Representing the Visibility Structure of a Polyhedral Terrain through a Horizon Map

Leila De Floriani Paola Magillo
Department of Information and Computer Science (DISI)
University of Genova
Viale Benedetto XV, 3
16132 Genova (Italy)
Tel.: +39-10-3538033
Fax: +39-10-3538028
Email: {DEFLO, MAGILLO} at DISI.UNIGE.IT

Keywords: Digital terrain models, Visibility, Data structures, Horizons

Short title: A horizon-based visibility map for terrains

Send correspondence to: Leila De Floriani
Tel.: +39-10-3538033
Fax: +39-10-3538028
Email: DEFLO at DISI.UNIGE.IT

Representing the Visibility Structure of a Polyhedral Terrain through a Horizon Map

Abstract

We present a model for describing the visibility of a polyhedral terrain from a fixed viewpoint, based on a collection of nested horizons. We briefly introduce the concepts of mathematical and digital terrain models, and some background notions for visibility problems on terrains. Then, we define horizons on a polyhedral terrain, and introduce a visibility model, that we call the *horizon map*. We present a construction algorithm and a data structure for encoding the horizon map, and show how it can be used for solving point visibility queries with respect to a fixed viewpoint.

1 Introduction

Describing a terrain through visibility information has a variety of applications, such as geomorphology, navigation and terrain exploration. Problems which can be solved based on visibility are, for instance, the computation of the minimum number of observation points needed to view a region, or the computation of optimal locations for television transmitters (Cole and Sharir 1989, Goodchild and Lee 1989, Lee 1991).

A terrain can be mathematically modeled as a surface described by a continuous function $z = \phi(x, y)$, defined over a connected, not necessarily convex, domain D. Two points on a terrain are considered mutually visible when they can be joined by a straight-line segment lying above the terrain. In practice, a mathematical terrain model is approximated through a digital model, built on the basis of a finite set of points belonging to the terrain. Visibility algorithms developed in the computational geometry literature usually operate on polyhedral digital models, i.e., planar-faced approximations of a terrain (Mulmuley 1989, Agarwal and Sharir 1986, De Berg et al. 1991, Katz et al. 1991, Overmars ans Sharir 1992, Reif and Sen 1988). Visibility algorithms operating on gridded DTMs are also available in GIS commercial packages. Such algorithms, though simple in structure and implementation, are not necessarily efficient and accurate: the visibility information they produce is not generally accurate because it is computed by using the grid cells as visibility units and by assuming arbitrary simplifications for the surface defined over each cell (Lee 1991); see also (Fisher 1993) for a discussion of uncertainty in information obtained by exsiting grid visibility algorithms.

In this paper, we address the problem of representing the visibility of a polyhedral terrain from a fixed viewpoint, to efficiently solving point visibility queries, i.e, to testing whether a given candidate point is visible from the viewpoint. We propose a solution based on a nested horizon structure.

Intuitively, the *i-th horizon* provides, for any radial direction, the i-th obstacle which blocks the view of an observer located at the viewpoint. We show that it is not necessary to store each single horizon for representing the visibility of a terrain, but only the segments forming the horizons (without taking into account which order of horizon they belong to). The horizon map introduced here consists of a decomposition of the $x - y$ plane defined by the projections of the terrain edges forming the horizons. Visibility maps based on horizons are described in (De Floriani and Magillo 1993).

We propose an algorithm for building the horizon map of a polyhedral terrain, a data structure to encode such a map as well as an algorithm to solve point visibility queries on it. The proposed approach has a spatial complexity of $O(d)$, for a polyhedral terrain in which the horizons are composed of d segments, and allows answering a point visibility query in $O(\log d)$ time.

In the computational geometry literature, various approaches exist for solving the point visibility problem either for a polyhedral terrain model or for a generic polyhedral scene in $\mathbb{E}^3$. In the most general method, valid for a generic three-dimensional scene, the problem is reduced to point location in the visible image of the scene (Lee and Preparata 1977, Kirpatrick 1983, Preparata and Shamos 1985, Sarnak and Tarjan 1986). Another method (Cole and Sharir 1989), specific for terrains, uses a tree of partial horizons (which are different from nested horizons defined here). We compare our solution with the above two methods, showing its validity in several practical cases.

The remainder of the paper is organized as follows. In Section 2, we introduce the concepts of mathematical and digital terrain model, and we provide a few basic definitions for visibility problems on terrains. In Section 3, we present an overview of the existing approaches for solving point visibility from a fixed viewpoint on a polyhedral terrain. In Section 4, we define nested horizons, first in the simplified case of a one-dimensional section of the terrain, and then in the two-dimensional case. In Section 5, we define the horizon map, a horizon-based visibility map for a polyhedral terrain. In Section 6, we consider an efficient implementation of the horizon map, and illustrate the related algorithms for computing the visibility of a candidate point, and for extracting a nested horizon sequence in a given direction. In Section 7, we propose a radial-sweep algorithm for computing the horizon map encoded as described in Section 6. Finally, Section 8 contains some concluding remarks.

2 Background

A natural terrain is mathematically modeled as a continuous surface in $\mathbb{E}^3$, which is the graph of a bivariate real-valued function $z = \phi(x, y)$, defined over a connected subset D of the $x - y$ plane, called the *domain* of the surface. Of course, this involves that a natural terrain must undergo some simplifications in order to be mathematically modeled: for example, a cave cannot be represented. Thus, a *Mathematical Terrain Model* (MTM), or, simply, a *terrain*, can be defined as a pair $\mathcal{M} \equiv (D, \phi)$.

In practice, *Digital Terrain Models* (DTMs) are used, which provide a discrete approximation of a terrain, built on the basis of a finite set $\mathcal{S}$ of data points, sampled on the surface. A *Digital Terrain Model* (DTM) built on $\mathcal{S}$ consists of a pair $\mathcal{D} \equiv (\Sigma, \Phi)$, where:

(1) Σ is a subdivision of a connected (not necessarily convex) plane domain, having the set of points $\{(x, y) \mid (x, y, z) \in \mathcal{S}\}$ as its vertices;

(2) Φ is a family of bivariate continuous functions, such that every function $\phi_i \in \Phi$ is defined over a closed region $f_i \in \Sigma$, and interpolates the elevations of the data points whose projections are the vertices of f_i (usually, linear or bilinear interpolating functions are used);

(3) for every pair of adjacent regions f_i and f_j, functions ϕ_i and ϕ_j assume the same value on the edge shared by f_i and f_j.

The graph of each function ϕ_i on region f_i will be called a *face* of the DTM; we will also use the terms *edge* and *vertex* of a DTM, denoting the graph of ϕ_i restricted to an edge and to a vertex, of f_i, respectively. Since a plane subdivision with n vertices is composed of $O(n)$ edges and regions, the spatial complexity of a DTM with n vertices is a linear function of n. *Polyhedral Terrain Models* (PTMs) are DTMs characterized by linear interpolating functions. Note that, for a polyhedral terrain, condition (3) is redundant, since it is implied by condition (2).

Points forming the data set $\mathcal{S}$, on which a DTM is built, can be either distributed on a regular grid, or irregularily sampled. Existing DTMs can be classified into *Regular Square Grids* (RSGs), and *Triangulated Irregular Networks* (TINs). RSGs are built from a regular distribution of points and characterized by a domain subdivision consisting of a regular rectangular grid. Since four points do not generally define a plane, RSGs are not necessarily polyhedral models: each function f_i is either a piecewise linear or a quadratic function obtained by linear interpolation along the edges of the square cell. TINs are characterized by a triangulation of the domain and by linear interpolating functions. They are PTMs, and, moreover, can be built from any distribution of sampled points.

In order to obtain an efficient terrain description, points should be sampled widely in rough areas, and sparsely in flat ones, to reflect the varying roughness of the terrain; moreover, it is useful to include in a surface model some special points (peaks, pits, or passes) or lines (ridges, rivers or valleys). Despite of its simplicity, the fixed regular structure makes RSGs not adaptive to terrain features. On the contrary, the irregular structure makes TINs capable of representing a terrain with accuracy using a smaller amount of data. Often, a Delaunay triangulation (Preparata and Shamos 1985) is used as a domain subdivision for a TIN, because of its good behavior in numerical interpolation (intuitively, a Delaunay triangulation avoids long and thin triangles). Note that an RSG can always be transformed into a TIN, by splitting each rectangular cell into two right triangles.

Given a mathematical terrain model $\mathcal{M} \equiv (D, \phi)$, two points P_1 and P_2 are said to be mutually *visible* if, for every point $Q \equiv (x, y, z)$ belonging to the open segment P_1P_2, either $(x, y) \notin D$, or $z > \phi(x, y)$. In other words, P_1 and P_2 are visible when the straight-line segment joining them lies above the terrain. Note that, according to the above definition, segment $P_1 - P_2$ is allowed to touch the surface at its two endpoints, but not to be tangent to the surface at some internal point. This is just an arbitrary choice: all the definitions, concepts and algorithms illustrated in the remainder of the paper could be in reformulated according to an alternative definition of visibility, in which condition "$z > \phi(x, y)$" is replaced by "$z \geq \phi(x, y)$", thus allowing two points to be mutually visible even if the segment joining them is tangent to the surface at some internal point. When the terrain is approximated by a DTM, we consider two points as visible if the above property is verified for every face of the surface decomposition.

In the remainder of the paper, we will consider as *viewpoint* a fixed point $V \equiv (x, y, z)$, lying on or above the terrain (i.e., $(x, y) \in D$ and $z \geq \phi(x, y)$). For simplicity, we will also denote with $\bar{o}$ the projection on the $x - y$ plane of a generic geometric entity o in the 3D space. We use the term *visual ray* to indicate a generic ray emanating from V. Given a spherical coordinate system centered at V, a visual ray r is identified by a pair (ρ, θ), where ρ is the angle between $\bar{r}$ and the positive x-axis outcoming from V, and θ is the angle between r and $\bar{r}$ (see Figure 1). We also call a *viewsphere* any sphere centered at V and large enough to contain the whole terrain inside. Intuitively, the viewsphere is used as a "projection screen" for the visible image of the terrain from the viewpoint.

The visible image of a scene (possibly a terrain) is a partition of the viewsphere into maximally connected regions such that only one face of the scene is visible in each region. Each region is labeled with the face visible inside it. The problem of computing the visible image of a scene is generally referred as the *Hidden Surface Removal* (HSR) problem. The general quadratic upper bound to the space complexity of the visible image (Devai

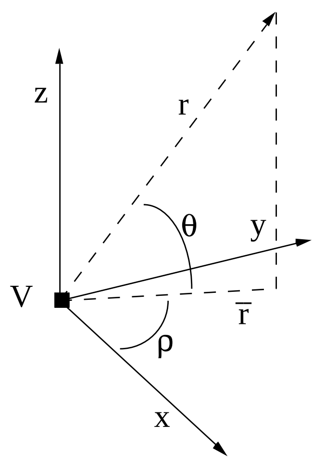


Figure 1: The two radial coordinates (ρ, θ), which identify a visual ray r in a spherical coordinate system centered at the viewpoint V.

1986, Schmitt 1981) applies also to a polyhedral terrain, thus giving a worst-case space requirement of $O(n^2)$ for a PTM with n vertices.

3 State of the Art

Testing whether a given candidate point Q is visible from a viewpoint V on a PTM can be done in a straightforward way by considering the edges of subdivision Σ which are intersected by the $x-y$ projection of segment $\vec{VQ}$, and checking, for each of such edges, whether or not the corresponding terrain edge hides Q from V. This simple approach takes $O(n)$ time for a generic PTM with $O(n)$ edges, while it requires only $O(\sqrt{n})$ time on a triangulated RSG, since each line of sight can cross at most grid edges $O(\sqrt{(n)})$. If the viewpoint V is fixed and a-priori known, better performances can be achieved by performing a preprocessing on the terrain.

The problem of efficiently solving point visibility queries on a PTM, related to a predefined viewpoint V, is commonly reduced to the more general problem of *ray shooting* from a fixed viewpoint. An instance of the ray shooting problem consists of finding the first face of a terrain which is hit by a visual ray emanating from V. The visibility of a candidate point Q can be determined by answering a ray shooting query related to the visual ray outcoming from V and passing through Q: point Q is visible from V if and only if Q and V lie on the same side with respect to the reported terrain face.

An efficient solution to ray shooting queries is usually obtained by preprocessing the terrain (or, in general, the input scene) with respect to the given viewpoint into a suitable data structure. For a PTM, a worst-case efficient ray shooting data structure has been proposed by Cole and Sharir (Cole and Sharir 1989): such structure consists of a balanced binary tree, built on the basis of a front-to-back order of the terrain edges with respect to the given viewpoint. The root of the tree is associated with the whole set of terrain edges. If $\mathcal{E}$ is the set of edges associated with a node u, then the half of the edges in $\mathcal{E}$, which lie closer to V, is associated with the left child of u, and the other half with the right child of u. Every parent node stores the upper rim of the terrain, restricted to the edge set associated with its left child. The *upper rim* of a terrain (elsewhere called the *horizon*), with respect to a viewpoint V, is a function r, which, for every radial direction ρ, provides the farthest point from V, lying on the terrain in direction ρ, and visible from V. In the terminology of Section 4, $r(\rho)$ corresponds, for every ρ, to the horizon of highest order defined in direction ρ. On a polyhedral terrain, every point of the upper rim belongs to an edge of the PTM. Thus, we can collect the maximal radial intervals in which point $r(\rho)$ belongs to the same terrain edge, and represent the upper rim as an ordered list of radial intervals, each labeled with the terrain edge corresponding to the upper rim inside it.

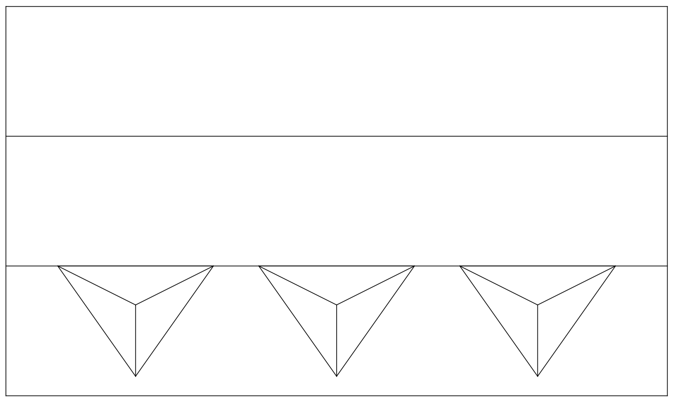
The combinatorial complexity of the upper rim of a polyhedral terrain with $O(n)$ edges is $O(n\alpha(n))$ in the worst-case (Hart and Sharir 1986), where $\alpha(n)$ is the inverse Ackermann's function. The problem reduces to the computation of the upper envelope, i.e., pointwise maximum, of the set of segments in the ρ-θ plane, obtained by projecting the terrain edges onto the viewsphere. The worst-case size of Cole and Sharir's structure is $O(n\alpha(n)\log n)$ for a PTM of size n. It can be computed in $O(n\alpha(n)\log n)$ time, and the time required for solving a ray shooting query (and, thus, a point visibility query) is equal to $O(\log n)$.

The size of the structure proposed by Cole and Sharir is determined just by the input PTM, without regarding its visibility situation with respect to the given viewpoint. Though worst-case efficient, it does not exploit the good properties that a PTM may have with respect to the selected viewpoint in practical cases (e.g., a terrain with many edges, only few of which visible from V). A data structure whose performance depends on the size of the visible image of the terrain rather than on the size of the terrain itself better fits user's expectations.

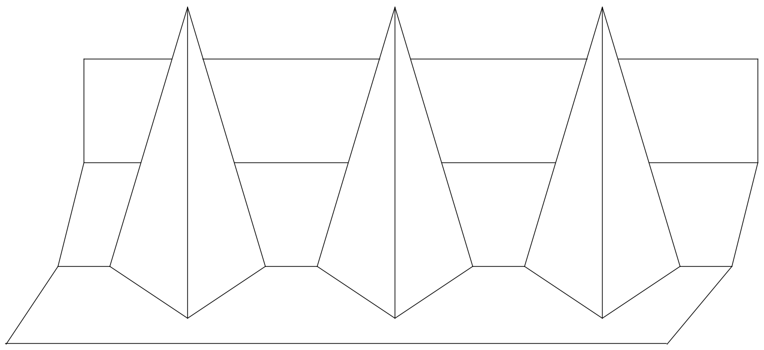
A general method, which fits the above "image-sensitive" requirement, reduces the ray shooting problem (and thus point visibility) to point location in the visible image of the scene with respect to the given viewpoint. Any visual ray emanating from V identifies a point (ρ, θ) on the viewsphere: answering a ray shooting query thus reduces to finding the region of the visible image which contains the point identified by the query ray.

The worst-case size of the visible image of a scene with $O(n)$ edges is equal to $O(n^2)$ (Devai 1986, Schmitt 1981). This bound holds also for the size v of the visible image of a PTM, as shown by Cole and Sharir (Cole and Sharir 1989). It is possible to construct a polyhedral terrain with $O(n)$ edges such that there are $O(n^2)$ *visible intersections* between projected edges on the viewsphere (see Figure 2, taken from (Cole and Sharir 1989)). In practical cases, however, v can be much smaller than $O(n^2)$. The worst-case terrain in (Cole and Sharir 1989) is characterized by long edges which project across the whole visible image. In practically used DTMs, such as regular DTMs, based on a $\sqrt{n} \times \sqrt{n}$ grid, or irregular TINs, built on a Delaunay triangulation, long and thin regions, and thus edges which project across the visible image, are avoided. Thus, it is unlikely to have $O(n^2)$ intersections between projected edges on the viewsphere. Moreover, not all the intersections may be visible, as also remarked in (Overmars ans Sharir 1992, Reif and Sen 1988). For example, high mountains near to the viewpoint can hide a large portion of the terrain behind. In summary, v is likely to be smaller than $O(n\alpha(n)\log n)$ for a large set of PTMs.

In the literature, several algorithms exist which compute the visible image of a PTM

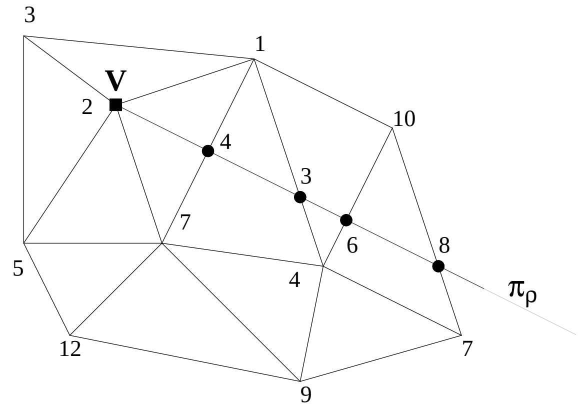


(a)



(b)

Figure 2: The worst-case construction for a terrain with $O(n)$ edges and an $O(n^2)$ visible image: (a): the terrain, seen projected onto the $x - y$ plane; (b) the corresponding visible image.



(a) (b)

Figure 3: (a) A polyhedral terrain model and a viewpoint V, and (b) the one-dimensional section of the same terrain in a radial direction ρ from V.

either in a worst-case optimal (McKenna 1987, Schmitt 1981) or in an output-sensitive way (Mulmuley 1989, Agarwal and Sharir 1986, De Berg et al. 1991, Katz et al. 1991, Overmars ans Sharir 1992, Reif and Sen 1988). By using, for example, the point location method of Sarnak and Tarjan (Sarnak and Tarjan 1986) (see also Section 6), the visible image can be preprocessed in $O(v \log v)$ time into a data structure of size $O(v)$, which allows a point location query (and, thus, a point visibility query) to be answered in $O(\log v)$ time.

4 Horizons on a Terrain

In this Section, we define the horizons on a terrain with respect to a predefined viewpoint V.

For simplicity, we first examine the case of a one-dimensional section of the terrain. Let $\mathcal{M} \equiv (D, \phi)$ be a mathematical terrain model. Given a radial direction $\rho \in [0, 2\pi)$, we consider the section of $\mathcal{M}$ obtained by intersecting $\mathcal{M}$ with the vertical half-plane Π_ρ, locus of the visual rays emanating from V in direction (ρ, θ) for $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ (see Figure 3). Such section will be called the *section of* $\mathcal{M}$ *in direction* ρ, and denoted by $\mathcal{M}_\rho$. $\mathcal{M}_\rho$ identifies a continuous line on Π_ρ, that we consider as oriented from left to right, taking as its left endpoint the vertical projection of V.

Informally, any horizon of V in direction ρ is a point on the terrain section $\mathcal{M}_\rho$, that blocks the view of an observer located at V and looking in direction ρ. The i-th horizon of V is the i-th such obstacle along $\mathcal{M}_\rho$.

A point P_ρ, belonging to the section $\mathcal{M}_\rho$, is a *transition from visible to invisible* if the points of $\mathcal{M}_\rho$ in the left neighborhood of P_ρ are visible from V, and the ones on the immediate right of P_ρ are not visible. The left endpoint of $\mathcal{M}_\rho$ is also considered a transition from visible to invisible if the points of $\mathcal{M}_\rho$ in its (right) neighborhood are invisible from V; conversely, the right endpoint of $\mathcal{M}_\rho$ is considered a transition from visible to invisible if the points of $\mathcal{M}_\rho$ in its (left) neighborhood are visible from V. The *i-th horizon* of V on

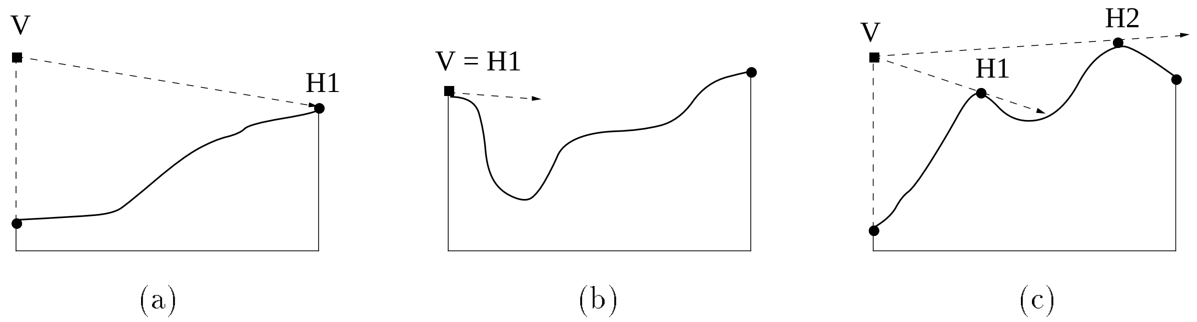


Figure 4: Horizons of V in a direction ρ: in (a), H_1 is the right endpoint and, in (b), it is the left endpoint of the terrain section (the drawn visual ray is tangent to the terrain); in both cases (a) and (b), H_i is undefined for any $i > 1$. In (c), H_1 is the first transition from visible to invisible, H_2 is the second such transition, and H_i is undefined for $i > 2$.

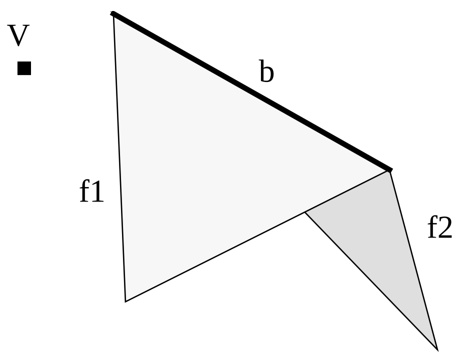


Figure 5: Faces f_1 and f_2 are face-up and face-down, respectively; b is a blocking edge.

$\mathcal{M}$ in direction ρ, denoted by H_i^ρ, is the point, belonging to $\mathcal{M}_\rho$, which corresponds to the i-th transition from visible to invisible encountered while walking along $\mathcal{M}_\rho$ from left to right (see Figure 4 for some examples of horizon configurations on a terrain section). The integer value i is called the *order* of the i-th horizon.

The first horizon H_1^ρ is always defined, since there is at least one transition from visible to invisible on $\mathcal{M}_\rho$. In different radial directions a different number of horizons may be defined.

The *i-th horizon* of V on a terrain $\mathcal{M}$ is a partial function h_i that, for every radial direction ρ, provides, if defined, the i-th horizon of V in direction ρ. The first horizon h_1 is a total function on $[0, 2\pi)$, while, in general, we have that the domain of h_{i+1} is contained in that of h_i. We call *nested horizons* the family of functions $[h_1, h_2, \ldots, h_{last}]$, where $last = \max\{i \mid h_i$ has a non-empty domain $\}$.

Based on the above definitions, we provide a characterization of horizons for a polyhedral terrain model $\mathcal{D} \equiv (\Sigma, \Phi)$, and estimate their space complexity as a function of parameters related to the size of $\mathcal{D}$. Given a viewpoint V, a face of $\mathcal{D}$ is said to be *face-up* with respect to V, if V lies in the upper half-space defined by the plane which contains the face, and *face-down*, if V lies in the lower half-space. An edge e of $\mathcal{D}$ is a *blocking edge* with respect to V if the face adjacent to e and looking towards V is face-up, and the face adjacent to e, opposite to V, is face-down (see Figure 5). We also call blocking edges those edges of the boundary of the terrain whose adjacent face either looks towards V and is face-up, or it looks away from V and is face-down.

On a polyhedral terrain, any i-th horizon is formed by visible points on blocking edges: every visible portion b of a blocking edge defines an obstacle to the view of an observer located at V, and makes the portion of terrain immediately behind b invisible.

For every horizon h_i, we collect the maximal radial intervals such that point $h_i(\rho)$ belongs to the same blocking edge e_j, and represent the i-th horizon as an ordered list $L = [I_1, \ldots, I_k]$ of maximal radial intervals, labeled with edges of the terrain: if a radial interval I_i has label e_j, then for all $\rho \in I_i$, point $h_i(\rho)$ is defined, and belongs to blocking edge e_j. In the following, we will use the term *horizon components* to indicate the maximal visible portions of the blocking edges of a PTM.

Worst-case upper bounds to the spatial complexity of the horizons for a polyhedral terrain model can be expressed as a function of the number n of vertices of the terrain, and of other two parameters: the number d of horizon components, and the maximum number p of horizons defined in a radial direction ρ. Clearly, the number d of maximal visible portions of blocking edges is less or equal than the global number of maximal visible portions of terrain edges, which is the same as the size v of the visible image of the PTM. Since $v = O(n^2)$, the same worst-case bound holds for d, even if in practical cases d can be much smaller than v (which in turn can be much less than $O(n^2)$, as pointed out in Section 3). In a polyhedral terrain, horizons are formed by visible portions of blocking edges; unfortunately, any of such portions may be subdivided, for different radial directions, into different orders of horizons. The maximum number p of horizons in a single radial direction is $O(n)$ in the worst case, because in any direction there can be at most as many horizons as terrain edges. Thus, the size of all horizons is $O(dp)$, that is, $O(n^3)$ in the worst case.

In Sections 5 and 6, we will show that simply storing the horizon components, i.e., the segments forming the horizons, without the information about the order of horizon they belong to, is sufficient to represent the visibility of a PTM, thus keeping the storage requirements down to $O(d) = O(n^2)$.

5 The Horizon Map

In this Section, we consider a polyhedral terrain model $\mathcal{D} \equiv (\Sigma, \Phi)$, and define a visibility structure for PTMs, called the horizon map, which consists of a partition of the $x-y$ plane, induced by nested horizons.

Let us consider, for simplicity, a one-dimensional section $\mathcal{D}_\rho$ of $\mathcal{D}$. We recall that $\bar{o}$ denotes the projection on the $x-y$ plane of a generic geometric entity o in the 3D space. Let l_ρ be the horizontal ray obtained by intersecting Π_ρ (the half-plane defined by visual rays emanating from V in direction ρ) with the $x-y$ plane. We define the (*one-dimensional*) *horizon map* of $\mathcal{D}$ in direction ρ as the partition of l_ρ into intervals defined by the vertical projections of the points $H_1^\rho, \ldots, H_{k_\rho}^\rho$, corresponding to the horizons of V in direction ρ.

Any interval of the horizon map in direction ρ (except, at most, for the first one) has the projection $\overline{H_i^\rho}$ of a horizon as its left endpoint, and it will be called the *influence interval*

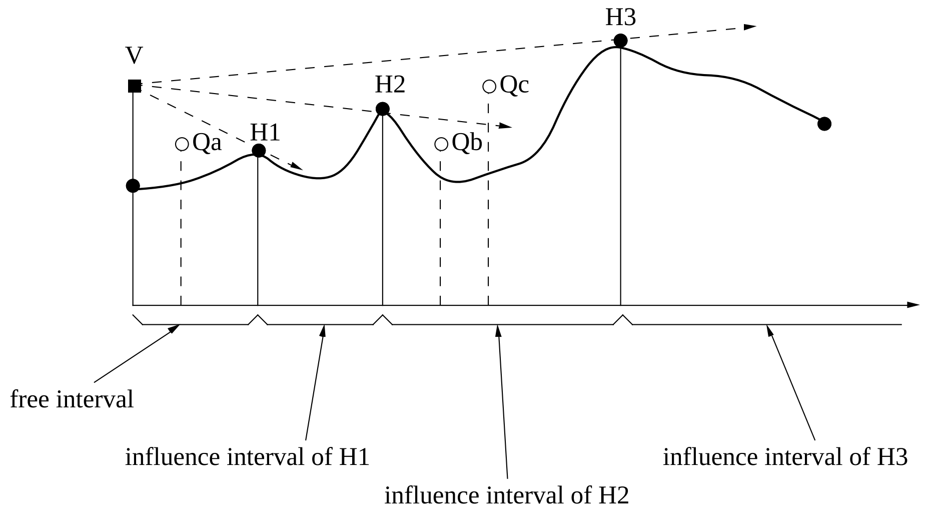


Figure 6: Solution of point visibility queries by using the nested horizon map in direction ρ: any point lying before the first horizon (as Q_a) is visible from V; the visibility of a point lying immediately behind a horizon H_i (e.g., Q_b and Q_c, which are behind the second horizon H_2) only depends on its position with respect to the visual ray $V - H_i$ (Q_b is not visible and Q_c is visible).

of H_i^ρ. If point H_i^ρ belongs to a blocking edge b, then the *influence interval* of H_i^ρ will be also called the influence interval of b. The first interval of the horizon map in direction ρ, if not yet the influence interval of H_1^ρ, will be called the *free interval.*

Given a candidate point Q_ρ, belonging to Π_ρ, the visibility of Q_ρ, can be determined by locating point $\overline{Q_\rho}$ in the horizon map in direction ρ, since the following property holds (see Figure 6):

- If $\overline{Q_\rho}$ lies in the free region, then Q_ρ is visible from V.
- If there exists $i > 0$ such that $\overline{Q_\rho}$ lies in the influence interval of a horizon H_i^ρ, then Q_ρ is visible from V if and only if Q_ρ lies above the visual ray emanating from V and passing through H_i^ρ.

In other words, point Q_ρ is visible from V if and only if the visual ray connecting V to Q_ρ is not blocked by the horizon of highest order lying before Q_ρ.

In the special case when $V \equiv H_1^\rho$ (i.e., for a polyhedral terrain, if V lies on the terrain), then the free interval in direction ρ is empty, and we take as visual ray passing through V and $H_1^\rho \equiv V$ the one tangent to the terrain face which lies on the immediate right of V (see Figure 7).

Now, let us consider the two-dimensional case, in which a polyhedral terrain model $\mathcal{D} \equiv (\Sigma, \Phi)$ and a viewpoint V are given. We project the horizon components of $\mathcal{D}$ with respect to V onto the $x - y$ plane, and consider the resulting set of semi-disjoint segments in $\mathbb{E}^2$. For every extreme point P of such segments, we draw a radial straightline outwards from P until it reaches another segment (if any). The resulting decomposition of the $x - y$ plane is called the *horizon map* (see Figure 8 (b)).

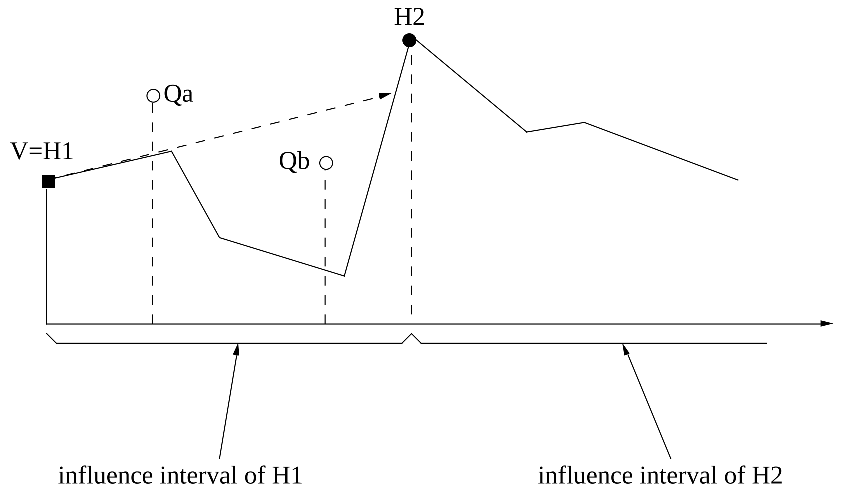


Figure 7: Computing the visibility of points lying in the influence interval of H_1 when $H_1 \equiv V$: query point Q_a is visible, Q_b is not visible.

The regions of the horizon map may have an arbitrary number of edges, and are, in general, not convex. Any region of the horizon map (except one) corresponds to a horizon component b and is formed by the union of the influence intervals of b for the range of those radial directions ρ for which the influence interval of b exists. Thus, we call it the *influence region* of b (in Figure 8 (c), the influence region of b is coloured in light gray). The influence interval of a horizon component b occurs for a range of consecutive values of ρ, and, for different values of ρ within that range, it may extend more or less far beyond $\bar{b}$. The influence region of b is formed by a union of quadrilaterals with two transversal edges (one corresponding to a fragment of $\bar{b}$ and the other one to a fragment of another projected horizon component) and two radial edges (corresponding to radial segments introduced during the construction). One region of the horizon map is formed by the union of all the free intervals, while varying ρ in $[0, 2\pi)$, and it is called the *free region* (in Figure 8 (c), the free region is coloured in dark gray). The free region consists of a union of triangles, each with two radial edges incident in $\bar{V}$ and one transversal edge, corresponding to a projected horizon component.

The visibility of a candidate point Q, whose vertical projection belongs to the influence region of a horizon component b, is only influenced by b, which means that possibly its invisibility can only be due to the shadow cast by a "curtain" (vertical trapezoid) hanging from b. Because of this property, it is possible to determine whether Q is visible from V by just testing whether Q lies above or below the plane passing through b and V (see Figure 9 for an example). All candidate points Q, whose projections lie in the free region, are visible from V.

Point visibility queries can thus be solved through point location in the *horizon map*. Intuitively, the horizon map can be regarded as a compressed version of the visible image: while the visible image of a PTM is built based on the visible portions of *every edge* of the terrain, only the visible portions of those terrain edges which are *blocking edges* with respect to V are considered in the horizon map (see Figure 8 (b) and (d)).

The space complexity of the horizon map is equal to $O(d)$, since one new vertex is

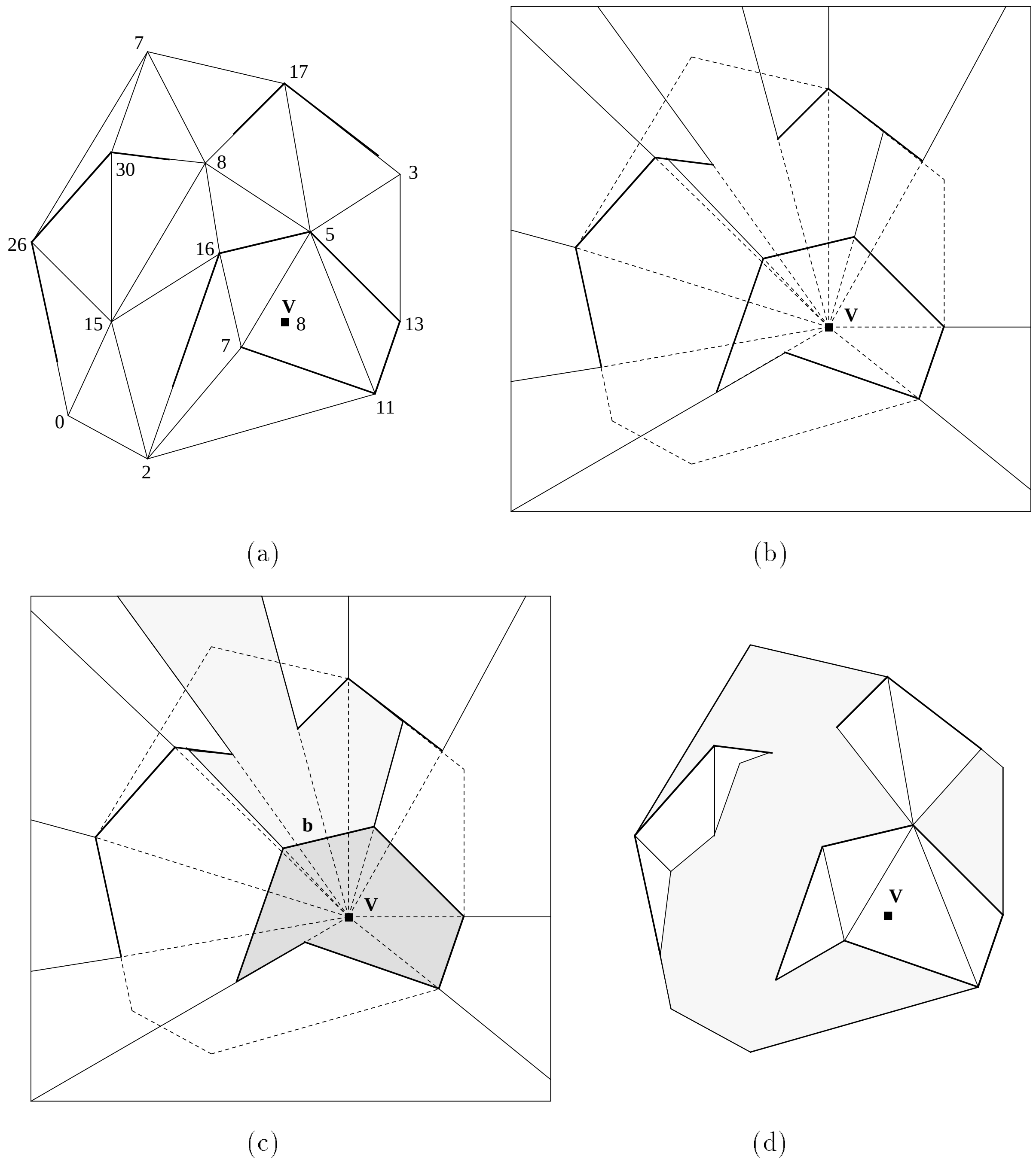


Figure 8: A PTM and a viewpoint V on it, seen projected onto the x-y plane (each vertex is labeled with its elevation); (b) the corresponding horizon map (the projections of the blocking edges are drawn in thick lines, the radial rays drawn from V and the boundary of the terrain domain are dashed); (c) the influence region of horizon component b (light gray) and the free region (dark gray); (d) the visible image, seen projected onto the x-y plane (invisible parts are shaded), with the edges taking part into the horizon map drawn as thick lines.

introduced for every endpoint of a projected horizon component. In the worst case, $d = O(v)$ (where v is the size of the visible image, in turn equal to $O(n^2)$), but it is likely to be smaller in many practical cases: in a detailed terrain representation, a single dome or valley will be represented by a network of several faces, with many non-blocking edges separating them.

6 Implementation and Use of the Horizon Map

We have shown that testing the visibility of a candidate point reduces to a point location query in the horizon map. To this aim, we encode the horizon map by using the data structure for point location proposed by Sarnak and Tarjan (Sarnak and Tarjan 1986). Given a polygonal plane subdivision Γ, with m edges, such structure allows a query time logarithmic in m by using only linear storage.

The basic idea is the following. The plane is decomposed into vertical slabs by drawing a vertical line through each vertex of Γ, and the edges of Γ are sorted from top to bottom withing each slab. A point location query, involving a point Q, is answered in $O(\log m)$ time through two binary searches, which locate the vertical slab containing Q, and the nearest edge e below Q within such slab, respectively: the region of Γ which contains Q is the region adjacent to e and lying above e (see Figure 10). Explicitly storing the ordered set of edges in each vertical slab would lead to an $O(m^2)$ data structure, since each edge of Γ can appear in $O(m)$ different slabs. Sarnak and Tarjan propose an approach which reduces the storage cost to $O(m)$.

By traversing Γ left-to-right in a sweep-line fashion, the ordered sets of edges within the vertical slabs correspond to different configurations of the sweep-line status. Sarnak and Tarjan achieve a linear space complexity by storing the "history" of such a sweep-line traversal of Γ, i.e., the left-to-right sequence of status configurations. In fact, starting from the initial configuration, we have only $2m$ constant-sized changes in the status (one insertion and one deletion for each edge of Γ) during the whole traversal. Linear storage is achieved by using a persistent binary tree to implement the sweep-line status. A data structure is said to be *persistent* if, after any update, the old version of the structure can still be accessed. A tree structure is made persistent by copying the paths which are modified at each update (details are described in Appendix A). At the end of the sweep process, the persistent tree which implements the status consists of a forest of $O(m)$ different trees having distinct roots and sharing common subtrees, with a total space complexity of $O(m)$.

For locating a point Q in Γ, we first select the root corresponding to the vertical slab containing Q, and then apply a binary search inside the corresponding tree. The preprocessing time, i.e., the cost of the sweep process for building the above point location structure is equal to $O(m \log m)$. In the case of the horizon map, $m = O(d)$, and, thus, preprocessing time, storage space and query time are $O(d \log d)$, $O(d)$ and $O(\log d)$ respectively. The point location data structure for the horizon map can be built through a *radial sweep* traversal, by using a radial ray emanating from $\bar{V}$ as sweep-line (see Figure 11). In Section 7 of this paper, we describe a radial-sweep algorithm for the horizon map itself.

The data structure obtained in this way can also be regarded as an implicit and compact

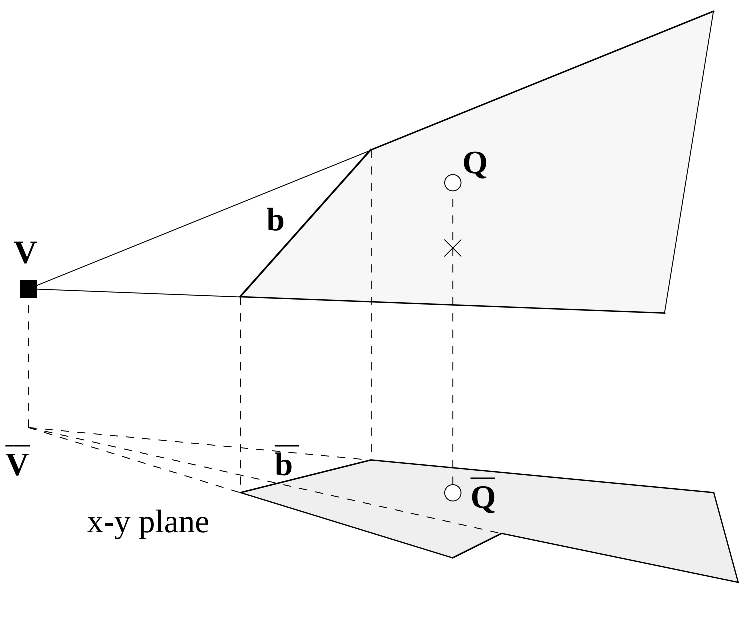


Figure 9: The vertical projection of point Q lies in the influence region of horizon component b. Q is visible from V since it lies above the plane passing through V and b.

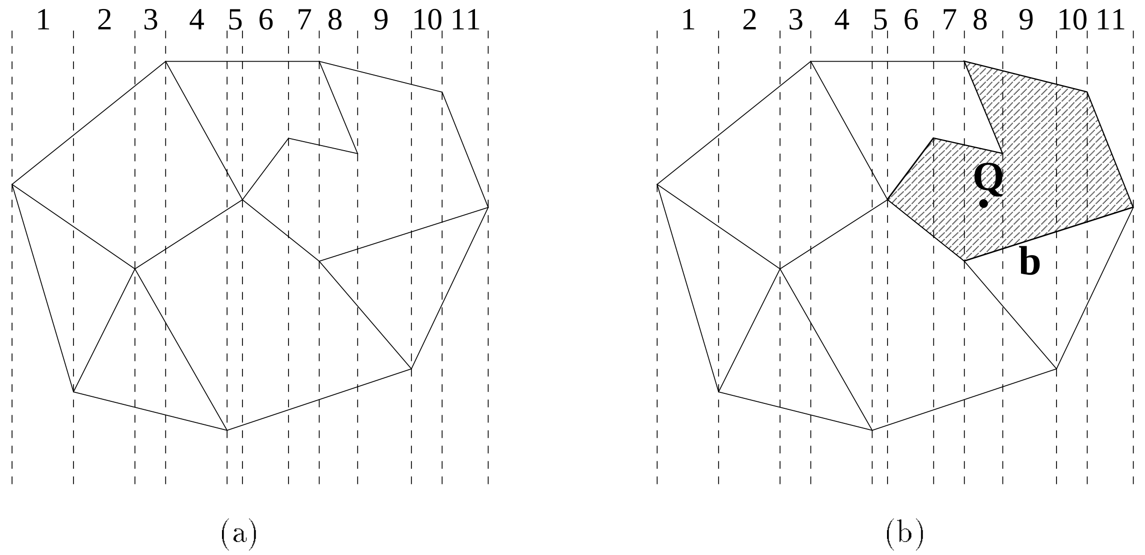


Figure 10: (a) A subdivision Γ and the corresponding decomposition of the plane into vertical slabs; (b) a visibility query: point Q lies in slab 8, the nearest edge below Q in such slab is b, the corresponding region is shaded.

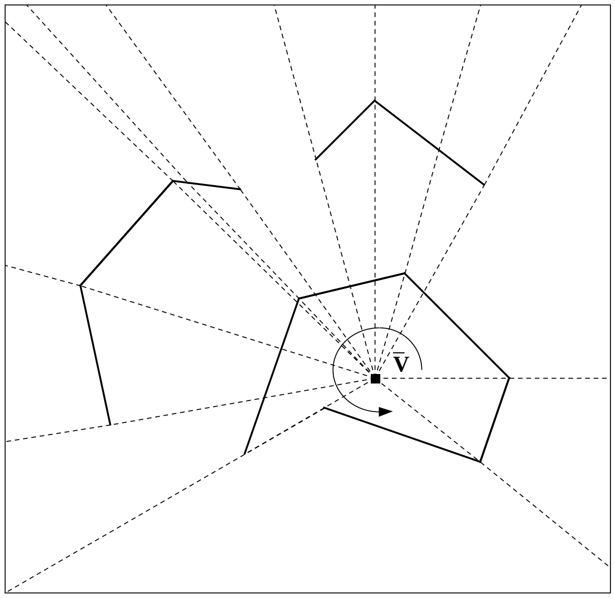


Figure 11: Radial sweep process applied to the $x-y$ projections of the horizon components of the terrain of Figure 8, and the resulting slab decomposition.

representation of the nested horizons, which use only $O(d)(=O(n^2))$ space instead of $O(dp)(=O(n^3))$. Given a generic radial direction $\rho \in [0, 2\pi)$, the sequence of all the horizons defined in direction ρ is given by the ordered sequence of blocking edges within the slab containing ρ (this is true since the data structure has been built with a radial sweep approach). The horizon sequence in a direction ρ can thus be retrieved in $O(\log d + k_\rho)$ time, where $O(\log d)$ is the time necessary for selecting the root corresponding to the proper vertical slab, and k_ρ is the number of reported horizons, which are the nodes of the selected binary tree.

7 Computing the Horizon Map

In this Section, we address the problem of computing the horizon components of a PTM. The related horizon map, encoded in the data structure introduced in Section 6, can be built starting from such components in $O(d \log d)$ time, by applying a radial sweep process to the set of segments obtained by projecting them onto the $x-y$ plane.

A possible approach is extracting the horizon components from the visible image of the terrain, computed with one of the existing HSR algorithm (see Section 3). For each edge of the visible image, corresponding to a visible portion b of a terrain edge, we test whether the terrain edge supporting b is blocking, and, in such case, b will be a horizon component. The cost of such approach is thus $O(v)$ plus the cost of computing the visible image (e.g., $O((n+v)\log n)$ by using the algorithms in Reif and Sen (1988) or Katz et al. (1991)).

If the computation of the whole visible image is not necessary for other purposes (e.g., for visualization), it seems rather time and space consuming determining the visibility of each terrain edge, including those which are not blocking. Alternatively, we can determine first the terrain edges which are blocking with respect to V, and then apply a hidden line algorithm to the set of "curtains" hanging from the blocking edges. In this way, the horizon components are obtained directly from the hidden line algorithm in $O((n' + d)\log n')$ time, where $n'(\leq n)$ is the number of blocking edges, and $d \leq v$.

In what follows, we propose an algorithm which computes the horizon components of a polyhedral terrain directly, by using a radial sweep technique. This method allows also the construction of the data structure which encodes the horizon map described in Section 6.

In the algorithm, a vertical half-plane Π, originating at the vertical line l through V, is moved radially around l. At each step, the one-dimensional section of the terrain intersected by Π is considered.

The *status* of the sweep process is represented by the blocking edges which are currently intersected by Π. Those blocking edges which are horizons in the current radial direction of Π are called *active* edges. The status also maintains both a horizontal and a vertical order of the blocking edges intersected by Π. The horizontal order is defined according to the distance of the blocking edges from V, measured on the $x - y$ plane; the vertical order is given by the order of θ-coordinate among the projections of such intersection points on the viewsphere (see Figure 12). A separated horizontal and vertical order, restricted to active edges, is also maintained. A blocking edge b is active if and only if every blocking edge preceding b horizontally ("closer" to V than b) precedes b also vertically (it is "below" b, and, thus, it does not obscure b).

The *events* of the sweep process, i.e., the radial directions ρ at which the status changes, are represented by the radial direction corresponding to intersection points between projections of blocking edges on the viewsphere, and by the radial directions corresponding to the endpoints of the blocking edges of the terrain.

During the sweep process, events are processed in a radial (e.g., counterclockwise) order around V. The algorithm needs also a preprocessing step, in which the blocking edges of the terrain are determined, and every blocking edge crossing the sweep half-plane Π at its initial position (e.g., $\rho = 0$) is split into two subsegments, lying on the right and on the left of Π, respectively. Such preprocessing requires $O(n)$ time.

At each event, the status of the sweep process is updated, according to the event type:

1) If the event is an intersection point between the projections of two blocking edges a and b (let a be before b in horizontal order), then we swap a and b in vertical order, and we update the active blocking edges. Three cases may occur:

   1.a) a was below b in vertical order, and now a is above b:
   b (if active) is deactivated, since b is now obscured by a, and cannot be a horizon anymore (see Figure 13 (a)).

   1.b) a was above b vertically, and now a is below b (in this case, b was inactive), and a was inactive:
   no changes occur, since b cannot become an horizon as some active blocking edge c exists obscuring both b and a (see Figure 13 (b)).

   1.c) a was above b vertically, and now a is below b (b was inactive), and a was active:
   b becomes active if and only if the active edge c which was immediately above

a (and now is immediately above b) comes after b in horizontal order. In fact, b is obscured by c if and only if c is nearer to V than b. Moreover, b cannot be obscured by another blocking edge c' different from c, otherwise c' would obscure c (see Figure 13 (c)).

2) If the event is a second endpoint of a blocking edge b, then we delete b from the status.

3) If the event is a first endpoint of a blocking edge b, then we insert b in the horizontal and vertical order. Let b be between two edges a and a' in vertical order. We look for the intersections between a, b and a', b, and (if they exists) we insert them among the events. Finally, b is made active if and only if the active edge c immediately above b comes after b in horizontal order (see Figure 14).

The maximal visible portions of a blocking edge b are represented by the portions of b between any two radial directions ρ_1 and ρ_2, corresponding, respectively, to an event where b becomes active, and to the next event where b is deactivated. For this purpose, we keep track, for every active edge, of the radial direction at which it became active.

During the algorithm, the size of the sweep status is bounded by $O(n)$, because, in the worst case, we have $O(n)$ blocking edges. The number of events is $O(n+k)$, where k denotes the number of intersections between pairs of blocking edges on the viewsphere. Processing an event requires $O(\log n)$ time, regardless of its type, thus leading to a $O((n+k)\log n)$ time complexity for the entire sweep process. In the worst case, $k = O(n^2)$, but it may be smaller in many practical cases, as pointed out in Section 3.

The above radial sweep algorithm can be combined with the radial sweep algorithm of Section 6, in order to organize the horizon components into the data structure of Sarnak and Tarjan. For this purpose, we also maintain the ordered set of currently active blocking edges (considered in their horizontal order) in a persistent tree. Such tree will be modified by inserting a new segment at each radial direction where a blocking edge becomes active, and by removing a segment every time a blocking edge is deactivated. Combining the two algorithms requires an additional $O(d \log d)$ time.

8 Concluding Remarks

In this paper, we have addressed the problem of describing the visibility of a polyhedral terrain, with respect to a fixed viewpoint, in a way which allows an efficient solution to point visibility queries. We have defined *nested horizons* on a terrain model, and we have introduced a visibility structure based on them, called the *horizon map*, which is a compact encoding of the visibility situation of a polyhedral terrain with respect to a predefined viewpoint. A suitable data structure has been proposed for encoding the horizon map, which allows both an efficient answer to point visibility queries and a retrieval of the sequence of horizons in a given direction.

The spatial complexity of the horizon map is not determined by the size n of the terrain itself, but it rather depends on the size of the visible image of the terrain with respect to the given viewpoint V. This feature allows a better performance in many practical cases of polyhedral terrains with many edges but only a small number of them visible from V. Moreover, the size d of the horizon map is even less than the size v of the whole visible image, since it is built based on a proper subset of the visible image, formed by the visible portions of those terrain edges which are blocking with respect to V. This leads to a further space reduction and increase in efficiency.

Our approach is specific for polyhedral terrain models. In practice, regular square grids, using smooth interpolation over rectangular cells, are widely used in GIS. We can deal with such models in two different ways. A model can be reduced to a TIN by splitting each grid cell in two right triangles and defining an interpolating plane over each triangle; the visibility data structure could be built based on such TIN. Since regular grids are usually very dense, this would generally be too expensive. A more convenient method consists of extracting from the regular grid a suitable subset of points, and building a TIN model starting from such set. Point selection can be done in such a way that the resulting TIN model approximates the original grid within a predefined tolerance (see, for instance, De Floriani 1989). Then, the proposed visibility structure can be build starting from such approximated TIN.

Future developments of the research presented in this paper are concerned with the use of horizons and of the horizon map to answer other types of visibility queries, for example, the visibility of a line segment or of a polygon. Another development is represented by the investigation of horizon based structures in the framework of hierarchical terrain models (De Floriani 1989, De Floriani and Puppo 1992).

A Appendix: Persistent binary trees (Sarnak and Tarjan 1986)

Persistent binary trees have been developed with the aim of maintaining a set S of items, when items are inserted and removed from S during the time. Each item has a key from a totally ordered set (e.g., an integer key).

Three operations are defined on set S:

- $access(x, t)$: access the set in its configuration at time t, and return the item with greatest key less or equal to x;
- $insert(i)$: insert item i (with predefined key);
- $delete(i)$: delete item i (if present).

Note that updates (insertions and deletions) can only be performed on the current version of set S, while accesses can also refer to the past versions.

A data structure for S is called *ephemeral* if only the current version of the structure can be accessed (i.e., parameter t in *access* operation must be the current time). A data structure for S is said to be *persistent* if, after any update, the old version of the structure can still be accessed (even if not modified).

A.1 The ephemeral case

An efficient ephemeral data structure for storing set S is a balanced binary tree. A balanced binary tree storing n items has a $O(\log n)$ depth and allows accesses, insertions and deletions to be performed in $O(\log n)$ time. Several versions of balanced binary trees exist. All these trees store one item per node plus some balance information; after each insertion or deletion the structure is rebalanced by performing a sequence of rotations along the path from the root to the inserted or deleted item. Here, we consider a special type of balanced binary trees, called *red-black trees*.

In red-black trees, balance information is represented as a colour red or black associated with each node, with the following constraints:

- *Missing node convention*: all missing nodes (e.g., missing children of nodes with less than two children) are regarded as black;
- *Black constraint*: all paths from the root to a missing node contain the same number of black nodes.
- *Red constraint*: the parent of any red node (different from the root) is black.

The colour constraints imply that the depth of a red-black tree with n nodes is at most $2 \log n$. Rebalancing after an insertion or deletion can be done in $O(1)$ rotations and $O(\log n)$ colour changes; moreover, the amortized number of colour changes per update is $O(1)$.

An access operation is performed as in an ordinary balanced binary tree:

- starting from the root, we compare the input key with the key of the current node;
- if the searched key is equal to the current key, then return the current node;
- if the searched key is less than the current key, we descend the left subtree; if there is no left subtree, then return a special null value;
- if the searched key is greater than the current key, then we descend the right subtree; if there is no right subtree, then return the current node.

The insertion of a new item i is performed as follows:

- We follow the access path for i until we reach a missing node. We create and attach a new *red* node v_i, containing the inserted item.
- This may violate the red constraint, since it may happens that the new red node has a red parent. In this case, we move the violation up the tree by applying the recoloring transformation of Figure 15 (a), until it cannot be applied any longer.
- If there is still a violation, then we apply suitable rotations to eliminate it, as illustrated in Figure 15 (b,c,d).

The deletion of an item i is performed as follows:

- follow the access path for i, and locate the node v_i containing the item i to be removed.
- if v_i has no left child, then skip.
- if v_i has a left child, then swap i with the item (with key less than the one of i) found by descending a left branch and then right branches until we find a node with no right child.
- Now item i is in a node with at most one child: remove the node and replace it with its child (if any). If the deleted node was black, then this operation violates the black constraint. Thus, we perform the recoloring operation of Figure 16 (a) until it does no longer apply. Then, we perform the rotation of Figure 16 (b) followed, if necessary, by one of the rotations in Figure 16 (c,d,e).

A.2 Making the structure persistent

In order to make red-black trees persistent, we must retain the old version when a new version is created by an update. Persistence can be obtained in a straight-forward method by copying the whole tree before each update, but, of course, this would be too expensive.

An alternative approach is represented by *path copying*, that is, copying only the nodes which are modified in the update. Since colours are only needed for updates, and updates always take place in the present version, the nodes which are *modified in an update* are those nodes whose content or whose child-pointers are modified, while we simply overwrite old colours at each update. Since every node is linked to its parent through a child-pointer, copying a node requires copying the entire path from the root to that node. The update time and space are $O(\log n)$ since each update involves only a single root-to-leaf path. After m updates, a persistent tree will consist of a forest of balanced binary trees, with m different roots (each representing a version), and sharing common subtrees. The m versions of a tree can require a $O(n + m \log n)$ space. The access time is $O(\log n \log m)$ where $O(\log m)$ is the time necessary to locate the root corresponding to the accessed version (assumed that roots are stored in an time-ordered array). The drawback of path copying is in its nonlinear storage space.

The design of space efficient trees is based on the idea of avoiding the copy of the entire access path at each update. A node can contain k extra childpointers in addition to the original pair. Each pointer has a time stamp. A node, which is changed in an update, is copied only if all its k pointers are already in use; otherwise, the update is performed in place, by using one of the free pointers, which is marked with the current time stamp. In the worst case, an update still involves a path copying and thus an $O(\log n)$ additional space. But, after a node has been copied once, then k updates can be performed on it without any copying. Moreover, when a node is copied, there is no need of copying all the nodes lying on its access path, since some of them may still have free pointers. An amortized analysis gives a $O(1)$ amortized space increase for each update. Thus, the total ammortized space is $O(n)$ (see (Sarnak and Tarjan 1986) for details).

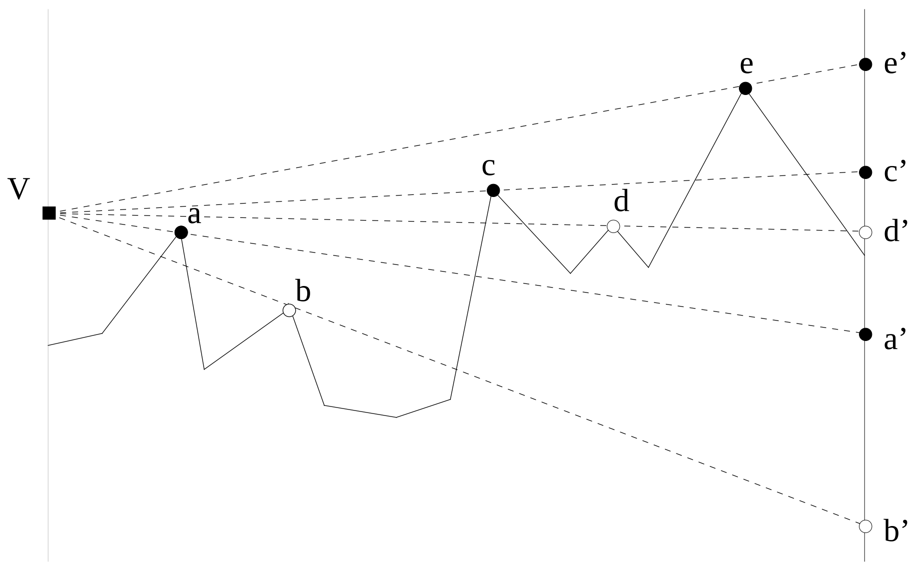


Figure 12: An example of sweep-line status. The horizontal order of the blocking edges is a, b, c, d, e; the vertical order is b, a, d, c, e; active edges are marked in black; not active edges in white.

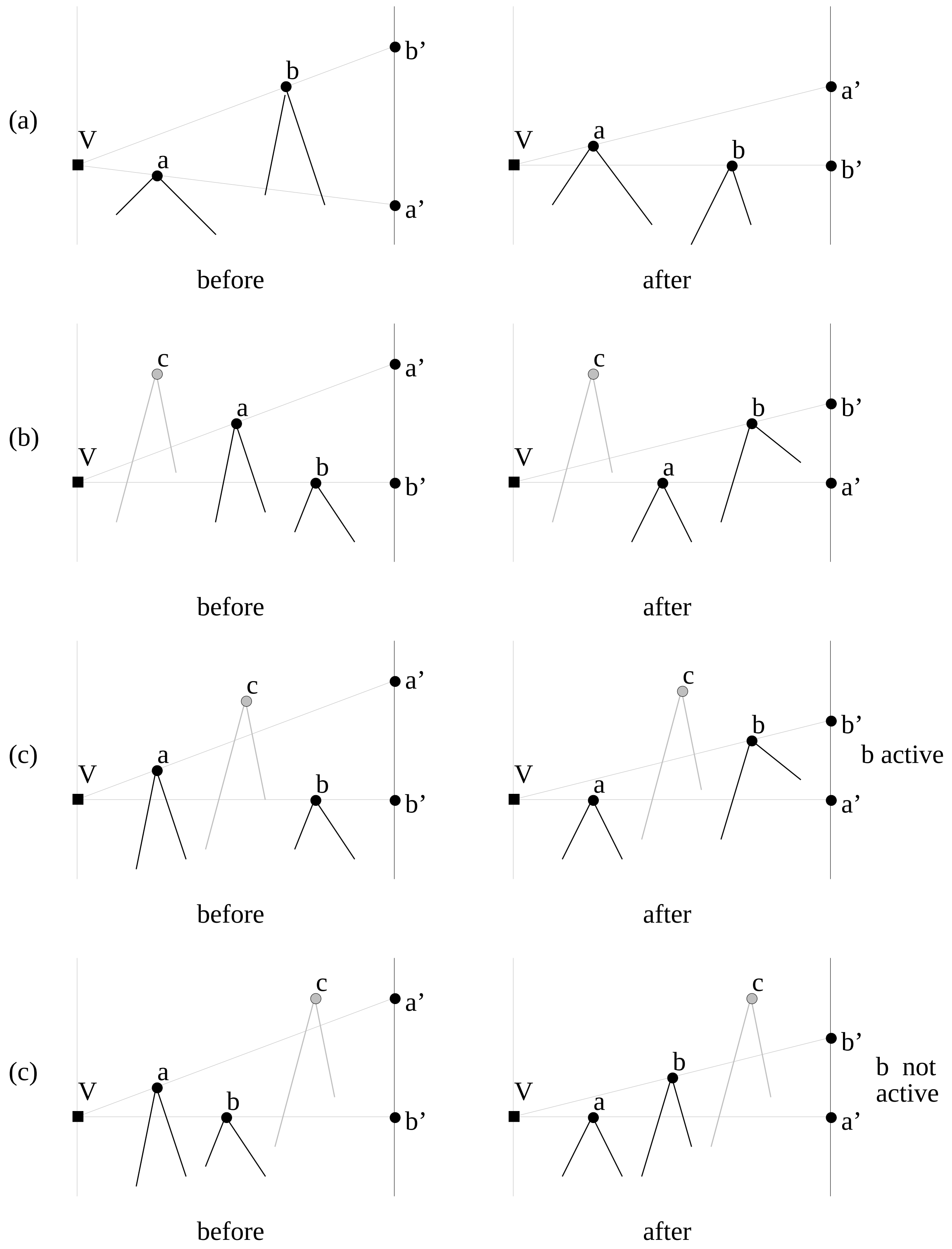


Figure 13: Modification of the sweep status at an event represented by the intersection between two blocking edges a and b, projected on the viewsphere.

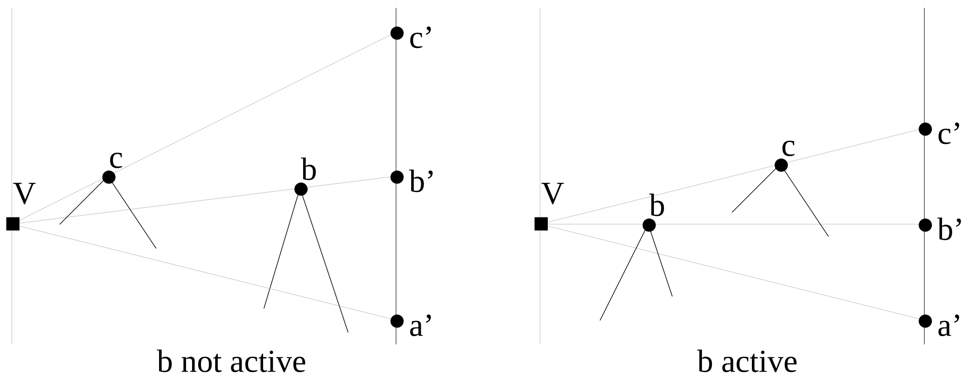


Figure 14: Modification of the sweep status at an event represented by the first endpoint of a blocking edge b.

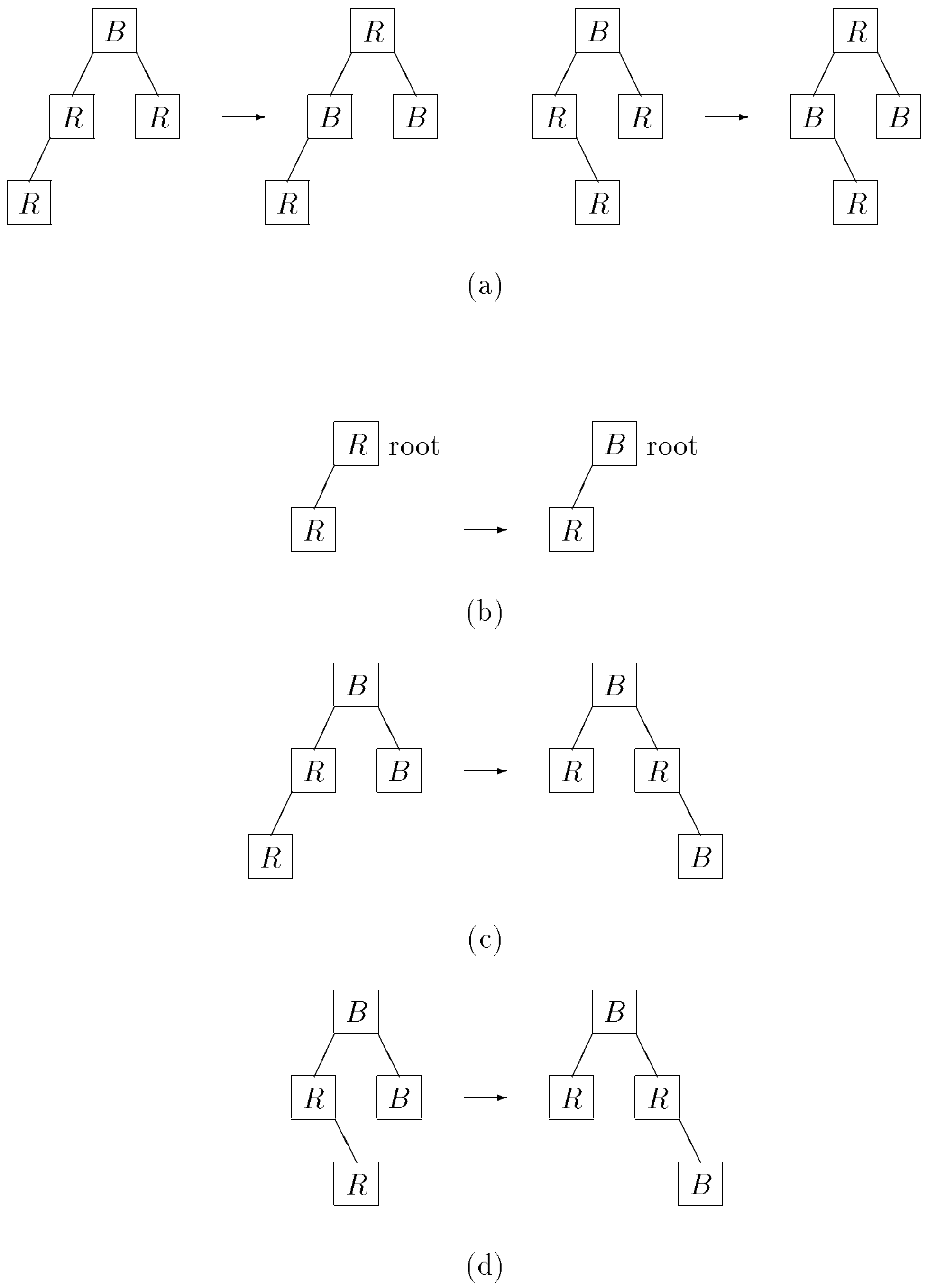


Figure 15: Rebalancing transformation during insertion in a red-black tree. Symmetric cases are omitted. In cases (c,d) the bottommost black node, shown in the picture, may be missing.

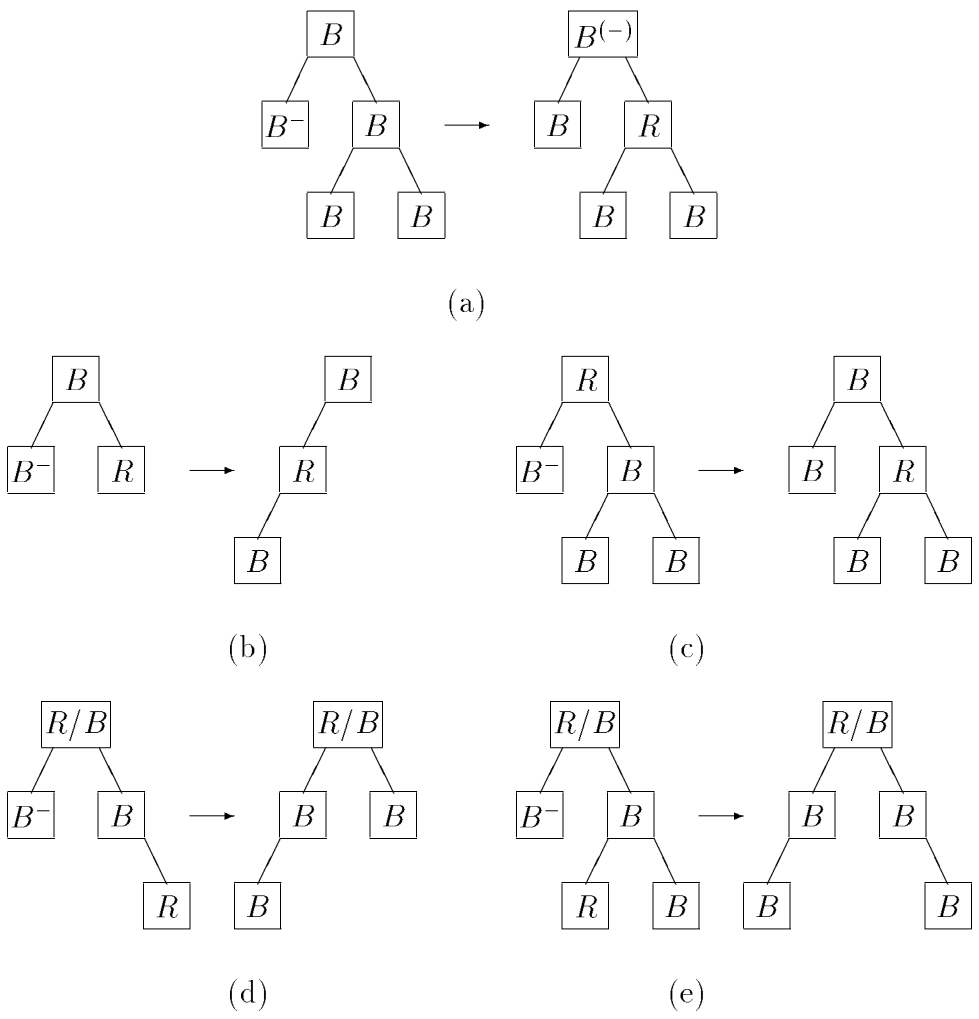


Figure 16: The rebalancing transformation during deletion in a red-black tree. Minus signs denote nodes for which the black constraint is violated (their path contains one black node less than all the others). In case (a) the node marked by $^{(-)}$ violates the black constraint unless it is the root.

References

AGARWAL, P.K., and SHARIR, M., 1986, Applications of a New Space Partitioning Technique. *Discrete and Computational Geometry*, 9, 11-38.

COLE, R., and SHARIR, M., 1989, Visibility Problems for Polyhedral Terrains. *Journal of Symbolic Computation*, 7, 11-30.

DE BERG, M., HALPERIN, D., OVERMARS, M., SNOEYINK, J., and VAN KREVELD, M., 1991, Efficient ray shooting and hidden surface removal. *Algorithmica: An International Journal in Computer Science*, 12, 30-53.

DE FLORIANI, L., 1989, A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 67-78.

DE FLORIANI, L., and PUPPO, E., 1992, A Hierarchical Triangle-based Model for Terrain Description. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, *Lecture Notes in Computer Science*, 639 (New York: Springer-Verlag), 236-251.

DE FLORIANI, L., and MAGILLO, P., 1993, Computing Visibility Maps on a Digital Terrain Model. In *Proceedings European Conference on Spatial Information Theory (COSIT)*, Elba Island, Italy, *Lecture Notes in Computer Science*, 716 (New York: Springer-Verlag), 248-269.

DEVAI, F., 1986, Quadratic bounds for hidden line elimination. In *Proceedings 2nd ACM Symposium on Computational Geometry*, Yorktown Heights, New York (New York: ACM Press), 269-275.

FISHER, P.F., 1993, Algorithm and implementation uncertainty in viewshed analysis. *International Journal of Geographical Information Systems*, 7, 331-347.

GOODCHILD, M.F., and LEE, J., 1989, Coverage problems and visibility regions on topographic surfaces. *Annals of Operation Research*, 20, 175-186.

HART, S., and SHARIR, M., 1986, Non-linearity of Davenport-Schintzel sequences and of generalized path compression schemes. *Combinatorica*, 6, 151-177.

KATZ, M.J., OVERMARS, M.H., and SHARIR, M., 1991, Efficient hidden surface removal for objects with small union size. In *Proceedings 7th ACM Symposium on Computational Geometry* (New York: ACM Press), 31-40.

KIRKPATRICK, D.G., 1983, Optimal search in planar subdivision. *SIAM Journal of Computing*, 12, 28-33.

LEE, J., and PREPARATA, F.P., 1977, Location of a point in a planar subdivision and its applications. *SIAM Journal of Computing*, 6, 594-606.

LEE, J., 1991, Analyses of visibility sites on topographic surfaces. *International Journal of Geographical Information Systems*, 5, 413-429.

McKenna, M., 1987, Worst case optimal hidden surface removal. *ACM Transactions on Graphics*, 6, 19-28.

Mulmuley, K., 1989, An efficient algorithm for hidden surface removal. In *Proceedings ACM Symposium on Computer Graphics*, 23 (New York: ACM Press), 379-388.

Overmars, M., and Sharir, M., 1992, A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11, 1-11.

Preparata, F.P., and Shamos, M.I., 1985, *Computational Geometry: An Introduction* (Berlin New York: Springer Verlag).

Reif, J.H., and Sen, S., 1988, An efficient output-sensitive hidden-surface removal algorithm and its parallelization. *Proceedings 4th ACM Symposium on Computational Geometry*, Urbana (New York: ACM Press), 193-200.

Sarnak, N., and Tarjan, R.E., 1986, Planar point location using persistent search trees. *Communications of the ACM*, 29, 669-679.

Schmitt, A., 1981, Time and space bounds for hidden line and hidden surface algorithms. In *Proceedings Eurographics* (Amsterdam: North Holland), 43-56.