Exercises

*9.1.2 Generalize the algorithm in this section so that it can handle intersecting polygons. Prove the bound in Theorem 9.1.4 for this algorithm, with appropriate new interpretation.

**9.1.3 (Dynamic hidden surface removal)

(a) Make the on-line hidden surface removal algorithm in Exercise 9.1.1 fully dynamic by allowing deletions of polygons. (Hint: Use dynamic shuffling.)

(b) Show that the expected structural change in the visibility map over a random (N, δ)-sequence is $O(\Theta(N, 0) \log n)$, if the signature δ is weakly monotonic.

(c) Similarly, show that the expected conflict change over a random (N, δ)-sequence is $O(\Theta(N, 1) \log n)$, if the signature δ is weakly monotonic.

(d) Show that the expected running time of the dynamic hidden surface removal algorithm over a random (N, δ)-sequence is $O(\Theta(N, 1)\text{polylog}n)$, if the signature δ is weakly monotonic.

9.2 Binary Space Partitions

In this section, we consider the problem of priority generation that arises in connection with the image space rendering of the visible view. Recall that the problem is the following. We are given a set N of n polygons in R^3. The goal is to break the polygons in N into pieces so that the resulting pieces do not admit overlapping cycles with respect to any viewpoint. Let $\Gamma(N)$ denote the resulting set of pieces. We want the size of $\Gamma(N)$ to be as small as possible. We also wish to have a data structure so that, given any viewpoint, a priority order over $\Gamma(N)$ with respect to this viewpoint can be generated quickly—say, in $O(|\Gamma(N)|)$ time. In this section, we shall present one method for decomposing the polygons in N and generating a priority order with respect to a given viewpoint. It is based on the so-called *Binary Space Partitions* (BSP).

9.2.1 Dimension two

We shall first illustrate the basic ideas underlying Binary Space Partitions in the simplest setting by considering the analogous problem in two dimensions. In two dimensions, the set N consists of segments in the plane rather than polygons in R^3, and the view from any viewpoint is one-dimensional rather than two-dimensional. Observe that in two dimensions, the problem of overlapping cycles cannot arise for any viewpoint. Then why is there any need to break the segments in N? Well, the problem is that, in general, it is difficult to generate the priority order on N with respect to the given viewpoint quickly. For this reason, we shall break the segments in N into a set $\Gamma(N)$ of fragments, so that the priority order over $\Gamma(N)$ can be generated quickly. Our

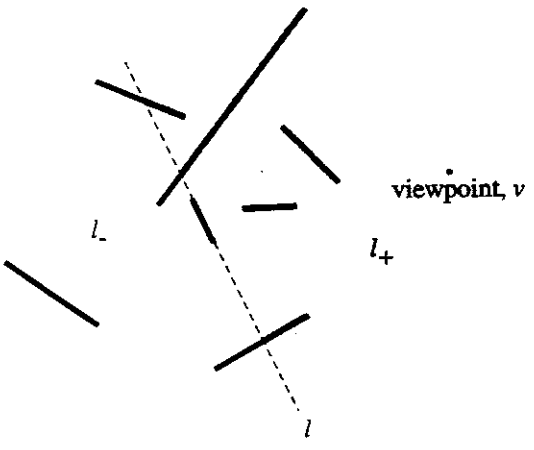

Figure 9.8: Divide and conquer.

method of decomposition will be randomized. The expected size of $\Gamma(N)$ will turn out to be $O(n \log n)$. Concurrently, we shall also build a data structure, called a *randomized BSP tree*, which will be used for generating a priority order over $\Gamma(N)$ with respect to the given viewpoint. The nodes of the BSP tree will be in one-to-one correspondence with the fragments in $\Gamma(N)$.

So let N be the given set of n segments. The basic idea underlying a BSP tree is based on the following divide-and-conquer paradigm (Figure 9.8). Fix any line l in the plane. Let l_+ and l_- denote the two half-spaces bounded by l. As a convention, we shall let l_+ denote the upper half-space bounded by l and let l_- denote the lower half-space bounded by l; if l is vertical, we label the half-spaces arbitrarily. We allow l to contain segments in N. If the segments in N are in general position—as we shall assume in what follows—then l can contain at most one segment in N. Cut the segments in N along l. Let $\bar{N}$ denote the set of resulting fragments (pieces). Partition $\bar{N}$ into three sets: $\bar{N}_0$, the set of pieces contained in l (possibly empty); $\bar{N}_+$, the set of pieces contained in l_+; and $\bar{N}_-$, the set of pieces contained in l_-. Observe that, for any viewpoint $v \in l_+$, a priority order for $\bar{N}$ can be obtained by concatenating the priority orders for $\bar{N}_+$, $\bar{N}_0$, and $\bar{N}_-$, in that order. This follows because no segment in $\bar{N}_-$ can overlap a segment in $\bar{N}_0 \cup \bar{N}_+$ as seen from v, and similarly, a segment in $\bar{N}_0$ cannot overlap a segment in $\bar{N}_+$ (Figure 9.8). If the viewpoint is contained in l_-, a priority order for $\bar{N}$ can be obtained by concatenating the priority orders for $\bar{N}_-$, $\bar{N}_0$, and $\bar{N}_+$, in that order.

This leads us to the following preprocessing algorithm. Given a set N of segments, the algorithm cuts these segments into pieces (fragments), and concurrently builds a so-called BSP (Binary Space Partition) tree. We shall denote this tree by BSP(N). The nodes of BSP(N) will be in one-to-one correspondence with the generated fragments. The reader should compare the resulting randomized BSP tree with a randomized binary tree (Section 1.3): a randomized binary tree can be thought of as a randomized BSP tree in one dimension. In the beginning, we shall sort the segments in N in a random order. Once this order on N is fixed, BSP(N) is completely determined as follows.

Algorithm 9.2.1 (Randomized BSP tree)

1. If N is empty, the BSP tree is NIL (empty).
2. Otherwise, randomly choose a segment $S \in N$. This is equivalent to choosing the first segment in N according to the initial (randomly chosen) order. Label the root of BSP(N) with S.
3. Let l denote the line through S. Cut the segments in N, other than S, along l. Let $\bar{N}_+$ and $\bar{N}_-$ be the resulting sets of fragments contained in the half-spaces l_+ and l_-, respectively. The orderings on $\bar{N}_+$ and $\bar{N}_-$ are derived from the ordering on N in a natural way.
4. Let the left (negative) and the right (positive) subtrees of BSP(N) be the recursively computed trees BSP($\bar{N}_-$) and BSP($\bar{N}_+$).

Each node σ in BSP(N) can be naturally identified with a string of $+$ and $-$ symbols that corresponds to the path from the root to σ. This string will be called the signature of σ. Figure 9.9 gives an example of a BSP tree. The nodes of the BSP tree in Figure 9.9(b) are labeled with their signatures. The fragments in Figure 9.9(a) are labeled with the signatures of the corresponding nodes in the BSP tree. We shall denote the fragment labeling a node σ in BSP(N) by $S(\sigma)$. The set of fragments labeling the nodes of the subtree rooted at σ will be denoted by $N(\sigma)$.

Observe that each node σ of BSP(N) corresponds in a natural way to a convex region $R(\sigma) \subseteq R^2$:

1. The root of BSP(N) corresponds to the whole of R^2.
2. Inductively, if $l = l(\sigma)$ is the line passing through the segment $S(\sigma)$ labeling σ, then the negative child of σ can be identified with the region $R(\sigma) \cap l_-$ and the positive child can be identified with the region $R(\sigma) \cap l_+$.

The regions in the partition shown in Figure 9.9(a) correspond to the leaves of the BSP tree shown in Figure 9.9(b). The two half-spaces in Figure 9.8 correspond to the nodes labeled $+$ and $-$.

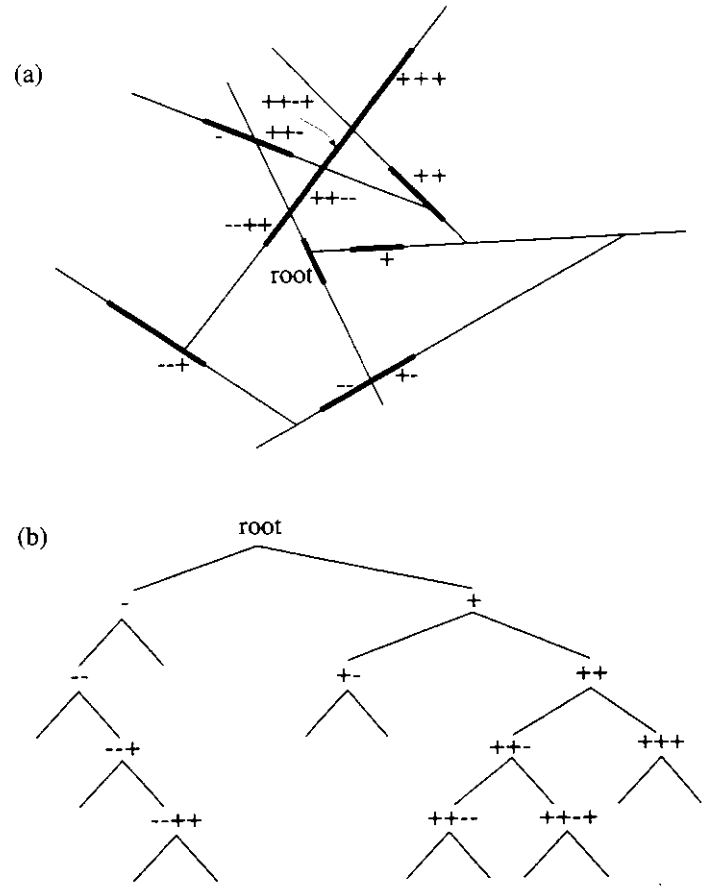

Figure 9.9: (a) A Binary Space Partition. (b) Corresponding BSP tree.

The association of the region $R(\sigma)$ with a node σ in BSP(N) is purely conceptual. We do not actually associate with σ the description of $R(\sigma)$. This would be costly as well as unnecessary. The convex regions associated with the leaves of BSP(N) constitute a partition of the plane (Figure 9.9(a)). The key property of this partition is that no segment in N intersects the interior of any of its regions.

Let $\Gamma(N)$ denote the set of fragments labeling the nodes of BSP(N). Given a view point v, a priority order on $\Gamma(N)$ can be generated by the in-order traversal of the BSP tree. The following algorithm outputs the fragments in $\Gamma(N)$ in the increasing priority order. It is initially called with σ pointing to the root of BSP(N).

Algorithm 9.2.2 (BSP tree traversal)

1. If $v \in l(\sigma)_+$:
   (a) Recur on the positive subtree of σ (if it is nonempty).
   (b) Output $S(\sigma)$.
   (c) Recur on the negative subtree of σ (if it is nonempty).
2. If $v \in l(\sigma)_-$, do the above three steps in reverse order.

The previous definition of a BSP tree was based on the divide and conquer paradigm. We can also give an equivalent definition that is based on the paradigm of randomized incrementation. We have already seen how a randomized binary tree can be thought of as the history of a randomized incremental algorithm (Section 1.3). One can do exactly the same for BSP(N) as well. For this, we imagine adding the segments in N, one at a time, in random order. Let N^k denote the set of the first k added segments. The initial BSP(N^0) consists of just one node that corresponds to the whole plane. In general, a leaf σ of BSP(N^k) corresponds to a convex region $R(\sigma)$ in the plane. Then, $N(\sigma)$ can be defined as the set of intersections of the segments in $N \setminus N^k$ with $R(\sigma)$. The convex regions that correspond to the leaves of BSP(N^k) constitute a partition of the plane. The addition of the $(k+1)$th segment S^{k+1} to BSP(N^k) (Figure 9.10) is done by splitting the regions intersecting S^{k+1}. The splitting is done along the line $l(S^{k+1})$ passing through S^{k+1}. Accordingly, the node for each region $R(\sigma)$ intersecting S^{k+1} gets two children. The fragments in $N(\sigma)$ are also cut along $l(S^{k+1})$.

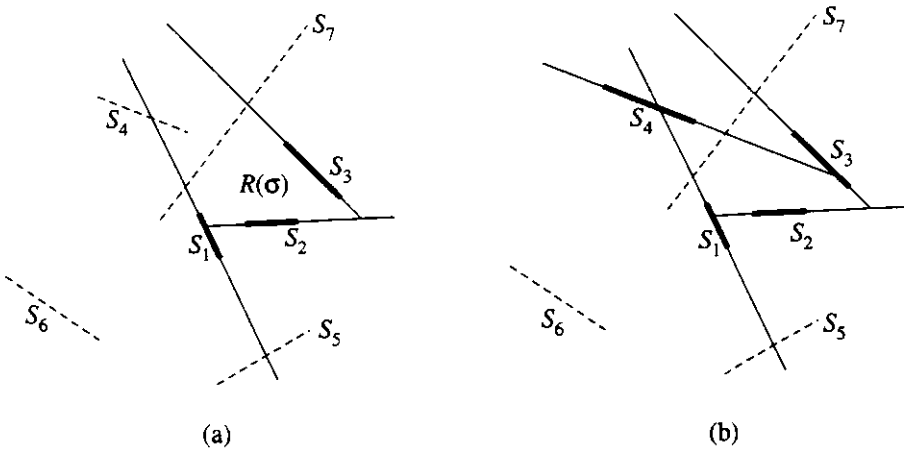

Figure 9.10: Addition of a segment to a BSP tree: (a) BSP(N^3). (b) BSP(N^4).

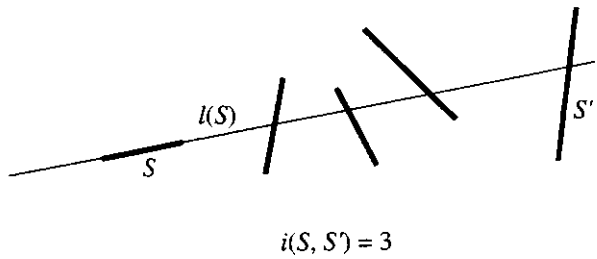

Figure 9.11: Index of a pair of segments.

Theorem 9.2.3 *Assume that the segments in N are non-intersecting. Then the expected size of* BSP(N) *generated by our algorithm is $O(n \log n)$. The expected cost of building* BSP(N) *is $O(n^2 \log n)$.*

Proof. It suffices to bound the expected size of BSP(N), because the time spent by the algorithm at each node of BSP(N) is clearly $O(n)$.

The size of BSP(N) is equal to the size of $\Gamma(N)$, the set of generated fragments labeling the nodes of $BSP(N)$. This is n plus the number of "cuts" that take place during the algorithm. Hence, it suffices to estimate the expected number of cuts.

Fix any two segments $S, S' \in N$. Let $l(S)$ denote the line passing through S. Let the index $i(S, S')$ denote the number of segments in N intersecting $l(S)$ between S and S' (Figure 9.11). By convention, $i(S, S') = \infty$, if $l(S)$ does not intersect S'.

We interpret BSP(N) as the history of a random N-sequence of additions. Let us say that S *cuts* S' during this sequence if S', or more precisely its fragment, gets cut along the line $l(S)$ during the addition of S. This can happen only if S is added before S' and the $i(S, S')$ segments between S and S'. (The converse is not true.) The probability of the latter event is $1/[2 + i(S, S')]$. Hence, the expected number of cuts is less than or equal to

$$\sum_S \sum_{S'} \frac{1}{i(S, S') + 2},$$

where S and S' range over all segments in N. The inner sum can be broken in two parts, by letting S' range over the segments intersecting $l(S)$ to the right or to the left of S. Each part is clearly bounded by $\sum_{i=1}^{n} 1/i = O(\log n)$. It follows that the expected number of cuts, and hence the expected size of BSP(N), is $O(n \log n)$. ∎

We shall end this section with one heuristic that should improve the preceding randomized construction of BSP trees. In the present scheme, the segment $S(\sigma)$ labeling a node $\sigma \in BSP(N)$ is chosen from the set $N(\sigma)$

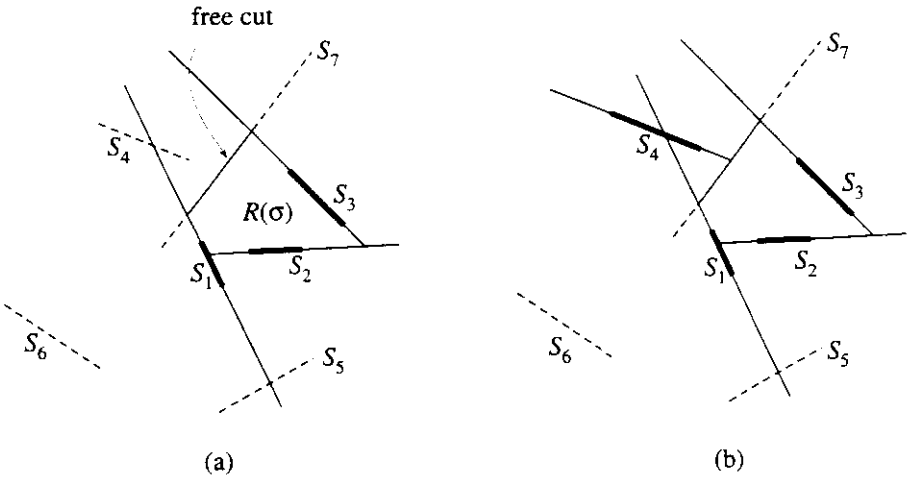

Figure 9.12: Effect of a free cut.

randomly. One can do better if $N(\sigma)$ contains a fragment $\bar{S}$ that spans $R(\sigma)$. By this, we mean that $\bar{S}$ cuts $R(\sigma)$ in two pieces; e.g., see S_7 in Figure 9.10(a). Choosing this fragment as $S(\sigma)$ provides a *free cut* in the sense that the remaining segments in $N(\sigma)$ are not cut by this choice. We only need to partition $N(\sigma) \setminus \{\bar{S}\}$ into two subsets of fragments—one for each half-space bounded by the line through $\bar{S}$. For example, in Figure 9.10(a) the region $R(\sigma)$ can be cut for free by the fragment of S_7 within it. Hence, the partition in Figure 9.10(a) can be refined further by a free cut, as shown in Figure 9.12(a). The new partition that results after the addition of four segments is shown in Figure 9.12(b). Compare it with the partition in Figure 9.10(b). Notice how the middle fragment of S_7 is not cut further in Figure 9.12(b). Note that this heuristic does not change the order of addition of segments (in the randomized incremental version of the algorithm). The only change is that we exploit free cuts, whenever possible. When a segment is added, we ignore its fragments that have been used in the earlier free cuts.

There is one subtle issue here. How do we know which fragments in $N(\sigma)$ span $R(\sigma)$? This is a valid question because we do not carry the description of $R(\sigma)$. Fortunately, the problem under construction is easy to handle. A fragment $\bar{S} \in N(\sigma)$ spans $R(\sigma)$ iff none of its endpoints coincides with an endpoint of the original segment in N containing $\bar{S}$. If several fragments in $N(\sigma)$ span $R(\sigma)$, we just choose the one with the lowest order; recall that the order on $N(\sigma)$ is inherited from the initial random order on N. If there is no spanning segment in $N(\sigma)$, we proceed as before.

Let us now summarize our refined construction of a BSP tree. We shall only give a randomized incremental version, leaving the divide-and-conquer version to the reader.

Initially, we put a random order on the set N. We shall add the segments in N in this fixed order. Let S_i denote the ith segment in this order, and let N^i denote the set of the first i segments in the order. The initial BSP(N^0) consists of one node τ that corresponds to the whole plane. We let $N(\tau) = N$. The addition of $S = S_{i+1}$ to BSP(N^i) is achieved as follows.

Algorithm 9.2.4 (Addition that exploits free cuts)

1. For each leaf σ of BSP(N^i) such that $N(\sigma)$ contains a fragment of S, cut $N(\sigma)$ along this fragment. By this, we mean: Cut the fragments in $N(\sigma) \setminus \{S\}$ so as to get two sets $N(\sigma)_+$ and $N(\sigma)_-$. Give σ two children σ_+ and σ_-. Let $N(\sigma_+) = N(\sigma)_+$ and $N(\sigma_-) = N(\sigma)_-$
2. While there is a leaf β, such that $N(\beta)$ contains a spanning fragment:
   (a) Choose the spanning fragment in $N(\beta)$ with the least order.
   (b) Cut $N(\beta)$ freely along this fragment.

We leave it to the reader to verify that free cuts can only decrease the size of the BSP tree.

Exercises

9.2.1 Rigorously verify that the free-cut heuristic does not increase the size of BSP(N).

†**9.2.2** Prove or disprove: For any set of n nonintersecting segments in the plane, there exists a BSP tree of $O(n)$ size.

9.2.3 How do the algorithms in this section perform when the segments in N are allowed to intersect?

9.2.2 Dimension three

Let us now turn to the real three-dimensional world of computer graphics. Here N is not a set of segments, but rather a set of polygons in R^3. All two-dimensional algorithms given before (Algorithms 9.2.1, 9.2.2, and 9.2.4) can be translated to the three-dimensional setting *verbatim*: Segments become polygons and dividing lines become dividing planes. We leave this straightforward translation to the reader. It only remains to see how the resulting algorithms perform in three dimensions. It is easy to see that, for any set N of n polygons, the BSP trees generated by these algorithms have $O(n^3)$ size. Can one say something better about the expected size, if the polygons in N are nonintersecting? For the algorithms given before, this seems rather difficult. But we shall see that a slight variation of the algorithm based on free

cuts (Algorithm 9.2.4) generates a BSP tree of expected $O(n^2)$ size. This bound is worst-case optimal: There exists a set N of polygons such that every BSP tree over N must have $\Omega(n^2)$ size (Exercise 9.2.6).

We shall now describe this three-dimensional variant of Algorithm 9.2.4. Let N be a set of n non-intersecting polygons in R^3. We assume that each polygon has a constant number of edges. Initially, we put a random order on the set N. We shall add the polygons in N in this order. Let S_i be the ith polygon in this order and let N^i be the set of the first i polygons in this order. We construct BSP(N^i) by induction on i. Each leaf β of BSP(N^i) will correspond in a natural way to a convex region $R(\beta) \subseteq R^3$. This correspondence is only conceptual. We do not maintain the description of $R(\beta)$ with β. The convex regions for all leaves of BSP(N^i) will constitute a convex partition of R^3. We shall denote this partition by $H(N^i)$. Each node of BSP(N^i) will be labeled with a fragment of a polygon in N^i. Initially, $H(N^0)$ consists of just one region, namely, the whole of R^3, and BSP(N^0) consists of one node τ that corresponds to this region. We let $N(\tau) = N$.

Now consider the addition of $S = S_{i+1}$ to N^i. In the three-dimensional analogue of Algorithm 9.2.4, only the regions of $H(N^i)$ that intersect $S = S_{i+1}$ would be cut. In the following modified algorithm, a region of $H(N^i)$ can be cut by the plane through S_{i+1} even if it does not intersect S_{i+1}. The modified addition of $S = S_{i+1}$ to BSP(N^i) works as follows.

Algorithm 9.2.5 (Addition)

1. For each leaf σ of BSP(N^i), do:
   If some fragment in $N(\sigma)$ intersects the plane $p(S)$ through S, then label σ with S, cut $N(\sigma)$ along the plane $p(S)$, and give two children to σ.
2. While there is a newly created leaf β such that $N(\beta)$ contains a spanning fragment, do:
   (a) Choose the spanning fragment in $N(\beta)$ with the least order. Label β with it.
   (b) Cut $N(\beta)$ along this fragment freely and give two children to β.

By a spanning fragment in $N(\beta)$, we mean a fragment that cuts the convex region $R(\beta)$ into two pieces. This happens iff the fragment is completely contained in the interior of the containing polygon in N. The cut via a spanning fragment is *free* in the sense that no polygon in $N(\beta)$ is actually cut by this choice.

Analysis

What is the expected size of the BSP tree generated by the above algorithm? First, we have to define the size of a BSP tree formally. We shall define it

as the total number of nodes in the tree. We could also have defined the size as the total face-length of the fragments labeling the nodes of BSP(N). Fortunately, the two definitions differ by at most a constant factor. To see this, fix a polygon $S \in N$. The fragments contained within S constitute a partition of S (Figure 9.13), which gives rise to a planar graph whose vertices have degree ≤ 3. It immediately follows from the Euler's relation for this planar graph (Exercise 2.3.4) that the total face-length of the fragments within S is proportional to their number. This implies that the preceding two definitions of size differ only by a constant factor. In what follows, by the size of BSP(N), we shall mean the total number of its nodes.

Let us now bound the expected size of the BSP tree produced by our algorithm, assuming that the polygons in N are non-intersecting. For this, it suffices to bound, for each i, the number of fragments that are cut by $p(S_{i+1})$ in the first step of Algorithm 9.2.5. In other words, for each polygon $Q \in N \setminus N^{i+1}$, we count the increase in the number of its fragments. We do not need to count the free cuts, because a generated fragment can be used for a free cut only once.

So fix a polygon $Q \in N \setminus N^{i+1}$. For any fixed N^{i+1}, we shall estimate the expected number of newly created fragments within Q during the $(i+1)$th addition, assuming that each polygon in N^{i+1} is equally likely to be S_{i+1}. Notice that the fragments of Q that occur in the lists $N(\beta)$'s associated with the leaves of BSP(N^i) must be *exterior*, i.e., they must be adjacent to the boundary of Q. This is because an interior fragment of Q would have been immediately used up in a free cut. Figure 9.14(a) illustrates this. It shows the arrangement formed within Q by the intersection of the planes $p(S_j)$, $1 \leq j \leq i$, with Q. The exterior fragments of Q are shown shaded. We are interested in the number of exterior fragments of Q that are cut by S_{i+1}. This number is obviously bounded by the number of the newly created exterior fragments within Q. Consider the set of exterior fragments within Q existing after $i+1$ additions. These correspond to the exterior regions in the arrangement that is formed within Q by the intersections of the planes $p(S_j)$, $1 \leq j \leq i+1$, with Q (Figure 9.14(b)). The newly created exterior regions are precisely the ones adjacent to $p(S_{i+1})$, or, more precisely,

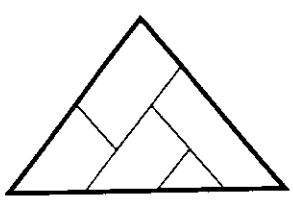

Figure 9.13: Partition of a polygon by its fragments.

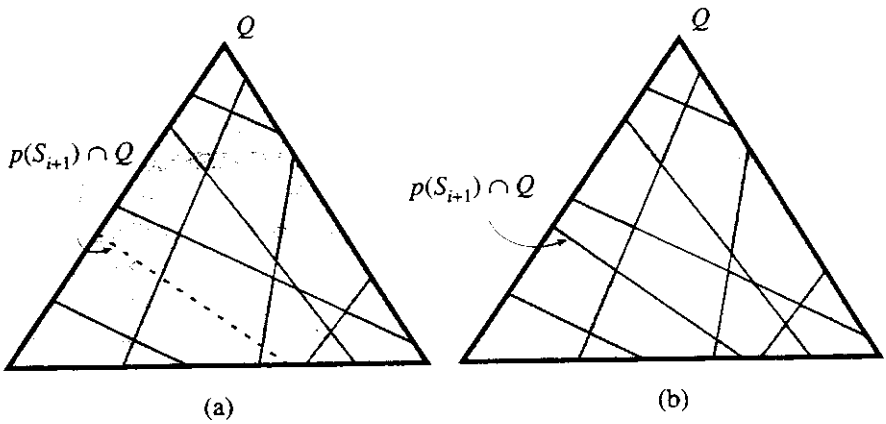

Figure 9.14: Exterior regions within Q (shown shaded): (a) After i additions. (b) After $i+1$ additions.

to $p(S_{i+1}) \cap Q$. Let us denote their number by $m(Q, S_{i+1})$. Now we shall analyze the $(i+1)$th addition backwards. For a fixed N^{i+1} (not N^i), what is the expected number of exterior fragments in Q that are newly created during the $(i+1)$th addition? Because of the random nature of additions, each polygon in N^{i+1} is equally likely to occur as S_{i+1}. Hence, the expected number of newly created exterior regions within Q is

$$\frac{1}{i+1} \sum_{S_{i+1} \in N^{i+1}} p(Q, S_{i+1}). \tag{9.6}$$

The latter sum is just the total face-length of all exterior regions in the arrangement formed within Q by $p(P) \cap Q$, $P \in N^{i+1}$. Each such exterior region belongs to the zone of some line bounding Q. Applying the Zone Theorem for line arrangements to each of the $O(1)$ lines bounding Q, it follows that this total face-length is $O(i+1)$. Thus, the expression in (9.6) is $O(1)$. In other words, for a fixed $Q \notin N^{i+1}$, the expected number of newly created fragments within Q during the $(i+1)$th addition is $O(1)$.

For the $(i+1)$th addition, it thus follows that the expected number of newly created fragments within all polygons in $N \setminus N^{i+1}$ is $O(n-i)$. Thus, the expected size of the BSP tree is $O(\sum_i n - i) = O(n^2)$.

What is the expected cost of generating the above BSP tree? First of all, what is the cost of detecting every leaf β of BSP(N^i), such that a fragment in $N(\beta)$ intersects $p(S_{i+1})$? (Refer to the first step in Algorithm 9.2.5.) We have already seen that, for a fixed $Q \in N \setminus N^i$, the fragments of Q that occur in the lists $N(\beta)$'s are the exterior fragments of Q and their number is $O(i)$ (Figure 9.14(a)). Hence, the total number of fragments that occur in the

lists $N(\beta)$'s associated with the leaves of BSP(N^i) is $O(i(n-i))$. For a fixed fragment, testing whether it intersects $p(S_{i+1})$ can take time proportional to the number of its vertices if the test is done by brute force. However, we can always maintain a balanced tree with the circular list of vertices of every fragment, so that this test takes only logarithmic time (Exercise 2.8.4). Thus, the total cost of detecting all leaves β's of BSP(N^i) such that $p(S_{i+1})$ intersects a fragment in $N(\beta)$ is $O(i(n-i)\log i)$. Over the whole algorithm, this adds up to $O(n^3 \log n)$.

As far as the cost of the rest of the algorithm is concerned, we only need to bound, for each node σ of BSP(N), the cost of cutting $N(\sigma)$. This is obviously bounded by the total number of edges of all fragments in $N(\sigma)$. But an edge generated by a fixed cut in the first step of Algorithm 9.2.5 can bound a fragment in $N(\sigma)$ for at most n choices of σ. This is because the depth of BSP(N) is at most n. We have already seen that the expected number of cuts is $O(n^2)$. Hence, the expected total cost incurred in this part of the algorithm is $O(n^3)$.

It thus follows that the expected cost of generating the BSP tree is $O(n^3 \log n)$. To summarize:

Theorem 9.2.6 *The expected size of the BSP tree generated by the algorithm is $O(n^2)$, if the polygons in N are non-intersecting. The expected cost of its generation is $O(n^3 \log n)$.*

Exercises

***9.2.4** Bring down the expected cost of BSP tree generation to $O(n^3)$.

9.2.5 Generalize the results in this section to arbitrary fixed dimension d. Let N be a set of $(d-1)$-dimensional polytopes in R^d. Assume that each polytope has a bounded size. Show that the d-dimensional variant of the algorithm in this section generates a BSP tree of $O(n^{d-1})$ expected size.

9.2.6 Construct a set N of n polygons in R^3, so that any BSP tree over N must have $\Omega(n^2)$ size.

9.3 Moving viewpoint

In this section, we consider the problem of hidden surface removal with respect to a moving viewpoint. One way of approaching this problem would be to compute the view with respect to every viewpoint from scratch. For example, one can traverse the BSP tree all over for every viewpoint. This will create a priority order among the given polygons, or, more precisely, their fragments, with respect to the given viewpoint. After this, the polygons can be painted onto the device screen in the decreasing priority order. One disadvantage of this method is that all fragments of the polygons in N, even the