An Efficient Algorithm For Hidden Surface Removal

Ketan Mulmuley
The University of Chicago

1 Abstract

We give an efficient, randomized hidden surface removal algorithm, with the best time complexity so far. A distinguishing feature of this algorithm is that the expected time spent by this algorithm on junctions which are at the "obstruction level" l , with respect to the viewer, is inversely proportional to l . This provably holds for any input, regardless of the way in which faces are located in the scene, because the expectation is with respect to randomization in the algorithm, and does not depend on the input. In practice, this means that the time complexity is roughly proportional to the size of the *actually visible* output times logarithm of the average depth complexity of the scene (this logarithm is very small generally).

2 Introduction

In this paper we give an efficient, randomized hidden surface removal algorithm, with the best time complexity so far. A basic theory behind this algorithm is a theory of probabilistic geometric games and a theory of a certain θ series, that can be associated with combinatorial arrangements [Mu1, Mu2, Mu3].

The concept of a θ series is essential to the discussion of this algorithm. So let us first see how one can associate such a series with a collection of faces in three dimensions. For the sake of simplicity, we shall assume that the faces are nonintersecting. Intersecting faces are treated in [Mu5]. We, of course, allow faces to share edges. Imagine an observer located at the origin. Assume, for the sake of simplicity, that the origin is located at $(0, 0, -\infty)$; this can be achieved by an appropriate perspective transformation. Project all faces orthonormally in the z direction onto the "view" plane. Abstractly speaking, the problem of hidden surface removal is concerned with finding a suitable partition of the view plane, and labelling each region of this partition with a face

that is visible there. When one projects all faces onto the view plane, this gives rise to several junctions formed by crossings of the xy -projections of the face boundaries. Only a few of these junctions are visible. An efficient algorithm should spend as little time as possible on the invisible junctions. To make this statement precise, we shall associate levels with all junctions. Consider a junction q located at an intersection of the xy -projections of the boundaries of two faces f and g . A face h is said to obstruct q , with respect to the observer o at $(0, 0, -\infty)$, if the presence of h makes q invisible to the observer. This means, that if we consider the line of sight from the observer, corresponding to the junction q , h occurs on this line of sight before f or g . Define the *obstruction level* of q , $level(q)$, as the number of faces in the scene, which obstruct q . Let V_l be the set of junctions at level $l - 1$, and let $v(l)$ be the size of V_l . Thus V_1 is precisely the set of visible junctions.

For a fixed collection of faces, define a θ series as follows: for every real number $s \geq 0$,

$$\theta(s) = \sum_l \frac{v(l)}{l^s}.$$

A similar θ series can also be associated with a collection of intersecting faces. The main motivation behind associating such a series is that it has a lot of combinatorial information encoded in it. For example, in [Mu3], we showed how a θ series can be associated with an arrangement of hyperplanes in any dimension d . It was then shown that the worst case behaviour of this series can be analyzed as a function of s , and that $s = \lceil d/2 \rceil$ is a critical point on the real axis where this behaviour changes abruptly. This has a lot of combinatorial implications. A similar theory can also be developed for the θ series associated with a collection of intersecting faces (in any dimension) [Mu5]. In this paper, we shall restrict ourselves to only an algorithmic implication of this theory in the context of hidden surface removal. The implication is that the running time of our randomized hidden surface removal algorithm depends linearly on the value of the θ function at $s = 1$: $\theta(1) = \sum_{l=1}^n v(l)/l$.

To see what this means, let us first see what $\theta(0)$ is. It is easy to see that $\theta(0)$ is simply the number of all junctions in the view plane, visible as well as invisible. Whether hidden surface removal can be performed in time that depends linearly on $\theta(0)$, in a strict theoretical sense, is itself a nontrivial question. The reason is that, this requires a method to find all m intersections of n segments in a plane in optimal $O(m + n \log n)$ time. This question has been

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

resolved in the affirmative recently in [Mu1,CE,Cl]. The optimal planar partition algorithm in [Mu1] can be easily extended so as to obtain an $O(n \log n + \theta(0))$ hidden surface removal algorithm. Thus, hidden surface removal can indeed be performed in time that is linear in $\theta(0)$. But that is still far from satisfactory, because most of the $\theta(0)$ junctions in the view plane are invisible. We surely do not want to spend equal time on all of them. Linear dependence of the expected running time of our algorithm on $\theta(1)$ achieves precisely that. Note that the expectation is with respect to the randomization used in the algorithm, very much as in Quicksort, and does not depend on the input. So we are assuming nothing about the input.

Linear dependence of the running time on $\theta(1)$ is, in essence, the characteristic feature of the new algorithm. What this means is that the work done by the algorithm on the junctions with level l is inversely proportional to l . Intuitively, the work done on junctions quickly decreases as we move farther from the observer. In practice, sizes of the various sets V_l are comparable. Hence, one approximately gets an harmonic series in the expansion for $\theta(1)$. And it follows that the running time of the algorithm is roughly proportional to the size of the actually visible output, i.e. $v(1)$, times logarithm of the (average) depth complexity of the scene (which is quite small generally). A intuitive explanation for the linear dependence of the running time on $\theta(1)$ is the following. The algorithm proceeds by adding one face at a time in a *random* order. The crux of the algorithm lies in ensuring that if a junction q is computed at some instant, during the algorithm, none of the faces added before that instant could obstruct q . Thus, intuitively, a junction is computed with a probability inversely proportional to its obstruction level. This naive explanation is, unfortunately, misleading and fallacious, because the dominant part of the algorithm is not the computation of the junctions, but the computation, and the management, of the so called conflicts. Conflict is a transient, imaginary junction, which has no real counterpart. For other uses of conflicts, see [Mu1,CS,Cl,Mu2,Mu3].

Note that $\theta(\infty)$ is precisely the number of visible junctions. Can one perform hidden surface removal in time that is linear in $\theta(\infty)$, or even $\theta(s)$, where $s > 1$? That is an open question. However, it seems plausible that, in the algebraic computation tree model, $\Omega(n \log n + \theta(1))$ is actually a lower bound for hidden surface removal. The only nontrivial bound that has been proved so far in this model is $\Omega(n \log n)$ (for sorting) [Be]. This is essentially based on the Milnor-Thom bound on the number of connected components of a real algebraic variety. To prove an $\Omega(\theta(1))$ bound in this model will definitely require a deeper insight into a similar aspect of algebraic geometry.

Another feature of our algorithm, in contrast to the previous methods [HG,NSS,Sut,Wa,WA,Sc,Mc], is its essential use of randomization. That randomization should help hidden surface removal is only to be expected. This is because hidden surface removal is a kind of sorting, and the power of randomization has already been demonstrated by Quicksort for sorting in dimension one. However, in higher dimensions, the situation is completely different. For example, the divide and conquer strategy of Quicksort fails to be as powerful in

higher dimensions. The division step of Quicksort is powerful because it results in two problems of a roughly equal size, whereas the division with respect to a geometric element, say a face, in a higher dimension invariably results in an unbalanced division, even if that geometric element were to be chosen randomly. This makes it necessary to follow a different approach in higher dimensions, to be able to exploit the power of randomization. In this approach, Quicksort is to be viewed not as a divide and conquer method, but as a method which exploits the random evolution of the underlying one dimensional configuration. Thus we are led to analyze the nature of a random evolution in higher dimensions, where it is much more interesting and complex than in dimension one. A theory of probabilistic geometric games and θ series in [Mu1,Mu2,Mu3] was developed precisely with the aim of analyzing this random evolution. This theory is a key to the algorithm in this paper. Also of related interest is a theory of random sampling [HW,Cl,CS].

The theoretical time complexity of the new method is much better compared to the previous methods, but can one expect it provide any improvement in practice? This seems quite plausible, because the algorithm is also very simple. With this in mind, we also provide (see the complete manuscript [Mu4]) another related hidden surface removal algorithm. This algorithm makes use of some ideas in the planar partition algorithm of [Mu1]. Its running time is not guaranteed to be linear in $\theta(1)$, but it is quite closely related to the original algorithm. So its observed behaviour might not be too different. Besides, it maintains less auxiliary information than the first algorithm, and this might be significant in practice. Both algorithms make no use of coherence in the image plane. This should make them robust with respect to the rounding errors and degeneracies in the input. This is especially so because randomization is well known for its suppressing the propagation of error.

On the other hand, our algorithm is a general purpose hidden surface removal algorithm, which can not detect cheaply special situations, such as a car in front of a grass field, a box with lots of things inside etc. In fact, if one could prove our above mentioned lower bound conjecture, that would provide a formal proof that a general purpose hidden surface algorithm must take at least $\Omega(\theta(1))$ time to detect such special situations. It is as important to use the usual heuristics, such as clipping and hierarchical comparisons, in conjunction with this method, as with any other method. We shall not deal with the question of heuristics in this paper. We shall, however, mention that [Mu2] gives a new clipping technique called *virtual clipping*, which has all advantages of the conventional clipping, in the amortized sense, but with a logarithmically small overhead in comparison.

For hidden surface removal in the presence of intersecting faces, curved faces etc. see [Mu5].

3 Algorithm

We assume that we are given some standard specification of the scene, which consists of a specification of the faces in the scene together with their edges and vertices. Let n be the number of faces. We assume that the faces are

nonintersecting (intersecting faces are treated in [Mu5]), and that the facelengths are bounded by a constant. But the faces are allowed to have arbitrary shapes. We also assume that the usual preprocessing operations such as perspective transformation, clipping against the view window, culling out irrelevant faces, have already been done. We are thus dealing with only a simple orthographic projection of the scene along the z axis. Given a face f in the scene, we shall denote its projection on the xy plane by $\bar{f}$. The boundary of $\bar{f}$ will be denoted by $\partial\bar{f}$. Similarly the xy projection of an edge e or a vertex v in the scene will be denoted by $\bar{e}$ and $\bar{v}$ respectively. We also assume that there is a special face O in the scene which serves as a background for the whole scene.

The overall organization of the algorithm is as follows. We first form an initial partition H_0 of the (view) window. H_0 is simply the whole window, and thus has just one region. This region will be labelled with the background face O , implying that, in the beginning of the algorithm, O is visible throughout the view window. Starting with H_0 , we form a sequence of *convex* partitions $H_0, H_1, \dots, H_n$, by "adding" the faces in the scene in a random order. This means that H_{k+1} will be obtained from H_k by adding to it a face f_{k+1} , which is randomly chosen from the set of remaining scene faces. As a convention, we shall let f_0 denote the background face O . By this convention, f_0 is the very first "randomly" chosen face. Each region R in H_k will be labelled with the face $f(R)$, which is "currently visible" in R , among the $k+1$ faces, $f_0, \dots, f_k$, that have been added so far. This means that the face $f(R)$ would be visible in R , if the scene were to consist only of the faces $f_0, \dots, f_k$. H_n will be the final *visibility partition* of the view window. H_n tells us which face in the scene is visible at any given location in the view window. Now one can scan H_n from left to right, and paint each region of H_n in accordance with the face visible in that region.

Let us specify in more detail the partition H_k of the window induced by the *randomly chosen* faces $f_0, \dots, f_k$. For this purpose, imagine that the scene consists only of the faces $f_0, \dots, f_k$. Consider an edge e of one of these faces. It is clear that, in general, one will only see parts of e — possibly none. The xy projections of these disconnected visible parts of e will be called fragments. Fragments of the edges of $f_0, \dots, f_k$ induce a partition of the view window. Unfortunately, the regions of this partition can have complicated boundaries. To overcome this difficulty, we pass a vertical attachment through each loose endpoint of a fragment, which extends in upward, as well as downward, direction until it hits either another fragment or a window border. For the input in fig 1a, we have shown in fig 1b the partition that results, if the first three randomly chosen faces are A, C, D . The partition obtained in this fashion is convex, but there is still one theoretical difficulty. A region of this partition, at least theoretically, can have arbitrarily many borders. To overcome this difficulty, we also pass a vertical attachment through every t -junction, where two fragments meet. This vertical attachment will, however, extend in only one direction (see fig 1c). What we have now is the partition H_k . Note that each region of H_k is a trapezoid; a triangle is regarded as a degenerate trapezoid. Hence, H_k can be called a trapezoidal decomposition induced by

the visible fragments of the edges of $f_0, \dots, f_k$. We point out, however, that in practice, it might be more desirable to impose just a convexity requirement, because a convex partition, as in fig 1b, is less expensive to maintain than a trapezoidal decomposition, as in fig. 1c. Accordingly, the algorithm, we are about to present, can be readily modified. This easy modification will be left to the reader.

We still have to specify, exactly how the trapezoidal decomposition H_k is to be represented. One possibility is an obvious planar graph representation. This has a disadvantage that the representation of a fixed region of H_k can be arbitrarily large, as it can have arbitrarily many vertices on its border. In [Mu1], we presented another representation of a planar partition, which does not have this defect. It is based on the following notion of visibility (this notion of visibility has nothing to do with visibility, as in the context of hidden surface removal):

Definition 1 A vertex v of the partition is said to be visible in a face R , if ∂R , the boundary of R , has a tangent discontinuity at v .

In the partition given in fig 1c, u is visible in R_0, R_1 and R_4 , whereas it is invisible in R_3 . Each trapezoid of H_k will be specified by a circular list of the *visible* vertices on its border. Thus the representation of R_3 in fig 1c is (a'_1, a_1, a_2) . If v is visible in R , we shall refer to the corresponding entry in the representation of R by v_R . In addition to specifying the faces of the partition, we also need to specify adjacency relationships at each vertex to complete the representation. If a vertex v is visible in R , we shall link v_R to $v_{R'}$, where R' is the next face, in the counterclockwise order, in which v is visible. Thus at the vertex u in fig 1c, we will have links in the order: $u_{R_4} \rightarrow u_{R_1} \rightarrow u_{R_0} \rightarrow u_{R_4}$. To avoid any confusion, we shall, henceforth, reserve the terms faces, edges, and vertices exclusively for those in the scene. Vertices of the partition H_k will be called *junctions*. A junction can be of three kinds: 1) it can be located at an intersection between two fragments, (e.g. the junction u in fig 1c), 2) it can be located at the projection of a scene vertex (e.g. a_0 in fig 1c), 3) or else it can be a point of attachment, i.e. a t -junction, where some vertical attachment meets a fragment or a window border (e.g. a'_1, a''_1, u' in fig 1c). Junctions of the first two kinds will be called *concrete*. A point of attachment, on the other hand, is not concrete. The faces of H_k will always be called trapezoids or regions. A border of a region in H_k is defined as usual. A border in H_k is called concrete if it lies on a fragment (e.g. the border (a'_1, a_2) of R_3 in fig 1c). A border which lies on a vertical attachment is not concrete (e.g. (a'_1, a_1) of R_3).

As already mentioned, each region R of H_k will also be labelled with the face, among the $k+1$ faces $f_0, \dots, f_k$ added so far, that is currently visible there. This face currently visible in R will be denoted by $f(R)$. Thus $f(R_3) = A, f(R_5) = C, f(R_1) = O$ etc.

This completes the specification of H_k . Unfortunately, there is still not enough information to make the "addition" of the next randomly chosen face f_{k+1} to H_k easy. The reason is that we have to somehow know which regions of H_k will be affected by the addition of f_{k+1} . With this in mind,

we also associate with H_k some conflict information. For the use of conflicts in other contexts see [Mu1, CS, Cl, Mu2, Mu3].

First let us make a few definitions. A face g , other than the added faces $f_0, \dots, f_k$, and a trapezoid $R \in H_k$ are said to be in conflict if

1. $\bar{g}$, the xy -projection of g , intersects R , and
2. $f(R)$ does not obscure g in the region R .

Intuitively, if g is in conflict with R , then g will be at least partially visible in R , if it were the next face to be chosen for addition to H_k . We shall say that $f(R)$ is a background face of this conflict between g and R .

If g is in conflict with $R \in H_k$ then one, or possibly more, of the following kinds of conflicts must occur.

1. **a trapezoid-vertex conflict:** A vertex v of g is in conflict with the trapezoid R , i.e. to say $\bar{v}$, the xy -projection of v , lies in R . This conflict is said to be located at $\bar{v}$.
2. **a border-edge conflict:** An edge e of g is in conflict with a border b of R , i.e. to say $\bar{e}$, the xy -projection of e , intersects b . This conflict is said to be located on b at the intersection of $\bar{e}$ and b .
3. **a junction-face conflict:** a concrete junction t of R is in conflict with g , i.e. to say $\bar{g}$, the xy -projection of g , contains t . This conflict is said to be located t . Note that we do not consider conflicts at junctions, which are not concrete.

Each of these conflicts is said to have $f(R)$ as its background face.

Examples:

In fig. 1d, we have overlayed on H_3 , for the sake of visualization, those parts of the boundaries of the remaining faces B and E , that are not occluded by the already added faces A, C, D . Referring to this figure, the following observations can be made. Face E conflicts with regions R_0, R_1, R_4 , but no other region, as it is occluded everywhere else. E conflicts with the junction u in R_1, R_0, R_4 . Because u is "invisible" (Definition 1) in R_3 , the question about a conflict in R_3 does not arise. If there were some unadded face, behind C looking from u , it would conflict with u in R_1 , but not in R_0 and R_4 . The background face of the conflict between u and E , in R_0 , is C , whereas the background face of the conflict between u and E , in R_1 , is O . These conflicts have the same locations but different backgrounds, hence they are completely different. On the other hand, the conflict between u and E , in R_0 , has the same location as well as background as the conflict between u and E in R_4 . Indeed, there is no real reason to regard these two conflicts as different. Hence, they can be identified. The vertex e_0 is in conflict with the trapezoid R_0 , whereas e_3 is in conflict with R_4 . Other vertices of E are all occluded by the added faces, and hence, are not in conflict with any trapezoids. The edge (e_2, e_3) of E is in conflict with the border (u, a_2) of R_4 . But it is *not* in conflict with the border (a'_1, a_2) of R_3 , which lies on the other side of the fragment (a_0, a_2) . On the other hand, the edge of (b_0, b_3) of B is in conflict with both borders, (u, a_2) of R_4 , as well as, (a'_1, a_2) of R_3 . Though the locations of both these conflicts with (b_0, b_3) are the same, their background faces are different (C and A respectively). Hence, these two conflicts

are to be regarded as completely different. The same edge (b_0, b_3) also conflicts with the vertical attachment (u, u') in R_0 , as well as, R_4 . Locations, as well as backgrounds, of these conflicts are the same. Again, there is no real reason to regard the border-edge conflicts such as these, which lie on the opposite sides of the same vertical attachment, as different. They too can be identified.

There are various ways to associate conflict information with H_k . Basically, one should be able to deduce in an efficient fashion, the various kinds of conflicts, as defined above, that occur within H_k . The scheme we will choose is not the most efficient, both in terms of memory and time. However, it has an advantage that it simplifies the description of the algorithm. The reader will be able to see many other alternative schemes.

The conflict information associated with H_k will be organized so that:

1. Given any concrete junction t in H_k , we know the unadded faces that are in conflict with t , and conversely, given any unadded face f , we know the concrete junctions in H_k that are in conflict with f . Note that we do not store conflicts with the points of attachment in H_k .
2. Given any border b in H_k , concrete or otherwise, we know the scene edges that are in conflict with b , and conversely, given any edge e , we know the borders in H_k that are in conflict with e . Moreover, we assume that the borders of all faces in the scene are oriented so that the corresponding orientations of their xy -projections is counterclockwise. This orients $\bar{e}$, the projection of e , too. We assume that all conflicts of e are kept ordered, by simple links, along the orientation of e .
3. Given any region $R \in H_k$, we know the scene vertices that are in conflict with R , and conversely, given any vertex v , we know the region in H_k that is in conflict with v . Moreover, we assume that all conflicts with a given region R are kept ordered, by simple links, according to the x -coordinates of their locations.

The conflict size of a region $R \in H_k$ is defined to be the total number of conflicts, of all three kinds, that are associated with R .

Now we are ready to describe how the addition of a randomly chosen face $f = f_{k+1}$ to the partition H_k is achieved.

3.1 Preliminary update along the boundary

In this step, we split the trapezoids of H_k that are in conflict with the boundary ∂f , i.e. the trapezoids which conflict with either an edge or a vertex in ∂f . Let us specify this step in more detail. Exploiting the ordering of the border-edge conflicts, we visit these trapezoids along the given counterclockwise orientation of ∂f . Note that we are not visiting all regions of H_k that intersect ∂f , but only the ones which conflict. Intuitively, these are the regions where ∂f is not occluded by any of the added faces $f_0, \dots, f_k$. Assume, for example, that the fourth randomly chosen face to be added to H_3 , in fig 1d, is E . In this case, we shall travel on ∂E from the conflict on (c'_0, u) to the one on (u, a_2) . Here the

conflicting part of $\partial \bar{E}$ is connected. In general, the conflicting part of ∂f_{k+1} can have many connected components. But that does not cause any problems. As we visit the conflicts on $\partial \bar{f}$ in the counterclockwise direction, we split every conflicting border that we visit. Moreover, when we split a border adjacent to a vertical attachment through some concrete vertex, say v , we retain only that split part which contains v . For example, in the addition of $f_4 = E$ to H_3 in fig 1d, we shall retain only the upper half of the border (u, u') . In addition, we pass a vertical attachment through every new intersection that corresponds to a conflict on a concrete border of H_k . We also create a new junction for every conflict that is located at the xy -projection of a vertex of f_{k+1} . Remember that we perform these actions, *as we travel*. It is possible that, during this travel, we will visit the same region of the old partition H_k more than once. But this should cause no problem. It only means that we have to split sometimes an already added vertical attachment even further.

When we travel in this manner on the conflicting parts of $\partial \bar{f}$, what we obtain at the end is a new trapezoidal decomposition H'_{k+1} , that is obtained from H_k by appropriately splitting the regions of H_k , that conflicted with ∂f . We also need to label each newly formed trapezoid R' with a face that will be visible in R' after the addition of $f = f_{k+1}$. This is easy to do. Notice that each newly formed trapezoid is either completely covered by $\bar{f}$ or is completely disjoint from $\bar{f}$, except at its boundary. If R' is covered by $\bar{f}$, we label it with f , else R' will retain the old label $f(R)$ of R .

We also have to associate conflict information with each newly formed region R' . R' , in general, consists of parts of some old trapezoids in H_k , say $R_1, \dots, R_l$, that were in conflict with ∂f . So we have to derive the conflict information, to be associated with R' , from the conflict information associated with $R_1, \dots, R_l$. Notice that an unadded face can conflict with R' only if it was in conflict with one of the trapezoids $R_1, \dots, R_l$. Moreover, a face g that was in conflict with some R_j , $1 \leq j \leq l$, can be in conflict with R' only if 1) $\bar{g}$, the xy -projection of g , intersects R' , 2) and, in case R' is covered by $\bar{f}$, the face g is not occluded by f in R' . With the help of these observations, it is easy to derive the conflict information, to be associated with R' , in time proportional to the sum of the conflict sizes of $R_1, \dots, R_l$.

As each $R \in H_k$, that was in conflict with ∂f , can be split into only a bounded number of trapezoids, the preliminary update along the boundary takes time that is $O(\sum_{R \in H_k} \text{conflict_size}(R))$, where R ranges over the trapezoids that were in conflict with ∂f . Needless to say, all trapezoids of H_k in conflict with ∂f and the conflict information associated with them can now be destroyed.

3.2 A preliminary update in the interior

In this step, we access all trapezoids in the partition, that are in conflict with f , and which are in the interior of $\bar{f}$, but are not adjacent to $\partial \bar{f}$. We update the conflict information associated with each of these trapezoids. At the end of this operation, the new trapezoidal decomposition H'_{k+1} , that was obtained above, will have proper, updated conflict

information associated with it.

It is easy to see that every conflicting trapezoid in the interior of $\bar{f}$ has two *concrete* junctions that are in conflict with f . Hence, using the list of junctions in conflict with f , we can readily access all conflicting regions in the interior of $\bar{f}$. For each region R , accessed in this fashion, we do the following:

1. We label R with f , because f will be the face visible in R after the addition of f .
2. We reduce the conflict information associated with R , by discarding every conflict with a face (or its edge or a vertex) that is occluded by f .

The second step requires us to know, for every face g that is in conflict with R , if f occludes (overlaps) g . The same face g might be in conflict with many of the accessed trapezoids. In this case, it is clearly desirable to carry out the required overlap comparison just once, if that is possible. This is indeed possible if f and g do not overlap cyclically, which is always the case if f and g are convex. In the absence of a cyclic overlap, we carry out the overlap comparison between f and g just once. To expediate this comparison, it is often advantageous to test first if the z -extents of f and g overlap; a complicated test is necessary only if they do [NSS].

At the end of this preliminary update in the interior, we have a partition H'_{k+1} , with the correct conflict information associated with it. However, H'_{k+1} might contain several fragments which have been rendered invisible during the addition of f . These fragments have to be removed from H'_{k+1} ; this step is called reconfiguration. After reconfiguration we shall update the conflict information once again. Let us first turn to reconfiguration.

3.3 Reconfiguration

Consider all trapezoids of H'_{k+1} which are labelled with $f = f_{k+1}$ at the end of the preliminary update. The union of these trapezoids form a region $\mathcal{C} = \mathcal{C}_{k+1}$ in the plane. Clearly the trapezoids of H'_{k+1} outside $\mathcal{C}$ need not be considered in reconfiguration. Note that the region $\mathcal{C}$ can have many connected components and the connected components themselves can have complicated, disconnected boundaries (see fig. 2). All these factors have to be taken into account during reconfiguration.

Let $\partial \mathcal{C}$ be the boundary of $\mathcal{C}$. It is easy to see that a vertical attachment can not form a part $\partial \mathcal{C}$. Thus every border of H'_{k+1} that lies on $\partial \mathcal{C}$ is concrete. Moreover, every such border on $\partial \mathcal{C}$ will remain visible after the addition of $f = f_{k+1}$. Thus the fragments which have become invisible after the addition of f must lie strictly in the interior of $\mathcal{C}$. And conversely, every fragment which lies in the interior of $\mathcal{C}$ has become invisible after the addition of $f = f_{k+1}$. Hence a naive method to reconfigure is the following: remove everything within $\mathcal{C}$, and then find a new trapezoidal decomposition of $\mathcal{C}$. This can turn out to be too expensive for two reasons. First, $\partial \mathcal{C}$ can be very complicated, hence one will need the power of a full fledged triangulation procedure. Second, we also have to "relocate" later the conflicts of the scene vertices, which were located within $\mathcal{C}$. (We need to relocate other kinds of conflicts too.) This means we will

also need a full fledged a planar point location algorithm. Obviously, this approach is too costly.

Fortunately, there is a less expensive way to reconfigure. It exploits the fact that what we have at our disposal is not just a boundary of $\mathcal{C}$, but some trapezoidal decomposition of $\mathcal{C}$. It is, of course, true that this trapezoidal decomposition has many unnecessary trapezoids. But it should still be possible to make use of this initially given decomposition somehow. The idea is to remove only a few of the unnecessary trapezoids at a time.

More precisely, let $A_1, \dots, A_h$ be the set of fragments within $\mathcal{C}$. We shall "remove" these fragments one at a time, *but in a random order*. This will be a randomized, decremental algorithm. In contrast, the overall hidden surface removal algorithm is randomized, and incremental. Clearly, the two strategies are complementary.

As we said earlier, we shall deal with the problem of updating conflicts later. So let us consider the removal of a single fragment in more detail. Fig. 3a shows the fragment $l = (l_0, l_1)$, which has been randomly chosen for removal, together with the adjacent trapezoids. Let $a_0, a_1, \dots, a_q$ be the junctions adjacent to the "lower" side of l and let $b_0, b_1, \dots, b_r$ be the junctions adjacent to the "upper" side of l . Some of these will be the junctions, where other fragments meet l and the remaining will be the points of attachment. Note that, by our definition 1, the junctions $a_0, \dots, a_q$ are "invisible" on the upper side of l , and $b_0, \dots, b_r$ are "invisible" on the lower side of l . So we need to merge the two sorted streams $a_0, \dots, a_q$, and $b_0, \dots, b_r$ to get a sorted stream $c_0, \dots, c_{q+r+1}$. Having done this, it is easy to remove l , by extending the vertical attachments ending on the upper side of l in the lower direction, and by extending the vertical attachments ending on the lower side of l analogously. See fig. 3b.

When this procedure is repeated, in a random order, for all fragments lying within $\mathcal{C}$, we obtain, at the end, the desired trapezoidal decomposition of $\mathcal{C}$.

This gives us the partition H_{k+1} , that we sought. But we still have to associate conflict information with the newly formed trapezoids. This is done next.

3.4 Relocating and generating conflicts

In this step, we derive the conflict information, to be associated with the trapezoids formed during reconfiguration, from the conflict information that was associated with those trapezoids in H'_{k+1} , that existed within $\mathcal{C} = \mathcal{C}_{k+1}$ before reconfiguration. We shall proceed by cases.

trapezoid-vertex conflicts

For every scene vertex v , that was in conflict with a trapezoid $R \in H'_{k+1}$ within $\mathcal{C}$, we need to find the new trapezoid $R' \in H_{k+1}$ within $\mathcal{C}$ that contains v . We call this step relocation of the conflict at $\bar{v}$. Relocating this conflict at the end of reconfiguration is a bit difficult. Instead, we shall modify the reconfiguration procedure a little, so that, at the end of reconfiguration, we also know the conflicting trapezoid R' for every scene vertex v , that was in conflict with some trapezoid $R \in H'_{k+1}$ within $\mathcal{C}$. The idea is to perform relocation of these conflicts along with reconfiguration.

Note that, because of the way in which we organize the

conflict information, the vertex-trapezoid conflicts located within any trapezoid of H'_{k+1} are kept ordered by their x -coordinates. We shall ensure that this invariant holds throughout reconfiguration. More precisely, at any stage of the reconfiguration procedure, all conflicts of the scene vertices within any given trapezoid will be kept ordered by their x -coordinates.

So reconsider the removal of a randomly chosen fragment $l = (l_0, l_1)$ as in fig. 3a. We shall perform the following extra steps during this removal. First, we concatenate the lists of the trapezoid-vertex conflicts lying in the trapezoids adjacent to the the upper side of l , and do the same for the lower side of l . Next we merge these two sorted lists. By a simple "interleaving" procedure, one can now split this merged list into various sorted lists of conflicts, corresponding to the new trapezoids formed after the removal of l (fig 3b). Now one is ready for the removal of the next randomly chosen fragment.

border-edge conflicts

Let us see how one can update the border-edge conflicts within $\mathcal{C}$. Note that we do not need to worry about the conflicts which lie outside $\mathcal{C}$, including the ones which lie on the "outer" side of $\partial\mathcal{C}$.

First, let us worry about the update on the inner side of $\partial\mathcal{C}$. Each new border-edge conflict on the inner side of $\partial\mathcal{C}$ is obtained from a similar conflict that existed in H'_{k+1} before reconfiguration. Hence, our task is basically to relocate the conflicts that existed in H'_{k+1} , on the inner side of $\partial\mathcal{C}$ before reconfiguration. Ignoring the vertical attachments in H_{k+1} , that end on $\partial\mathcal{C}$, one sees that $\partial\mathcal{C}$ is a union of many linear segments. Group the above conflicts in H'_{k+1} according to the segments in $\partial\mathcal{C}$ containing their locations. Let l be one of the segments on $\mathcal{C}$, and let S_l be the set of border-edge conflicts located on the inner side l . At the end of reconfiguration, the vertical attachments ending on $\partial\mathcal{C}$ can split the inner side of l into many borders. Hence, for each conflict in S_l , we need to find out the new border that will contain this conflict. We shall find out this border by a simple sequential search along l .

So assume that we have relocated the conflicts on the inner side of $\partial\mathcal{C}$. We also need to generate the border-edge conflicts, that lie on the vertical attachments within $\mathcal{C}$. This will be done next.

If g is scene face, define a *concrete* conflict of ∂g to be a conflict of a vertex on ∂g with some trapezoid, or a conflict of an edge in ∂g with some *concrete* border. At this stage of the algorithm, we know all concrete conflicts within $\mathcal{C}$. Consider a concrete conflict of ∂g within $\mathcal{C}$. This is either a conflict of an edge of ∂g with a border on $\partial\mathcal{C}$, or a conflict of a vertex in ∂g with a trapezoid within $\mathcal{C}$. We shall travel from this conflict to the next concrete conflict of ∂g within $\mathcal{C}$, in the counterclockwise direction along ∂g , if it is possible to do so without leaving $\mathcal{C}$. This travel only involves skipping an appropriate number of vertical attachments in H_{k+1} , until the next concrete conflict is reached. For every vertical attachment encountered on the way, we generate a border-edge conflict. This procedure is repeated for every concrete conflict of ∂g within $\mathcal{C}$.

The same can be done for every ∂g which had a conflict within $\mathcal{C}$. This procedure will generate all border-edge con-

licts on the vertical attachments within $\mathcal{C}$.

junction-face conflicts

Fortunately, nothing elaborate needs to generate these conflicts. What makes our task easy is the fact that we store these conflicts only at concrete junctions. It is easy to see that, for every concrete junction t in H_{k+1} , the faces which conflict with t in H_{k+1} are precisely the ones which conflict with t in H'_{k+1} . This makes the updating task straightforward.

At the end of this step, H_{k+1} has been associated with proper conflict information. Now we are ready to add the next randomly chosen face to H_{k+1} .

This finishes the description of the hidden surface removal algorithm.

4 Probabilistic games

A basic idea behind the analysis is to analyze a random evolution of the underlying partition H_k and the associated conflict information. As this global evolution is overwhelmingly complex, we need a tool to break this complex behaviour into tractable parts. The tool, that we will use, is a probabilistic theory of geometric games, which was developed in [Mu1, Mu2, Mu3]. We shall summarize in this section the results from this theory, that we will need in our analysis.

Suppose that we are given a universe set N of some elements. The game we are going to play in this section, consists in drawing the elements of N in a random order without replacement. Fix two disjoint subsets S and R of N throughout this section. Let $r = |R|$, and $s = |S|$. Let us associate an observer with this pair of sets S and R . The following rule will determine the active phase of the observer during the game. The observer is active at any given instant of the game if the following two conditions are satisfied at that instant:

- 1) all elements from R have been chosen,
- 2) and no element from S has been chosen so far.

If the observer does indeed become active during the game, he will go into the inactive phase as soon as the second condition is violated. The sets R and S will be called the sets of triggers and stoppers respectively.

Now assume that we are given one more subset M of the universe N , and let $m = |M|$. M need not be disjoint from either S or R . But we shall assume that M is linearly ordered. Imagine, just for the sake of visualization, putting elements of M on the positive real axis, according to the ordering of M , with the ordering increasing in the positive direction. Place the observer o , associated with the pair of sets S and R , at the origin. Assume that the observer can "see" only along the positive real axis. We shall soon make this notion precise. We say that an element $b \in M$ was observed by o , at some instant of the game, if b was chosen at that instant and no element $c \in M$, s.t. $c < b$, was chosen before this instant. The idea is that the chosen elements in M are supposed to act as barriers to the sight of the observer. Let O be the number of elements observed by o in his active phase. If the observer never became active in the game, O is defined to be zero.

Theorem 1

$$\begin{aligned} E(O) &= O\left(\frac{1}{\binom{s+r}{r}}\right), \quad \text{for } t \geq 1 \\ &= O(1 + \ln(m+1)), \quad \text{for } s = 0, t = 0 \\ &= O\left(1 + \ln\left(1 + \frac{m}{s}\right)\right), \quad \text{for } s > 0, t = 0. \end{aligned}$$

Now let us define the visibility span of the observer, at any instant during the game, as follows: Let b be the least element in M chosen at or before this instant. Then the visibility span of the observer at this instant consists of all elements in M less than b . Notice that none of the elements in the visibility span at a given instant, could be chosen before that instant. Thus the visibility span at any instant consists of the elements in M , not yet chosen, which lie within the visibility extent of the observer o . Let V be the number of elements in the visibility span of the observer, at the instant he became active. V is defined to be zero, if the observer does not become active at all. We shall define one more random variable W as follows. Charge the observer a cost equal to the length of his visibility span, i.e. V , at the moment he became active. After this, every time the observer observes in his active phase some element of M , say b , we charge the observer a cost equal to the number of elements $\leq b$ in M . (Thus the cost is equal to the length of the visible span at the instant b was observed.) Let W be the total cost charged to the observer. Then it is clear that $V \leq W$.

Theorem 2

$$\begin{aligned} E(V) \leq E(W) &= O\left(\frac{r}{\binom{s+r}{r-1}}\right), \quad \text{for } t \geq 2 \\ &= O(\ln(m+1) + 1), \quad \text{for } s = 0, t = 1. \\ &= O\left(1 + \ln\left(1 + \frac{m}{s}\right)\right), \quad \text{for } s > 0, t = 1. \end{aligned}$$

5 A part of the analysis

We shall first state the expected running time of the algorithm. Let $c(f)$, the coupling coefficient of a face f , be the number of faces g whose projected borders ∂g intersect $\bar{f}$. For a fixed point q in the view plane, let $c(q) = \max\{c(f)\}$, where f ranges over the faces whose projections cover q . If q is an intersection of $\partial \bar{f}$ and $\partial \bar{g}$, recall from Section 2 that the *obstruction level* of q , $level(q)$, is the number of faces in the scene, which obscure either f or g at q . Intuitively, the presence of any of these $level(q)$ faces will make the junction between the boundaries of f and g invisible at q . If q is the xy -projection $\bar{v}$ of a vertex v of some face f , define $level(q)$ to be the number of faces in the scene which occlude v at q . Let $depth(q)$ be the number of faces whose projections cover q .

Theorem 3 *The expected running time of the hidden surface removal algorithm is $O(n \log n + n \log(1+c1) \log(1+r) + \theta(1) + \log(1+c2)\theta(2))$, where r is the average of the ratio*

$depth(q)/level(q)$ over the projections of the scene vertices, and c_1 and c_2 are the averages of the coupling coefficients $c(q)$ over the projections of the vertices, and intersections of the projected boundaries respectively.

The factor $\log(1+c_2)\theta(2)$ is smaller than $\theta(1)$ in practice, but in theory, it is a nuisance. This factor can be removed from the running time, at the cost of a considerable complication in the algorithm [Mu5]. On the other hand, we have not been able to remove, so far, the factor $n \log(1+c_1) \log(1+r)$. This factor is, in practice, much smaller compared to $\theta(1)$. It is plausible that the algorithm as presented here, or its variant, has $O(n \log n + \theta(1))$ expected running time. That is an open question.

A complete proof of Theorem 3 can be found in the complete manuscript of the paper [Mu4]. Here we shall deal with only a part of the analysis, that deals with bounding the expected number of conflicts created in the algorithm. In this section, we shall prove the following

Theorem 4 *The expected number of conflicts created and destroyed in the whole course of the algorithm is $O(n \log n + \theta(1))$.*

Our prime goal is to illustrate the use of probabilistic games in Section 4. Complete analysis is, in a sense, a long exercise in applying these games over and over again.

Recall our convention to regard two conflicts with same locations but different background faces as different. This makes Theorem 4 more interesting. Thus, if many conflicts with different background faces were created, during the algorithm, at the same location, they are all counted as distinct in the theorem. A created conflict can be destroyed but once, hence we need not worry about the destroyed conflicts anymore. In the proof, we shall treat the three different kinds of conflicts separately.

Junction-face conflicts

Recall that, in the algorithm, we stored conflicts of this kind at the concrete junctions only.

Lemma 1 *The expected number of conflicts, created in the algorithm, that are located at a concrete junction q is $O(\frac{1}{1+level(q)})$, if q is located at an intersection of the projections of two face boundaries, and $O(1 + \log(\frac{1+depth(q)}{1+level(q)}))$, if q is located at the projection of a vertex. (If a concrete junction was never created at q , during the algorithm, the number of conflicts created at q is defined to be zero.)*

Proof.

We shall only consider the first case, where q is located at an intersection of say ∂f and $\partial \bar{g}$. Fix f, g and q . Refer to fig. 4a. Note that, by our Definition 1, q is invisible in the zone Q_0 . Thus no conflict located at q can come into existence in Q_0 . Also note that among the conflicts located at q , the ones which occur in the zone Q_1 , and the ones which occur in the zone Q_2 have different background faces. Hence one has to treat the conflicts in these two zones separately. We shall only treat here conflicts in the zone Q_1 . A similar treatment of the conflicts in the zone Q_2 will be omitted.

The lemma will be derived by applying the probabilistic game of Section 4. The setup is the following. The universe

set N will be the set of all faces in the scene. (This will be the choice for the universe set throughout the analysis of the algorithm.) We shall place an observer at q who can "see" only in the positive z -direction. M , the set of objects that the observer can see, is defined to be the set of faces in the scene, whose projections cover q . The set M is linearly ordered according to the increasing z -coordinate of the faces in M at the location q . The sets R and S of stoppers and triggers, that will determine the active phase of the observer, are defined as follows. Let $R = \{f, g\}$, and let S be the set of $level(q)$ faces which obscure f or g at q . Intuitively, the observer remains active as long as the junction q remains visible in the algorithm, and it is easy to see that q is visible iff all elements of R are chosen, and no element from S is chosen. In this setup, it is easy to see that the number of conflicts, created at q in the zone Q_1 , is precisely the random variable W defined in Theorem 2. This takes into account the fact that the conflicts located at q , but having different background faces, are to be considered different. Now the lemma follows from Theorem 2. $\square$

Trapezoid-vertex conflicts

Let v be a vertex of some face f . Fix f as well as v .

Lemma 2 *The expected number of trapezoid-vertex conflicts, created in the algorithm, that are located at $q = v$, is $O(1 + \log(\frac{1+depth(q)}{1+level(q)}))$.*

Proof. Place, as in Lemma 1, an observer at q , who can only see in the positive z -direction. Let M be the set of faces, whose projections cover q , and let S be the set of faces, which obscure v at q . The set of triggers R is now defined to be an empty set. The reason is that the trapezoid-vertex conflicts located at q can come into existence even when the face f has not been chosen for addition by the algorithm, assuming, of course, that no face from S has been chosen. In this set up, it is clear that the number trapezoid-vertex conflicts which are created at q is precisely the random variable O defined in Theorem 1: everytime such a conflict with a background face h is created, the observer at q can "see" the face h that is being added at the time of creation. Now apply Theorem 1. $\square$

Border-edge conflicts

Each border-edge conflict, created in the algorithm, is located either on a vertical attachment or on a fragment passing through some concrete junction.

Lemma 3 *The expected number of border-edge conflicts, created in the algorithm, that are located on a vertical attachment or a fragment through a concrete junction q , is $O(\frac{1}{1+level(q)})$, if q is located at an intersection of the projections of two face boundaries, and $O(1 + \log(\frac{1+depth(q)}{1+level(q)}))$, if q is located at the projection of some vertex. (If a concrete junction was never created at q , during the algorithm, the number of conflicts created at q is defined to be zero.)*

Proof. We shall only consider the first case. So suppose that q is located at an intersection of the projected borders ∂f and $\partial \bar{g}$ of two faces f and g . Fix f, g , as well as q . We will only estimate the conflicts created on a vertical attachment through q ; the ones that are created on a

fragment through q can be estimated similarly. Assume, without loss of generality, that f is in front of g at q . Notice that, depending upon how ∂f and ∂g are situated at q , a vertical attachment through q will either extend upwards or downwards. Assume, without loss of generality, that it extends downwards. Referring to fig.4, notice that a vertical attachment lies either in the region that does not intersect $\bar{g}$ or in the region that intersects $\bar{g}$. We will only consider the first case, and omit the second (easier) case.

Let L be the set of $level(q)$ faces which obscure either f or g at q . Let S_q be the set of faces, other than f and g , whose projections cover q .

Fix a face $h \in S_q$. We shall estimate the expected number of conflicts, created on a vertical attachment through q , which have h as their background face. First notice that h can be a background face for such a conflict only if $h \notin L \cup \{f, g\}$. Assume, henceforth, that $h \notin L \cup \{f, g\}$. Let $level_q(h)$ be the obstruction level of h at q , i.e. the number of faces, other than f or g , which obscure h at q . Obviously $level_q(h) \geq level(q)$.

Claim: The expected number of conflicts created on a vertical attachment through q , which have h as their background is $O(\frac{1}{(2+level_q(h))^2})$.

This will prove the lemma since, by elementary calculus,

$$\sum_h \frac{1}{(2+level_q(h))^2} \leq \frac{1}{1+level(q)},$$

where h ranges over all faces in S_q which do not belong to $L \cup \{f, g\}$.

Let $S_h \subseteq S_q$ be the subset of those faces which are in front of h at q . Let T be the semi-infinite imaginary line through q extending downwards. A conflict with a border ∂k of a face k , with h as a background face of the conflict, can come into existence on T , only if k obscures h somewhere on T , but not at q . This condition is just necessary, and not sufficient. Let M_h be the set of faces k , which satisfy the above condition. Linearly order M_h by letting $k_1 < k_2$ iff ∂k_1 intersects T before ∂k_2 , i.e. the intersection between ∂k_1 and T , that is closest to q , is closer to q than the similar closest intersection between ∂k_2 and T . The following observation is crucial:

Observation 1 If $k_1 < k_2$, and k_1 is chosen for addition in the algorithm before k_2 , then ∂k_2 cannot conflict with a vertical attachment through q , with the face h in the background. (This does not hold, if the background is allowed to vary.)

Now let $R_h = \{f, g, h\}$. Letting R_h , S_h and M_h play the roles of R , S , M in Theorem 2, it follows from the above observation that the number of conflicts, that come into existence on T with h in the background, is bounded, upto a constant factor, by the random variable V in that theorem: all faces whose boundaries create these conflicts are in the "visibility span" of the observer located at q , when the face h was added. The claim now follows from that theorem. $\square$

Using Lemma 1, Lemma 2, Lemma 3, and summing over all concrete junctions, we conclude that the expected number of conflicts created during the algorithm is $O(n \log n + \theta(1))$.

This finishes the proof of Theorem 4.

References

- [Be] Ben-Or M., "Lower bounds for algebraic computation trees", *Proc. of the 15th STOC*, 83.
- [CE] Chazelle B., Edelsbrunner H., "An optimal algorithm for intersecting line segments in the plane", *Proceedings of the FOCS*, 1988.
- [Cl] Clarkson K., "Applications of random sampling to computational geometry, II", *Proc. 4th Ann. ACM Symposium on Computational Geom.*, 1988.
- [CS] Clarkson K., Shor P., "Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental", *Proc. 4th Ann. ACM Symposium on Computational Geometry*, 1988.
- [HG] Hamlin G., Gear C., "Raster-Scan hidden surface algorithm techniques", *Computer Graphics*, vol. 11, no. 2, pp. 206-213.
- [HW] Haussler D., Welzl E., "Epsilon nets and simplex range queries", *Proc. 2nd Ann. ACM Symposium on Computational Geometry*, 86.
- [Mc] McKenna M., "Worst-case optimal hidden surface removal algorithm", *ACM transactions on graphics*, vol.6, no. 1, 1987.
- [Mu1] Mulmuley K., "A fast planar partition algorithm, I", *Proceedings of the 29th FOCS*, 1988, full version to appear in a special computational geometry issue of the *Journal of Symb. Logic*.
- [Mu2] Mulmuley K., "A fast planar partition algorithm, II", To appear in the Proceedings of the 5th Ann. ACM symposium on Computational Geometry, 89, full version submitted to JACM.
- [Mu3] Mulmuley K., "On levels in arrangements and Voronoi diagrams", *Technical report, TR 88-21, University of Chicago, December, 88*, submitted to the *Journal of Discrete and Computational Geom.*
- [Mu4] Mulmuley K., "An efficient algorithm for hidden surface removal, I", a complete manuscript.
- [Mu5] Mulmuley K., "An efficient algorithm for hidden surface removal, II", in preparation.
- [NSS] Newell M., Newell R., Sancha T., "A new approach to the shaded picture problem", *Proc. ACM. National Conf.*, 1972.
- [Sc] Schmitt A. "Time and space bounds for hidden line and hidden surface algorithms", *Eurographics*, 81.
- [Sut] Sutherland, I. E., R. F. Sproull, and R. A. Schumaker, "A characterization of ten hidden surface algorithms", *Computing Surveys* 6: 1-55, 1974.
- [Wa] Warnock J., "A hidden surface algorithm for computer generated half-tone pictures", *computer science dept., University of Utah, TR 4-15*, 1969.
- [WA] Weiler K., Atherton P., "Hidden surface removal using polygon area sorting", *computer graphics (Proc. SIGGRAPH)*, July, 77.

Figure 1a: Input

Figure 1c: H_3

Figure 4a

Figure 1b: H'_3