The Visibility–Voronoi Complex and Its Applications[*]

Ron Wein
School of Computer Science
Tel-Aviv University, Israel
wein@tau.ac.il

Jur P. van den Berg
Institute of Information and Computing Sciences
Utrecht University, The Netherlands
berg@cs.uu.nl

Dan Halperin
School of Computer Science
Tel-Aviv University, Israel
danha@tau.ac.il

ABSTRACT

We introduce a new type of diagram called the $\mathrm{VV}^{(c)}$-diagram (the Visibility–Voronoi diagram for clearance c), which is a hybrid between the visibility graph and the Voronoi diagram of polygons in the plane. It evolves from the visibility graph to the Voronoi diagram as the parameter c grows from 0 to ∞. This diagram can be used for planning natural-looking paths for a robot translating amidst polygonal obstacles in the plane. A natural-looking path is short, smooth, and keeps — where possible — an amount of clearance c from the obstacles. The $\mathrm{VV}^{(c)}$-diagram contains such paths. We also propose an algorithm that is capable of preprocessing a scene of configuration-space polygonal obstacles and constructs a data structure called the VV-complex. The VV-complex can be used to efficiently plan motion paths for any start and goal configuration and *any clearance value* c, without having to explicitly construct the $\mathrm{VV}^{(c)}$-diagram for that c-value. The preprocessing time is $O(n^2 \log n)$, where n is the total number of obstacle vertices, and the data structure can be queried directly for any c-value by merely performing a Dijkstra search. We have implemented a CGAL-based software package for computing the $\mathrm{VV}^{(c)}$-diagram in an *exact* manner for a given clearance value, and used it to plan natural-looking paths in various applications.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*

[*]This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE — Motion Planning in Virtual Environments), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

General Terms

Algorithms

Keywords

Visibility graphs, Voronoi diagrams, motion planning, path optimization

1. INTRODUCTION

We study the problem of planning a natural-looking collision-free path for a robot with two degrees of motion freedom moving in the plane among polygonal obstacles. By "natural-looking" we mean that the robot should select a path that will be as close as possible to the path a human would take in the same scene to reach the goal configuration from the start configuration. This essentially means the following: (a) the path should be *short* — that is, it should not contain long detours when significantly shorter routes are possible; (b) it should have a guaranteed amount of *clearance* — that is, the distance of any point on the path to the closest obstacle should not be lower than some prescribed value; and (c) it should be *smooth*, not containing any sharp turns. Requirements (b) and (c) may conflict with requirement (a) in case it is possible to considerably shorten the path by taking a shortcut through a narrow passage. In such cases we may prefer a path with less clearance (and perhaps containing sharp turns).

The motion-planning problem for a robot with two degrees of freedom (a disc robot, or a polygonal robot that can only translate — and not rotate — in the plane) moving amidst polygonal obstacles can be efficiently solved by computing a complete representation of the free configuration space [15], for instance using a trapezoidal map [4]. However, this approach yields paths that are piecewise linear (hence not smooth) and do not compute shortest paths. It is also possible to solve this variant of the motion-planning problem using Probabilistic Roadmaps (PRMs — see, e.g., [14]) — but the output paths in this case are also piecewise linear and may be far from the shortest possible paths. Indeed, it is possible to perform path smoothing as a post-processing stage and produce a more natural-looking path (see [9] for a summary of applicable smoothing techniques), but as there is no guarantee that the initial path is in the same homotopy class as the best path possible, the smoothed path may be different from the most natural-looking path.

The *visibility graph* is a well-known data structure for computing the shortest collision-free path between a start and a goal configuration (see, e.g., [6, Chapter 15]). However, shortest paths are in general tangent to obstacles, so a path computed from a visibility graph usually contains semi-free configurations (the robot is in contact with an obstacle, but their interiors do not intersect) and therefore does not have any clearance. This not only looks unnatural, it is also unacceptable for many motion-planning applications. On the other hand, planning motion paths using the *Voronoi diagram* of the obstacles [20] yields a path with maximal clearance, but this path may be significantly longer than the shortest path possible, and may also contain sharp turns.

We suggest a hybrid of these two latter approaches, called the *$VV^{(c)}$-diagram* (the Visibility–Voronoi diagram for clearance c), yielding natural-looking motion paths, meeting all three criteria mentioned above (with the reservation mentioned above regarding narrow passages). It evolves from the visibility graph to the Voronoi diagram as c grows from 0 to ∞, where c is the preferred amount of clearance. The $\mathrm{VV}^{(c)}$-diagram contains the visibility graph of the obstacles dilated with a disc of radius c. The dilated obstacle vertices become circular arcs in this case, and the visibility edges are bitangent to these arcs. This guarantees that the paths in the diagram are not only *short* but also *smooth*. However, due to this obstacle inflation, narrow passages in the scene may disappear, which implies that it is not possible to pass through these narrow passages with a clearance of at least c. As we still want to keep the option of traversing these narrow passages, we integrate into the diagram paths with the maximal possible clearance in regions where the preferred clearance c can not be obtained. It is easy to see that these paths are portions of the Voronoi diagram of the original obstacles.

Beside the straightforward algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram for a given clearance value c, we also propose an algorithm for preprocessing a scene of configuration-space polygonal obstacles and constructing a data structure called the *VV-complex*.[1] The VV-complex can be used to efficiently plan motion paths for any start and goal configuration and any given clearance value c, without having to explicitly construct the $\mathrm{VV}^{(c)}$-diagram for that c-value. The preprocessing time is $O(n^2 \log n)$, where n is the total number of obstacle vertices, and the query takes $O(n \log n + k)$ time, where k is the number of edges encountered during the search. Furthermore, we reduce the number of costly geometric operations in the query stage and perform the most time-consuming computations in the preprocessing stage.

A direct application of the VV-complex is planning natural motion paths for a polygonal robot among polygonal obstacles. We can compute the Minkowski sum of each obstacle with the robot rotated by 180° to obtain a set of configuration-space obstacles, which are also polygonal. Constructing the $\mathrm{VV}^{(c)}$-diagram of these configuration-space obstacles and giving adequate weights to the diagram edges yield more natural motion paths, compared, for example, to the implementation of [4].

Another interesting application is motion planning for a group of entities in a two-dimensional environment cluttered with polygonal obstacles. Kamphuis and Overmars [12] solve this problem by planning a collision-free path for a single entity, then "inflating" this backbone path up to a diameter of a preferred group width w, wherever possible, and governing the motions of the individual entities inside this inflated path using a potential field. They use a PRM with cycles [19] to compute the initial path, then apply smoothing techniques on it to achieve natural-looking motions. However, this method is not complete and does not guarantee that the best available path can be computed. The $\mathrm{VV}^{(c)}$-diagram of the environment for $c = \frac{w}{2}$ contains all natural-looking paths between a start and a goal configuration and is ideal for computing the initial path, especially if the weight given to the diagram edges is proportional to the *time* it takes the group to traverse each edge. Furthermore, the time-consuming smoothing stage is not needed. We have implemented our algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram and used it in the two applications described above.

The principles of our construction may also be applied for sensor-based coverage using a robot with a limited sensor radius r — see for example two work of Acar et al. [3].

The rest of this paper is organized as follows: In Section 2 we give a short review of the geometric data structures we use for constructing the $\mathrm{VV}^{(c)}$-diagram. In Section 3 we present the $\mathrm{VV}^{(c)}$-diagram in more detail and explain how to construct it, given a scene with obstacles and a preferred clearance value c. In Section 4 we introduce the VV-complex, show how to efficiently construct it and explain how to query it. In Section 5 we review the software we have developed to robustly compute the $\mathrm{VV}^{(c)}$-diagram of a set of obstacles and a given c and show some experimental results.

2. PRELIMINARIES

Visibility Graphs. The *visibility graph* of a set of pairwise interior-disjoint polygons is an undirected graph whose set of vertices is the set of polygon vertices, and whose set of edges consists of those pairs of vertices that are mutually visible. Two vertices are *mutually visible* if the straight line segment connecting them does not intersect the *interior* of any of the polygons — in this case, we call this segment a *visibility edge*. In fact, it is sufficient to consider only the edges that are bitangent to the polygons they connect. These are called *valid* visibility edges.

The visibility graph can be used to compute shortest paths amidst polygonal obstacles, where the polygons are considered as open sets. It can straightforwardly be computed in $O(n^2 \log n)$ time (see, e.g., [6, Chapter 15]), or using an output-sensitive algorithm in $O(n \log n + k)$ time [10], where n is the total number of polygon vertices and k is the number of visibility edges. To compute the shortest path between a start and a goal configuration, one simply needs to connect them to the visibility graph and execute Dijkstra's algorithm, starting from the vertex representing the start configuration, where each edge is given a weight equal to the Euclidean distance between its end-vertices (see more on shortest paths in [18]).

Voronoi Diagrams of Polygons. Given a set of pairwise interior-disjoint polygons in the plane, the *Voronoi diagram* of this set is the planar subdivision into maximal connected cells such that the points in each *Voronoi cell* are closer (under the Euclidean distance metric) to a specific polygon than to any other polygon in the set. The *Voronoi vertices* are points equidistant to features of three (or more) different polygons (a polygon *feature* is either a vertex or an edge). The vertices are connected by continuous *chains* of *Voronoi*

[1]Despite the similarity in names, our structure is different from the *visibility complex* introduced in [21].

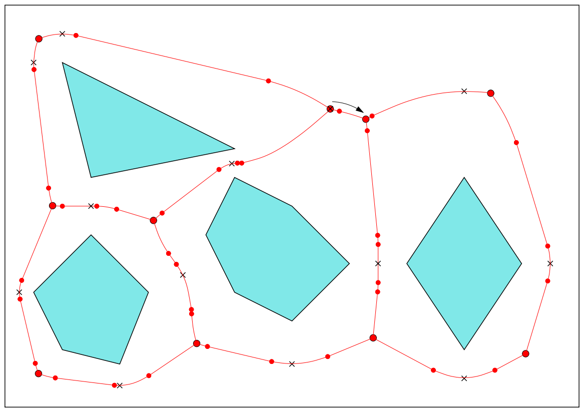


Figure 1: The Voronoi diagram of four convex polygons contained inside a rectangle. Small dots mark the endpoints of each Voronoi arc, while the larger dots mark the Voronoi vertices. The point of minimum clearance along each chain is marked by $\times$. Notice that the chain marked by an arrow is a monotone chain and obtains its minimal clearance on its left Voronoi vertices — so when we traverse it in the arrow's direction, the clearance only increases.

arcs. An arc may be equidistant to two vertices or to two polygon edges — in which case it is a straight line segment, or to a polygon vertex and a (non-incident) polygon edge — in which case it is a segment of a parabola (parabolic arc). Each arc has two *endpoints*, which either connect it to the next arc in the chain or to a Voronoi vertex.

If we examine the clearance value along a Voronoi chain, we notice that in most cases the minimum clearance value is obtained in the interior of a vertex–vertex or a vertex–edge arc inside the chain (note that the interior of an edge–edge arc will never contain a clearance minimum). In such cases, the clearance value increases as we move from this minimum point toward either of the chain's end-vertices. However, for some chains the minimum clearance value is obtained at one of their end-vertices, and grows as we move along the chain toward its other end. We call such a chain a *monotone Voronoi chain* (see Figure 1 for an illustration).

The Voronoi diagram can be used to compute paths with maximal amount of clearance from the obstacles. It can be shown that the total complexity of the Voronoi diagram is $O(n)$, where n is the total number of polygon vertices, and that it can be constructed in $O(n \log n)$ time (see, e.g, [5, 16]).

Minkowski Sums. The *Minkowski sum* of two given sets $A, B \in \mathbb{R}^d$, denoted $A \oplus B$, is defined as:

$$A \oplus B = \{a + b \mid a \in A,\ b \in B\} \ .$$

In particular, if we are given a polygon P, the set of points whose distance from P is less than ρ is the Minkowski sum $P \oplus B_\rho$, where B_ρ is a disc of radius ρ. The union of the Minkowski sums of a set of polygons having n vertices with a disc has $O(n)$ complexity and can be computed in $O(n \log n)$ time using a randomized algorithm (see [6, Chapter 13] and [11] for further discussions and more references).

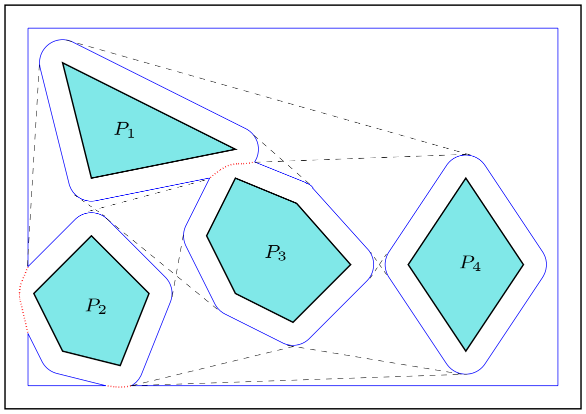


Figure 2: The VV$^{(c)}$-diagram for four convex obstacles located in a rectangular room. The boundary of the union of the dilated obstacles is drawn in a solid line, the relevant portion of the Voronoi diagram is dotted. The visibility edges are drawn using a dashed line. Notice that an endpoint of a visibility edge may either lie on a circular arc or on the intersection of two dilated obstacle boundaries (a chain point).

3. THE VV$^{(c)}$-DIAGRAM

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of pairwise interior-disjoint polygons in the plane, having n vertices in total, representing two-dimensional configuration-space obstacles. Given a start configuration, a goal configuration and a preferred clearance value $c > 0$, we wish to find a shortest path between the query configurations, keeping a clearance of at least c from the obstacles where possible, but allowing to get closer to the obstacles in narrow passages when it is possible to make considerable shortcuts.

We begin by dilating each obstacle by c, by computing the Minkowski sum of each polygon with a disc of radius c. The visibility graph of the dilated obstacles contains all shortest paths with a clearance of at least c from the obstacles. Moreover, as each convex polygon vertex becomes a circular arc of radius c, the valid visibility edges are bitangents to two circular arcs (note that the dilated polygon edges are also valid visibility edges). This guarantees that a shortest path extracted from such a visibility graph is $\mathcal{C}_1$-smooth, and contains no sharp turns. The only disadvantage in this approach is that narrow, yet collision-free, passages can be blocked when we dilate the obstacles (for example, in Figure 2 there exists such a narrow passage between P_1 and P_3). It is clearly not possible to pass in such passages with a clearance of at least c, but we still wish to allow a path with the maximal clearance possible in this region. To do this, we compute the portions of the free configuration space that are contained in at least two dilated obstacles, and add their intersection with the Voronoi diagram of the original polygons to our diagram. The resulting structure is called the VV$^{(c)}$-diagram.

Formally, given a collection of disjoint convex obstacles $P_1, \ldots, P_m$ (we will later discuss non-convex obstacles as well) and a preferred clearance c, we dilate every obstacle P_i with a disc of radius c, that is, we construct the Minkowski sum $M_i^{(c)} = P_i \oplus B_c$ for every obstacle P_i, where B_c is a disc with radius c. Note that the inflated obstacles $M_i^{(c)}$ may no longer be disjoint. We now compute the union $\mathcal{M}^{(c)}$ of all $M_i^{(c)}$. The boundary of $\mathcal{M}^{(c)}$ consists of circular arcs, straight line segments and reflex vertices. The reflex vertices are the intersection of the boundary arcs of two dilated obstacles, and we refer to them as *chain points*, as they lie on Voronoi chains, since their distance from both relevant polygons is exactly c.

We now compute the modified visibility graph $\mathcal{G}^{(c)}$ of $\mathcal{M}^{(c)}$. This graph contains every free bitangent of two circular arcs of the boundary of $\mathcal{M}^{(c)}$ (the edges that form the boundary of $\mathcal{M}^{(c)}$ are also regarded as bitangents to two neighboring dilated vertices), every free line segment between two chain points, and every free line segment from a chain point tangent to a circular arc. We finally construct $\mathcal{V}$, the Voronoi diagram of the original set of polygons and compute the intersection $\mathcal{V} \cap \mathcal{M}^{(c)}$, namely the portion of the Voronoi diagram that is contained within the union of the dilated obstacles. We combine the corresponding Voronoi arcs (and sub-arcs) with $\mathcal{G}^{(c)}$ to connect the chain points via narrow passages and form the final VV$^{(c)}$-diagram.

In case our polygons are not convex, we decompose them to obtain a set of convex polygons and compute $\mathcal{M}^{(c)}$ for this set. Note that not every reflex vertex of $\mathcal{M}^{(c)}$ is now a chain point, since reflex vertices can also be induced by reflex vertices of the original polygons. These reflex vertices of $\mathcal{M}^{(c)}$ are not taken into account in the VV$^{(c)}$-diagram.

Given a start and a goal configuration we just have to connect them to our VV$^{(c)}$-diagram and compute the shortest path between them using Dijkstra's algorithm. To this end, we have to associate a weight with each diagram edge. The weight of a visibility edge can simply be equal to its length, while for Voronoi edges we may add some penalty to the edge length, taking into account its clearance value, which is below the preferred c-value. It should be noted that if a path contains a portion of the Voronoi diagram it may not be smooth any more. This is however acceptable, as we consider making sharp turns inside narrow passages to be natural.

It takes $O(n^2 \log n)$ time to construct the VV$^{(c)}$-diagram of an input set $\mathcal{P}$ of pairwise interior-disjoint polygons for a given c-value if we use a straightforward approach. We note that it may also be possible to improve the running time to be $O(n \log n + k)$, where k is the number of visibility edges, based on the work of [21].[2]

4. THE VV-COMPLEX

The construction of the VV$^{(c)}$-diagram for a given c-value is straightforward, yet it requires some non-trivial geometric and algebraic operations that should be computed in a robust manner — see Section 5 for more details. Moreover, if we wish to plan motion paths for different c-values and select the best one (according to some criterion), we must construct the VV$^{(c)}$-diagram for each c-value from scratch. In this section we explain how to efficiently preprocess an input set of polygonal obstacles and construct a data structure called the VV-complex, which can be queried to produce a natural-looking path for every start and goal configuration and for any preferred clearance value c.

[2]The main difficulty here is that we handle the dilated obstacles, which may *not* be disjoint.

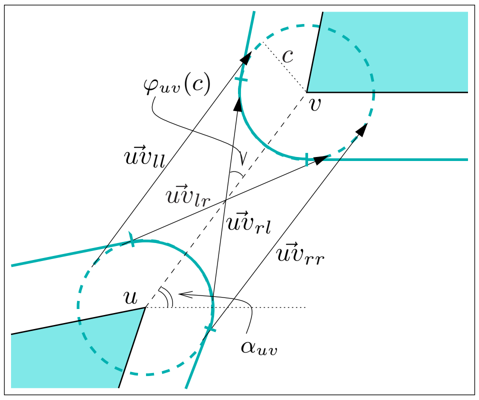


Figure 3: The four possible bitangents to the circles $B_c(u)$ and $B_c(v)$ of radius c centered at two obstacle vertices u and v. Notice that in this specific scenario only the bitangent $\vec{uv}_{rl}$ is a valid visibility edge.

Let us examine what happens to the VV$^{(c)}$-diagram as c continuously changes from zero to infinity. For simplicity, we consider only convex obstacles in this section. As we mentioned before, VV$^{(0)}$ is the visibility graph of the original obstacles, while VV$^{(\infty)}$ is their Voronoi diagram, so as c grows visibility edges disappear from VV$^{(c)}$ and make way to Voronoi chains. We start with a set of visibility edges containing all pairs of the polygonal obstacle vertices that are mutually visible, regardless whether these edges are bitangents of the obstacles.[3] We also include the original obstacle edges in this set, and treat them as visibility edges between two neighboring polygon vertices. Furthermore, we treat our visibility edges as directed, such that if the vertex u "sees" the vertex v, we will have two directed visibility edges $\vec{uv}$ and $\vec{vu}$.

As c grows larger than zero, each of the *original* visibility edges potentially spawns as many as four bitangent visibility edges. These edges are the bitangents to the circles $B_c(u)$ and $B_c(v)$ (where $B_r(p)$ denotes a circle centered at p whose radius is r) that we name $\vec{uv}_{ll}$, $\vec{uv}_{lr}$, $\vec{uv}_{rl}$ and $\vec{uv}_{rr}$, according to the relative position (left or right) of the bitangent with respect to u and to v (see Figure 3).[4] The two bitangents $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ retain the same slope, while the slopes of the other two bitangents change for increasing c-values.

Note that for a given c-value, it is impossible that all four edges are valid (at most three can be valid, and the ll- and rr-edges can never be valid simultaneously). Our goal is to compute a *validity range* $R(e) = [c_{\min}(e), c_{\max}(e)]$ for each edge e, such that e is part of the VV$^{(c)}$-diagram for each $c \in R(e)$.[5] If an edge is valid, then it must be tangent to

[3]Visibility edges are only *valid* when they are bitangents, otherwise they do not contribute to shortest paths in the visibility graph. However, as c grows larger these edges may become bitangents, so we need them in our data structure.

[4]Recall that edges in the visibility graph are *undirected*, thus our *directed* visibility edges come in pairs. According to our notation, $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ are equivalent to the opposite edges $\vec{vu}_{rr}$ and $\vec{vu}_{ll}$, respectively, while $\vec{uv}_{lr}$ and $\vec{uv}_{rl}$ are equivalent to $\vec{vu}_{lr}$ and $\vec{vu}_{rl}$, respectively. A pair of opposite edges always become valid or invalid simultaneously.

[5]Liu and Arimoto [17] use a similar notion to construct a structure that answers shortest-path queries for disc robots, where the radius of the robot is given in the query. They do

both circular arcs associated with its end-vertices. There are several reasons for an edge to change its validity status: (a) The tangency point of e to either $B_c(u)$ or to $B_c(v)$ leaves one of the respective circular arcs; (b) The tangency point of e to either $B_c(u)$ or to $B_c(v)$ enters one of the respective circular arcs; (c) The visibility edge becomes blocked by the interior of a dilated obstacle.

The important observation is that at the moment that a visibility edge $\vec{uv}$ gets blocked, it becomes tangent to another dilated obstacle vertex w, so essentially one of the edges associated with $\vec{uv}$ becomes equally sloped with one of the edges associated with $\vec{uw}$ (see Figure 4(a)). The first two cases mentioned above can also be realized as events of the same nature, as they occur when one of the $\vec{uv}$ edges becomes equally sloped with $\vec{uw}_{lr}$ (or $\vec{uw}_{rl}$), when v and w are neighboring vertices in a polygonal obstacle — see Figure 4(b).

This observation stands at the basis of the algorithm we devise for constructing the VV-complex: We sweep through increasing c-values, stopping at critical *visibility events*, which occur when two edges become equally sloped.[6] We note that the edge $\vec{uv}_{ll}$ (or $\vec{uv}_{lr}$) can only have events with arcs of the form $\vec{uw}_{ll}$ or $\vec{uw}_{lr}$, while the edge $\vec{uv}_{rl}$ (or $\vec{uv}_{rr}$) can only have events with arcs of the form $\vec{uw}_{rl}$ or $\vec{uw}_{rr}$. Hence, we can associate two circular lists $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$ of the left and right edges of the vertex u, respectively, both sorted by the slopes of the edges. Two edges participate in an event at some c-value only if they are neighbors in the list for infinitesimally smaller c. At these event points, we should update the validity range of the edges involved, and also update the adjacencies in their appropriate lists, resulting in new events.

In the rest of this section, we use the notation $\vec{uv}$ to represent any of the four edges $\vec{uv}_{ll}$, $\vec{uv}_{lr}$, $\vec{uv}_{rl}$ or $\vec{uv}_{rr}$. We also use $\mathcal{L}(u)$ to denote either $\mathcal{L}_l(u)$ or $\mathcal{L}_r(u)$ (whether we choose the "left" or the "right" list depends on the type of edge involved).

As mentioned in Section 3, an endpoint of a visibility edge in the VV$^{(c)}$-diagram may also be a chain point, so we must consider chain points in our algorithm as well. As a Voronoi chain is either monotone or has a single point with minimal clearance, we can associate at most two chain points with every Voronoi chain. Our algorithm will also have to compute the validity range for edges connecting a chain point with a dilated vertex or with another chain point. For that purpose, we will have a list $\mathcal{L}(p)$ of the outgoing edges of each chain point p, sorted by their slopes (notice that we do not have to separate the "left" edges from the "right" edges in this case).

In the next subsection we review the algorithmic details of the preprocessing stage for constructing the VV-complex, and describe how to query this data structure in Section 4.2. We conclude the presentation of the algorithm by a proof of correctness in Section 4.3 and a complexity analysis in Section 4.4. We finally explain how the algorithm can be generalized for non-convex polygons in Section 4.5.

not, however, incorporate portions of the Voronoi diagram in their construct.

[6]Our visibility events are somewhat reminiscent of the *merge events* and *split events* that occur in the algorithm for drawing "fat" planar edges, as suggested by Duncan et al. [7].

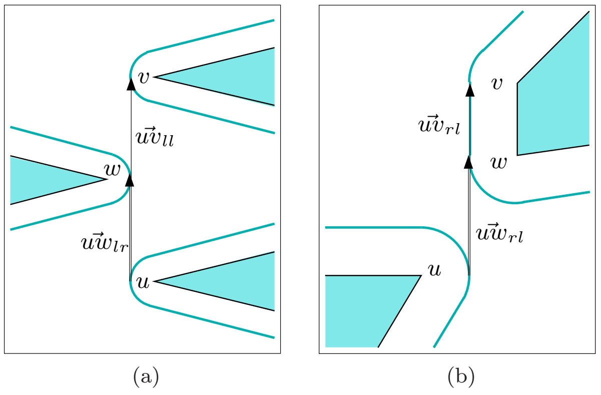


Figure 4: Visibility events involving u, v and w: (a) The dilated vertex w blocks the visibility of u and v. (b) As $\vec{uw}_{rl}$ becomes equally sloped with $\vec{uv}_{rl}$ (where vw is an obstacle edge), it becomes a valid visibility edge.

4.1 The Preprocessing Stage

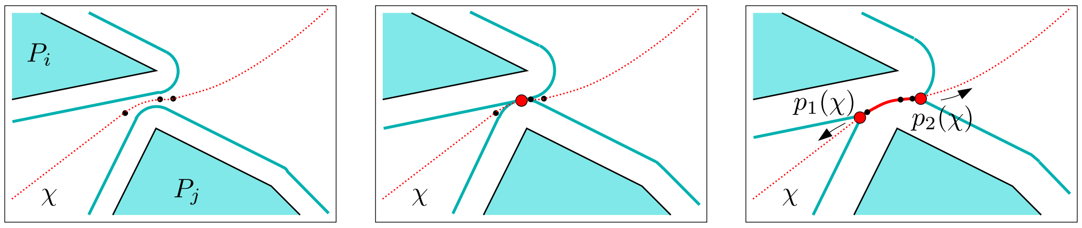
Given an input set $P_1, \ldots, P_m$ of polygonal obstacles as described above, we start by computing their visibility graph and classifying the visibility edges as valid (bitangent) or invalid. We examine each bitangent visibility edge uv: For an infinitesimally small c only one of the four $\vec{uv}$ edges it spawns is valid — we assign 0 to be the minimal value of the validity range of this edge (and of the opposite $\vec{vu}$ edge). As our algorithm is event-driven, we initialize an empty event queue $\mathcal{Q}$, storing events by increasing c-order. For each obstacle vertex u we construct $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$, based on the visibility edges we have just computed, and examine each pair of adjacent edges e_1, e_2 in $\mathcal{L}_l(u)$ and in $\mathcal{L}_r(u)$. We compute the c-value at which the adjacent e_1 and e_2 become equally sloped, if one exists, and insert the *visibility event* $\langle c, e_1, e_2 \rangle$ into $\mathcal{Q}$.

As our VV-complex also contains Voronoi chains, we have to compute the Voronoi diagram of the polygonal obstacles. For each *non-monotone* Voronoi chain, locate the arc a that contains the minimal clearance value $c_{\min}$ of the chain in its interior and insert the *chain event* $\langle c_{\min}, a \rangle$ into $\mathcal{Q}$.

While the event queue is not empty, we proceed by extracting the event in the front of $\mathcal{Q}$, associated with minimal c-value, and handle it according to its type. It is possible to express the critical c-values as algebraic numbers (i.e., roots of polynomials with integer coefficients), assuming that the input vertices have rational coordinates.

Visibility event: In a visibiliy event some edges become blocked and reach the end of their validity range, while some new edges may become valid. Visibility events always come in pairs — that is, if $\vec{uv}$ becomes equally sloped with $\vec{uw}$, we will either have an event for the opposite edges $\vec{vu}$ and $\vec{vw}$, or for the opposite edges $\vec{wu}$ and $\vec{wv}$. We therefore handle a pair of visibility events as a single event. Let us assume that the edges $\vec{uv}$ and $\vec{uw}$ become equally sloped for a clearance value c', and at the same time the edges $\vec{vu}$ and $\vec{vw}$ become equally sloped (see Figure 4).

As the edges $\vec{uv}$ and $\vec{vu}$ now become blocked, we assign c' to be the maximal c-value of the validity range of these edges. We also remove the other event, if any,



(a) (b) (c)

Figure 5: A chain event associated with the Voronoi chain χ (dotted) induced by the two obstacles P_i and P_j. The endpoints of the arcs forming χ are drawn as small black dots. (a) The clearance value c is less than the minimal clearance of the chain χ, so this chain does not contribute to the VV$^{(c)}$-diagram. (b) c equals the minimal clearance of the chain χ and a chain event occurs. Note that the two dilated obstacles now begin to intersect. (c) When c grows the two chain points $p_1(\chi)$ and $p_2(\chi)$, that define the portion of χ lying inside the VV$^{(c)}$-diagram(drawn in a solid line) move along the arcs of the chain χ toward its end-vertices (not shown in this figure).

involving $\vec{uv}$ (based on its other adjacency in $\mathcal{L}(u)$) from $\mathcal{Q}$, and delete this edge from $\mathcal{L}(u)$. We examine the new adjacency created in $\mathcal{L}(u)$ and insert its visibility event into the event queue $\mathcal{Q}$. We repeat this procedure for the opposite edge $\vec{vu}$.

If the edge $\vec{uv}$ was valid before it was deleted and the edges $\vec{uw}$ and $\vec{vw}$ do not have a minimal validity value yet, we assign c' to it, because these edges have become bitangent for this c-value (see Figure 4(b) for an illustration).

Chain event: The value c equals the minimal clearance of a Voronoi chain χ_a, obtained on the arc a, which is equidistant from an obstacle vertex u and another obstacle feature (see Figure 5(b)). Let z_1 and z_2 be a's endpoints. We initiate two chain points $p_1(\chi_a)$ and $p_2(\chi_a)$ associated with the Voronoi chain χ_a. As c grows, $p_1(\chi_a)$ moves toward z_1 and $p_2(\chi_a)$ moves toward z_2 (see Figure 5(c) for an illustration).

For all edges e incident to u, we compute the c-value c' for which e becomes incident to one of the chain points $p_i(\chi_a)$, and insert a *tangency event* $\langle c', e, p_i(\chi_a)\rangle$ into the event queue. If a is equidistant from u and from another obstacle vertex v, do the same for the edges incident to v.

Finally, we create two *endpoint events*, $\langle c_1, p_1(\chi_a), z_1\rangle$ and $\langle c_2, p_2(\chi_a), z_2\rangle$, associated with the clearance c_1 and c_2 values obtained at z_1 and z_2, respectively.

When dealing with a chain event, we introduced two additional types of events, used to handle chain points: tangency events and endpoint events. For a small enough c value, the endpoints of all visibility edges lie on dilated obstacle vertices, but as c grows these endpoints gradually become chain points. A *tangency event* occurs when a visibility edge becomes incident to a chain point. The *endpoint events* are used to transfer the chain points along Voronoi chains. We next explain how we deal with these events.

Tangency event: A visibility edge $e = \vec{ux}$ (the endpoint x may either represent a dilated vertex or a chain point) becomes tangent to $B_{c'}(u)$ at a chain point $p(\chi_a)$ associated with the Voronoi arc a (see Figure 6 for an illustration). In this case we have to replace e by $\overrightarrow{p(\chi_a)x}$ associated with the chain point $p(\chi_a)$: We assign c' to be the maximal validity value of the edge e, and remove this edge from $\mathcal{L}(u)$. We now insert a reincarnate of e to $\mathcal{L}(p(\chi_a))$, and assign c' as its minimal validity value. We examine the new adjacencies in $\mathcal{L}(p(\chi_a))$ and insert new visibility events into $\mathcal{Q}$. Finally, we replace the edge $\vec{xu}$ in $\mathcal{L}(x)$ by $\overrightarrow{xp(\chi_a)}$, recompute the critical c-values of the visibility events of this edge with its neighbors (notice that the slope of $\overrightarrow{xp(a)}$ becomes a different function of c from now on) and modify the corresponding visibility events in $\mathcal{Q}$.

In case x is a dilated obstacle vertex, we may have another tangency event in the queue, associated with $\vec{xu}$, which was computed under the (false) assumption that the tangency point of the edge on x coincides with a chain point before the one on u does. In this case, we have to locate the tangency event from $\mathcal{Q}$ that is associated with $\vec{xu}$ and recompute the c-value associated with it.

Endpoint event: A chain point $p(\chi_a)$ reaches the endpoint z of the Voronoi arc a. We should examine a few cases here, the simplest one occurs when z is incident only to two Voronoi arcs a and a' belonging to the same chain ($\chi_a = \chi_{a'}$). In this case the chain point $p(\chi_a)$ is transferred from a to a', and we only have to examine the adjacencies in $\mathcal{L}(p(\chi_{a'}))$ and modify the corresponding visibility events in the queue (as the slopes of these arc become a different function of c from now on). We also have to deal with the opposite edges, as we did in the tangency-event procedure. Furthermore, we take care of tangency events that occur in the range of the new Voronoi arc. Thus, if one of the polygon features associated with the new arc a' is a vertex u, iterate over all edges incident to u and check whether each edge has a tangency event in the range of the new Voronoi arc a' — if so, add this event to the queue $\mathcal{Q}$. If a' is associated with two vertices u and v, we repeat this procedure for v as well.

If z is a Voronoi vertex *and* a local maximum of the clearance function, there are multiple endpoint event associated with it, so we should just assign a maximal validity value to all edges in the edge lists of all chain points coinciding with z. Otherwise, z is the endpoint

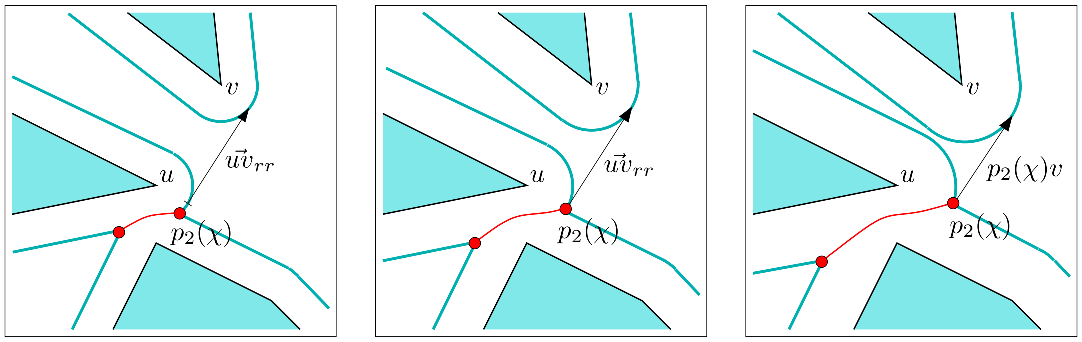


Figure 6: A tangency event: (a) The chain point $p_2(\chi)$, whose creation is depicted in Figure 5, lies on the supporting circle of the dilated vertex u. (b) The visibility edge $\vec{uv}_{rr}$ becomes tangent to $B_c(u)$ exactly at $p_2(\chi)$, so a tangency event occurs. (c) The reincarnated visibility edge $\overrightarrow{p_2(\chi)v}$ replaces $\vec{uv}_{rr}$ as c grows. Note that this edge is not tangent to $B_c(u)$ any more.

of the chain χ_a (i.e., a Voronoi vertex) and it is *not* a local maximum of the clearance function. In this case we may have several chains $\chi_1, \chi_2 \ldots$ ending at z, having a synchronous endpoint event, and a single monotone chain $\hat{\chi}$ beginning at z (see for example the marked vertex in Figure 1). We create a new chain point $p(\hat{\chi})$ associated with the monotone chain, assign a maximal validity value c' to each edge in $\mathcal{L}(p(\chi_1)), \mathcal{L}(p(\chi_2)), \ldots$, where c' is the clearance value at z. We remove all visibility events associated with these edges from $\mathcal{Q}$ and insert their reincarnates into $\mathcal{L}(p(\hat{\chi}))$. We examine all adjacencies in $\mathcal{L}(p(\hat{\chi}))$ and add the appropriate visibility event to $\mathcal{Q}$. We also have to deal with the opposite edges.

We note that in order to avoid duplicate work, when we have several events occurring at the same c-value, we deal with endpoint events first, to make sure that edges are associated with the correct chain. We can then handle the visibility events, chain events and finally the tangency events.

4.2 The Query Stage

The result of the preprocessing stage is the VV-complex $\langle \mathcal{V}, \mathcal{T} \rangle$: $\mathcal{V}$ is the Voronoi diagram of the polygonal obstacles; we also store the clearance value $c(z)$ of each vertex z in the Voronoi diagram, and for each non-monotone chain χ we store its minimal clearance value $c_{\min}(\chi)$. In addition, we have a set $\mathcal{T}$ of interval trees: For each obstacle vertex u, $\mathcal{T}_u$ contains the edges incident to u and their validity ranges (namely the intervals are the c-ranges). For each Voronoi chain χ, $\mathcal{T}_{\chi,i}$ is an interval tree storing the incident edges and incident Voronoi arcs to the ith chain point ($i \in \{1, 2\}$) of the chain χ, along with their validity ranges.

A query on the VV-complex is defined by a triple $\langle s, g, \hat{c} \rangle$, where s and g are the start and goal configurations, respectively, and $\hat{c}$ is the preferred clearance value. We assume that s and g themselves have a clearance larger than $\hat{c}$. Given a query, we start by computing the relevant portion of the Voronoi diagram: For each Voronoi chain we can examine the clearance values of its end-vertices, as well as the chain minimum, and determine which portion of the chain (if at all) we should consider. This way we also obtain all the chain points for the given c-value $\hat{c}$.

Next we need to find the incident edges of s and g. This means that we should obtain two lists $\mathcal{L}(s)$ and $\mathcal{L}(g)$ containing the visibility edges emanating from s and g (respectively) to every visible circular arc and chain point. This can be done using a radial sweep-line algorithm. We can now start searching the graph we have implicitly constructed using a Dijkstra-like search to find the "shortest" path between s and g (see the discussion in Section 3 about the weight we give the graph edges). The way we select the distance measure for the graph edges may depend on the path-planning strategy we employ. All visibility edges (and portions of the circular arcs which need to be traversed) have a clearance of at least $\hat{c}$, so their distance measure depends only on their length. For the portions of the Voronoi diagram, the limited amount of clearance may add extra weight.

Note that when we reach a vertex x (a dilated polygon vertex or a chain point) we query $\mathcal{T}_x$ with the given c-value $\hat{c}$ to obtain the valid edges incident to x, as we do not have an explicit representation of the graph. In addition, we add g to the list of x's neighbors if $x \in \mathcal{L}(g)$. If x is an obstacle vertex, we should keep in mind to add the length of the portion of the corresponding circular arc to the distance. We proceed until the goal configuration g is reached.

4.3 Proof of Correctness

Theorem 1. *Every visibility edge has only one continuous range $[c_{\min}, c_{\max}]$ of c-values for which it is valid. Thus, once it has been deleted it should not become valid again for a higher c-value.*

Proof. When we construct the VV-complex by gradually increasing the c-value, edges can only be deleted when a visibility event occurs and they become blocked by some dilated vertex.[7] Here we show that once an edge becomes blocked, it does not become unblocked again for a higher c-value.

Consider a visibility edge $\vec{uv}$ (it may either be invalid or valid) tangent to the supporting circles of the dilated vertices u and v, and a dilated vertex w whose supporting circle

[7]An edge can also reincarnate as a different edge, but in this case we can treat the validity range of its reincarnate as a direct continuation of the range of the original edge.

will start blocking the edge $\vec{uv}$ for some c-value c'. At the moment $c = c'$, the edge $\vec{uw}$ has the same slope as $\vec{uv}$, and $\vec{uv}$ is deleted. Assume without loss of generality that the slope of $\vec{uw}$ was *smaller* than the slope of $\vec{uv}$ for $c < c'$, then for c-values infinitesimally larger than c' the slope of $\vec{uw}$ is larger than the slope of uv (assuming that we do not delete $\vec{uv}$).

Obviously, $\vec{uv}$ is blocked by the dilated vertex w as long as the slope of $\vec{uw}$ is larger than the slope of $\vec{uv}$, so $\vec{uv}$ can only become unblocked again, if for some clearance value $c > c'$ the slope of $\vec{uw}$ will become smaller again than the slope of $\vec{uv}$. We conclude that there must exist some value $c'' > c'$ for which the edges $\vec{uv}$ and $\vec{uw}$ should become equally sloped for the second time.

It is easily proved that such a value c'' does not exist. It is not difficult to show that to compute for which c-values two edges are equally sloped one needs to solve a quadratic equation that has one positive and one negative solution. Hence, there is only one c-value between 0 and ∞ for which the two bitangent edges are equally sloped. This means that the edge $\vec{uv}$ cannot become unblocked once it has been deleted.

The proof above holds for a visibility edge $\vec{uv}$ tangent to two dilated vertices, but in our algorithm we also consider edges $\vec{xv}$ between a chain point and a dilated vertex. Assume that $\vec{xv}$ is blocked by a dilated vertex w, so it becomes equally sloped with $\vec{xw}$. To compute the c-values for which $\vec{xv}$ and $\vec{xw}$ become equally sloped we have to solve a polynomial of degree 4, that can have as many as four distinct solutions. However, two of these solutions are negative, and one of the positive solutions in not relevant as it relates to the second chain point associated with the same chain. We conclude that there is only one positive c-value for which the edge $\vec{xv}$ becomes equally sloped with $\vec{xw}$, so once w blocks $\vec{xv}$ the edge cannot become unblocked again. The same argument holds for blocking an edge $\vec{xy}$ between two chain points. The equation we need to solve in this case has as many as eight solutions, but only two of them relate to the correct pair of chain-points, of which only one is positive. Hence, after an edge has been deleted, it will not become valid again. $\square$

4.4 Complexity Analysis

Theorem 2. *The preprocessing stage takes $O(n^2 \log n)$ in total, where n is the total number of obstacle vertices.*

Proof. In the initialization of the preprocessing stage we first have to compute the visibility graph, which can be performed in $O(n^2 \log n)$ time — this also accounts for the time needed to construct the initial edge lists $\mathcal{L}(u)$ for each obstacle vertex u (we need $O(n \log n)$ time to construct each of the $2n$ edge lists) and label the valid visibility edges. The construction of the Voronoi diagram can be performed in $O(n \log n)$, and the complexity of the diagram (the number of arcs) is linear in n.

After the initialization, the priority queue $\mathcal{Q}$ contains $O(1)$ events associated with each of the $O(n^2)$ visibility edges, and in addition $O(n)$ chain events. Any operation on the event queue thus takes $O(\log n)$. The initialization takes $O(n^2 \log n)$ time in total.

As the preprocessing algorithm proceeds, it starts handling events: In total, by Theorem 1 we have $O(n^2)$ visibility events,[8] each of them can be handled in $O(\log n)$ time as it involves a constant number of operations on the queue and on the edge lists. There are $O(n)$ chain events, each of them can be handled in $O(n \log n)$ time. Each chain event spawns $O(n)$ tangency events, so in total there are $O(n^2)$ tangency events, each of them can be handled in $O(\log n)$ time. Finally, there are $O(n)$ endpoint events, and we need $O(n \log n)$ time to handle each of these events. $\square$

The query phase takes in any case $O(n \log n)$ time, which is spent on calculating the valid visibility edges emanating from s and g. Calculating the relevant portions of the Voronoi diagram takes $O(n)$ time (note that the Voronoi diagram itself has already been constructed in the preprocessing phase).

The rest of the query phase consists of executing Dijkstra's algorithm, or an equally suited A*-algorithm. The worst-case running-time of these algorithms is $O(n \log n + \ell)$ where $\ell = O(k)$ is the number of edges encountered during the search. In practice, Dijkstra's algorithm turns out to be very fast, because hardly any geometric operations have to be performed anymore. In particular the A*-variant of Dijkstra may be the method of choice here, as it biases the search toward the goal configuration, which keeps the number ℓ low.

As we noted in Section 3, the VV$^{(c)}$-diagram for a fixed c-value may be constructed in $O(n \log n + k)$ time (recall that k is the number of visibility edges) so it may seem we do not need any preprocessing stage, and it is better to construct the VV$^{(c)}$-diagram from scratch whenever we are given a preferred clearance value. However, this algorithm involves the construction of the planar arrangement of line segments, circular arcs and parabolic arcs, which is very complicated when carried out in a *robust* manner (see the next section). Such an approach will require longer running times than the query stage of the second algorithm. We note that Dijkstra's algorithm, whose running time theoretically dominates the query phase, is in practice very fast if after preprocessing our set of input obstacles in an exact manner, we switch to machine-precision floating-point arithmetic in the query stage.

4.5 Handling Non-Convex Obstacles

So far we described the algorithm for constructing a VV-complex for a set of convex polygonal obstacles. Our algorithm can however be easily adapted to work with non-convex obstacles as well. The only thing that is changed is the way in which the Voronoi diagram is constructed.

Due to the non-convexity of the obstacles, some obstacles may contain reflex vertices. These reflex vertices are treated as normal vertices in the initial construction (for $c = 0$) of the visibility graph. Note that the visibility edges emanating from reflex vertices will never be part of a shortest path, but we still need to keep track of these edges, as they may induce visibility events that give other valid edges the correct c-values of their validity ranges.

As c grows, the reflex vertices will be treated as chain points. These chain points move over monotone Voronoi chains originating in the reflex vertices themselves. To this

[8]Note that we consider all potential events in our analysis. In practice, some of these events that were computed under false assumptions (see Section 4.1) and are eventually discarded.

end, the definition of the Voronoi diagram should be adapted such that Voronoi arcs can be equidistant to two edges of the same polygon as well. Still, this new Voronoi diagram is an instance of the Voronoi diagram of line segments, so this change is easily carried through. The rest of the algorithm remains unchanged. Also, the complexity analysis is still valid, since the construction time and the complexity of both the visibility graph and the Voronoi diagram is not affected by the non-convexity of its input obstacles.

5. IMPLEMENTATION DETAILS

CGAL, the Computational Geometry Algorithms' Library [1] offers the infrastructure we need for developing a robust software for computing the $\mathrm{VV}^{(c)}$-diagram. We use the software components developed by Hirsch and Leiserowitz [11] for constructing the union of the Minkowski sums of the polygonal obstacles with a disc of radius c. The Voronoi diagram of the polygons is computed using a recently implemented CGAL package by Karavelas [13] for computing Voronoi diagrams of line segments: We simply add a label to each segment (polygon edge) that identifies its source polygon, and disregard Voronoi chains induced by features of the same polygon (we omit the technical detail related to chains induced by non-convex polygons).

The intersection between the dilated obstacles and the Voronoi arcs is robustly computed using the conic-arc traits [22] of CGAL's arrangement package [8]. We exploit the fact that our polygonal obstacles are given as sequences of points with *rational* coordinates, so that the supporting curves of each dilated obstacle boundary and each Voronoi arc can be represented as algebraic curves of degree 2 with rational coefficients if the squared clearance value is also rational (see below), to robustly maintain the arrangement of such curves.

LEMMA 3. *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of pairwise interior-disjoint simple polygons, such that all polygon vertices have* rational *coordinates. Then all Voronoi arcs have supporting algebraic curves of degree 2 at most with rational coefficients and all chain minima are also points with rational coordinates. Moreover, for a clearance value c such that c^2 is rational, the dilated obstacle boundaries are also supported by algebraic curves of degree 2 with rational coefficients.*

5.1 Voronoi Arcs

An arc a of the Voronoi diagram corresponds to the locus of all points equidistant from two polygon features, and the following cases are possible:

Vertex–vertex arc: The arc is equidistant from two polygon vertices u and v. The equation of its supporting curve, a line in this case, is simply given by (throughout this section we use the squared distance, in order to avoid the square-root operation):

$$\begin{aligned}(x-x_u)^2+(y-y_u)^2 &= (x-x_v)^2+(y-y_v)^2\\ 2(x_v-x_u)x+2(y_v-y_u)y &= x_v^2+y_v^2-(x_u^2+y_u^2)\ . \quad (1)\end{aligned}$$

Note that this line is perpendicular to the line segment connecting u an v and bisects it. The point with minimal clearance on the arc is therefore the midpoint between u and v, $z_{\min} = \frac{1}{2}(x_u + x_v, y_u + y_v)$, and its clearance is of course $c_{\min} = \frac{1}{2}d(u, v)$.

Vertex–edge arc: The arc is equidistant from a polygon vertex u and a polygon edge vw, whose supporting line is denoted $\ell :\ Ax+By+C = 0$, where A, B and C are rational (since the vertices have rational coordinates). The equation of its supporting curve, a parabola in this case, is thus given by:

$$\frac{(Ax+By+C)^2}{A^2+B^2} = (x-x_u)^2+(y-y_u)^2\ . \qquad (2)$$

In this case, to find the point with minimal clearance on the arc we compute a line perpendicular to ℓ that passes through u. The equation of this line is $\ell^{\perp} :\ By - Ax + (Ay_u - Bx_u) = 0$, and the point with minimal clearance is the midpoint between u and the intersection point of ℓ and $\ell^{\perp}$:

$$\begin{aligned} z_{\min} &= \frac{1}{2}\left(x_u + \frac{B^2x_u - A(By_u+C)}{A^2+B^2}\ ,\right.\\ &\qquad\qquad \left. y_u + \frac{A^2y_u - B(Ax_u+C)}{A^2+B^2}\right)\ .\end{aligned}$$

The minimal clearance is half the distance between u and the line ℓ.

Edge–edge arc: The arc is equidistant from two polygon edges, whose supporting lines are denoted $\ell_1 :\ A_1x + B_1y + C_1 = 0$ and $\ell_2 :\ A_2x + B_2y + C_2 = 0$, respectively. The supporting curve of this edge is a line, but in general this line cannot be represented as a linear curve with rational coefficients. Instead, we represent the edge as a segment of a pair of perpendicular lines, which form the two angle bisectors of ℓ_1 and ℓ_2:

$$\frac{(A_1x+B_1y+C_1)^2}{A_1^2+B_1^2} = \frac{(A_2x+B_2y+C_2)^2}{A_2^2+B_2^2}\ . \qquad (3)$$

As we mentioned before, such an arc is always monotone — that is, as we traverse it from the endpoint with smaller clearance value to the other endpoint, we get further away from the obstacles.

5.2 Dilated Obstacle Boundaries

Dilated vertex: Each convex polygon vertex u induces a circular arc, which is a segment of the circle $B_c(u)$, given by the equation:

$$(x-x_u)^2+(y-y_u)^2 = c^2\ . \qquad (4)$$

Since x_u, y_u and c^2 are all rational, $B_c(u)$ has rational coefficients.

Dilated edge: The edges of the dilated obstacles are formed by offsetting the polygon edges parallel to themselves. However, it is impossible to represent a dilated edge as a linear curve with rational coefficients. Instead, we represent it as a segment of a pair of parallel lines, representing the locus of all points whose distance from the line $\ell :\ Ax + By + C = 0$ supporting the original polygon edge equals c:

$$\frac{(Ax+By+C)^2}{A^2+B^2} = c^2\ . \qquad (5)$$

The two endpoints of the segment lie of course on one of the two lines given by the equation above, and not on the other.

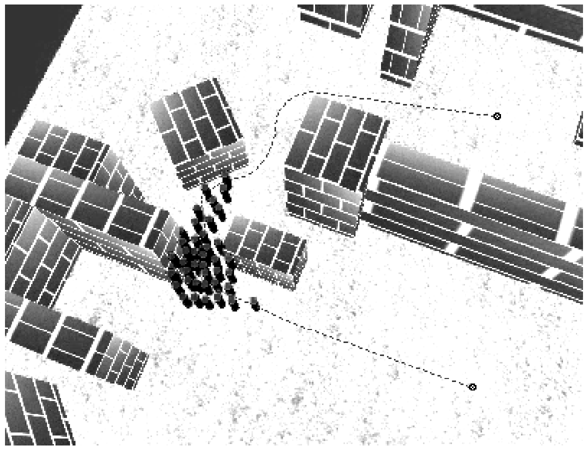


Figure 7: A group of 40 entities moving in a virtual scene along a backbone path, drawn with a dashed line. (Courtesy of Arno Kamphuis.)

5.3 Experimental Results

Our software is implemented using CGAL 3.1 and the exact number types are supplied by CORE 1.7 [2]. As we wish to obtain an exact representation of the $VV^{(c)}$-diagram, we may spend some time on the diagram construction, especially if it contains chain points, which are algebraically more difficult to handle. For example, the construction of the $VV^{(c)}$-diagram depicted in Figure 2 takes about 10 seconds (running a Pentium IV 2 GHz machine with 512 MB of RAM), but if we choose a smaller clearance value for the same scene, such that no chain points appear in the diagram, the construction time drops to 2.5 seconds. However, once the $VV^{(c)}$-diagram is constructed, it is possible to use a floating-point approximation of the edge lengths to speed up the time needed for answering motion-planning queries,[9] so that the average query time is only a few milliseconds.

We also used the $VV^{(c)}$-diagram to generate convincing group motions in a more complex scene, as the one depicted in Figure 7. The construction of such diagrams takes about 40–60 seconds (for clearance values that induce chain points), but the average query time was only a few milliseconds. This is a considerable improvement over previous techniques, which require smoothing operations in the query stage, taking about one second on average.

Acknowledgements

We thank Arno Kamphuis for providing us with screen shots from his coherent group-motion demo (Figure 7).

6. REFERENCES

[1] The CGAL project homepage. `http://www.cgal.org/`.
[2] The CORE library homepage. `http://www.cs.nyu.edu/exact/core/`.
[3] E. U. Acar, H. Choset, and P. N. Atkar. Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and Voronoi diagrams. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 1305–1311, 2001.
[4] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Computational Geometry: Theory and Applications*, 21:39–61, 2002.
[5] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter V, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 2nd edition, 2000.
[7] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. In *Proc. 9th Intern. Sympos. Graph Drawing*, pages 162–177, 2001.
[8] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. In *Proc. 12th Europ. Sympos. Algorithms*, pages 664–676. Springer-Verlag, 2004.
[9] R. Geraerts and M. H. Overmars. Clearance based path optimization for motion planning. In *IEEE Intern. Conf. Robotics and Automation*, pages 2386–2392, 2004.
[10] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.
[11] S. Hirsch and E. Leiserowitz. Exact construction of Minkowski sums of polygons and a disc with application to motion planning. Technical Report ECG-TR-181205-01, Tel-Aviv University, 2002.
[12] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In R. Boulic and D. K. Pai, editors, *Eurographics/ACM SIGGRAPH Sympos. Computer Animation*, pages 1–10, 2004.
[13] M. I. Karavelas. Segment Voronoi diagrams in CGAL, 2004. `http://www.cgal.org/UserWorkshop/2004/svd.pdf`.
[14] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12:566–580, 1996.
[15] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete and Computational Geometry*, 1:59–70, 1986.
[16] D.-T. Lee and R. L. Drysdale III. Generalization of Voronoi diagrams in the plane. *SIAM J. on Computing*, 10(1):73–87, 1981.
[17] Y. H. Liu and S. Arimoto. Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph. *IEEE Trans. Robotics and Automation*, 11:682–691, 1995.
[18] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 27, pages 607–642. CRC, 2nd edition, 2004.
[19] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE Intern. Conf. Robotics and Automation*, pages 446–452, 2004.
[20] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disk. *J. Algorithms*, 6:104–111, 1985.
[21] M. Pocchiola and G. Vegter. The visibility complex. *International J. of Computational Geometry and Applications*, 6(3):279–308, 1996.
[22] R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. 10th Europ. Sympos. Algorithms*, pages 884–895. Springer-Verlag, 2002.

[9]Indeed, we lose some accuracy here, but as our constructed diagram is topologically correct, the worst thing that can happen is that we may compute a path that is only slightly longer than the shortest possible path.