coverage map*, $\omega_O : \mathbb{M} \to \mathbb{R}_{\geq 0}$ as follows:

$$\omega_O(p) = 1 - e^{-cV_O(p)}$$

where $c > 0$ is a constant.

For a unit-velocity curve $\Pi : [0, L] \to \mathbb{M}$ and a fixed set O of observers, we define the cost of the curve to be:

$$c(\Pi) = \int_0^L \omega(\Pi(t))\, \mathrm{d}t\,.$$

Given two points $a, b \in \mathbb{M}$, the cost of the most occluded path between a and b is defined as

$$\inf_{\Pi} c(\Pi)$$

where inf is taken over all paths in $\mathbb{M}$ from a to b.

Grid DEM. As is common in terrain modeling, we use a grid DEM to represent the terrain Σ. More precisely, we have a parameter $\rho > 0$ and assume that $\mathbb{M}$ is a square of side length $\rho 2^L$ for some integer $L \geq 0$. We partition $\mathbb{M}$ into $2^L \times 2^L$ *grid cells*, each of length ρ. For each grid cell (i, j), $0 \leq i, j < L$, let q_{ij} denote its center. Set $\mathbb{Q} = \{q_{ij} \mid 0 \leq i, j < 2^L\}$. We sample the height of the center of each grid cell $h_{ij} = h(q_{ij})$ and define $\hat{q}_{ij} = (q_{ij}, h_{ij})$. Set $\hat{\mathbb{Q}} = \{\hat{q}_{ij} \mid q_{ij} \in \mathbb{Q}\}$.

The surface Σ is defined from $\mathbb{Q}$ by linear interpolation across a planar triangulation of $\mathbb{Q}$. We triangulate $\mathbb{Q}$ by inserting edges between grid points. For a point $q_{ij} \in \mathbb{Q}$ we insert edges to horizontally adjacent grid point $q_{(i+1)j}$ if $i < 2^L - 1$, to vertically adjacent grid point $q_{i(j+1)}$ if $j < 2^L - 1$, and the diagonal between to $q_{(i+1)(j+1)}$ if both i and j are less than $2^L - 1$. Let $\Delta(\mathbb{Q})$ denote the resulting 2D triangulation. We "lift" $\Delta(\mathbb{Q})$ to 3D by associating the vertices with the elevation values for the correponding points in $\hat{\mathbb{Q}}$. The resulting polyhedral surface is the terrain Σ.

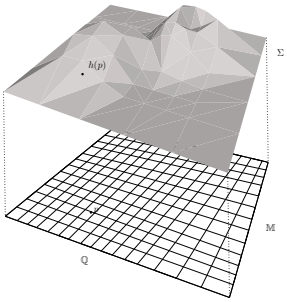


Figure 1. Terrain Σ on top of the domain $\mathbb{M}$.

We now construct a weighted graph $\mathcal{G} = (\mathbb{Q}, E)$, where each grid point q_{ij} is connected to its eight neighbors. The weight of an edge (q, q') is

$$c(q, q') = \frac{\omega(q) + \omega(q')}{2} \|q - q'\|.$$

The cost of a path $\Pi = v_0 v_1 \cdots v_k$ in $\mathcal{G}$ is defined to be

$$c(\Pi) = \sum_{i=0}^{k-1} c(v_i, v_{i+1}).$$

Given two grid points $a, b \in \mathbb{Q}$, let $H(a, b)$ denote the minumum-cost path from a to b in $\mathcal{G}$. We refer to H as the *most occluded path* between a and b on Σ.

3. COMPUTING THE VISIBILITY MAP

Given a terrain Σ over a domain $\mathbb{M}$ and an observer $o \in \mathbb{R}^3$ lying above Σ, we wish to compute the visibility map of Σ from o, as defined above. Computing $V_o : \mathbb{M} \to \{0, 1\}$ is expensive, especially when Σ is large, and we wish to compute the map from various points, so we focus on computing an approximate visibility map $\widetilde{V}_o : \mathbb{M} \to \{0, 1\}$. The quality of the approximation can be tuned by setting two parameters appropriately. Although we need to compute $\widetilde{V}_o$ only at the

grid points in $\mathbb{Q}$, it will be simpler for most of the discussion to think that $\widetilde{V}_o$ is being computed over the entire domain $\mathbb{M}$.

The approximation of $\widetilde{V}_o$ is performed in two stages. The first stage approximates the terrain Σ to a simplified terrain $\widetilde{\Sigma}$, with significantly fewer vertices, and the second stage computes an approximate visibility map of $\widetilde{\Sigma}$. We first describe the construction of $\widetilde{\Sigma}$, then we describe the overall approach for computing $\widetilde{V}_o$, and finally we describe a GPU based algorithm for constructing $\widetilde{V}_o$. Throughout this section we assume that the observer o is fixed, and we drop o from various notation.

3.1 Terrain simplification

Our approach for simplifying Σ is similar to the ones used in computer graphics and GIS for visualizing large scenes [15], the so-called levels of detail. Roughly speaking, we take advantage of the fact that it gets progressively harder to discern detailed terrain features as the distance between o and these features increase. We rely on a quadtree data structure to represent Σ hierarchically and to compute an adaptive approximation of Σ.

Quad tree. We construct a quad tree $\mathcal{T}$ on $\mathbb{M}$. $\mathcal{T}$ is a 4-way tree, each of whose leaves is associated with a grid cell of $\mathbb{M}$. Each internal node, u, of $\mathcal{T}$ is also associated with a square $\square_u$, which is the union of the squares associated with the four children of u. The root of $\mathcal{T}$ is associated with the square $\mathbb{M}$ itself. The squares associated with the children of u are obtained by splitting each side of $\square_u$ into two halves. $\mathcal{T}$ has L levels, with leaves being at level 0. For a node u, we set $\mathbb{Q}_u = \mathbb{Q} \cap \square_u$. If u is at level l, then the side length of $\square_u$ is $\rho 2^l$ and $|\mathbb{Q}_u| = 2^{2l}$. The nodes at level l partition $\mathbb{M}$ into a $2^l \times 2^l$ grid.

We fix a parameter b and store a set P_u of at most b^2 grid points, as defined below[1]. We also store a set Ξ_u of triangles at u, which form a triangulation of P_u. Abusing the notation, we use Ξ_u to denote the set of triangles as well as the planar subdivision induced by these triangles. For $l \leq \log_2 b$, $P_u = \mathbb{Q}_u$. For $l > \log_2 b$, we let P_u be a set of b^2 grid points defined recursively from its four children. Let $c_1, \ldots, c_4$ be these four children with c_j corresponding to the j'th quadrant of $\square_u$. Each c_j is averaged into a $b/2 \times b/2$ grid where the elevation of grid point (α, β) for $0 \leq \alpha, \beta \leq b$ is the average of the elevation of the grid points $(2\alpha, 2\beta)$, $(2\alpha+1, 2\beta)$, $(2\alpha, 2\beta+1)$, and $(2\alpha+1, 2\beta+1)$ in P_{c_j}. These four sub-grids are put together to form P_u which has $b \times b$ vertices. Define r_u, the resolution of P_u, to be the distance between two vertices (i, j) and $(i+1, j)$ in P_u. The resolution of a node at level k is

$$r_u = \begin{cases} 2^{k - \log b}\rho & k \geq \log b, \\ \rho & 0 \leq k < \log b. \end{cases}$$

Approximate terrain surface. Let $\Psi \in \mathcal{T}$ be a top subtree of $\mathcal{T}$, i.e., if a node $u \in \mathcal{T}$ is in Ψ, then its parent also belongs to Ψ, and let Λ be the leaves of Ψ; see Figure 3. Then the squares associated with Λ partition $\mathbb{M}$, i.e., $\square_u \cap \square_v = \varnothing$ for $u \neq v \in \Lambda$ and $\cup_{u \in \Lambda} \square_u = \mathbb{M}$. We form $\widetilde{\Sigma}$ by taking the union of Ξ_u, for $u \in \Lambda$, stitching along the borders of triangulations Ξ_u, and lifting the resulting triangulation to 3D. The last step is achieved by lifting each point $p \in P_u$, for $u \in \Lambda$, to $\hat{P}$ as

[1] for simplicity, we assume b is of the form 2^k for some $k \in \mathbb{N}$

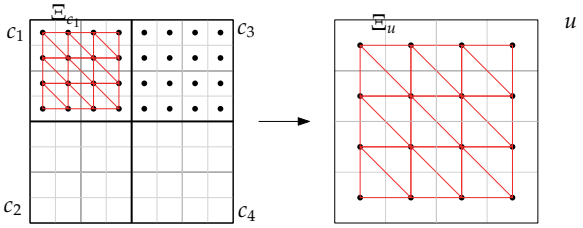


Figure 2. Constructing Ξ_u from its four children. Each grid point of Ξ_u is the average of four corresponding points in one of its children.

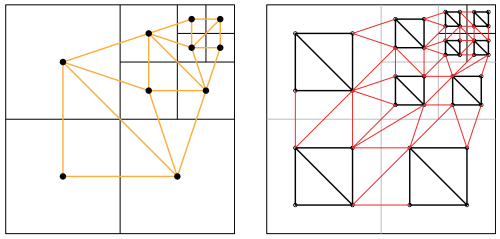


Figure 4. (a) The dual graph G (orange) of the partition of $\mathbb{M}$ induced by the partition $\square^\Lambda$. (b) The set of triangles Ξ_u (black) and stitching triangles $\partial\Xi_u$ (red) for $u \in \Lambda$. Here $b = 2$.

described above, so we describe how to choose the subtree Ψ and how to stitch the borders together.

A *selection rule* is used to choose the top subtree $\Psi \subseteq \mathcal{T}$ for a given observer. The selection rule is a function $s(o)$ that maps each observer o to a set of quad-tree nodes Λ such that the union of their cells $\cup_{u \in \Lambda} \square_u$ forms a partition over $\mathbb{M}$. Then Ψ is the subtree with leaves Λ. We define a monopole condition for node u: $||\hat{u} - o|| > \mu r_u$ for some constant *monopole coefficient* μ. Our selection rule $s(o)$ returns the set of nodes Λ consisting of the highest nodes in the tree that satisfy the monopole condition. Thus each node $u \in \Lambda$ satisfies the monopole condition but its parent that does not satisfy the condition. A node at a level below $\log_2 b$ is never chosen, since a node at level $\log_2 b$ has the same resolution, ρ, as its children. If some portion of the tree has no nodes that satisfy the monopole condition, then the nodes at level $\log_2 b$ are chosen. Note that our selection rule has the following property: for two nodes u_1 and u_2 in Λ, $l(u_1) < l(u_2) \Rightarrow ||\hat{u}_1 - o|| < ||\hat{u}_2 - o||$. This property means that only nodes at the same level or lower may come between any given node and the observer.

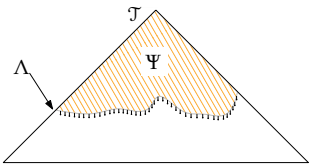


Figure 3. Top tree Ψ of $\mathcal{T}$.

Next we describe the *stitching procedure*. Let G denote the dual graph of the partition of $\mathbb{M}$ induced by the set $\square^\Lambda = \{\square_u \mid u \in \Lambda\}$. That is, the vertex set of G is $\square^\Lambda$ and two vertices $\square_u, \square_v$ are connected by an edge if the interior of the edges of $\square_u$ and $\square_v$ intersect; see Figure 4(a). Each edge of G corresponds to a fixed side e_u of $\square_u$ and a fixed side e_v of $\square_v$. The corresponding boundary edges of Ξ_u and Ξ_v form the boundary of a monotone polygon, and we can triangulate this polygon using any standard polygonal triangulation algorithm; see e.g. [6]. We repeat this procedure for every edge of G. This may leave some holes around the corners of squares

in $\square_\Lambda$, which we fill by adding $O(1)$ triangles for each such hole. Let $\partial\Xi_u$ be the set of triangles added to the boundary of Ξ_u by the stitching procedure; see Figure 4(b). It can be checked that $|\partial\Xi_u| = O(\sqrt{\Xi_u}) = O(b)$. We set $F_u = \Xi_u \cup \partial\Xi_u$ and refer to F_u as the set of *local* triangles at u.

3.2 Computing the visibility map

We begin by describing an algorithm for computing the exact visibility map of $\widetilde{\Sigma}$ from o and then describe how we approximate it.

Recall that the algorithm described in Section 3.1 computes a set Λ of nodes of Ψ such that the squares in $\square^\Lambda$ partition $\mathbb{M}$. For each node $u \in \Lambda$, we construct the visibility map, $\widetilde{V}_u : \square_u \to \{0,1\}$, of $\widetilde{\Sigma}$ restricted within $\square_u$. Putting these maps together, we obtain the visibility map of the entire terrain.

Computing $\widetilde{V}_u$. Fix a node $u \in \Lambda$. The visibility of a point $p \in \square_u$ is not only affected by the triangles in Ξ_u and $\partial\Xi_u$ but also by the triangles of $\widetilde{\Sigma}$ outside $\square_u$. More precisely, if the segment $o\hat{p}$ intersects a triangle $\hat{t}$ of $\widetilde{\Sigma}$, then $\widetilde{V}_u(p) = 0$. Instead of considering all triangles of $\widetilde{\Sigma}$, we compute $\widetilde{V}_u$ by considering a small subset of triangles of $\widetilde{\Sigma}$ lying outside $\square_u$. For a triangle $\hat{t} \in \widetilde{\Sigma}$, let Φ_t be the *shadow frustum* of $\hat{t}$ (with respect to o), i.e., the points in $\mathbb{R}^3$ that are occluded by $\hat{t}$. Formally,

$$\Phi_t = \{z + \lambda\overrightarrow{oz} \mid z \in \hat{t}, \lambda \geq 0\}.$$

Let $z_\perp \in \mathbb{R}$ be the minimum height of a vertex of $\widetilde{\Sigma}$. We say that a triangle $\hat{t} \in \widetilde{\Sigma}$ is an *occluder* for u if Φ_t intersects the unbounded box $\square_u \times [z_\perp, \infty]$. Let O_u be the set of all occluders for u. More precisely, $O_u = \{t \mid \hat{t} \in \widetilde{\Sigma} \text{ is an occluder for } u\}$. Set $\Delta_u = F_u \cup O_u$; Δ_u is the set of *relevant triangles* for u, in the sense that no triangle of $\widetilde{\Sigma} \setminus \Delta_u$ can impact the map $\widetilde{V}_u$.

After having computed the set Δ_u, $\widetilde{V}_u$ can be computed in a straightforward manner. In the next subsection we describe an efficient GPU based algorithm for computing $\widetilde{V}_u$.

Computing the occluder set. Computing the set O_u, for each $u \in \Lambda$, independently by traversing all triangles of $\widetilde{\Sigma}$ will be expensive. We describe a more efficient algorithm by processing the nodes of Λ in a particular sequence.

For two nodes $u, v \in \mathcal{T}$ such that $\square_u \cap \square_v = \varnothing$, we say that *u precedes v*, $u \prec v$, if there is a ray γ emanating from o^*, the projection of o onto the xy-plane, such that γ intersects $\square_u$ before intersecting $\square_v$. It is well known that this relationship induces a partial order on the nodes in Λ; see e.g. [21]. Furthermore, a total order of the nodes of Λ that is consistent with this partial order can be computed by traversing $\mathcal{T}$ carefully in the previous step—when we compute Λ. In particular, suppose we are at a node $u \in \mathcal{T}$ and we decide to visit the children $u_1, \ldots, u_4$ of u. Then we visit u_i before u_j if $u_i \prec u_j$. Let $v_1, \ldots, v_k$ be the sequence of nodes of Λ in the order computed by the algorithm. The following lemma suggests how to compute the occluder set for each u efficiently.

LEMMA 1. *For $i \leq k$, if a triangle $t \in O_{u_i}$, then there is a neighbor v_j of v_i in the graph G such that $j < i$ and $t \in \Delta_j$.*

PROOF. Suppose $t \in O_{v_i}$. Then there is a ray γ in $\mathbb{R}^2$ emanating from o^* that first intersects t and then $\square_{v_i}$. Let $\square_{v_j}$ be the square in $\square^\Lambda$ that γ intersects immediately before $\square_{v_i}$.

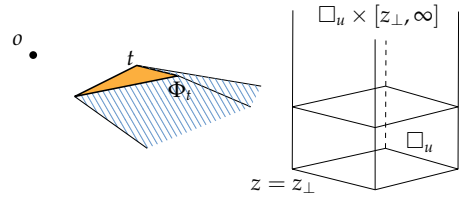


Figure 5. Triangle $t \in \widetilde{\Sigma}$ and its shadow frustum Φ_t looking at u.

Then v_j is a neighbor of v_i and $j < i$. If $t \in F_{v_j}$, then we are done. Otherwise the shadow frustum Φ_t also intersects the box $\square_{v_j} \times [z_\perp, \infty]$, implying that $t \in O_{v_j}$. This completes the proof of the lemma. □

In view of Lemma 1, we visit the nodes of Λ in the computed order. When we process the node u_i, we take all the relevant triangles for the nodes that are neighbors of u_i and that precede u_i, and discard all those triangles whose shadow frustums do not intersect the box $\square_{u_i} \times [z_\perp, \infty]$. This gives the set O_u.

Approximating $\widetilde{V}_u$. Instead of computing $\widetilde{V}_u(q)$ explicitly for each grid point $q \in \mathbb{Q}_u$, we compute it at a coarser grid if the size of $\mathbb{Q}_u$ is large. We choose a parameter $m \leq b$, where b is the resolution at which $\widetilde{\Sigma}$ was computed within u. We choose a subset of $\mathbb{Q}_u$ of $m \times m$ grid points and compute $\widetilde{V}_u$ at each of them using the algorithm described in Section 3.3. For each grid point $q \in \mathbb{Q}_u$ that was not chosen, we set $\widetilde{V}_u(q)$ to be the value of its nearest point that was chosen.

3.3 Computing $\widetilde{V}$ on the GPU

The GPU is responsible for drawing 3D scenes, composed of many objects, onto a 2D image plane of pixels as seen from a viewpoint β. Because of their simplicity and flexibility, these objects are almost always triangles. For each pixel $\pi = (x, y)$ where x, y is a global coordinate, the GPU finds all objects Ω that ray $\vec{\beta\pi}$ intersects. To store the information about the scene, the GPU keeps several two-dimensional arrays of pixels called buffers. The *color buffer* $\mathbb{C}$ stores the color of the scene as viewed from β. For our purposes, the color of pixel π in $\mathbb{C}$ is the color of object $\omega \in \Omega$ whose intersection with the ray $\vec{\beta\pi}$ is closest to β. Internally, the color buffer uses a *depth buffer* to find the closest point.

We use the GPU to compute the approximate visibility map $\widetilde{V}$, as follows. For simplicity, we first describe the algorithm for computing the unattenuated visiblility map. Since the terrain, and thus the derived visibility map, can be very large, we cannot handle the entire computation in one pass using the GPU. Instead we go through the nodes $u \in \Lambda$ and compute the visibility map $\widetilde{V}_u$ for each of these separately. We choose the parameters m and b, the resolution of the $\widetilde{V}_u$ and the nodes of $\mathcal{T}$, to maximize the use of the GPU by setting it to the maximum texture size of the GPU.

To calculate $\widetilde{V}_u$, we triangulate the boundary of the shadow frustum of each triangle in $\Delta_u = F_u \cup O_u$. Let D_u denote the resulting set of *shadow* triangles. We set the color of the (shadow) triangles in D_u to 0 and the color of (local) triangles in F_u to 1. We render the triangles in $D_u \cup F_u$ by setting the viewpoint at $z = +\infty$ (i.e., orthographic projection in the $(-z)$-direction). We set the view-port to the bounding box of $\square_u$, and render onto an image plane with $m \times m$ pixels: each grid

point is then at the center of a pixel. Due to the orthographic projection, the color buffer $\mathbb{C}$ at pixel (i, j) will hold the highest triangle in the $+z$ direction at grid point (i, j). If a pixel is colored 0, then the terrain at the corresponding grid point is covered by a shadow frustum and is thus occluded; if the pixel is colored 1, then the terrain at that point is visible, since no shadow frustum contains it. We copy the color buffer from the GPU to CPU, and set the values of $\widetilde{V}_u$ at point (i, j) to the value of the color buffer at that pixel. In the case of calculating the attenuated visibility map, we color terrain triangles with the value $\alpha(\|p - o\|)$ instead of 1 (see section 2). The same procedure then applies.

4. COMPUTING OCCLUDED PATHS

Given a terrain Σ and two points $a, b \in \Sigma$, we describe algorithms for computing a highly occluded path from a to b on Σ. If we know the set O of observers, then using the algorithm in Section 3 as the building block, we construct the coverage map ω_O and then compute the path $H(a, b)$ in $\mathcal{G}$. On the other hand, if we do not have any information about the observers, we first guess the location of observers and then construct a highly occluded path with respect to the guessed observers.

4.1 Known observer placement

Suppose we are given a set $O = \{o_1, \ldots, o_k\}$ of observers in advance. We do not require each o_i to lie on Σ. Typically they will be above the ground, say, on a watchtower. For each o_i, using the algorithm in Section 3.3, we construct an approximate visibility map $\widetilde{V}_{o_i}$. We upscale $\widetilde{V}_{o_i}$ to each grid point of $\mathbb{Q}$ by setting the visibility map value of $q \in \mathbb{Q}$ to the value $\widetilde{V}_{o_i}(c_q)$ where c_q is the center of the grid cell of $\widetilde{\Sigma}$ that contains q. Recall that $\widetilde{V}_{o_i}$ is computed at a coarser resolution in areas far away from o_i. Next, we compute the map $\widetilde{V}_O$, where $\widetilde{V}_O(q) = \sum_{i=1}^{k} \widetilde{V}_{o_i}(q)$, and finally we compute the coverage map $\omega(q)$ for each grid point $q \in \mathbb{Q}$. We then construct the weighted graph $\mathcal{G} = (Q, E)$, as defined in Section 2. We use Dijkstra's algorithm to compute the minimum-cost path $H(a, b)$ in $\mathcal{G}$.

4.2 Unknown observer placement

If we have no prior information about the observer locations we want to find a path that is likely to be concealed regardless of how the set of (unknown) observers O' was chosen. Our hypothesis is that O' is picked by an adversary in a rational way and that the adversary has no knowledge of a and b but strives to minimize the family of occluded paths (blind spots) in the terrain.

Finding a good occluded path in this setting is equivalent to finding a good guess O^* of possible observer locations. If the observer placements of O^* are close to O', occluded paths under one should translate well to the other. Even if O' and O^* differ, our hypothesis is that if O^* accurately captures the highly visible areas of Σ then paths that are occluded under O^* likely remain so under the set O' of adversary-chosen observers.

We explore two approaches for selecting O^*; one using topological information of Σ, and the other directly working with visibility maps and maximizing the terrain-coverage explicitly. Let k be our budget of observers. There are several possibilities to estimate the value of k, e.g. how much terrain is covered by k observers.

Topology-based placement. For a grid point $q \in \mathbb{Q}$, let $N(q)$ be the set of its neighbors in $\Delta(\mathbb{Q})$, sorted in clockwise order, and let $N^-(q)$ be the subsequence of $N(q)$ of points whose heights are less than that of q. Let $\mathbb{M}_t = \{p \in \mathbb{M} \mid h(p) \leq t\}$, i.e., the region corresponding to the portion of Σ lying below the height t. The point q is called a *minimum*, *maximum*, or *saddle* if $N^-(q) = N(q)$, $N^-(q) = \varnothing$, or $N^-(q)$ consists of two or more contiguous portions of $N(q)$, respectively; they are called *critical points* of $\mathbb{Q}$. Otherwise q is called a *regular* vertex. A saddle q is called *positive* if a connected component of $\mathbb{M}_t$ splits at $h(q)$, and negative if two connected components of $\mathbb{M}_t$ merge at $h(q)$.

Intuitively, it makes sense to place observers at (or above) high maxima or negative saddles of Σ with the hope that large portions of Σ are visible from there. However, a single hill may have multiple maxima, so it's not wise to simply choose the k highest maxima and saddles as possible locations of observers; see Figure 9. Instead we rely on persistent homology [7]. Roughly speaking, *persistence* attaches a measure of significance with each critical point of Σ. A contour, a connected component of a level set of Σ, is created at a minimum or a positive saddle and destroyed at a negative saddle or a maximum. The persistence homology pairs a minimum q_0 with the negative saddle q^- at which the contour created at q_0 is destroyed, and their persistence is $h(q^-) - h(q_0)$. Similarly it pairs a positive saddle q^+ with a maximum q_1 and their persistence is $h(q_1) - h(q^+)$. The persistence of the global maximum is set to ∞ (here we assume that the global minimum is at $-\infty$). Going back to the example of a hill with multiple maxima, only one of the maxima will have high persistence, and the others will have small persistence; see [7] for a detailed discussion on persistent homology.

Agarwal et al. [1] developed a simple scalable algorithm for computing the persistence of all critical points on a terrain. Using their algorithm to get a list of critical points with associated persistence, we pick O^* to be the k maxima and negative saddles with the highest persistence. The persistence ensures that only one point is selected from each feature, and thus, the points will be spread out amongst saddles and maxima.

Coverage-based placement. The persistence-based selection strategy above does a good job putting observers at maxima and saddles in the terrain. However, the significant hills also may not be evenly distributed across the terrain, so we also explore *coverage-based* placement, an observer placement strategy that more directly tries to get a good coverage of the terrain by computing visibility maps as part of the selection strategy.

Our coverage-based placement strategy is a variant of the *art gallery* problem [19, 25], and chooses a set of observers such that each observer sees as much of the terrain as possible.

This can be done by a simple greedy algorithm. First set $O^* = \varnothing$ and randomly sample a set $\Gamma \subset \mathbb{Q}$ of $m = |\Gamma|$ grid points. We then compute $\widetilde{V}_\Gamma$ using the algorithm in Section 3 and find $q' = \arg\max_{q \in \mathbb{Q}} \widetilde{V}_\Gamma(q)$. We then add $o' = \hat{q}'$ to our set of observers O^*. Let $A \subseteq \Gamma$ be the set of points that are visible to o', i.e. the set of points $p \in \Gamma$ where $\widetilde{V}_{o'}(p) = 1$. Note that depending on the application we may want observers to have height $h(q') + h_\top$, instead of just $h(q')$, for some application-dependent $h_\top$, e.g. we could choose $h_\top$ to be the height of an observer tower. Therefore, the points

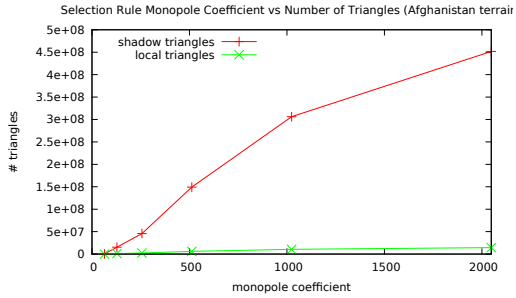


Figure 6. Total triangle counts for different values of μ. Visibility map computation times are 0.1 seconds for $\mu = 64$ and 167 seconds for $\mu = 2048$.

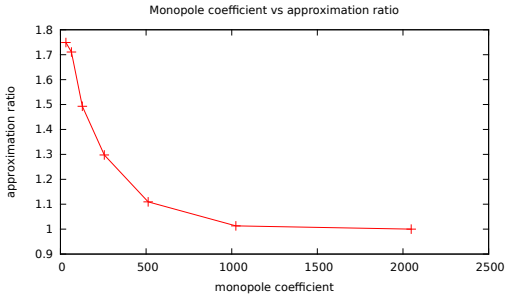


Figure 7. Graph showing how the approximation ratio varies with μ.

in Γ that see q' may not be identical to the points of Γ that can be seen by o'[2]. We then pick the next observer q' as $q' = \arg\max_{q \in Q} V_{\Gamma - A}(q)$ and repeat this process until we are done, i.e. $|O^*| = k$, or until we have covered all of the sample points of Γ, i.e. $A = \Gamma$. In the latter case we pick a new set Γ' and and run the procedure again, repeating as necessary until $|O^*| = k$.

This procedure effectively picks an observer that can see the highest number of points at each iteration and its fidelity can be tweaked by selecting m, the size of Γ. As k increases to $|Q|$ we get closer to picking a set O^* that sees the maximal number of points in $\hat{Q}$ but since we compute V_γ for each $\gamma \in \Gamma$ this algorithm becomes expensive for large m.

Hybrid placement. Finally, we combine the ideas from the above two strategies. We first choose a set T of $t > k$ observers using the topological persistence-based algorithm—they are placed on significant maxima and saddle points. We then compute the visibility map for each $p \in T$ and use the idea from the coverage-based placement strategy to greedily add the point $p \in T$ to our candidate set O^* whose visibility map covers the largest region that has not been covered by other observers in O^*. We repeat this greedily until $|O^*| = k$, or until every point in Σ is visible by some point in O^*. This strategy differs from the coverage-based strategy in that that our initial sample T consists of candidate observers whereas the coverage based strategy uses the sample to find observers that can see many sample points. The advantage of the hybrid algorithm is that it narrows the set of candidate targets to locations that are likely to be good (those with reasonably high persistence value) and thus the sample size can be smaller than the one in the coverage-based strategy.

5. EXPERIMENTS

This section presents the results of an experimental study on real data sets to evaluate the effectiveness of our algorithm. First, we report on the performance of our algorithm — the trade-off between accuracy and efficiency by using approximation. Second, we compare the different strategies for choosing observers. Finally, we study how observer-placement strategies affect the visibility of paths.

Platform & data. We ran our experiments on a computer with an Intel Core i7-3770 CPU running at 3.40 GHz and 32GB of internal memory. The machine has a NVIDIA GeForce GTX 660 graphics card which we interfaced with using OpenGL. The implementation was written in C++. We used two terrain models in our experiments. One dataset covers a roughly 2km by 4km region in Afghanistan terrain at a resolution of 2 meters (data courtesy of TEC ERDC). The terrain model consists of about 7.1 million grid points with mostly mountainous topology. It is a so-called *Digital Surface Model* (DSM) and as such contains non-terrain features relevant for visibility, such as trees and a few buildings. The second data set is a larger, higher-resolution 1 meter grid covering a 12km by 12km region in Ft. Leonard Wood in Missouri (data courtesy of the U.S. Army Corps of Engineers). The model has 144 million grid points with mostly rolling hills and a prominent riverbed. It is a *Digital Terrain Model* (DTM) containing only bare-earth elevation data.

Terrain simplification and efficiency. In this section we study the speed-accuracy tradeoff arising from the monopole coefficient μ. We use the Afghanistan terrain and a fixed observer in the corner of the study area. We first study the effect of μ on the performance of computing the visibility map $\widetilde{V}_o$. Computing the set Λ defining $\widetilde{\Sigma}$ is negligible compared to the cost of rendering the triangles in $\sum_{u \in \Lambda} F_u \cup D_u$. For $\mu = 2048$ the computation of $\widetilde{V}_o$ took 167 seconds, out of which only 1.5 seconds were spent computing Λ. On larger terrains, the cost of computing the quad tree $\mathcal{T}$ can be on the same order of magnitude as computing a single visibility map, but $\mathcal{T}$ is independent of o and only has to be computed once, making its construction cost trivial when the set of observers is large. Figure 6 shows the relationship between μ and number of local triangles, $\sum_{u \in \Lambda} |F_u|$ as well as the number of shadow triangles, $\sum_{u \in \Lambda} |D_u|$. For $\mu = 2048$ no simplification is done (i.e., $\widetilde{\Sigma} = \Sigma$), and there are 1.4×10^7 and 4.5×10^8 local and shadow triangles, respectively. As the simplification gets heavier further away from o, the number of triangles in both sets decreases, e.g., to 1.2×10^6 local and 1.5×10^7 shadow triangles for $\mu = 128$. As stated above, the total time for computing $\widetilde{V}_o$ for $\mu = 2048$ was 167 seconds, the time for computing using $\mu = 128$ was two magnitudes faster at 4.9 seconds. Besides the number of triangles, the number of rendering passes, determined by $|\Lambda|$ has a significant impact on performance since each pass involves transferring the computed visibility map from GPU memory to CPU memory, for $\mu = 128$ the number of leaves was $|\Lambda| = 13$ but for $\mu = 2048$ the number was an order of magnitude higher at $|\Lambda| = 256$.

Lowering μ has clear performance benefits and in the following we investigate how it affects the quality of the computed paths. Figure 8(top) shows how $H(a, b)$ varies with

[2]The fact that observer at height h above point $p_1 \in \Sigma$ can see point $p_2 \in \Sigma$ does not imply that an observer at height h above p_2 can see p_1

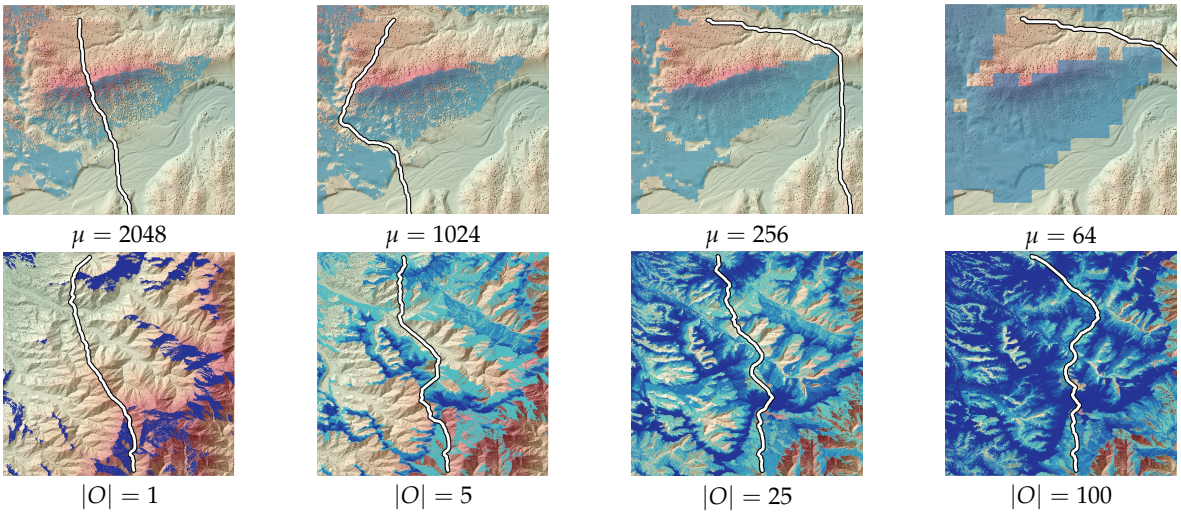


Figure 8. (top) Paths between the same two locations with the same set of observers for different values of μ The blue-shaded area denotes points visible from an observer. (bottom) Most occluded path between two points for different sizes of O, the set of observers. The blue shaded area is the coverage map ω_O.

μ. The area shown in the figure is far away from the observer position where the DSM contains a number of trees. For $\mu = 2048$, the path goes through the hidden patches provided by the trees. As μ decreases the approximation becomes coarser ($\mu = 1024$) and since the averaging algorithm used in the terrain simplification acts as a low-pass filter, the trees (high-frequency components) disappear, exposing that region of the terrain to the observer. Thus, the path changes to follow a riverbed adjacent to the forest instead. A further decrease in μ (to $\mu = 256$) eliminates some narrow corridors of invisibility in the riverbed, so the path changes again to avoid the riverbed and the forest completely. The final figure shows the result when increasing even further to $\mu = 64$, at this stage the path is diverted even further to the right, likely because the little valley in the lower left of the figure has disappeared in the simplification.

We now explore the quality of the paths generated using the simplified terrain. We use the Afghanistan terrain and a set O of 5 observers in the terrain, these observers were placed using the topology-based strategy from Section 4.2. We randomly picked 100 different pairs of points $(a_1\ b_1), \ldots, (a_{100}, b_{100})$ in the terrain (by picking 10 different source points, and for each of these picking 10 different destination points). We computed the path $\Pi_i^\mu = H(a_i, b_i)$ for each i using a monopole coefficient of μ. Let $c(\Pi_i^\mu)$ be the cost of this path and let $c^*(\Pi_i^\mu)$ be the cost of this path evaluated on the original terrain Σ. $c(\Pi_i^\mu)/c^*(\Pi_i^\mu)$ is the *approximation ratio* of Π_i^μ. In Figure 7 we have plotted the average of the approximation ratios as a function of μ. Not surprisingly, the approximation ratio increases as μ decreases and the amount of simplification increases. However, it also shows that this increase is not dramatic, for instance, paths computed using $\mu = 256$ are only 30% more expensive on average than what one would have gotten using the full resolution, and the lower value of μ reduces the number of triangles by about two orders of magnitude, causing a similar decrease in computation time.

Observer selection. In this section we compare five different strategies for selecting an observer set O to evaluate the effectiveness of the strategies chosen in Section 4. Besides the topology, coverage and hybrid strategies discussed in Section 4 we used two additional, trivial, strategies, as a baseline for comparison. The first simply selects k random points from the set of maxima of Σ. Another simple strategy is to sort the maxima by height and pick the k highest elevation vertices.

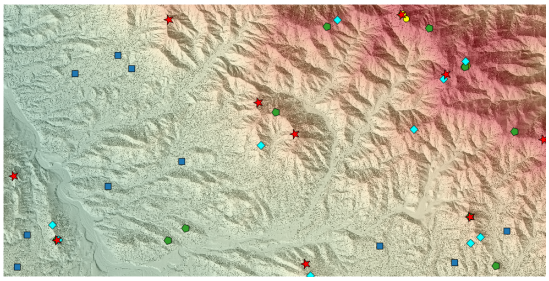


Figure 9. The $k = 10$ observers selected using each observer selection strategy on the Afghanistan terrain. O_{random} (blue square), O_{topology} (green pentagons), O_{coverage} (turquoise diamonds), O_{top} (yellow circles), O_{hybrid} (red stars).

Let $\mathcal{S} = \{\text{random}, \text{top}, \text{topology}, \text{coverage}, \text{hybrid}\}$ be the set of strategies, which gives rise to $|\mathcal{S}|$ sets of observers and coverage maps O_s and ω_s, respectively for $s \in \mathcal{S}$. Figure 9 shows the sets O_s for $k = 10$ on the Afghanistan terrain. The observers in O_{top} are clustered around the two tallest two mountain peaks. Here persistence helps greatly, and O_{topology} is distributed on the k largest mountains. The observers in O_{coverage} are also at (or close to) the peaks of mountains, and is in fact very similar to O_{topology} although the observers are placed slightly differently on the mountains. O_{hybrid} is distributed well on the terrain with observers placed on peaks. O_{random} is fairly well distributed, but it fails to capture many high-visibility points.

Figure 10(left) shows the average value of the coverage map ω_{O_s} over the Afghanistan and Ft. Leonard Wood terrains as a function of $k = |O_s|$ using each of the O_s for $s \in \mathcal{S}$. The graph suggest that we do not need a large number of observers to get a good coverage of the terrain using O_{hybrid}, O_{topology} and O_{coverage}.

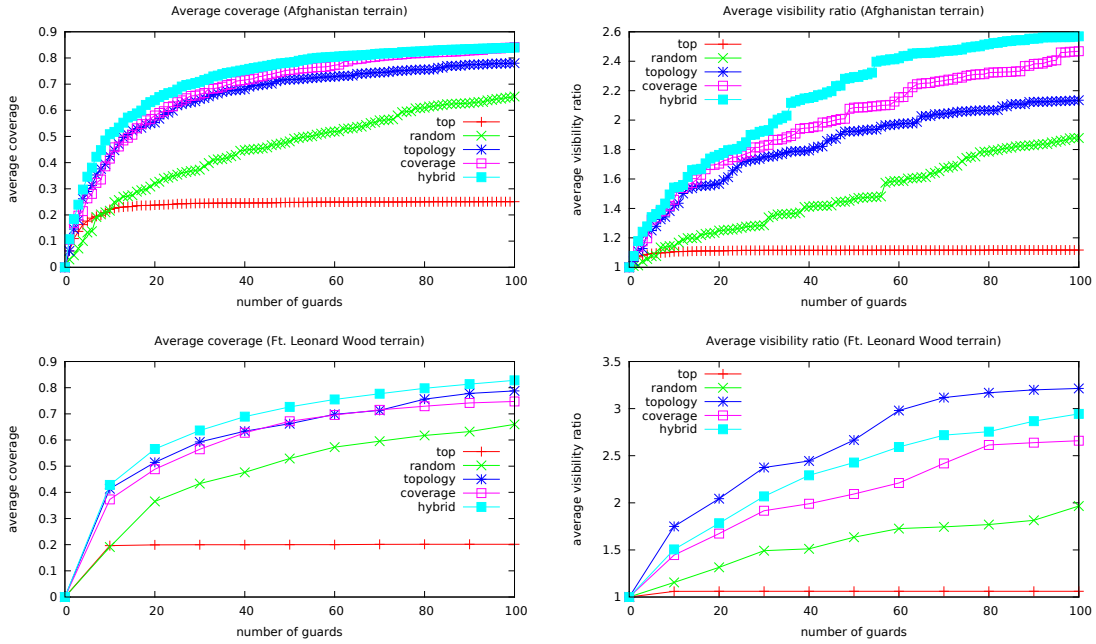


Figure 10. (left) Coverage of terrain. (right) Average minimum path cost ratio.

We also consider the actual costs of occluded paths under the different strategies. We do this by picking 10 random points $b_1, \ldots, b_n \in \mathbb{Q}$ and computing the most occluded path $\Pi_i^k = H(a, b_i)$ from the point a (chosen as the midpoint of $\mathbb{Q}$) using k observers. Let $c'(\Pi_i^k)$ be the cost of Π_i^k on the original terrain Σ and with a constant visibility map $\omega(\cdot) = 1$. Let $c(\Pi_i^k)/c'(\Pi_i^k)$ denote the *visibility ratio* of Π_i^k—the path cost relative to the shortest path on Σ if visibility is ignored. Figure 10(right) shows the average visibility ratio of $\Pi_0^k, \ldots, \Pi_{10}^k$ as a function of k under each of the strategies $\mathcal{S}$. We note that $c'(\Pi)$ is independent of the set of observers used to compute Π. This implies that we can directly compare visibility ratios of paths computed under different observer placement strategies with the same k. Figure 10(right) indicates the baseline strategies result in lower ratios, which implies larger number of observers must be chosen to reduce the number of cheap occluded paths. The hybrid strategy consistently outperforms the coverage-based strategy, but for the Ft. Leonard Wood data the topology-based strategy does best.

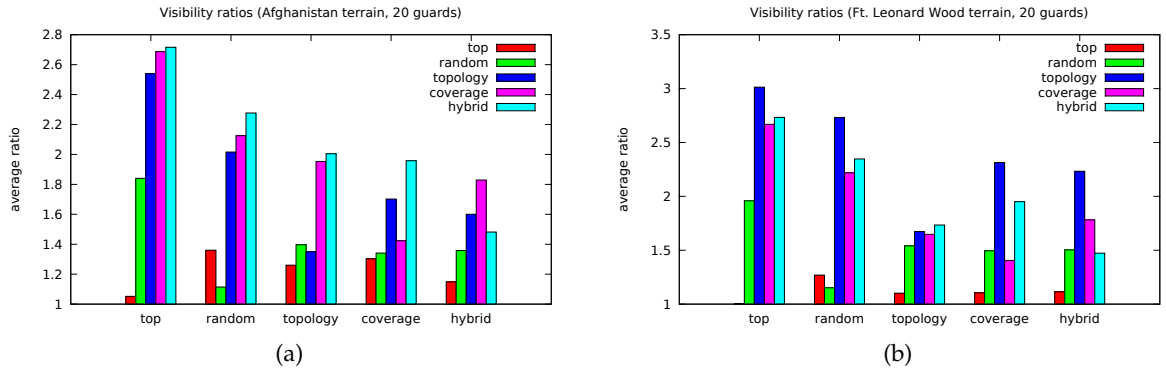
Highly occluded path quality. The previous discussion showed that the hybrid and topology-based strategies consistently outperform the baseline strategies. Thus, if we are given Σ with no information about the true set of observers O' we can use any of these to find another set of observers O^* and hope that this set captures roughly the same visibility information as O'. We investigate this by calculating the path $\Pi = H(a, b)$ using one of the strategies s of $\mathcal{S}$ and then computing the cost of Π when the coverage map $\omega_{O_{s'}}$ is computed using another strategy $s' \in \mathcal{S}$. This tests if the path Π is occluded, not just for O_s but for the other sets of observers that the adversary could have chosen as well. More precisely, we fix the source point a and choose 10 random destination points $b_1, \ldots, b_{10}$. For $i \leq 10$ and $s \in \mathcal{S}$ let $\Pi_s^i = H(a, b_i)$ be the most occluded path computed using the coverage map ω_{O_s}. For a path Π and for a strategy $s' \in \mathcal{S}$, let $c_{s'}(\Pi)$ denote its cost with respect to the coverage map $\omega_{O_{s'}}$.

Figure 11 illustrates the average $\beta_{s_1 s_2}$ of the visibility ratios, i.e. $\beta_{s_1 s_2} = \frac{1}{10} \sum_i^{10} c_{s_1}(\Pi_{s_2}^i)/c'(\Pi_{s_2}^i)$, for all pairs of strategies s_1 and s_2 of $\mathcal{S}$, grouped by s_2. That is, we fix s_2, and report $\beta_{s_1 s_2}$ for all $s_1 \in \mathcal{S}$. A lower value of $\beta_{s_1 s_2}$ indicated that paths computed using strategy s_2 are highly occluded under strategy s_1 as well, so a strategy s_2 is universally effective if $\beta_{s_1 s_2}$ is small for all $s_1 \in \mathcal{S}$. On the Afghanistan data case, the hybrid strategy seems to be the most effective strategy. However, for the Ft. Leonard Wood dataset the topology data strategy appears to be slightly better. In both cases top seems to be the least effective and the topology and coverage strategies perform better than the random strategy.

6. CONCLUSION

In this paper we presented a model for computing highly occluded paths over a terrain with or without knowledge of observers on the terrain. We described an efficient algorithm for computing approximate visibility maps and demonstrated its performance experimentally. We presented several different observer selection strategies and demonstrated the effectiveness of topology- and coverage-based placement. Finally, we showed that assuming a hybrid strategy using both topology- and coverage-based placement gives highly occluded paths regardless of an adversary's strategy.

We are currently working on making our algorithm scalable to even larger terrains. In particular, we are investigating using more sophisticated visibility-preserving simplification algorithms while still allowing us to tune the simplification to each individual observer. Furthermore, our present algorithm assumes that $\mathbb{Q}$ and $\mathcal{T}$ fit in memory, an unrealistic assumption for truly massive terrain. External quad tree data structures exist but external algorithms for computing shortest paths are not known. We are developing external memory algorithms for computing the path after having computed the approximate visibility maps for every observer.



(a) (b)

Figure 11. Cost of paths using different guessed strategies with 20 observers on (a) Afghanistan terrain (b) Ft. Leonard Wood terrain

References

[1] P. K. Agarwal, L. Arge, and K. Yi, I/O-efficient batched union-find and its applications to terrain analysis, *ACM Trans. Algorithms* 7 (2011), Article 11.

[2] L. Aleksandrov, H. Djidjev, A. Maheshwari, and J.-R. Sack, An approximation algorithm for computing shortest paths in weighted 3-d domains, *Discr. Comput. Geom.*, 50 (2013), 124–184.

[3] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, Approximation algorithms for geometric shortest path problems, *Proc. 32nd ACM Sypos. Theory. Comput.*, 2000, pp. 286–295.

[4] C. Cai and S. Ferrari, Information-driven sensor path planning by approximate cell decomposition, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 39 (2009), 672–689.

[5] J. Chen and Y. Han, Shortest paths on a polyhedron, *Int. J. Comput. Geometry Appl.*, 6 (1996), 127–144.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer, 2010.

[7] H. Edelsbrunner and J. Harer, *Computational Topology: An Introduction*, American Mathematical Society, 2010.

[8] W. R. Franklin, M. Inanc, Z. Xie, D. M. Tracy, B. Cutler, and M. V. A. Andrade, Smugglers and border guards: the geostar project at rpi, *Proc. ACM-GIS*, 2007, pp. 30:1–8.

[9] M. Hielsberg, R. Tsai, P. Guo, and C. Chen, Visibility-based urban exploration and learning using point clouds, *Proc. IEEE/RSJ Intl. Conf. Intell. Robots Sys.*, 2013.

[10] K. Klein and S. Suri, Capture bounds for visibility-based pursuit evasion, *Proc. 29th Sympos. Comput. Geom.*, 2013, pp. 329–338.

[11] Y. Landa and R. Tsai, Visibility of point clouds and exploratory path planning in unknown environments, *Commun. Math. Sci.*, 6 (2008), 881–913.

[12] S. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[13] J. Lee and D. Stucky, On applying viewshed analysis for determining least-cost paths on digital elevation models, *Intl. J. Geog. Info. Sci.*, 12 (1998), 891–905.

[14] M. Lu, J. Zhang, P. Lv, and Z. Fan, Max/min path visual coverage problems in raster terrain, *Computer-Aided Design and Computer Graphics*, 2007, pp. 497–500.

[15] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney, *Level of Detail for 3D Graphics*, Elsevier Science Inc., 2002.

[16] L. Lulu and A. Elnagar, A comparative study between visibility-based roadmap path planning algorithms, *Proc. IEEE/RSJ Intl. Conf. Intell. Robots Sys.*, 2005, pp. 3263–3268.

[17] M. S. Marzouqi and R. A. Jarvis, New visibility-based path-planning approach for covert robotic navigation, *Robotica*, 24 (2006), 759–773.

[18] J. S. B. Mitchell, Approximating watchman routes, *Proc. 24th ACM-SIAM Sympos. Discrete Algo.*, 2013, pp. 844–855.

[19] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.

[20] M. Rao, G. Dudek, and S. Whitesides, Minimum distance localization for a robot with limited visibility, *Proc. IEEE Intl. Conf. Robotics Automat.*, 2005, pp. 2438–2445.

[21] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, 2005.

[22] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, 2003.

[23] T. Siméon, J.-P. Laumond, and C. Nissoux, Visibility-based probabilistic roadmaps for motion planning, *Advanced Robotics*, 14 (2000), 477–493.

[24] K. R. Varadarajan and P. K. Agarwal, Approximating shortest paths on a nonconvex polyhedron, *SIAM J. Comput.*, 30 (2000), 1321–1340.

[25] Wikipedia. Art gallery problem — wikipedia, the free encyclopedia, 2013. [Online; accessed 1-July-2013].

[26] C. Zheng, H. Yin, J. Li, and M. Lu, A particle swarm optimization algorithm for least visual path problem in raster terrain, *Proc. Intl. Conf. Intell. Comput. Bio-Medical Instr.*, 2011, 228–231.