INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

Repetitive Hidden-Surface-Removal for Polyhedra

Marco Pellegrini*
Dept. of Computer Science, King's College London

TR-93-032

July 1993

Abstract

The repetitive hidden-surface-removal problem can be rephrased as the problem of finding the most compact representation of all views of a polyhedral scene that allows efficient on-line retrieval of a single view. In this paper we present a novel approach to this problem. We assume that a polyhedral scene in 3-space is given in advance and is preprocessed off-line into a data structure. Afterwards, the data structure is accessed repeatedly with view-points given on-line and the portions of the polyhedra visible from each view-point are produced on-line. This mode of operation is close to that of real interactive display systems. The main difficulty is to preprocess the scene without knowing the query view-points.

Let n be the number total of edges, vertices and faces of the polyhedral objects and let k be the number of vertices and edges of the image. The main result of this paper is that, using an off-line data structure of size m with $n^{1+\epsilon} \leq m \leq n^{2+\epsilon}$, it is possible to answer on-line hidden-surface-removal queries in time $O(k \log n + \min\{n \log n, kn^{1+\epsilon}/m^{1/2}\})$, when the scene is composed of c-oriented polyhedra. This data structure accommodates dynamic insertion and deletion of polyhedral objects. The polyhedra may intersect and may have cycles in the dominance relation. We also improve worst-case time/storage bounds for the repetitive hidden surface removal problem when the polyhedral scene is composed of unrestricted polyhedra.

Preliminary version of this work is in the Proceedings of the 1993 Workshop on Algorithms and Data Structures.

*Dept. of Computer Science, King's College, Strand, London WC2R 2LS U.K. marco@dcs.kcl.ac.uk

1 Introduction

1.1 The problem

The Hidden Surface Removal problem (HSR for short) for polyhedral scenes in 3-space is important for graphical applications and has attracted much interest in the research community in recent years (e.g. [15],[7],[38],[5],[23], [32],[17] and [31]). A good survey of old and recent results can be found in [14, 40].

Given a set P of objects bounded by polygonal faces which we assume to be opaque, a view-point v, and a view-plane V, the problem consists in computing the visible portion of the scene as viewed from v and projected on V. We consider the "object space" variant of this problem where the output of the algorithm is a planar graph representing the subdivision of V into maximal connected regions in each of which a single face of an object (or the background) is seen. This subdivision is called *visibility map* of P from v, denoted by $M(v, P)$. Other solutions are classified as "image space" since the aim is to compute for each pixel of the screen the object visible at that pixel [40].

We assume in this paper that the polyhedra are given in advance and are preprocessed into a data structure. This step is critical since we do not know at this time which viewpoints will be used. Therefore, our data structure must be independent from any fixed viewpoint. Afterwards, the data structure is accessed repeatedly with query view-points and the visibility map for each viewpoint is produced on-line (*repetitive mode*). This scenario should be compared with the *single-shot mode*, in which *both* the polyhedral scene *and* a view-point are processed together to produce the output visibility map. No data structure is saved and if a new viewpoint is given the process is repeated from scratch. In the real use of interactive display systems the repetitive mode of operation is more natural than the single-shot one. For example in virtual reality applications the user should have on the screen the impression of a smooth walk through an environment. In most cases it is neither possible nor desirable to determine beforehand the positions of the view-points. On the other hand the images should be produced at an high speed in response to interactive changes of the view-point. These constraints lead us to design a system where we allow massive off-line computations to take place if we can lower the time needed to produce the image once the viewpoint is known.

Our first objective in this paper is to achieve optimal output-sensitive query time (up to polylogarithmic factors) and minimize the worst case storage requirement. Then we modify this initial solution by allowing to trade-off query-time and storage. We consider first the case of c-oriented polyhedra (i.e. polyhedra whose facets have one of c predefined orientations) for which more effective time bounds can be proved. In the last part of the paper we apply the same approach to the case of general polyhedra thereby improving worst-case time/storage bounds.

1.2 The aspect graph approach

Traditionally, the problem of efficiently computing the visibility map of a polygonal scene has been tackled using the *aspect graph* approach [25, 26, 35, 21]. For a survey of results on aspect graphs see [21, 22]. The aspect graph of a polyhedral scene is a graph in which each node is associated with a region of R^3 where visibility maps with the same combinatorial

structure are visible. Provided that we can store efficiently the visibility maps at the nodes of the graph and that, given a view-point we can find efficiently the corresponding node in the graph, the storage depends mainly on the number of nodes and edges in the aspect graph.

In [21] Gigus, Canny and Seidel give an algorithm for computing the aspect graph of *orthographic views* (i.e with viewpoint at infinity) of a general polyhedral scene. They use $O(n^4 \log n + |G| \log |G|)$ time to produce the aspect graph G. This graph is stored in a data structure of size $O(|G|)$ and orthographic HSR-queries can be solved in time $O(\log |G| + k)$, where k is the size of the output visibility map. The method in [21] does not seem to generalize immediately to computing also the aspect graph of *perspective views* (i.e. with a proper point as view-point). For perspective views a method in [36] computes the aspect graph in time $O(n^5 + n^2|G|)$ and $O(n^2|G|)$ working storage.

In [36] it is shown that for n non-intersecting triangles the number of orthographic views is $\Theta(n^6)$ in the worst case and the number of perspective views is $\Theta(n^9)$. In [39] it is shown that for axis oriented polyhedra with n edges the number of orthographic views can be $\Theta(n^6)$. In [16] it is shown that for a polyhedral terrain with n edges there can be $\Omega(n^5\alpha(n))$ orthographic views and $\Omega(n^8\alpha(n))$ perspective views.

The aspect graph approach, in a nutshell, consists in computing all possible views and storing them compactly by exploiting coherence between neighbouring views. This approach is only partially satisfactory. On one hand it shows a degree of sensibility to the "complexity" of the scene measured by the number of possible different combinatorial views. The drawback is that this measure of complexity is quite rough and leads quite easily to high storage requirements even for fairly simple scenes. No storage/query-time trade off is possible and the method is not sensible to restrictions in the orientations of the input polyhedra.

In this paper we prove that for c-oriented polyhedra $O(n^{2+\epsilon})$ preprocessing time and storage are always sufficient to answer efficiently *both orthographic and perspective HSR queries* (Theorem 7). In this paper we prove that for general polyhedra $O(n^{4+\epsilon})$ preprocessing time and storage are always sufficient to answer efficiently *both orthographic and perspective HSR-queries* (Theorem 10). Moreover, it is possible to trade off continuously storage and query time. The main conclusion suggested by the new results is that in general the number of possible views of a polyhedral scene is not a lower bound on the storage needed for a compact representation of a scene.

1.3 Other previous results on repetitive HSR

In [31], Mulmuley builds a spatial decomposition $D(P)$ for a general non-intersecting polyhedral scene P. The size of $D(P)$ is $\Theta(n^2)$ in the worst case, and ranges from linear to quadratic depending on the input. Given a query view-point v the visibility map from v can be computed on-line by traversing this decomposition. With $\mathcal{V}(v, P)$ we denote the set of all point in R^3 visible from v. Note that the visibility map $M(v, P)$ is embedded in the boundary of $\mathcal{V}(v, P)$. The time bound for the construction of $M(v, P)$ is proportional to the number of features of $\mathcal{V}(v, P) \cap D(P)$. As recognized in [31] it is not difficult to devise an example (P, v) in which the size of $M(v, P)$ is constant but but the size of $\mathcal{V}(v, P) \cap D(P)$ is quadratic. In this paper we solve the problem of computing the visibility map in time

provably proportional to the size of the output map, up to polylogarithmic factors.

A variation of the HSR problem that received some attention in the research community is obtained when we constrain the view-point to be on a given line L. This problem has applications in flight simulation. The idea in [8] and [31] is to precompute all topologically different visibility maps for viewpoints on L using a sweeping approach. Such results are thus close to the aspect graph approach. Let us define by K the number of topological changes in the visibility map $M(v, P)$ for v moving on L. With K_t we denote the transparent topological changes, that is the number of changes in the projection of edges of P as v moves on L. With K_s we denote the semi-opaque topological changes (i.e. changes on the projection of edges in P, such that the segments involved can see each other locally). In general $1 \leq K \leq K_s \leq K_t \leq n^3/3$. The algorithm in [31] computes all K changes in time $O((K_s + n^2\alpha(n)) \log n)$. The algorithm in [8] computes all K changes in time $O((n^2 + K_t) \log n)$. Since since we can build examples in which K is $O(1)$ and $K_s, K_t = \Omega(n^2)$, these solutions do not exhibit a guaranteed output-sensitive behaviour. The methods in [31, 8] do not seem to be able to make use of a restriction of the input to axis-oriented polyhedra and no trade-off between storage and query time seems possible.

1.4 Previous results on one-shot HSR for c-oriented polyhedra

Solutions to the one-shot HSR problem have developed following ideas quite different from those mentioned above. One major difficulty for some time has been the presence of cycles in the dominance relation among objects. A polyhedron p_1 *dominates* polyhedron p_2 if p_1 partially obstructs the view of p_2 from v. We shall classify output sensitive HSR algorithms in two categories according to their reliance on the existence of an acyclic dominance relation for the objects as seen from the viewpoint. Preparata et al. [37] compute $M(v, P)$ for axis-oriented polyhedra in time $O((n + k) \log n \log \log n)$ provided that the dominance relation is acyclic. The algorithm proposed by de Berg and Overmars in [17] for HSR on c-oriented polyhedra works in *single-shot mode*, with a time complexity $O((n + k) \log n)$, without requiring acyclicity of the dominance relation. In [13] there is a method for maintaining the view of c-oriented polyhedra from a *fixed point of view* when we are allowed to insert and delete polyhedra.

Since our algorithm borrows some tricks from [17] it is worth to discuss this algorithm in more detail. The method in [17] is based on using some primitive operations: (i) ray-shooting queries on "curtains" and (ii) vertex-visibility queries. Each component of $M(v, P)$ is "traced" using these two primitives. The $O(n \log n)$ overhead term accounts for the construction of the data structures supporting efficient answer to the queries, and for performing a vertex-visibility query for each vertex in P.

If we try to use this method for the solution of repetitive HSR we run into the following complications. The data structures for ray-shooting on curtains and for vertex-visibility depend on the viewpoint v and cannot be updated in sublinear time for a new viewpoint v'. Moreover, in order to obtain optimal output sensitive query time we cannot afford to consider independently each vertex for visibility. We shall see in this paper how these difficulties can be overcame.

1.5 Previous results on one-shot HSR for general polyhedra

When the dominance relation on general non-intersecting polyhedra is acyclic, Overmars and Sharir [32] give two algorithms running in time $O(n\sqrt{k}\log n)$ and $O(n^{4/3}\log^{2/3} n + k^{3/5}n^{4/5+\epsilon})$. The second algorithm has been improved in [5] to run in time $O(n^{2/3-\epsilon}k^{2/3+\epsilon} + n^{1+\epsilon} + k^{1+\epsilon})$. For objects with the additional property that the union of their projections has small complexity Katz et al. [23] give an algorithm running in time $O((U(n)+k)\log^2 n)$, where $U(n)$ is the maximum size of the union of the projections of n such objects. This methods applies to disks and balls ($U(n) = O(n)$), fat triangles ($U(n) = O(n\log\log n)$) and terrains ($U(n) = O(n\alpha(n))$). For terrains Reif and Sen [38] obtain an $O((n+k)\log n\log\log n)$ algorithm. The first output-sensitive HSR algorithm that works well even in presence of cycles is in [15] and runs in time $O(n^{1+\epsilon}\sqrt{k})$. The method in [15] is improved in [4] to run in time $O(n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon} + k)$.

All methods for one-shot HSR use at least time $O(n)$ to read the input and process it. Whenever $k < n$ we cannot charge the overhead to the output size and we fall short of the query time $O(k)$ which is optimal for the repetitive HSR problem. In all of the methods mentioned above a change in the view-point triggers an extensive reconstruction of the data structures. Our aim is to improve the time bound for small values of k.

We will borrow a few tricks from [15]. The basic idea in [15] is to use a sweeping line approach to the construction of $M(v, P)$. The event queue is initialized with the vertices of P and the computation proceeds using primitives for ray-shooting on curtains and ray-shooting from the viewpoint v. As before, the adaptation of this schema to repetitive HSR is not trivial. We cannot afford to initialize the queue with n vertices most of which are potentially invisible. Moreover, the data structures depend heavily on v and cannot be updated efficiently for a new view-point.

1.6 Summary of results

For a set of c-oriented polyhedra with a total of n vertices, edges and faces we build off-line data structures of size m, where m is a parameter to be chosen between $n^{1+\epsilon}$ and $n^{2+\epsilon}$. Afterwards the visibility map $M(v, P)$ from a view-point v is produced in time $O(k\log n + \min\{n\log n, kn^{1+\epsilon}/m^{1/2}\})$. If we allow the maximum storage $m = n^{2+\epsilon}$ the query time is $O(k\log^2 n)$, which is almost optimal for all values of k. If we allow the minimum amount of storage $m = n^{1+\epsilon}$ the query time is $O(k\log n + \min\{n\log n, kn^{1/2+\epsilon}\})$ (see Figure 1).

We consider also the dynamization of our data structure under insertion and deletion of c-oriented polyhedral objects form the scene. The main new idea is that the amortized cost of an update (insertion or deletion) can be split between an *on-line update cost* and an *off-line update cost*. The objective is to minimize the on-line update costs since these are more critical for the performance of the algorithm. We suppose that we have a mixed sequence of HSR-queries, insertion and deletions. Let n be the number of edges of polyhedra initially present in the scene and let n_I, n_D be the edges of the polyhedra inserted and deleted over the whole sequence of operations. We obtain on-line amortized update time that depend only on n_I and n_D, not on n. This result in significant when $n_I, n_D << n$, which is to be expected in many real applications during a session of use of the system (Section 5.2). We place no other restriction on the polygonal scene formed by c-oriented polyhedra. In

particular, we do not require disjointness of the input polyhedra. As a consequence the polygonal scene can be specified as the union of polyhedral objects with fewer edges. We do not require acyclicity in the dominance relation.

The same general strategy used to solve the HSR problem for c-oriented polyhedra works for general polyhedra with somewhat degraded time/storage performance. For a general set of (intersecting, non-convex) polyhedra with n vertices, edges and faces we can build a data structure of size m, with $n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$ such that any HSR-query from a point $v \in R^3$ is answered in time $O(kn^{1+\epsilon}/m^{1/4})$. This result improves over the best one-shot method for small values of k. Thus, running this algorithm and the best one-shot algorithm in parallel we can achieve query time $O(\min\{kn^{1+\epsilon}/m^{1/4}, n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon} + k\})$ (see Figure 2).

1.7 The method

The main underlying theme of this research is to extend some methods used for one-shot HSR in order to solve the repetitive HSR problem, thus improving on the results attained through the aspect graph approach. In order to obtain our result on repetitive HSR we mix and modify the approaches in [17] and [15]. We use a sweeping line approach, but we avoid initializing the queue with all the vertices. Instead, we discover the visible vertices on the fly. Vertices that are not visible are never put in the queue. The second challenge is to make our data structure independent of v. We pay this freedom with an increase in the storage requirement. Then we return to linear storage by trading off storage with query time.

In order to achieve these two goals we must support primitive operations more powerful than those in [17] and [15]. In particular, besides supporting ray-shooting queries we must be able to count efficiently how many edges of the polyhedral scene meet query *triangles* and query *pyramids*. Moreover, we apply Megiddo's parametric search technique [29] in order to obtain even more powerful operations. Intuitively, we need to detect efficiently the first moment in which a growing triangle or pyramid intersects a feature of P. We implement all these operations for c-oriented polyhedra by a reduction to planar point location and half-plane range queries. The algorithm in Section 3 is quite general and works also for general polyhedral scenes when we provide an implementation for the primitive operations. The implementation of the primitive operations for general polyhedra is based on results in [34] on collision-free simplices.

The paper is organized as follows. In Section 2 we review some geometric tools used to derive the results. In Section 3 we describe an high-level generic algorithm for solving hidden-surface removal queries which we will denote as the HSRA algorithm. In Section 4 we discuss the implementation of the primitive operations for a set of c-oriented polyhedra. In Section 5 we discuss query-time/storage trade-offs and dynamization of the data structures under insertions and deletions. In Section 6 we discuss the case of general polyhedral scenes.

2 Geometric and algorithmic preliminaries

In this section we survey some geometric and algorithmic results which lay the groundwork for the main results of the subsequent sections.

1. **Arrangements**. A finite set H of hyperplanes in R^d defines a decomposition of R^d into convex cells of various dimensions, which is called the *arrangement* $\mathcal{A}(H)$ of H [18]. If $|H| = n$ the maximum number of cells in $\mathcal{A}(H)$ is $O(n^d)$ and the arrangement $\mathcal{A}(H)$ can be computed in optimal $O(n^d)$ time [20]. One d-dimensional cell of $\mathcal{A}(H)$ is bounded by $O(n^{\lfloor d/2 \rfloor})$ cells of any dimension [18].

2. **Random sampling and cuttings**. Given a random sample R of a set of hyperlanes H, with $|R| = r \leq n$, let us consider the arrangement $\mathcal{A}(R)$. A triangulation $\triangle\mathcal{A}(R)$ is a subdivision of each cell of $\mathcal{A}(R)$ into disjoint simplices such that the vertices of each simplex are vertices of $\mathcal{A}(R)$. The number of simplices in $\triangle\mathcal{A}(R)$ is $O(r^d)$. The random sampling theory of Clarkson [11] states that with probability at least $1/2$ the interior of each simplex $s \in \triangle\mathcal{A}(R)$ does not meet more than $O(n/r \log r)$ hyperplanes of H. A set as $\triangle\mathcal{A}(R)$ is called a *cutting* for H. Given H we can build a data structure which uses $Cn^{d+\epsilon}$ storage, for each $\epsilon > 0$, where the constant C depends on ϵ, such that a query point is located in $\mathcal{A}(H)$ in $O(\log n)$ time [11]. This data structure is built in expected time $O(n^{d+\epsilon})$ [11]. For $d = 2$ Matoušek [27] gives a deterministic method that, for a parameter $r < n$, subdivides the plane into $O(r^2)$ triangles in time $O(nr)$ such that the interior of each triangle meets only n/r lines in H. Cuttings are the basis of many recent divide-and-conquer algorithms in computational geometry (see [1] for a survey).

3. **Halfspace range searching**.

   A problem intimately connected to the point-location problem is the half-space range searching problem. Given a set S of n points in R^d, build a data structure such that, for every query half-space h^+, the number of points in $S \cap h^+$ is computed efficiently. This problem is solved in [10, 28] using *partition trees*. In a partition tree, each node is associated with a region in R^d such that only a fraction of the children intersect the hyperplane h supporting the query half-space. During the query we retrieve the number of points of S within the regions completely contained in h^+ and we recurse the query on the children associated with regions intersected by h. Partition trees are quite versatile and they can be used to set up multi-level data structures.

4. **Multi-level data structures**. Multilevel data structures are a basic paradigm in computational geometry [30]. They are used to search for elements satisfying a complex property. Usually the complex property is split into elementary properties and each elementary property is tested at a specific level of the data structure. For example, in [10, 28] sets of points are organized in multilevel partition trees to answer simplex range queries, where each level of the data structure tests the position of the data points with respect to the hyperplane spanning a facet of the simplex. We have this fundamental theorem in [10]:

   Theorem 1 (Theorem 3.1 in [10]) *Simplex range searching in n points in R^d can be performed in $O(n^{1+\epsilon}/m^{1/d})$ query time, for every $\epsilon > 0$, using a data structure of size m (for any m between n and n^d) which can be computed in $O(m^{1+\epsilon})$ randomized expected time.*

Matoušek in [28], using a different partition scheme, is able to make the preprocessing deterministic and to reduce the query time to $O(n \log^{O(1)} n/m^{1/d})$.

5. **Parametric search**. Parametric search can be described informally as a meta-algorithm which, under quite general conditions, transforms an algorithm to test some property into an algorithm to find the minimum value of a parameter for which this property is true.

   More formally, suppose we have an algorithm P' to compute a predicate $P(i,t)$, which depends on an input i and a real parameter t. The algorithm P' only uses the parameter t to perform branching tests based on the evaluation of fixed degree polynomials which depend on i and t. Moreover, suppose that the predicate $P(i,t)$ is monotone in t, meaning that it is false for $t = 0$ and once it is true it remains so for larger values of t. Megiddo's parametric search technique [29, 12] is a method that transforms P' into an algorithm P'' for finding the minimum value of t for which the predicate $P(i,t)$ is true.

   The simplest form of parametric search is such that, if P' runs in time $T_{P'}$, the modified algorithm runs in time $T_{P'}^2$.

   A more sophisticated type of parametric search uses as an intermediate step a parallel version of the program P', which we call P_{par}. The parallel time is $T_{P_{par}}$ and the number of processors is p. The new algorithm P'' simulates sequentially P_{par} without specifying the value of t. When each of the p processors is stopped at a branching step which requires the evaluation of a polynomial in t we compute all the roots of the polynomials at the branching steps and we sort them. Then we use the sequential algorithm P' to perform a binary search on the sequence of roots in order to find the interval where the minimum value t^* lies. Once we have this interval we can compute the sign of the branching polynomials at t^*. The algorithm branches accordingly to these values. The algorithm completes the simulation and determines a final interval whose left endpoint is exactly t^*. The total time of the simulation is given by $O(pT_{P_{par}} + T_{P'}T_{P_{par}} \log p)$. Applications of Megiddo's parametric search to geometric problems are in [2] [4] [9].

3 The HSRA Algorithm

In this section we give a skeleton algorithm for solving HSR-queries. This algorithm uses the off-line data structures as black boxes (i.e asking queries and receiving answers) and is independent of any particular implementation of these data structures.

The general structure is reminiscent of Bentley and Ottman line-sweep algorithm for reporting intersections of segments in the plane [6]. We take a fixed plane at pre-processing time, independently of any viewpoint, as our view-plane V onto which the visibility map is projected. We require that v does not lie on V[1]. We also fix a direction on V which we call vertical.

[1] It is easy to overcome this restriction by selecting four different planes in general position. Any point is disjoint from at least one such plane.

The visibility map $M(v, P)$ will be constructed by sweeping a vertical line $l(x)$ on V. As in [6] we need to keep two data structures: a dictionary $Line\_status(x)$ which stores the intersection of $M(v, P)$ with the current line $l(x)$, and the priority queue $Event\_queue(x)$ which stores the list of "events" in sorted order along the x-axis.

As a matter of convention we will denote with e an edge in 3-space and with e' its projection of V. Let e' be an edge stored in $Line\_status(x)$ which is the projection of edge e in 3-space. While e' is traced ahead (left-to-right) we shall detect the *first* of the following events.

(i) The right end-point of e. At this event the edge e' should be deleted from $Line\_status$.

(ii) Edge e_1 from face f_1 obscures e, which is therefore no longer visible. We delete e' from $Line\_status$ and insert e'_1 if not already present.

(iii) Edge e_1 from face f_1 is partially obscured by e. We insert e'_1 in $Line\_status$ if not already present.

(iv) Edge e intersects face f_1. If f is the face containing e, then the intersection of f and f_1 is a new visible edge, which is inserted in $Line\_status$.

(v) The intersection of three facets of polyhedra in P. All new visible edges from this intersection point are inserted in $Line\_status$.

There is a further event that cannot be discovered by tracing a single edge present in $Line\_status(x)$:

(vi) A visible vertex q of a face in P projects on $l(x)$ from v. In this case all the visible edges incident to v are inserted in the data structure.

Event (vi) is quite important because it allows us to find on-line all the connected components of the visibility map. By a discussion in [17, 14] these are all and only the interesting events. Since each edge is traced until visible and discarded afterwards until it might be visible again, at each instant $Line\_status(x)$ stores a faithful representation of $M(v, P) \cap l(x)$. Since each event corresponds to a feature of the visibility map it can be charged to the output size k. Each event is not detected more then a constant number of times. An edge can be inserted and deleted several times but each operation corresponds to a distinct feature of $M(v, P)$. The algorithm proceeds by executing the update corresponding to the next event in $Event\_queue$ and by updating $Event\_queue$ itself with the new events induced by the edges which have been manipulated. All the operations on $Line\_status$ and $Event\_queue$ take $O(\log n)$ time using standard data structures. The primitive operations needed to discover events are the following:

1. **Ray-shooting queries**. Given a point p and a ray ρ from p, determine the first face of a polyhedron in P intersected by ρ, and the intersection point.

2. **Triangle-not-emptiness queries**: Given a triangle t in 3-space determine whether t intersects any face or edge or vertex in P.

3. **Minimal-not-empty triangle queries**: Given a family of triangles $t(s)$ depending on a positive real value s such that $s_1 < s_2$ implies $t(s_1) \subset t(s_2)$, report the minimum value of s for which $t(s)$ intersects a feature of P and the feature of P for which this happens.

4. **Minimal-not-empty pyramid queries**: Given a family $\Gamma(s)$ of pyramids with quadrangular base, depending on a positive real value s such that $s_1 < s_2$ implies $\Gamma(s_1) \subset \Gamma(s_2)$, report the minimum value of s for which $\Gamma(s)$ intersects a feature (vertex, edge, face) of P and the feature of P for which this happens.

Setting up Line_status. Initially we set up $Line\_status(0)$ in the following way. We take a point a on $l(0)$ and we *shoot a ray* from v towards a. We determine the point q on the first face f hit by the ray. By definition q is visible. Then we take on the face f the line through q which projects from v onto $l(0)$. We parametrize this line the generic point $p(s)$ where $p(0) = q$ and s is a positive real parameter. Let $\bar{s} > 0$ be the value of s for which $p(s)$ is on the boundary of f. We consider the triangle $t(s)$ of vertices $v, q, p(s)$. We make a triangle emptiness query with $t(\bar{s})$, if $t(\bar{s})$ is empty we have discovered the first point on a visible edge of $M(v, P)$. By shooting a ray from v towards $p(\bar{s})$ we find a point on the first visible face behind f (or the background) and we repeat the tracing procedure upwards on this new face (or background).

If $t(\bar{s})$ is not empty, we use the minimal-not-empty triangle query on the family of triangles $t(s)$ and we find a point on a visible edge of $M(v, P)$. This point belongs to an edge of a face f' different from f. We repeat this tracing procedure upwards on f'. Repeating the tracing procedure from q downwards, we complete the construction of the initial data structure $Line\_status(0)$.

Computing events. Let e' be an edge in $Line\_status$, e the edge in 3-space generating e', q a visible point of e projecting from v onto $l(x)$, and f a visible face containing e (if both faces incident to e are visible, choose one arbitrarily). We parametrize the edge e by a generic point $p(s)$ where $p(0) = q$ and s is a positive real value. Let $\bar{s} > 0$ be the value of s for which $p(s)$ is the right endpoint of e.

1) If the triangle $v, q, p(\bar{s})$ is empty then we have an event (i). Otherwise, by asking a a minimum-not-empty triangle query on $v, q, p(s)$ we detect the first event of type (ii), (iv) or (v).

2) By shooting a ray from v towards q we find a point q' on the first visible face f' different from f. By projecting e onto f' and repeating the steps at 1) we can detect an event of type (iii). Note that we deal implicitly also with the case when both faces incident to e are visible. In this case f' is one of these faces.

3) In order to detect events of type (vi) we proceed in the following way. We consider a pair of consecutive edges e'_1 and e'_2 in $Line\_status$. Let f be the face of P visible between e_1 and e_2 at $l(x)$. We project both e_1 and e_2 onto this face obtaining e''_1 and e''_2. We consider a parametrization of e''_1 and e''_2 with generic points $p_1(s)$ and $p_2(s)$ such that the segment $p_1(s)p_2(s)$ projects on V into a segment parallel to $l(x)$. We ask minimum-not-empty pyramid queries with a pyramid of vertices

$v, p_1(0), p_2(0), p_1(s), p_2(s)$. The value of s returned corresponds to a new visible vertex of P or to some other event (i)-(v) relative to the edges e'_1 and e'_2. In order to detect vertices and edges lying on the face f we must consider the query pyramid as a closed set.

Initially we compute events for the edges and pairs of consecutive edges in $Line\_status(0)$. On the fly, at each update of $Line\_status$, we insert new events in $Event\_queue$ relative to the edges just updated and their neighbours in the linear order stored in $Line\_status$.

When a new visible vertex q is discovered (event (vi)), we insert in $Line\_status$ the new visible edges incident to q. For c-oriented polyhedra in general position the number of edges incident to a vertex is bounded by a constant, therefore we can afford to check every edge for its visibility. The case of general polyhedral scenes is more complex since the degree of a vertex is not bounded.

If q projects from v on the sweeping line at $l(\bar{x})$, we consider the perturbed position $l(\bar{x}+\delta)$ for an infinitesimal $\delta > 0$. We choose a point a to the right of q that projects on $l(\bar{x}+\delta)$ and we use the procedure for setting up $Line\_status$ in the neighborhood of q (i.e. at $l(\bar{x}+\delta)$ instead of $l(0)$). Using a as the initial target for ray-shooting. We stop the procedure when we find a visible edge not incident to q above and below a. Any such edge must be already present in $Line\_status(\bar{x})$, so it is easy to check this halting condition. All the visible edges discovered during this scan are incident to q. Conversely, any visible edge incident to q is discovered. The number of primitive queries is proportional to the number of visible edges incident to q. We summarize this discussion with the following theorem:

Theorem 2 *Algorithm HSRA requires $O(k)$ calls to off-line data structures to produce $M(v,P)$. All other operations require total time $O(k \log n)$.*

4 Primitive operations for c-oriented polyhedra

4.1 A computational lemma

Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of geometric objects, let A be a finite subset of $\mathcal{A}$, and let b be an element of $\mathcal{B}$. Our objective is to compute efficiently $|\{a \in A | F(a,b)\}|$, where $F(a,b) = \bigwedge_{j=1}^{w} C_j(a,b)$ is a conjunction of a *constant* number w of *elementary conditions*. Let $S_a^j(.)$ be a polynomial whose structure depends on j and $\mathcal{A}$ and whose coefficients depend only on $a \in A$. Let p_b^j is a tuple of real numbers depending only on $b \in \mathcal{B}$. Let $S_b^j(.)$ be a polynomial whose structure depends on j and $\mathcal{B}$, whose coefficients depend only on $b \in B$. Also, let p_a^j be a tuple of real numbers depending only on $a \in A$. The elementary conditions $C_j(a,b)$ must be of the form $S_a^j(p_b^j) \geq 0$. Moreover, we require that we can rewrite each polynomial in each conjunct C_j in the dual form $S_b^j(p_a^j) \geq 0$.

For technical reasons we assume that $C_0 = TRUE$. If $S_a^j(p_b^j) > 0$ we say that the *point* p_b^j is on the *positive side* of the *surface* $S_b^j(.) = 0$. From now on we will use the terms *point* and *surface* in this context. Note that, for a given j, an element $a \in A$ (resp. $b \in B$) is mapped to a point or to a surface. We call the point and the surface *dual* to one another.

The problem stated above is discussed in full generality in [34]. In this paper we discuss a restricted cases relevant for deriving the main result on the HSR problem. We use the

additional hypothesis that all of the surfaces S_a^j and S_b^j are hyperplanes in a 1-dimensional or 2-dimensional space. We define $\mathcal{N}_j(A,b)$ to be $|\{a \in A | \bigwedge_{i=0}^{j} C_i(a,b)\}|$, where j is any integer between 0 and w. We are interested thus in $\mathcal{N}_w(A,b)$. Let n be the cardinality of A.

Lemma 1 *For every $0 \leq j \leq w$, we can build in time and storage $O(n^{2+\epsilon})$ a data structure depending only on A such that, for every $b \in \mathcal{B}$, $\mathcal{N}_j(A,b)$ is computed on-line in time $O(\log n)$.*

Proof. Let $T_j(n)$ be the time needed to build the data structure up to level j. We prove the claim by induction on j. For $j = 0$, $T_0(n) = O(1)$ therefore the time bound is satisfied.

Assume $j > 0$. Applying Matoušek [27] partitioning technique on the set of hyperplanes $\{S_a^j(.) = 0 | a \in A\}$ we compute in time $O(nr)$ decomposition of the 2-dimensional space into $O(r^2)$ elementary cells (triangles as a matter of fact) with the property that each elementary cell is intersected by no more than n/r surfaces in $\{S_a^j(.) = 0 | a \in A\}$. Also we build in time $O(r)$ a fast planar location data structure using the method in [19, 24], which attains $O(\log r)$ query time.

For each elementary cell τ we select a point in its interior and we compute the set S_τ^+ of surfaces not intersecting the region τ and in positive position with respect to τ. We associate the cardinality of S_τ^+ with the elementary cell τ and we recurse this construction on the surfaces intersecting τ. Also, for each region τ and for the subset A_τ of A whose corresponding surface is in S_τ^+ we build the data structure to compute $\mathcal{N}_{j-1}(A_\tau, b)$, which can be built in time $T_{j-1}(n)$. The total time and space for this construction satisfies the recurrence:

$$T_j(n) \leq \begin{cases} O(1) & \text{if } n = O(1) \\ O(r^2)T_j(n/r) + O(r^2)T_{j-1}(n) + O(nr) + O(r) & \text{otherwise} \end{cases} \quad (1)$$

Assuming by induction hypothesis that $T_{j-1} = O(n^{2+\epsilon'})$, the solution to this recurrence is $T_j(n) = O(n^{2+\epsilon})$ for an $\epsilon > \epsilon'$. Note that the result for the time and storage holds also when we choose r to be a small power of n.

Given b we compute the query point p_b^j and we locate the region τ containing it using the fast point-location data structure. Than we recurse on the the data structures associated with τ using p_b^j in one and the query point p_b^{j-1} in the other. At the last level we collect the counters of the positive regions and we sum them up to form the final result. The query time $Q_j(n)$ satisfies this recurrence:

$$Q_j(n) \leq Q_j(n/r) + Q_{j-1}(n) + O(\log r)$$

If we assume that $r = n^\nu$ and the induction hypothesis that $Q_{j-1} = c' \log n$, we obtain the total query time $Q_j(n) = c \log n$ for a slightly larger value $c \geq c'$. ∎

In the rest of this section we discuss the implementation of the four primitive operations used by algorithm HSRA when the input is restricted to c-oriented polyhedra. Initially we shall see how to answer each query in polylogarithmic time at the expenses of quadratic storage. The general idea is to show that any of the queries used by the HSRA algorithm is reducible to the computation of $\mathcal{N}_w(A,b)$ for some formula F of the type described above.

Is is sufficient to show a reduction to any boolean formula F composed of elementary conditions. Afterwards, F can be put in disjunctive normal form $F_{DNF} = \bigvee_i F_i$, where each F_i is a conjunct. Moreover we can transform further F by imposing that at most one disjunct is verified for a given pair (a, b) by setting

$$F = \bigvee_i (F_i \wedge (\bigwedge_{j=0}^{i-1} \neg F_j)).$$

If the initial formula has a constant number of elementary conditions each conjunct in the new formula will have constant length.

4.2 Ray shooting on c-oriented polyhedra

Theorem 3 *Given a set of c-oriented polyhedra with n edges there exists a data structure of size $O(n^{2+\epsilon})$ to answer ray shooting queries in $O(\log^2 n)$ time. The data structure can be built using $O(n^{2+\epsilon})$ time and storage.*

Proof. As noted in [4] we obtain a method for ray-shooting by applying the parametric search transformation on an algorithm to test whether a segment does not meet any facet of a c-oriented polyhedra. Thus it sufficient to give an algorithm for counting incidences between segments and c-oriented facets in 3-space. Counting problems are easily decomposable. If $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ is the set of facets bounding the c-oriented polyhedra, the answer for $\mathcal{F}_1$ and the answer for $\mathcal{F}_2$ can be combined in $O(1)$ time to give the answer for $\mathcal{F}$. We partition the facets of the c-oriented polyhedra into c sets of parallel facets, and we solve the counting problem on each set separately. After applying a suitable affine transformation we can consider one such set of parallel facets as horizontal. Any such c-oriented horizontal facet has at most $2(c-1)$ edges and moreover each such edge is orthogonal to one of c fixed planes $P_1, .., P_c$ *which depend only on the initial choice of the c orientations.* We denote with q^i the projection of a set q upon the plane P_i. Let $l(s)$ be the line spanning the segment s and $aff(f)$ be the plane spanning the facet f. We have the following easy lemmas:

Lemma 2 *For any segment s and facet f in 3-space which are not not co-planar:*

$$s \cap f \neq \emptyset \iff s \cap \mathrm{aff}(f) \neq \emptyset \wedge l(s) \cap f \neq \emptyset$$

Note that the first conjunct of the formula in Lemma 2 reduces to compare the endpoints of s with the plane spanning the facets f. Since these planes are all parallel the comparison is indeed a comparison in a 1-dimensional space.

Lemma 3 *Given a c-oriented horizontal polygon f, and planes $P_1, ..P_c$ as above, a line l intersects f if and only if l^i meets f^i, for all $i = 1, .., c$.*

Proof. Let $aff(f)$ be the plane spanning f and let $p = l \cap aff(f)$. If $p \in f$ then for every i, $p^i \in f^i$. If $p \notin f$ then there is an edge e of f separating p from f. If P_j is the projection plane associated with e, then $p^j \notin f^j$. ∎

Given a segment s on the plane, let $l(s)$ be the line supporting s, $rhp(s)$ the right endpoint and $lhp(s)$ the left endpoint. $Slope(l)$ is the slope of line l.

Lemma 4 *Two segments s_1 and s_2 on the plane intersect if and only if $l(s_1)$ meets s_2 and $l(s_2)$ meets s_1.*

Lemma 5 *A line l and a segment s on the plane meet each other in either of these two cases:*

A) $(Slope(l) < Slope(l(s))) \wedge (lhp(s)\ below\ l) \wedge (rhp(s)\ above\ l)$

B) $(Slope(l) > Slope(l(s))) \wedge (lhp(s)\ above\ l) \wedge (rhp(s)\ below\ l)$

where the above and below relations refer to the vertical direction on the plane containing l and s.

Proof. Follows from elementary geometry. ∎

Lemmas 2, 3, 4 and 5 imply that we can write a formula of type F to denote the fact that a segment meets a c-oriented facet. Moreover, all of the elementary conditions involve testing a point versus a line in some 2-dimensional space or a point versus a point in a 1-dimensional space. Thus from Lemma 1 we can count efficiently how many c-oriented facets meet a query segment, and thus also decide if the query segment is empty of intersections. ∎

As a matter of fact, by extending a slightly different approach to this ray-shooting problem developed in [33] for axis-oriented polyhedra (a special case of c-oriented polyhedra) it is possible to avoid using parametric search and to reduce the query time to $O(\log n)$. We choose to present this weaker result since it is more similar to the solution for the other queries used by the HSRA Algorithm.

Using this ray-shooting primitive we can also check whether the view-point v is interior to any polyhedron in P. We consider the polyhedra as solid objects, thus being in the interior of one object can be regarded as a non-physical situation. We send a ray from v in an arbitrary direction. If the ray meets a face of P coming from the interior of the polyhedron containing this face we conclude that v is an interior point of P. From now on we consider v to be external to P.

4.3 Empty triangle queries

Given a set of c-oriented polyhedra P and a triangle t we want to check whether t meets any polyhedron in P. In our application we use triangles t such that at least one vertex in external to any polyhedron in P. The following lemmas hold easily:

Lemma 6 *A triangle t with a vertex disjoint from P intersects P of and only if an edge of P meets t or an edge of t meets a face in P.*

Lemma 7 *For any segment s and triangle t in 3-space which are not co-planar:*

$$s \cap t \neq \emptyset \iff s \cap \mathrm{aff}(t) \neq \emptyset \wedge l(s) \cap t \neq \emptyset$$

Now we are set to prove the following theorem.

Theorem 4 *Given a set of c-parallel polyhedra P of complexity n, we can answer empty triangle queries of algorithm HSRA in time $O(\log n)$ using data structures of total size $O(n^{2+\epsilon})$.*

Proof. We can easily check whether an edge of t meets a face of P using three ray-shooting queries. In order to check an edges of P against t we partion those edges into $\binom{c}{2}$ sets of parallel edges and we test each set independently.

Let S be a set of parallel segments in 3-space and t our query triangle. It is crucial for the analysis of the storage required by this data structure to recall that the planes $aff(t)$ used in by the HSRA algorithm in Section 3 are of a special kind. Every such plane must contain an edge of P, or or must be parallel to the sweeping line. We have thus $\binom{c}{2}+1$ sub-families of planes, where planes in each sub-family must all meet a fixed point at infinity. We project the segments in S from such point at infinity onto the auxiliary plane obtaining a planar set of segments S'. We project the plane $aff(t)$ from such point at infinity onto the auxiliary plane obtaining a line l_t.

The first test in the formula of Lemma 7 is thus reduced to detecting the segments in S' intersected by the line l_t for which we can invoke the formula of Lemma 5.

The second test in the formula of Lemma 7 is reduced to the following computation We project the segments onto a plane orthogonal to the common direction of segments in S, obtaining a set of points S''. Also we project triangle t onto this plane obtaining a triangle t'. We now require that a point of S'' is in the intersection of three half-planes defined by the lines spanning edges of t'. All of these conditions are elementary conditions. We can thus write a formula of type F for counting the number of edges intersecting a query triangle t and thus also determine if the triangle is empty. The time and storage bounds derive directly from Lemma 1. ∎

4.4 Minimum empty triangle queries

Let us consider a family of planar triangles in 3-space $t(s)$ indexed by a positive real parameter s, forming an *inclusion chain* (i.e. $s_1 < s_2 \Rightarrow t(s_1) \subset t(s_2)$). It is easy to see that the property of intersecting P is monotone for such set of triangles. There is a value s^* such that, for all $s < s^*$, $t(s)$ does not meet P and, for all $s > s^*$, t meets P. Therefore we can use Megiddo's parametric search technique to transform the algorithm of Theorem 4 into an algorithm for finding s^* [29]. The query time of the transformed algorithm is the square of the time of the original one.

Theorem 5 *Given a set of c-parallel polyhedra P of complexity n, we can answer minimum-not-empty triangle queries of algorithm HSRA in time $O(\log^2 n)$ using data structures of total size $O(n^{2+\epsilon})$.*

4.5 Minimal-not-empty pyramid queries

Let us consider a family of pyramids in 3-space $\Gamma(s)$ indexed by a positive real parameter s, forming an *inclusion chain* (i.e. $s_1 < s_2 \Rightarrow \Gamma(s_1) \subset \Gamma(s_2)$). Again, the property of $\Gamma(s)$ to intersect P is monotone in s: there is a value s^* such that, for all $s < s^*$, $\Gamma(s)$ does not meet P and, for all $s > s^*$, $\Gamma(s)$ meets P. Therefore we can use Megiddo's parametric

search technique to transform an algorithm for detecting empty pyramids into an algorithm for finding s^*.

Lemma 8 *A pyramid Γ with a vertex disjoint from P intersects P of and only if an edge of P meets a face of Γ or an edge of Γ meets a face in P or a vertex of P is contained in Γ.*

Theorem 6 *Given a set of c-parallel polyhedra P of complexity n, we can answer minimum-not-empty pyramid queries of algorithm HSRA in time $O(\log^2 n)$ using data structures of total size $O(n^{2+\epsilon})$.*

Proof. From the above discussion is is sufficient to set up a test for pyramid emptiness and apply the parametric search transformation. From Lemma 8 we have three cases to check. We can check whether an edge of Γ meets a face in P using the ray-shooting data structure. We can test whether a face of Γ (triangle or quadrangle) meets an edge of P using the result of Theorem 4 (though we need a slight modification to deal with the quadrangular face of Γ within the stated storage bound). The last case we need to test is an instance of simplex range searching on the set of vertices of P. The pyramid has faces belonging to $\binom{c}{2} + 1$ subfamilies of planes. Each subfamily is constrained to contain a fixed point at infinity. We can thus write down a formula for testing whether a point is within a pyramid. We just have to check the position of the vertices of P with respect to each plane spanning a facet of the query pyramid.

One such test is solved by projecting both the plane and the points from the point at infinity constraining the plane. Thus the problem reduces to an half-plane range searching problem in 2-space. Since there is a constant number of different classes of pyramids according to the possible combinations of orientations of the facets we can write a formula F to denote the fact that a point is within the query pyramid. Time and storage bounds follow from Lemma 1. ∎

4.6 Putting the pieces together

Theorem 2 and the implementation of the primitive operations in Section 4 prove the following theorem:

Theorem 7 *Given a scene P composed of c-oriented polyhedra with n vertices, edges and faces, we can build a data structure $\mathcal{D}_1(P)$ of size $O(n^{2+\epsilon})$ such that for a query point v the visibility map $M(v, P)$ can be computed in time $O(k \log^2 n)$, where k is the output size.*

5 Trade offs and dynamization

5.1 Storage-query time trade offs

The primitive operations of algorithm HSRA are implemented in Section 4 by a reduction to (several) point location queries on arrangements of lines in 2-spaces. As a matter of fact we are interested only in the sign of some bilinear forms depending on P and v. The same result can be obtained by dualizing point and surfaces of Lemma 1, in this case the problem is transformed in a series of halfplane range searching on sets of points in 2-spaces.

Such queries can be solved using the techniques in [10, 28] in $n^{1+\epsilon} \leq m \leq n^{2+\epsilon}$ storage and $T = O(n^{1+\epsilon}/m^{1/2})$ query time. When we apply the parametric search technique to superlogarithmic algorithms (as it is the case for the query time in the trade-off case) we use the more sophisticated form of parametric search that needs a parallel version of the halfplane range searching algorithm [29]. The data structures in [10, 28] are based on a partition-tree approach and the query time depends on the number of nodes in the partition tree visited during the query. We can allocate dynamically processors to the nodes visited during the search, thus we need $p = O(n^{1+\epsilon}/m^{1/2})$ processors. The parallel query time T' is given by the depth of the tree which in $O(\log n)$. After the parametric-search transformation we have a total query time $O(T'p + TT' \log n) = O(n^{1+\epsilon}/m^{1/2})$, for a slightly greater value of ϵ. Also, we can run in parallel the algorithm in [17], which has better performances when k is large. The above discussion leads to the following theorem:

Theorem 8 *Given set P of c-parallel polyhedra with n vertices, edges and faces, we can build a data structure $\mathcal{D}_2(P, m)$ of size $n^{1+\epsilon} \leq m \leq n^{2+\epsilon}$ such that for a query point v the visibility map $M(v, P)$ can be computed in time $O(k \log n + \min\{n \log n, kn^{1+\epsilon}/m^{1/2}\})$, where k is the output size.*

Corollary 1 *Given a set P of c-parallel polyhedra with n vertices, edges and faces, we can build a data structure $\mathcal{D}_3(P)$ of size $O(n)$ such that for a query point v the visibility map $M(v, P)$ can be computed in time $O(k \log n + \min\{n \log n, kn^{1/2+\epsilon}\})$, where k is the output size.*

If we allow only linear storage we obtain a query time roughly order of $k\sqrt{n}$ which is asymptotically better than the one-shot algorithm of [17] for $k \leq \sqrt{n}$. The result of Corollary 1 is significant for those scenes in which only a small part of the total number of polyhedra is visible at any given time. A graph of the running time $T(n, k)$ as a function of k in logarithmic scale is in Figure 1, for the case of linear storage.

5.2 Dynamization

Since all data structures are multilayer data structures based on the CSW scheme [10, 5] or on Matoušek 's scheme [28] we can use a dynamization result in [5]:

Theorem 9 (Theorem 3.1 in [5]) *Given a set of n points in R^d and a parameter $n^{1+\epsilon} \leq m \leq n^{d+\epsilon}$ one can maintain the CSW-partitioning structure in $O(m/n^{1-\epsilon})$ amortized time as we insert or delete a point, and can answer half-space range queries in time $O(n^{1+\epsilon}/m^{1/d})$.*

Thus the off-line amortized update time for $\mathcal{D}_2(P, m)$ is $O(m/n^{1-\epsilon})$. On-line we can do better under a reasonable assumption. Notice that all of the data structures for answering primitive queries of the HSRA algorithm *count* the number of features (vertices, edges, faces) intersecting the query objects. Let us suppose that we have a mixed sequence of update operations and queries. Let P be the initial set of c-oriented polyhedra, P_I the set of c-oriented polyhedra inserted up to the present moment and P_D the set of c oriented polyhedra deleted form P and P_I up to the present moment. Let $N_g(S, q)$ be the number of

features of type g (vertex, edge, face) of a set of polyhedra S intersecting the query object q (ray, triangle, pyramid). Clearly:

$$N_g((P \cup P_I)/P_D, q) = N_g(P, q) + N_g(P_I, q) - N_g(P_D, q) \tag{2}$$

Thus we need to modify the $\mathcal{D}_2(., m)$ only to accommodate insertion. We build three separate data structures for P, P_I and P_D. The three data structures storing the sets P, P_I and P_D will use three parameters of storage m, m_I and m_D with $n^{1+\epsilon} \leq m \leq n^{2+\epsilon}$, $n_I^{1+\epsilon} \leq m_I \leq n_I^{2+\epsilon}$ and $n_D^{1+\epsilon} \leq m_D \leq n_D^{2+\epsilon}$. Parameters m, m_I and m_D can be chosen so to tune the performance of the system. The on-line amortized time for an insertion is now only $O(m_I/n_I^{1-\epsilon})$. Since in real applications we expect that over a session of use of the system $n_I << n$, we obtain a very small slowdown of the system due to reconstructions. A similar argument holds for P_D. The query time is changed only by a constant factor. Extensive reconstruction of the data structure in order to maintain the overall performance can be done off-line between sessions using the method of Theorem 9.

6 Primitive operations for general polyhedra

In this section we discuss the implementation for the several primitive operations used in the HSRA algorithm in the case of general polyhedral scenes.

1. **Ray-shooting queries**. From results in [33, Theorem 8] we can answer ray-shooting queries in time $O(\log n)$ using $O(n^{4+\epsilon})$ storage.

2. **Empty triangle queries**. From results in [34, Theorem 3] we can answer triangle emptiness queries in time $O(\log n)$ using $O(n^{4+\epsilon})$ storage.

3. **Minimal-empty triangle queries**. We use the data structure in [34, Theorem 3] to answer triangle emptiness queries. We modify the query method by applying Megiddo's parametric search [29], thus obtaining a method to determine the smallest empty triangle in a family of triangles forming a chain of inclusions. The query time becomes $O(\log^2 n)$.

4. **Minimal-empty pyramid queries**. A data structure described in [34, Theorem 4] answers simplex emptiness queries in time $O(\log n)$ using $O(n^{4+\epsilon})$ storage. This method can be easily adapted to answer emptiness queries with any convex polyhedron with a constant number of facets (e.g. a pyramid with quadrangular base). Similarly to the previous case we modify the query algorithm by applying Megiddo's parametric search [29]. Minimal-empty pyramid queries can be solved in time $O(\log^2 n)$ using $O(n^{4+\epsilon})$ storage.

The HSRA Algorithm together with the above primitives leads to the following result:

Theorem 10 *Given a set P of polyhedra with n vertices, edges and faces, we can build a data structure $\mathcal{D}_4(P)$ of size $O(n^{4+\epsilon})$ such that for a query point v the visibility map $M(v, P)$ can be computed in time $O(k \log^2 n)$, where k is the output size.*

Using results in [3], as mentioned in [34], it is possible to trade off storage and query time.

Corollary 2 *Given a set P of polyhedra with n vertices, edges and faces, we can build a data structure $\mathcal{D}_5(P,m)$ of size $n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$ such that for a query point v the visibility map $M(v,P)$ can be computed in time $O(kn^{1+\epsilon}/m^{1/4})$, where k is the output size.*

The method of Corollary 2 compares favourably with the one-shot methods only for quite small values of k, unless a large amount of storage is allowed. Thus the best bound on the time needed to produce a view is given by combining Corollary 2 with the best one-shot method. We obtain a bound $O(\min\{kn^{1+\epsilon}/m^{1/4}, n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon} + k\})$. In Figure 2 it is shown a graph of the bound in logarithmic scale for $m = n^2$.

7 Conclusions

In this paper we have shown several results on the repetitive hidden-surface removal problem for polyhedral scenes. We considered both c-oriented polyhedra, for which better bounds are attained, and general polyhedral scenes. We reduce the HSR problem to combinations of point-location and half-plane range searching, in conjunction with a general sweeping line algorithmic skeleton. For several visibility problems in 3-space we improve on the previously known best bounds.

References

[1] P. Agarwal. Geometric partitioning and its applications. Technical Report CS-1991-27, Dept. of Computer Sci. Duke University, July 1991.

[2] P. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 321–331, 1990.

[3] P. Agarwal and J. Matoušek. Range searching with semialgebraic sets. In *Proc. of the 17th Symp. on Mathematical Foundations of Computer Science*, number 629 in Lecture Notes in Computer Science, pages 1–13, 1992.

[4] P. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 517–526, 1992.

[5] P. K. Agarwal and M. Sharir. Applications of a new space partitioning technique. In *Proceedings of the 1991 Workshop on Algorithms and Data Structures*, number 519 in Lecture Notes in Computer Science, pages 379–391. Springer Verlag, 1991.

[6] J. Bentley and T. Ottman. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, C-28:643–647, 1979.

[7] M. Bern. Hidden surface removal for rectangles. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 183–192, 1988.

[8] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 107–117, 1990.

[9] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric search. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 120–129, 1992.

[10] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 23–33, 1990.

[11] K. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.

[12] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. of ACM*, 34:200–208, 1987.

[13] M. de Berg. Dynamic output-sensitive hidden surface removal for c-oriented polyhedra. *Computational Geometry: Theory and Applications*, 2:119–140, 1992.

[14] M. de Berg. *Efficient Algorithms for ray-shooting and hidden surface removal.* PhD thesis, Utrecht University, Dept. of Comp. Sci., 1992.

[15] M. de Berg, D. Halperlin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray-shooting and hidden surface removal. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 21–30, 1991.

[16] M. de Berg, D. Halperlin, M. Overmars, and M. van Kreveld. Sparse arrangements and the number of views of polyhedral scenes. Manuscript, June 1992.

[17] M. de Berg and M. Overmars. Hidden surface removal for axis-parallel polyhedra. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 252–261, 1990.

[18] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, 1987.

[19] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, (15):317–339, 1986.

[20] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computing*, pages 341–363, 1986.

[21] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 13:542–551, 1991.

[22] Z. Gigus and J. Malik. Computing the aspect graphs for line drawings of of polyhedral objects. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 12:113–122, 1990.

[23] M. Katz, M. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 31–40, 1991.

[24] D. Kirkpatrick. Optimal search in planar subdivision. *SIAM Journal of Computing*, (12):28–35, 1983.

[25] J. Koenderlink and J. van Doorn. The singularities of visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

[26] J. Koenderlink and J. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[27] J. Matoušek. Cutting hyperplane arrangements. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 1–9, 1990.

[28] J. Matoušek. Efficient partition trees. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 1–9, 1991.

[29] N. Megiddo. Applying parallel computation algorithms in the design of sequential algorithms. *J. of ACM*, 30:852–865, 1983.

[30] K. Mehlhorn. *Multidimensional Searching and Computational Geometry.* Springer Verlag, 1984.

[31] K. Mulmuley. Hidden surface removal with respect to a moving view point. In *Proceedings of the 23th Annual ACM Symposium on Theory of Computing*, pages 512–522, 1991.

[32] M. Overmars and M. Sharir. Output-sensitive hidden surface removal. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 598–603, 1989.

[33] M. Pellegrini. Ray shooting and isotopy classes of lines in 3-dimensional space. To appear in *Algorithmica*. Preliminary version in *Proceedings of the 1991 Workshop on Algorithms and Data Structures*, number 519 in Lecture Notes in Computer Science, pages 20–31. Springer Verlag, 1991.

[34] M. Pellegrini. On collision-free placements of simplices and the closest pair of lines in 3-space. To appear in *SIAM J. on Computing.* Preliminary version in the 8th ACM Symp. on Comp. Geom. with the title 'Incidence and nearest-neighbor problems for lines in 3-space' pp. 130-137, 1992.

[35] W. Plantiga and C. Dyer. An algorithm for constructing the aspect graph. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 123–131, 1986.

[36] W. Plantiga and C. Dyer. Visibility, occlusion and the aspect graph. *Int. J. of Computer Vision*, 5:137–160, 1990.

[37] F. Preparata, J. Vitter, and M. Yvinec. Output sensitive generation of the perspective view of isothetic parallelepipeds. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, number 447 in Lecture Notes in Computer Science, pages 71–84. Springer Verlag, 1990.

[38] J. Reif and S. Sen. An efficient output-sensitive hidden-surface removal algorithm and its parallelization. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 193–200, 1988.

[39] J. Snoeyink. The number of views of axis-parallel objects. *Algorithmic Review*, 2:27–32, 1991.

[40] I. Sutherland, R. Sproul, and R. Schumaker. A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1):1–55, 1974.

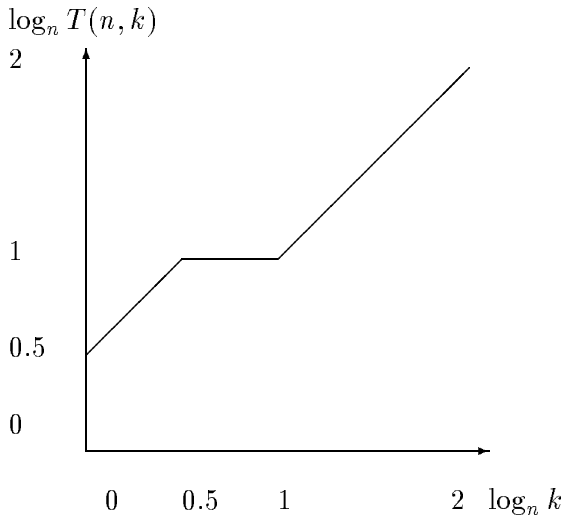

Figure 1: Time in logarithmic scale as function of k for c-oriented polyhedra and linear storage.

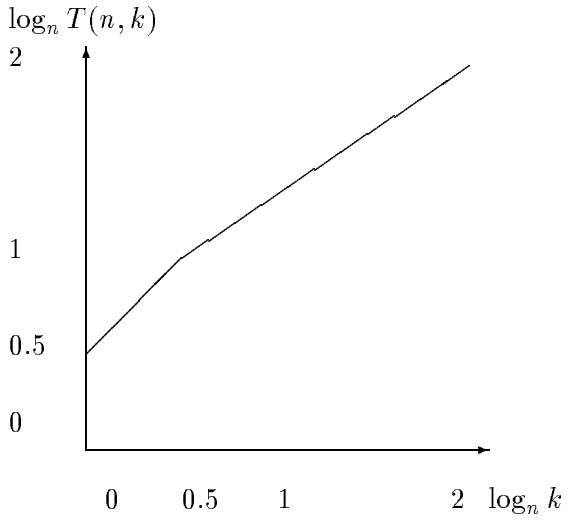

Figure 2: Time in logarithmic scale as function of k for general polyhedra and quadratic storage.