

The Minimal Sum of Altitudes Decomposition for Coverage Algorithms

Wesley H. Huang
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180

December 2000

Abstract

The coverage problem in the field of robotics is the problem of moving a sensor or actuator over all points in a given region. Example applications of this problem are lawn mowing, spray painting, and aerial or underwater mapping. In this paper, I consider the single-robot offline version of this problem, i.e. given a map of the region to be covered, plan an efficient path for a single robot that sweeps the sensor or actuator over all points.

One basic approach to this problem is to decompose the region into subregions, select a sequence of those subregions, and then generate a path that covers each subregion in turn. This paper addresses the problem of creating a good decomposition. Under certain assumptions, the cost to cover a polygonal subregion is proportional to its minimum altitude. An optimal decomposition then minimizes the sum of subregion altitudes.

This paper describes an algorithm to find the minimal sum of altitudes (MSA) decomposition of a region with a polygonal boundary and polygonal holes. This algorithm creates an initial decomposition based upon multiple line sweeps and then applies dynamic programming to find the optimal decomposition. This paper describes the algorithm and reports results from an implementation. Several appendices give details and proofs regarding line sweep algorithms.

1 Introduction

Coverage algorithms have received much attention in the past several years because of demining operations in several parts of the world. Landmines must be detected and removed in order to make these areas safe for public activity, and we would like to use robots to carry out the demining. Such a robot must pass a specialized sensor over all points in an area. These robots should be guaranteed to cover the entire area and should perform this task efficiently.

Coverage algorithms also have much broader applicability. Like demining, some applications require that a sensor be passed over all points in a given area. These applications include mapping (creating image mosaics), inspection, and search and rescue operations. Other applications require some sort of actuator to be passed over a given area: spraying coatings (such as paint), vacuum cleaning, lawnmowing, and various agricultural tasks. An application such as snow removal requires not only coverage but consideration of all the snow that is collected!

There have been several coverage algorithms published in the robotics literature, including on-line and offline algorithms, and single and multiple robot algorithms; however, none address the cost of the path generated to cover the given area.

Efficiency is important in many coverage applications. Time is critical in search and rescue operations, and in industrial applications, even a small improvement in efficiency can result in large cost savings through improved cycle times or reduced material use.

A coverage algorithm must generate what we will call a *coverage path*, i.e. a detailed sequence of motion commands for a robot that sweep the given sensor or actuator over a specified region. An optimal coverage algorithm would return a coverage path that minimized, for example, the time required to execute that path.

Several existing algorithms take the following basic approach to generating a coverage path: the region to be covered is decomposed into subregions, a traveling salesman algorithm is applied to generate a sequence of subregions to visit, and a coverage path is generated from this sequence that covers each subregion in turn. These algorithms all use a single line sweep in order to decompose the coverage region into subregions, and these subregions are individually covered using a back and forth motion in rows perpendicular to the *sweep direction*. In existing algorithms, all subregions use the same sweep direction.

After finishing one row, the robot must turn around to start the next row, and we claim that minimizing the number of these turns is the most important factor in an efficient solution. The number of turns is directly related to the altitude of the subregion (measured along the sweep direction), so our optimality criterion is to minimize the sum of subregion altitudes.

In Appendix A and B, we show that the optimal uniform line sweep decomposition for convex and nonconvex polygonal worlds is generated by using a sweep direction perpendicular to one of the boundary or obstacle sides. By trying all these sweep directions and choosing the one which results in a minimum sum of subregion altitudes, this provides an optimal decomposition for existing line-sweep based algorithms.

However, by allowing different sweep directions to be assigned to each subregion of a decomposition, we can produce a lower sum of subregion altitudes and thus a cheaper coverage path. We refer to such a decomposition as the *minimum sum of altitudes* (MSA) decomposition. We propose an algorithm to generate this decomposition; it creates an initial decomposition based on multiple line sweeps and then uses dynamic programming to group subregions and assign sweep directions to each subregion.

After reviewing related work, we describe our assumptions, give a general problem statement, and then describe our algorithm. Additional appendices give details regarding the line sweep algorithm (for inputs not in general position).

1.1 Previous work

Two similar and relatively recent algorithms for coverage are by Choset and Pignon [2] and Hert *et al.* [6].

Choset and Pignon describe an offline planning algorithm for polygonal worlds which explicitly performs a line sweep decomposition (the “Boustrophedon” decomposition) and creates a sequence of subregions (cells) using an heuristic Traveling Salesman algorithm. This work includes experiments on a synchro-drive mobile robot. Hert *et al.* describe an online algorithm for nonpolygonal worlds which implicitly uses a line sweep decomposition and a heuristic Traveling Salesman algorithm. This work is described in the context of an autonomous underwater vehicle that creates an image mosaic of the ocean floor.

Schmidt and Hofner [9] describe a floor cleaning robot which has nonholonomic constraints. They use an offline planning algorithm to generate a coverage path based on a line sweep decomposition. A vocabulary of “basic motion macros” are used to maneuver the robot (i.e. for the portions of the coverage path that are not straight lines).

Kurabayashi *et al.* [8] describe an offline algorithm for planning coverage paths for multiple robots. It appears to generate a single coverage path, based on both “direction parallel” and “contour-parallel” motion. Zelinsky *et al.* [10] describes a grid-based coverage algorithm.

More recent results include: Gabriely and Rimon [5], who formulated a coverage algorithm based on traveling about the perimeter of a minimum spanning tree that fills the coverage region; Butler *et al.* [1], who created a distributed algorithm for multiple robots to cover an unknown recti-

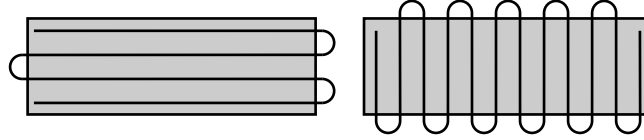


Figure 1: The number of turns is the main factor in the cost difference of covering a region along different sweep directions.

linear environment; and Choset *et al.* [3] who have extended the Boustrophedon decomposition to higher dimension Euclidean spaces.

2 The Coverage problem

A planar coverage problem is defined by the following three items:

- *coverage region* — the region to be covered is planar (or can be embedded in the plane), is connected, and is defined by an outer boundary and possibly holes in the interior. In this paper we will assume that both the region to be covered and the holes are polygonal.
- *robot* — may have nonholonomic constraints, and the shape of the robot can be unrelated (in shape and size) to the sensor/actuator pattern. The starting or ending configuration of the robot may be specified, or we may insist that the robot end in the same place it starts.
- *sensor/actuator pattern* — the sensor or actuator has a one or two dimensional “coverage pattern” which sweeps out a two dimensional area as the robot moves. Common sensor/actuator patterns include a circle (for radially limited sensors), a rectangle (e.g. a video camera), a line (snowplow). We assume the sensor/actuator pattern does not move relative to the robot.

A coverage algorithm must return:

- *coverage path* — a detailed sequence of motion commands for the robot that result in the sensor or actuator being swept over all points in the coverage region.

In the rest of this section, we discuss several issues regarding the general coverage problem and the assumptions upon which the remainder of the paper is based.

2.1 Optimal coverage

Optimal coverage is a difficult problem because it is not clear what an optimal coverage path would look like; there are many qualitatively different coverage paths for a given region. We subscribe to the approach of decomposing the coverage region, determining a sequence of subregions, and generating a coverage path that covers each region and then moves on to the next.

In order to generate an optimal coverage path, we focus on decomposing the coverage region into subregions which can be covered efficiently. The planar line sweep, upon which our decompositions are based, divides the coverage region into monotone subregions. These subregions can then be easily covered by back and forth motion along rows perpendicular to the sweep direction. Our preliminary work suggested that back and forth motion can cover a convex polygon more quickly than other methods, such as a spiral pattern.

The time to cover a subregion using back and forth motion consists of the time to travel along the rows plus the time to turn around at the end of a row. As illustrated in Figure 1, covering a subregion in different directions produces rows of approximately the same total length; however, there can be a large difference in the number of turns required. Furthermore, turns take a significant

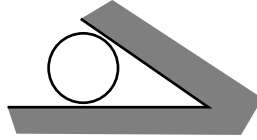


Figure 2: A circular robot with a coincident sensor/actuator pattern cannot reach into a corner where there are obstacles at the boundary.

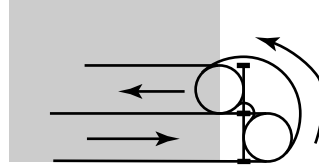


Figure 3: A lawnmower-like platform must drive outside the boundary in order to turn around efficiently.

amount of time — the robot must slow down, make the turn, and then accelerate. We therefore wish to minimize the number of turns required, and this is proportional to the altitude of the subregion measured along the sweep direction.

An additional cost we have not yet addressed is that of traveling from one subregion to another. In selecting a sequence of subregions, we can take into account the cost of traveling from one region to another, but in general, the decomposition of the coverage region cannot be independent of choosing a sequence of subregions to visit and generating a coverage path from that sequence. We shall put these concerns aside for now by assuming that the cost of traveling from one subregion to the next for is approximately constant for any reasonable decomposition.

Under these assumptions, the MSA decomposition will result in optimal coverage because the total cost of covering the subregions is the minimum for this class of solutions. We do not address the more straightforward problems of determining a good subregion sequence (a variation of the traveling salesman problem) and generating a coverage path from the subregion sequence and sweep directions.

2.2 Boundaries & obstacles

We differentiate between the boundaries of the coverage region and obstacles within or outside the region. Sometimes the two may be coincident, and sometimes they may be independent. For example, when spray painting, a “mask” may define the boundary of the coverage region, but it is permissible to spray beyond this boundary. Alternatively, when mowing a lawn, it is not acceptable to mow over any flowers surrounding the lawn.

Coverage problems may be ill-posed. As illustrated in Figure 2, a circular robot and sensor/actuator pattern cannot reach into a corner where the boundary and “boundary obstacles” are coincident.

These observations lead us to the following assumption: the coverage region given as input to our algorithm has sufficient room between the boundary and any obstacle (beyond the outer boundary or inside a hole) to turn around. This can be accomplished by assuming that the first time the robot encounters a hole or the boundary, it makes one or more complete circuits about the perimeter to create this “buffer zone.”

This assumption also helps us deal with nonholonomic constraints of a robot platform. Figure 3 illustrates a simple 180 degree turn for a lawnmower-like robot (differential drive with a circular sensor/actuator pattern in front of the wheels). Note that it must travel beyond the boundary so

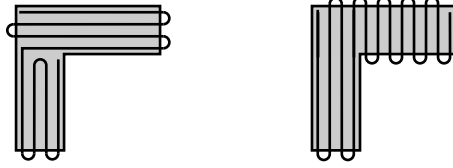


Figure 4: A simple example where a nonuniform decomposition is better than the uniform decompositions.

that two rows in the coverage region are completely covered. Smaller turns at the end of rows are possible but involve several maneuvers, so they are more costly.

3 Minimal sum of altitudes decomposition

Figure 4 shows a simple example where assigning a different sweep direction to each subregion results in a better decomposition (i.e. fewer turns). In this section, we describe an algorithm for generating the minimal sum of altitudes (MSA) decomposition.

This algorithm first decomposes the the coverage region into cells based on multiple line sweeps and then applies dynamic programming to group the cells into subregions and assign sweep directions to each subregion.

3.1 Decomposition of the coverage region

For each edge orientation (of the boundary, a hole, or their convex hulls), we perform a line sweep using a sweep direction perpendicular to such an edge. Each line sweep is done independently, but we overlay all decompositions, in effect taking the dividing lines introduced by all line sweeps. The resulting cells are monotone with respect to all sweep directions under consideration. In addition, we extend all nonconvex edges until they hit an obstacle boundary or the coverage region boundary.

That the (optimal) MSA decomposition can be formed from combinations of the cells of this initial decomposition and with the set of sweep directions perpendicular to an edge of the boundary or a hole is currently a hypothesis. Intuitively, it appears true based on the fact (shown in Appendices A and B) that the best uniform MSA decomposition results from one of these sweep directions.

3.2 Dynamic programming formulation

From the initial decomposition, we create an adjacency graph (each node represents a cell, and two nodes are connected if they share an edge). This adjacency graph may have cycles, even if there are no holes in the coverage region.

We start with the graph G which is the entire adjacency graph from the initial decomposition. The basis of the dynamic programming formulation is to either split this graph in two, thus creating two smaller subproblems, or to create one subregion from all the cells in G (and cover the entire subregion under one sweep direction). If we split the graph, we create two (individually) connected subgraphs G_1 and G_2 . These subgraphs together contain all the edges from G except those that connect a node from G_1 to a node in G_2 .

We define the minimum sum of altitudes to be:

$$S(G) = \min \left\{ C(G), \min_i S(G_1^i) + S(G_2^i) \right\} \quad (1)$$

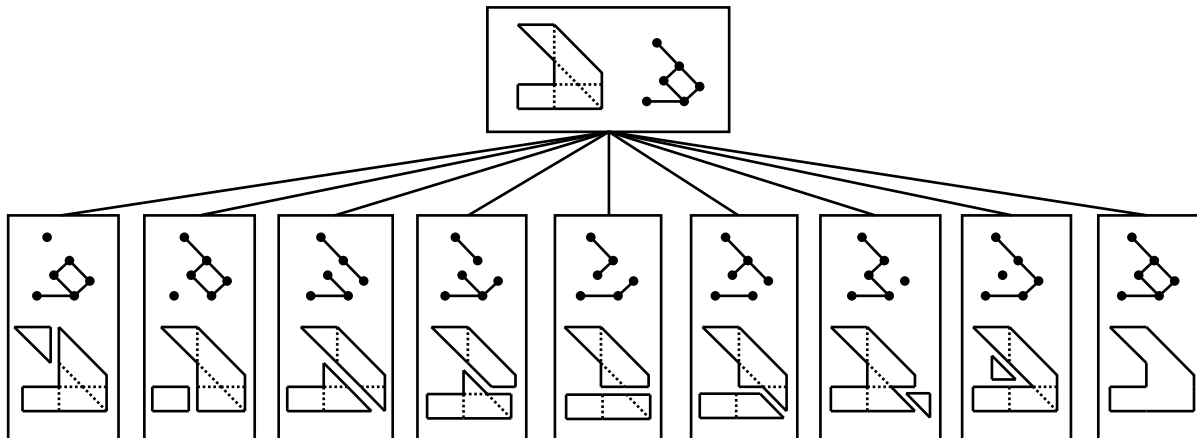


Figure 5: The top box shows a coverage region and the initial decomposition, and the corresponding adjacency graph for the proposed MSA decomposition algorithm. There are 8 ways that this graph can be decomposed into two separate connected graphs, and the corresponding division of the coverage region is shown. The rightmost choice represents covering all cells as a single region.

where i iterates over all possible ways to split the graph G into two connected subgraphs and $C(G)$ returns the cost of covering all cells corresponding to nodes in G as one subregion. When there is only one node in the graph, $S(G) = C(G)$.

The function $C(G)$ must consider all the sweep directions under consideration to determine the cost for covering the cells in G as a single region. For some (or possibly all) coverage directions, this subregion may not be monotone, in which case we assign the cost $+\infty$. $C(G)$ returns the minimum cost over all sweep directions under consideration.

Figure 5 shows an example of the first level of decomposing a problem; Figure 6 shows the three line sweep decompositions and the optimal solution produced by the MSA decomposition algorithm.

3.3 Dynamic programming subproblems

Although this algorithm produces the optimal solution, the dynamic programming component must solve an exponential number of subproblems. A value for $S(G)$ will be computed for all connected subgraphs containing 1 through n nodes. For a complete graph, there are 2^n such subgraphs, but it is not possible to have a fully connected adjacency graph (except in trivial cases). We may therefore expect many fewer than 2^n subgraphs; however, the number of subgraphs is most likely still exponential. The number depends not only on the geometry of the graph but also its maximum degree.

3.4 Implementation notes and results

We have implemented this algorithm in C++ using the CGAL library [4] for computational geometry support. Running on a Sun SPARC Ultra 10 workstation, our current implementation can compute solutions to problems where the initial decompositions contain about 16 cells in less than one hour. The actual runtime varies depending on the structure of the adjacency graph.

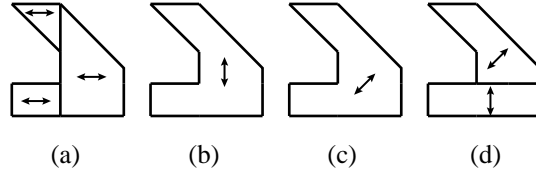


Figure 6: Figures (a) through (c) show the three line sweep decompositions of this region. Figure (d) shows an optimal MSA decomposition produced by our algorithm. The sum of subregion altitudes for Figures (a) through (d), respectively, is 10, 7, 6.4, and 5.5.

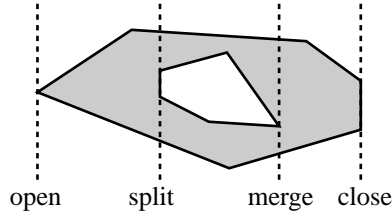


Figure 7: Illustration of the four main types of events in a line sweep from left to right.

4 Conclusions

We have given a general overview of the coverage problem and described our assumptions and formulation for optimal coverage. We follow the basic approach of decomposing the coverage region into subregions, selecting a sequence of subregions, and generating a coverage path that covers each subregion in turn. This paper introduces a measure of optimality for coverage paths which we translated into a measure of optimality for the coverage region decomposition.

The MSA decomposition divides a coverage region into subregions such that the sum of subregion altitudes is minimized. Behind this criterion is the idea that subregions are covered with back and forth motion along rows perpendicular to the sweep direction. The number of turns at the end of rows is most the important factor affected by the orientation of the sweep direction. The number of turns is directly related to the subregion altitude (with respect to the sweep direction).

We have given an algorithm for this decomposition which creates an initial decomposition by performing multiple line sweeps and extending nonconvex edges. Dynamic programming is then used to group cells in the initial decomposition into subregions and to assign each subregion a sweep direction.

A Optimal line sweep decomposition for convex worlds

Current coverage algorithms can be improved by determining the optimal sweep direction for a planar line sweep decomposition. This section address this problem for convex worlds; the follow-

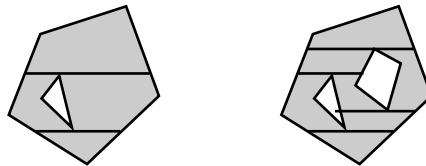


Figure 8: Introducing convex holes increases the subregion altitude sum by the altitude of the holes.

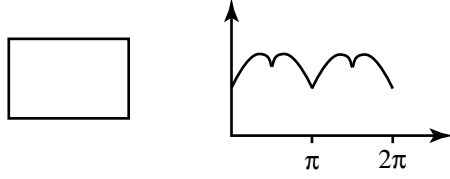


Figure 9: An example diameter function.

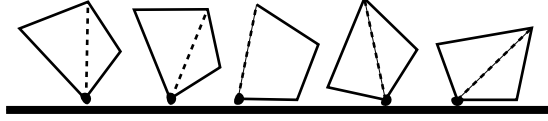


Figure 10: Creating the diameter function by rolling the polygon.

ing section addresses nonconvex worlds.

A planar line sweep decomposes a region into monotone subregions by adding diving lines at certain “events” as the sweep line moves across the region. (A subregion is monotone with respect to a sweep direction if any line perpendicular to the sweep direction intersects the region to form a connected set of points.) Figure 7 shows the four types of events in a line sweep. For convex worlds (i.e. where the boundary and the holes are convex polygons), there will be one OPEN event, one CLOSE event (both on the boundary), and for every hole, there will be one SPLIT and one MERGE event. It is easy to see, as illustrated in Figure 8, that the sum of subregion altitudes is simply the altitude of the boundary plus the altitude of all holes.

To determine the optimal sweep direction, we can express the sum of subregion altitudes as a function of sweep direction and then find the sweep direction that minimizes this sum.

A.1 Altitude sum in terms of diameter functions

For convex polygons we can use the *diameter function* $d(\theta)$ to describe the altitude of the polygon as a function of the line sweep direction. For a given angle θ , the diameter of a polygon is determined by rotating the polygon by $-\theta$ and measuring the difference between its highest and lowest point. The diameter function can be thought of as polygon “height” as measured by parallel jaw grippers rotated (from the horizontal) by θ . An example diameter function is shown in Figure 9. The altitude of a polygon for a sweep direction at an orientation of α is $d(\alpha - \frac{\pi}{2})$.

We can then express the sum of subregion altitudes as:

$$S(\theta) = d_B(\theta) + \sum_i d_{H_i}(\theta) \tag{2}$$

where $d_B(\theta)$ is the diameter function of the boundary, and $d_{H_i}(\theta)$ is the diameter function of hole i . The optimal decomposition is then determined by the the line sweep orientation $\alpha = \theta + \frac{\pi}{2}$ that minimizes S .

A.2 Form of diameter functions

The form of a diameter function can be understood by considering its height as it rolls along a flat surface. Assume we start with one edge resting on the surface. As illustrated in Figure 10, we can draw a “diameter line” from the pivot vertex to another vertex of the polygon, and the height of the polygon will be determined by this vertex. Whenever the polygon has rolled on to the next side or when an edge at the top of the polygon becomes parallel to the surface, we will change to a different diameter line (possibly from a different top vertex to a different pivot vertex).

The diameter of the polygon is determined by projecting the diameter line onto the vertical, i.e. multiplying the length of the diameter line by the sine of its angle from the horizontal. This angle can be expressed as the sum of the orientation of the polygon θ and the relative orientation ϕ_i of the diameter line with respect to the reference frame of the polygon. Therefore, a diameter function (for an n sided polygon) has the following form:

$$d(\theta) = \begin{cases} k_1 \sin(\theta + \phi_1) & \theta \in [\theta_0, \theta_1) \\ k_2 \sin(\theta + \phi_2) & \theta \in [\theta_1, \theta_2) \\ \vdots & \\ k_{2n} \sin(\theta + \phi_{2n}) & \theta \in [\theta_{(2n-1)}, \theta_{2n}) \end{cases} \quad (3)$$

where $\theta_0 = 0$ and $\theta_{2n} = 2\pi$. The diameter function is piecewise sinusoidal, has continuous derivatives everywhere except at the “breakpoints” θ_i , and its minima correspond to angles where the sweep line is parallel to a side of the polygon. More importantly, notice that the portion of the sine curve used is in the range $\theta \in (0, \pi)$. Therefore, the second derivative of the radius function is negative (except at the breakpoints at which it is not defined).

A.3 Minimizing the altitude sum

The minimum of the function $S(\theta)$ must lie either at a critical point ($S'(\theta) = 0$) or at a breakpoint of one of its constituent diameter functions.

However, for any critical point in between breakpoints:

$$S''(\theta) = d_B''(\theta) + \sum_i d_{H_i}''(\theta) < 0 \quad (4)$$

which means that it corresponds to a maximum! Therefore, the minimum must lie at a breakpoint of one of the diameter functions which corresponds to the sweep direction being perpendicular to an edge of the boundary or a hole (i.e. the sweep line is parallel to the edge).

The minima can be determined by testing each of these line sweep directions.

B Optimal line sweep decomposition for nonconvex worlds

With a nonconvex boundary or nonconvex holes, a planar line sweep still places dividing lines at SPLIT and MERGE events, but now the boundary and any hole can both produce any of the four types of events, and the sum of subregion altitudes is greater than the sum of diameters of the holes and boundary.

In this section, we will reexamine the line sweep algorithm and then proceed to show that the optimal line sweep decomposition for a nonconvex world corresponds to a sweep direction that is perpendicular to an edge of the boundary or of an hole, just as for convex worlds. Like for convex worlds, we consider the optimal decomposition to be that which minimizes the sum of subregion altitudes as measured along the sweep direction.

B.1 Line sweeps for nonconvex worlds

The line sweep will essentially be the same as for convex worlds, but the boundary and the holes can all produce OPEN, CLOSED, SPLIT, and MERGE events. For purposes of demonstrating the optimal decomposition, we will introduce another event, the INFLECTION event. This event occurs when one or more consecutive vertices of the boundary or of a hole lie on the sweep line and when it is not classified as one of the four main types of events.

In performing an actual decomposition, it is not necessary to divide the region at such an event because the entire region will still be monotone. However, we will make use of this event in our following proof.

B.2 Proof outline

To show that the optimal line-sweep decomposition corresponds to a sweep direction perpendicular to an edge of the boundary or an hole, we first show that we can match every OPEN and SPLIT event with a CLOSE or MERGE event. For each pair, we will associate a stack of polygonal cells and consider a line connecting the OPEN/SPLIT event to a CLOSE/MERGE event. (Assume for the moment, that each event consists of only a single vertex.) The altitude of a stack of subregions is the projection of this line onto the sweep direction. This line is essentially the same as the “diameter line” from the previous section. The sum of subregion altitudes is the sum of the projections of the lines for all OPEN/SPLIT and CLOSE/MERGE pairs.

As the sweep direction changes, these lines rotate, so the altitude sum is a sum of sinusoids, as in the case of convex worlds. Also like the convex world case, we only take portions of the sine curve from 0 to π , so any critical points of the altitude sum are maxima. The minima can only occur at the breakpoints of the functions, and these occur only when we need to change the mapping from OPEN/SPLIT events to CLOSE/MERGE events. Changing the mapping corresponds to selecting different lines which will rotate with the sweep direction. We only need to revise this mapping when an edge of the boundary or an hole is perpendicular to the sweep direction.

B.3 Creating stacks of polygons

Unlike the convex world case, the sum of subregion altitudes is not just the sum of the boundary and hole altitudes. We still draw “diameter lines” to measure the altitude of a group of polygonal cells. For the nonconvex case, we must be more explicit about how all the polygonal cells are grouped to account for their altitude. We associate a “stack” of cells with a OPEN/SPLIT–CLOSE/MERGE event pair, and there are certain times when we must revise this grouping.

In a planar line sweep, there are the same number of OPEN/SPLIT events as there are CLOSE/MERGE. In the basic line sweep algorithm, a new cell is started by an OPEN or a SPLIT event and is finished by a CLOSE or MERGE event. We create stacks of polygonal cells in the following manner:

- An OPEN event starts one cell, which we add to the stack corresponding to that event.
- A SPLIT event finishes one cell and starts two new cells. We place the cell on the left side of the SPLIT event on the same stack as the cell that is closed by this event. We start a new stack of polygons associated with the SPLIT event and place the polygon on the right side of the event on this stack.
- A MERGE event finishes two polygons and starts one new polygon. We place the new polygon on the stack of the polygon to the left of the merge event, and we complete a stack of polygons with the polygon on the right of the event. This merge event is associated with the OPEN or SPLIT event that started this stack.
- A CLOSE event finishes one polygon; we associate this event with the OPEN or SPLIT event that started this stack.

B.4 Drawing lines between event pairs

We have included every polygon in the decomposition into one of the stacks, and these stacks are continuous — connecting the each OPEN/SPLIT event to a CLOSE/MERGE event. In the general case (when all events consist of a single vertex), we can draw a line from the OPEN/SPLIT event vertex to the CLOSE/MERGE event vertex. The altitude of this stack of polygons is therefore determined by projecting the line connecting these vertices onto the sweep direction.

As the sweep direction rotates, these lines still measure the altitude of the stack of polygons, even though the polygons themselves are changing. One can imagine the dividing line placed at each event as rotating around the event vertex as the sweep direction changes.

As the sweep direction rotates, the projection of all these lines change sinusoidally. Like the convex world case, the sum of the projections of these lines is a sum of sinusoids which draw only from the portion of the sine curve from 0 to π , so any critical point must be a maximum. The minima of this function must then lie at the breakpoints where the diameter lines are changes. The remainder of this section shows that this occurs when an edge of the boundary or a hole is perpendicular to the sweep direction.

B.5 Remapping event pairs

As the sweep direction continues to rotate, there are certain situations where we must adjust the mapping of event pairs. Adjusting this mapping means that we need to select different vertices to connect with the lines that we're projecting onto the sweep direction. We assume for the following discussion that the sweep direction is rotating counter-clockwise.

When an edge of the boundary or hole lies on the sweep line, we need to switch the vertex to/from which we are drawing a line. One of the endpoints of this edge will (just before the sweep direction rotated to this point) will have been an event of some sort. We need simply to switch to the other vertex. For OPEN and SPLIT events, this new vertex will lie to the right of the original vertex on the sweep line. For CLOSE and MERGE events, the new vertex will lie to the left of the old.

When an edge of the boundary or hole which is not on the convex hull lies on the sweep line, this creates a INFLECTION event. If the two vertices of this event were separate events before they became a single inflection event, they will have a pair: either an OPEN-MERGE pair or a SPLIT-CLOSE pair. We simply remove this pair from the event mapping — as the sweep line continues to rotate, neither will be events in the line sweep. If the vertices were not events in the line sweep, then we assign the rightmost to be an OPEN event and the left to be a MERGE event. (They will become these events as the sweep line continues to rotate.)

Another possibility (which does not require remapping the events, is when two separate events become simultaneous. This may result in one or more polygons disappearing from the stacks, but it does not change the mapping (which can be done as before), and it does not change the line which connects the bottom of a stack to its top.

Therefore, the only occasions that the mapping of events needs to be changes is when an edge of the boundary or a hole becomes perpendicular to the sweep direction.

C Line sweep algorithm

This section presents the line sweep algorithm to decompose a polygonal nonconvex region (possibly with polygonal nonconvex holes) into monotone regions with respect to a given sweep line.

The basic algorithm for performing a line sweep is well known. Here, however, we do not assume that the input is in general position: any number of points may be collinear.

The basic algorithm is as follows. As the sweep line sweeps across the region, it encounters various types of "events" — instants where vertices of the boundary or obstacles lie on the sweep line. Depending on the type of event, the region is subdivided by adding a line segment along the sweep line.

Normally, an event consists of a single vertex and no two events may occur simultaneously. Without the general position assumption, however, an event may also consist of a sequence of consecutive vertices and multiple events may occur simultaneously. Our algorithm addresses extends the basic line sweep algorithm to deal with these situations.

C.1 Definitions and terminology

- sweep line — the line which is swept across the coverage region

- sweep direction — the direction that the sweep line is swept over the coverage region. The sweep direction is perpendicular to the sweep line.
- boundary — a (possibly nonconvex) polygon which we assumed is stored as a list of points in CCW order so that the interior of the coverage region lies to the left as one walks around the boundary in this order.
- holes — (possibly nonconvex) polygons which we assume are stored as a list of points in CW order so that the interior of the coverage region is to left as one walks around the hole in this order.
- cell — a monotone (with respect to the sweep direction) polygon created by the line sweep algorithm
- open cell — a cell which is still being created. As the sweep line is swept across the coverage region, open cells are created, points are added to them, and ultimately, they are “closed” (making them regular cells)
- open segments — while in the process of being created, an open cell contains vertices that lie on one side of the sweep line. The open segments are the line segments connecting a vertex belonging to an open cell to a vertex on the other side of the sweep line.

C.2 Events

For the following definitions, assume that the sweep direction is horizontal and points to the right and that the sweep line is vertical. When one or more consecutive vertices lies on the sweep line, they become an event. The type of event is determined by the:

- *entering segment* — the line segment from the previous vertex of the polygon to the first vertex in the event
- *leaving segment*: — the line segment from the last vertex in the event to the next vertex in the polygon.

The event may be classified as one of the following events:

- OPEN — the entering and leaving segments lie on the right side of the sweep line, and the entering segment is above the leaving segment.
- CLOSE — the entering and leaving segments lie on the left side of the sweep line, and the entering segment is below the leaving segment.
- SPLIT — the entering and leaving segments lie on the right side of the sweep line, and the entering segment is below the leaving segment.
- MERGE — the entering and leaving segments lie on the left side of the sweep line, and the entering segment is above the leaving segment.
- INFLECTION — the entering and leaving segments lie on opposite sides of the sweep line.

In certain circumstances, we may create one additional type of event:

- INTERSECTION — corresponds to a new vertex which is the intersection of the sweep line and an open segment.

C.3 The algorithm

C.3.1 LINE-SWEEP

LINE-SWEEP(world W , sweep-direction \vec{s})

1. Let L be a list of events from the world W with respect to the sweep direction \vec{s} .
2. Sort L along the sweep direction (in increasing order of their sweep direction coordinate)
3. Let:
 - O be a list of open cells, initially empty
 - C list of closed cells, initially empty
4. Repeat:
 - let E be a list containing the next event (or events if simultaneous) from L
 - PROCESS-EVENTS(E, O, C)Until all events from L have been processed
5. Return C

C.3.2 PROCESS-EVENTS

PROCESS-EVENTS(event-list E , open-list O , closed-list C)

1. Set `contains-split/merge` to false
2. If E contains a SPLIT or MERGE event,
 - Create all INTERSECTION events for this position of the sweep line and add them to E .
 - Sort E according to the sweep line coordinate (assume increasing order)
 - Set `contains-split/merge` to true
3. Let e be the first event on E .
4. If e is of type:
 - OPEN, then start a new open cell and add it to O . Let e be the next event on E .
 - CLOSE, then find the cell on O which this event closes. Add the closed cell to C . Let e be the next event on E .
5. Otherwise, if `contains-split/merge` is false, then e must be an INFLECTION event, so:
 - Find the open cell on O which this point extends and update the open cell. Let e be the next event on E .
6. Otherwise, e must be either an INFLECTION or INTERSECTION event.
 - If the next event on E is an INFLECTION or INTERSECTION event, then process both events as follows:
 - INFLECTION: Find the open cell on O which this point extends and update the open cell. Let e be the next event on E .
 - INTERSECTION: do nothing

- Otherwise, the next event on E is a SPLIT or MERGE event, so
 - call PROCESS-SPLIT/MERGE-SEQUENCE(e, E, O, C)
 - Let e be the event returned by this procedure
7. Repeat steps 4–6 until there are no more events on E to process

C.3.3 PROCESS-SPLIT/MERGE-SEQUENCE

When this procedure is called, e will be an INTERSECTION or INFLECTION event, and the next event in E will be a SPLIT or MERGE event. This procedure takes care of a sequence of possibly simultaneous split and merge events, beginning and ending with an INTERSECTION or INFLECTION event. Note that the “first” and “last” points of an event refer to the order in which those consecutive vertices are stored in the boundary or hole polygons. “Above” and “below” refer to the height of the points in an event on the sweep line. (Recall that we assume the sweep direction is to the right, and we are processing simultaneous events from the bottom to the top of the sweep line.)

PROCESS-SPLIT/MERGE-SEQUENCE(event e , event-list E, open-list O, closed-list C)

1. let `open-left` be the open cell to which e belongs. Add the first point from e . If the remaining points are above the first point, add them to `open-left` too.
2. let `open-right` be a new open cell. Add the last point from e . If the other points of e are above the last point, add them to `open-right` too.
3. Let e be the next event on E
4. If e is of type SPLIT, then:
 - add all points from e to `open-left`
 - add the first point from e to `open-right`
 - put `open-right` on O
 - let `open-right` be a new open cell, adding the last point from e
5. Otherwise, if e is of type MERGE, then:
 - add all points from e to `open-right`
 - add the last point from the event to `open-left`
 - put `open-left` on C
 - let `open-left` be the cell from O which has an open segment that connects to the first point of e . Add the first point of e to `open-left`
6. If e is of type INTERSECTION or INFLECTION, then:
 - add the first point of e and all points in e below the last point on the sweep line to `open-right`
 - add `open-right` to O
 - add the last point from e and all points in e below the last point on the sweep line to `open-left`
 - add `open-left` to C
 - return the next event from E
7. go to step 3

References

- [1] Z. Butler, A. Rizzi, and R. Hollis. Complete distributed coverage of rectilinear environments. In *Fourth international workshop on the algorithmic foundations of robotics*, 2000.
- [2] H. Choset and P. Pignon. Coverage path planning: the boustrophedon cellular decomposition. In *Proceedings of the International Conference on Field and Service Robotics*, December 1997.
- [3] H. Choset, E. Acar, A. Rizzi, and J. Luntz. Exact cellular decompositions in terms of critical points of Morse functions. In *IEEE International Conference on Robotics and Automation*, 2000.
- [4] Computational Geometry Algorithms Library (CGAL). <http://www.cgal.org/>
- [5] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. Submitted to *Annals of Mathematics and Artificial Intelligence*.
- [6] S. Hert, S. Tiwari, and V. Lumelsky. A terrain-covering algorithm for an auv. *Autonomous Robots*, 3(2-3):91-119, June-July 1996.
- [7] W. Huang. Optimal line-sweep-based decompositions for coverage algorithms. Rensselaer Polytechnic Institute Computer Science technical report 00-3, September 2000.
- [8] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 1744-1749, 1996.
- [9] G. Schmidt and C. Hofner. An advanced planning and navigation approach for autonomous cleaning robot operations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1230-1235, 1998.
- [10] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. *International Journal of Robotics Research*, 13(4):315-329, 1994.