The Visibility–Voronoi Complex and Its Applications

Ron Wein

Jur P. van den Berg

Dan Halperin

institute of information and computing sciences, utrecht university

technical report UU-CS-2005-040

www.cs.uu.nl

The Visibility–Voronoi Complex and Its Applications *

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

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

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

Abstract

We introduce a new type of diagram called the $VV^{(c)}$-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 $VV^{(c)}$-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 $VV^{(c)}$-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 $VV^{(c)}$-diagram in an *exact* manner for a given clearance value, and used it to plan natural-looking paths in various applications.

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

*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.

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, as suggested by Kedem *et al.* [17]. This approach was simplified, by decomposing the configuration space into pseudo-trapezoidal cells and constructing a roadmap of the free cells, and was robustly implemented for a polygonal robot [4] and for a disc robot [13]. However, the trapezoidal-map approach yields paths that are piecewise linear (hence not smooth) and that are often not the shortest paths. Another popular approach for solving motion-planning problems is using Probabilistic Roadmaps (PRMs — see, e.g., [16]) — but the output paths in this case are also piecewise linear and may be far from the shortest possible paths. Indeed, in both cases it is possible to perform path smoothing as a post-processing stage and produce a more natural-looking path (see [11] 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., [7, 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 [23] 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 $\mathrm{VV}^{(c)}$-*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 keeping a distance of at least c from the obstacles. As we still want to keep the option of traversing these narrow passages (for example when a pass through a narrow passage is significantly shorter than any alternative path), we integrate into the diagram paths with the maximal possible clearance in regions where the preferred clearance c cannot 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

[1]Despite the similarity in names, our structure is different from the *visibility complex* introduced by Pocchiola and

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, by performing a Dijkstra search on an implicitly constructed graph encoded by the VV-complex. The preprocessing time is $O(n^2 \log n)$, where n is the total number of obstacle vertices, and the query takes $O(n \log n + \ell)$ time, where ℓ 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.

Applications

A direct application of our constructs 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 (see the discussion in Section 3) 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 [14] 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 [21] to compute the initial path, then apply smoothing techniques on it to achieve natural-looking motions. While the smoothing procedure outputs more natural paths, it has several drawbacks: First, it is an expensive operation, so to obtain real-time performance it can only be done for the final selected path, and not for other candidate paths that can be computed using the PRM. Note that as smoothing may considerably deform the original path, it is not guaranteed that we get the shortest path possible if we smooth the shortest path obtained from the PRM. Also, as the PRM method is not complete, it is not guaranteed that the PRM contains all possible paths between the start and goal locations. It is even possible that the two locations cannot be connected using the PRM although there exists a path connecting them. On the other hand, 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 $\mathrm{VV}^{(c)}$-diagram does not require any smoothing step, which saves a precious amount of time and enables real-time performance.

The principles of our construction may also be applied to sensor-based coverage using a robot with a limited sensor radius. Acar *et al.* [3] devised an algorithm for a disc robot of radius r, carrying a detector with a range $R > r$, to detect all points in an unknown environment. They decompose the free space into *vast* cells, where the robot traverses the boundary of the obstacles dilated by radius R, and *narrow* cells, where the obstacles are within the detector range and the robot has to follow the Voronoi diagram. It is possible to use the $\mathrm{VV}^{(c)}$-diagram in this case for traversing the narrow cells, as it naturally connects the relevant portions of the Voronoi diagram to the vast cells.

We have implemented our algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram for a given clearance value and applied it to the problem of motion planning for coherent groups of entities. The paths

Vegter [24] for efficiently computing the visibility among disjoint convex objects in the plane.

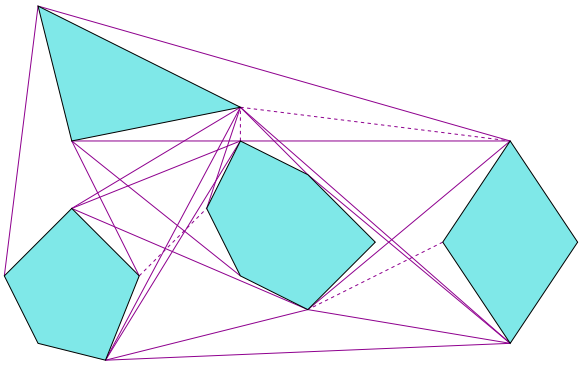


Figure 1: The visibility graph of a set of four convex polygons. The valid visibility edges are drawn with solid lines, while some invalid edges are also shown, drawn with dashed lines. Notice that all obstacle edges are also valid visibility edges.

contained in the $\mathrm{VV}^{(c)}$-diagram yield convincing group motions, and the approach we propose has several advantages over methods used so far to generate group paths. We note that the robust construction of the $\mathrm{VV}^{(c)}$-diagram involves many non-trivial geometric procedures and requires careful algebraic computations, which we also discuss in this paper.

Outline

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-value. We finally show some experimental results in Section 6 and give concluding remarks in Section 7.

2 Preliminaries

2.1 Visibility Graphs

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of simple pairwise interior-disjoint polygons having n vertices in total. The *visibility graph* of $\mathcal{P}$ is an undirected graph defined on the set of polygon vertices, 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 $\mathcal{P}$ — in this case, we call this segment a *visibility edge*.

The visibility graph can be used to compute shortest paths amidst configuration-space polygonal obstacles, where the polygons are considered as open sets. Each edge is given a weight equal to the Euclidean distance between its two end-vertices. To find a 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. In fact, it is sufficient to consider only the edges that are bitangent to the polygons they connect, namely edges that can be infinitesimally extended without penetrating any polygon. Such bitangent edges are called *valid* visibility edges (see Figure 1 for an illustration).

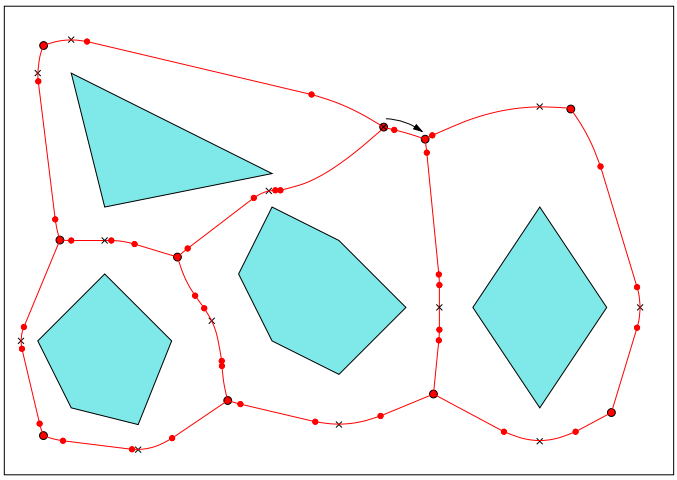


Figure 2: The Voronoi diagram of four convex polygons contained inside a rectangle. Small dots mark the endpoints of each Voronoi arc, while the Voronoi vertices are marked by larger dots. The point of minimum clearance along each chain is marked by $\times$. Notice that the chain marked by an arrow is a monotone chain and obtains its minimal clearance on its left Voronoi vertex — so when we traverse it in the arrow's direction, the clearance only increases.

The visibility graph can straightforwardly be computed in $O(n^2 \log n)$ time, performing a radial sweep around each of the polygon vertices (see, e.g., [7, Chapter 15]). Ghosh and Mount [12] were the first to give an output-sensitive algorithm for computing the visibility graph in optimal $O(n \log n + k)$ time, where k is the number of visibility edges in the output visibility graph. For more information on shortest paths, see [20].

2.2 Voronoi Diagrams of Polygons

Given a set S of geometric entities in $\mathbb{R}^d$ and a distance metric $\|\cdot\|$, the *Voronoi diagram* of S, denoted $\mathrm{Vor}(S)$, is the subdivision of $\mathbb{R}^d$ into maximal connected cells, such that the points in each *Voronoi cell* are closer to a specific entity of S than to all other entities of S.

There are many variants of Voronoi diagrams (see [5, 10] for extensive reviews), here we focus on the Voronoi diagram of a set of pairwise interior-disjoint polygons in $\mathbb{R}^2$ under the Euclidean distance metric, which can be regarded as a special case of a Voronoi diagram of line segments [18]. The *Voronoi vertices* in this case 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 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 2 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, 18]). For more details on the connection between Voronoi diagrams and motion planning see [22, 23, 25].

2.3 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. This Minkowski sum of a set of polygons $\mathcal{P}$ as above and a disc has $O(n)$ complexity and can be computed in $O(n \log^2 n)$ time using a divide-and-conquer algorithm [17], where n is the number of polygon vertices. It is also possible to use an incremental randomized algorithm that achieves a running time of $O(n \log n)$ [6]. See [7, Chapter 13] and [13] for further discussions and more references.

3 The $\mathrm{VV}^{(c)}$-Diagram

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of simple pairwise interior-disjoint polygons in the plane, having n vertices in total, representing two-dimensional configuration-space obstacles. Let c be the preferred distance we wish to keep from these obstacles. Our goal is to preprocess $\mathcal{P}$, so that given a start configuration s and a goal configuration g, we can efficiently compute a shortest path between s and g, 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 — that is, 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 3 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 $\mathrm{VV}^{(c)}$-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 value c, we perform the following steps:

1. 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.

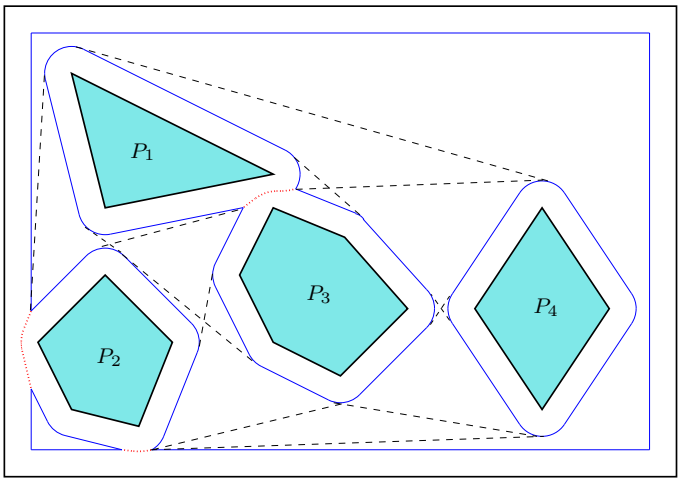


Figure 3: The $\mathrm{VV}^{(c)}$-diagram for four convex obstacles located in a rectangular room. The boundary of the union of the dilated obstacles is drawn in a solid line, the relevant portion of the Voronoi diagram is dotted. The visibility edges are drawn using a dashed line. Notice that an endpoint of a visibility edge may either lie on a circular arc or on the intersection of two dilated obstacle boundaries (a chain point).

2. We compute the union $\mathcal{M}^{(c)}$ of all $M_i^{(c)}$. The boundary of $\mathcal{M}^{(c)}$ consists of circular arcs and straight line segments. Reflex vertices may appear on the boundary of $\mathcal{M}^{(c)}$, which 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.

3. We compute the modified visibility graph $\mathcal{G}^{(c)}$ of $\mathcal{M}^{(c)}$. This graph consists of 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.

4. We 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 $\mathrm{VV}^{(c)}$-diagram.

As mentioned in Section 2.3, step 1 can be carried out in linear time while step 2 takes $O(n \log^2 n)$ time (or $O(n \log n)$ time using a randomized algorithm). As step 4 takes linear time, step 3, which takes $O(n^2 \log n)$, clearly dominates the running time of the $\mathrm{VV}^{(c)}$-diagram construction process. We conclude that it takes $O(n^2 \log n)$ time to construct the $\mathrm{VV}^{(c)}$-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 might also be possible to improve the running time to be $O(n \log n + k)$, where k is the number of visibility edges, by constructing the visibility complex of the dilated polygons [24].[2]

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 in this case 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.

[2]The main difficulty here is that we handle *dilated* obstacles, which may *not* be disjoint. Moreover, the obstacles (and of course the dilated obstacle) are not of constant complexity.

However, these reflex vertices of $\mathcal{M}^{(c)}$ can be easily identified and are not taken into account in the VV$^{(c)}$-diagram (namely the diagram does not contain visibility edges emanating from these vertices).

Querying the VV$^{(c)}$-Diagram

Having constructed the VV$^{(c)}$-diagram, once we are given a start configuration s and a goal configuration g we just have to connect them to our diagram and compute the shortest path between s and g using Dijkstra's algorithm. It takes $O(n \log n)$ to connect s and g to the diagram, by performing radial sweep from each configuration. The execution of Dijkstra's algorithm takes $O(n \log n + \ell)$, where ℓ is the number of diagram edges we encounter during the search.

As mentioned before, we may compromise of the amount of clearance our motion path keeps from the obstacles if we can make a considerable shortcut by traversing through a narrow passage. 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). In order to balance between the length and the clearance of the selected path we have to associate the appropriate weight with each diagram edge, so the Dijkstra algorithm outputs the path which is most suitable for our application. The weight of a visibility edge can simply be equal to its length (the lengths of the circular arcs we traverse must also be taken into consideration), while for Voronoi edges we may add some penalty to the edge length, taking into account their clearance values, which are below the preferred c-value. For example, if the minimal clearance of a Voronoi arc is $c' < c$, we can give it the weight of its length multiplied by $\left(\frac{c}{c'}\right)^{\kappa}$, where $\kappa > 0$ is a parameter controlling the amount of extra weight given to Voronoi arcs.

Another option of weighting the edges, especially suitable for the application of coherent group motion (see Section 1) is to estimate the time it takes the group to traverse each edge: For edges with a clearance of at least $c = \frac{w}{2}$, where w is the preferred group width, this time is clearly proportional to the edge length. On the other hand, for Voronoi edges the actual clearance of the edge would also be taken into account, as the moving entities will have to traverse this edge in a long row. The resulting path will therefore be the one enabling the group to reach its goal as quickly as possible.

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.

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,

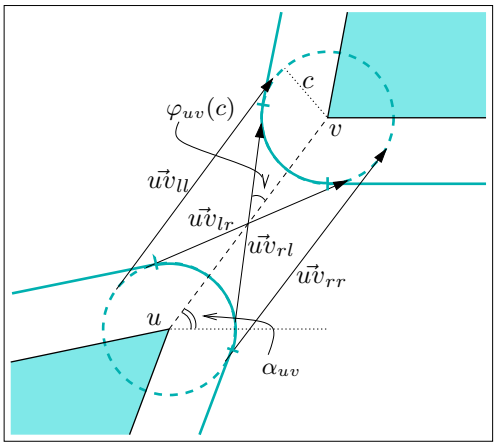


Figure 4: The four possible bitangents to the circles $B_c(u)$ and $B_c(v)$ of radius c centered at two obstacle vertices u and v. Notice that in this specific scenario only the bitangent $\vec{uv}_{rl}$ is a valid visibility edge.

regardless whether these edges are bitangents of the obstacles.[3] We also include the original obstacle edges in this set, as they can be viewed as visibility edges between two adjacent 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 in our structure, $\vec{uv}$ 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 4).[4] Let α_{uv} be the angle between the vector $\vec{uv}$ and the x-axis, and $d(u, v)$ the Euclidean distance between u and v, then it is easy to see that the two bitangents $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ retain the same slope α_{uv} for increasing c-values. The slope of the other two bitangents changes as c grows: $\vec{uv}_{rl}$ rotates counterclockwise and $\vec{uv}_{lr}$ rotates clockwise by the same amount, both around the midpoint $\frac{1}{2}(u + v)$ of the original edge, so their slopes become $\alpha_{uv} + \varphi_{uv}(c)$ and $\alpha_{uv} - \varphi_{uv}(c)$, respectively, where $\varphi_{uv}(c) = \arcsin(\frac{2c}{d(u,v)})$. For $c > \frac{1}{2}d(u, v)$ the two edges $\vec{uv}_{rl}$ and $\vec{uv}_{lr}$ disappear.

Note that for a given c-value, it is impossible that all four edges are valid (at most three can be valid, and the edges $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ 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 $\mathrm{VV}^{(c)}$-diagram for each $c \in R(e)$.[5] If an edge is valid, then it must be tangent to both circular arcs associated with its end-vertices. There are several reasons for an edge to change its validity status:

- The tangency point of e to either $B_c(u)$ or to $B_c(v)$ leaves one of the respective circular arcs.
- The tangency point of e to either $B_c(u)$ or to $B_c(v)$ enters one of the respective circular arcs.
- The visibility edge becomes blocked by the interior of a dilated obstacle.

[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 the invalid 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 [19] 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 not, however, incorporate portions of the Voronoi diagram in their construct.

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 5(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 adjacent vertices in a polygonal obstacle — see Figure 5(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 be involved in visibility 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 one of these lists 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.

As mentioned in Section 3, an endpoint of a visibility edge in the $\mathrm{VV}^{(c)}$-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 ranges of 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 continue 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.

4.1 The Preprocessing Stage

4.1.1 Initialization

Given an input set $P_1, \ldots, P_m$ of convex interior-disjoint polygonal obstacles, 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.

We proceed by constructing the circular lists $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$ for each obstacle vertex u, based on the visibility edges we have just computed. We examine each pair of adjacent edges e_1, e_2 in $\mathcal{L}_l(u)$ (and in $\mathcal{L}_r(u)$), compute the c-value at which e_1 and e_2 become equally sloped — if one exists — and insert the *visibility event* $\langle c, e_1, e_2 \rangle$ into the event queue. In a visibility event some edges become blocked and reach the end of their validity range, while some new edges may become valid.

[6] Our visibility events are reminiscent of the *merge events* and *split events* that occur in the algorithm for drawing "fat" planar edges, as suggested by Duncan *et al.* [8].

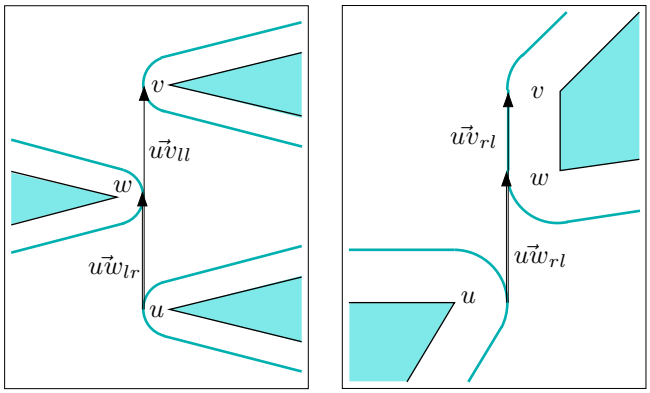


Figure 5: Visibility events involving u, v and w: (a) The dilated vertex w blocks the visibility of u and v. (b) As $u\vec{w}_{rl}$ becomes equally sloped with $u\vec{v}_{rl}$ (where vw is an obstacle edge), it becomes a valid visibility edge.

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 we 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 $\mathcal{Q}$. A chain event occurs when a Voronoi chain starts contributing to the $\mathrm{VV}^{(c)}$-diagram, namely when we sweep through its minimal clearance value.

4.1.2 Event Handling

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.

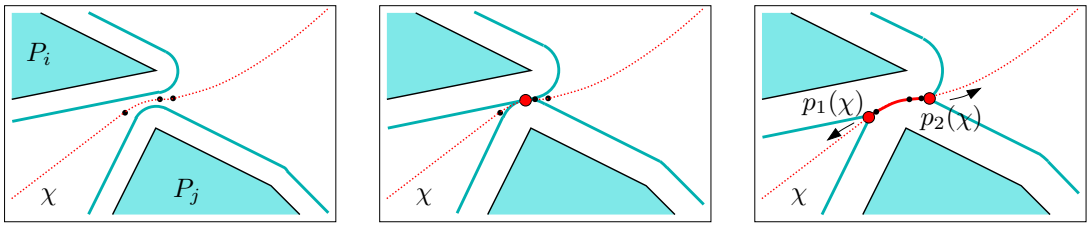
Visibility event: Visibility events always come in pairs — that is, if $u\vec{v}$ becomes equally sloped with $u\vec{w}$,[7] we will either have an event for the opposite edges $v\vec{u}$ and $v\vec{w}$, or for the opposite edges $w\vec{u}$ and $w\vec{v}$. We therefore handle a pair of visibility events as a single event. Let us assume that the edges $u\vec{v}$ and $u\vec{w}$ become equally sloped for a clearance value c', and at the same time the edges $v\vec{u}$ and $v\vec{w}$ become equally sloped (see Figure 5).

As the edges $u\vec{v}$ and $v\vec{u}$ 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, involving $u\vec{v}$ (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 $v\vec{u}$.

If the edge $u\vec{v}$ was valid before it was deleted and the edge $u\vec{w}$ (or $v\vec{w}$) does not have a minimal validity value yet, we assign c' to it, because this edge has become bitangent for this c-value (see Figure 5(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 Fig-

[7] In the rest of this section, we use the notation $u\vec{v}$ to represent any of the four edges $u\vec{v}_{ll}$, $u\vec{v}_{lr}$, $u\vec{v}_{rl}$ or $u\vec{v}_{rr}$. 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).



(a) (b) (c)

Figure 6: 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 $\mathrm{VV}^{(c)}$-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 $\mathrm{VV}^{(c)}$-diagram(drawn in a solid line) move along the arcs of the chain χ toward its end-vertices (not shown in this figure).

ure 6(b)).[8] 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 6(c) for an illustration).

As we increase c, larger portions of χ_a will enter the $\mathrm{VV}^{(c)}$-diagram and visibility edges will become incident to its chain points, rather than to dilated vertices. We therefore have to examine all edges e incident to u, compute the c-value c' for which e becomes incident to one of the chain points $p_i(\chi_a)$, and insert the *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 (i.e., a is a vertex–vertex Voronoi arc), we 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 values c_1 and c_2 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 (smaller than the clearance value of any point on the Voronoi diagram) 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 $e = u\vec{x}$ (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 7 for an illustration). In this case we have to replace e by the visibility edge $p(\vec{\chi_a})x$ associated with the chain point $p(\chi_a)$: We assign c' to be the maximal validity value of the edge e, and remove it 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 adjacency in $\mathcal{L}(p(\chi_a))$ and insert, if necessary, a new visibility event into $\mathcal{Q}$.[9] Finally, we replace the edge $x\vec{u}$ in $\mathcal{L}(x)$ by

[8] Recall that a Voronoi arc equidistant to two polygon edges is always monotone with respect to the clearance and can never contain a chain minimum in its interior.

[9] Note that even though $\mathcal{L}(p(\chi_a))$ is represented as a circular list, the direction opposite to the direction of $p(\chi_a)$'s move actually divides it to a regular list. We note that a tangency event always results in the insertion of a new edge at one of the list ends, so only one true adjacency is created.

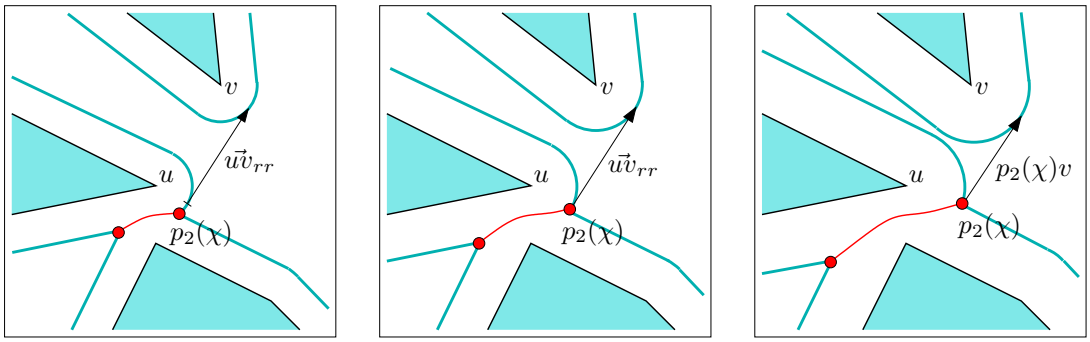


Figure 7: A tangency event: (a) The chain point $p_2(\chi)$, whose creation is depicted in Figure 6, lies on the supporting circle of the dilated vertex u. (b) The visibility edge $\vec{uv}_{rr}$ becomes tangent to $B_c(u)$ exactly at $p_2(\chi)$, so a tangency event occurs. (c) The reincarnated visibility edge $p_2(\vec{\chi})v$ replaces $\vec{uv}_{rr}$ as c grows. Note that this edge is not tangent to $B_c(u)$ any more.

$\vec{xp(\chi_a)}$, recompute the critical c-values of the visibility events of this edge with its neighbors (notice that the slope of $\vec{xp(a)}$ 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 in $\mathcal{Q}$ 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 consider the following cases here:

- The endpoint z is incident only to two Voronoi arcs a and a' belonging to the same chain (i.e., $\chi_a = \chi_{a'}$). 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 arcs become a different function of c from now on). We also have to handle the opposite edges, as we did in the tangency-event procedure. Moreover, if there are tangency events associated with the opposite edges we should modify them as well.

  As the chain point $p(\chi_a)$ now moves on the Voronoi arc a', we have to take care of tangency events that occur in the range of this new arc. Thus, if one of the polygon features associated with a' is a vertex u, we 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, we insert the appropriate tangency event into the event queue.[10] In case a' is a vertex–vertex arc, 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 events associated with it. In non-degenerate cases, the edge lists of all chain points coinciding with z are already empty. Only in degenerate cases, chain points

[10]Note that edges that had a tangency event in the range of the previous Voronoi arc a have already been deleted from the incident-edge list of the vertex at the moment this endpoint event occurs.

involved in an endpoint-event at z may still have incident edges, and in this case we should just assign a maximal validity value to these edges and empty the edge lists associated with these chain points.

- Otherwise, z is the endpoint 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 simultaneous endpoint event, and a single monotone chain $\hat{\chi}$ beginning at z (see for example the left Voronoi vertex of the marked chain in Figure 2). We therefore 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 and modify any tangency events they are involved in.

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 Querying the VV-Complex

The result of the preprocessing stage is the VV-complex $\langle \mathcal{V}, \mathcal{T} \rangle$, where:

- $\mathcal{V}$ 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 $c_{\min}(\chi)$.
- $\mathcal{T}$ is a set of interval trees: For each obstacle vertex u, $\mathcal{T}_u \in \mathcal{T}$ contains the edges incident to u and their validity ranges (namely the intervals are the c-ranges). For each Voronoi chain χ, $\mathcal{T}_{\chi,i} \in \mathcal{T}$ 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 implicitly constructed $\mathrm{VV}^{(\hat{c})}$-diagram using a Dijkstra-like search to find the "shortest" path between s and g.

Note that when we reach a vertex x (a dilated polygon vertex or a chain point) during the Dijkstra search 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 $x \in \mathcal{L}(g)$ (that is, if the goal is visible from x). If x is an obstacle vertex, we should

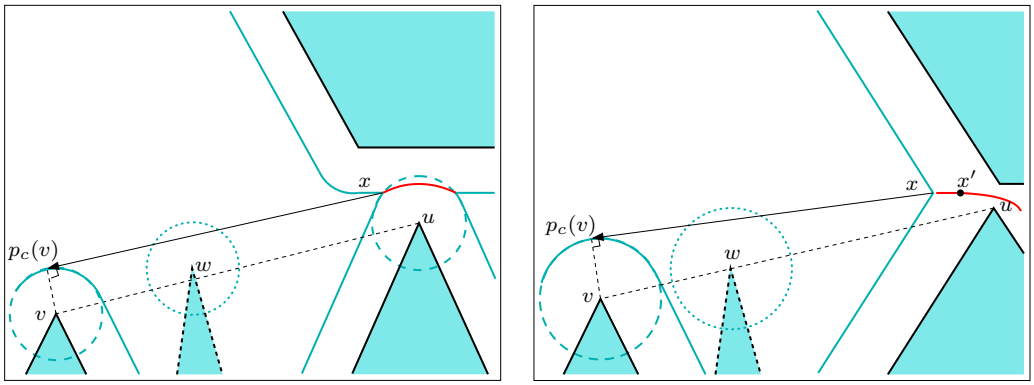


Figure 8: Visible dilated vertices from a chain point x. (a) If a vertex v is visible from a vertex–edge Voronoi arc, it must be also visible from the vertex u inducing this arc, and cannot be blocked by the dashed polygon. (b) In case of an edge–edge Voronoi arc, we consider the vertex u, which lies closest to the arc endpoint with the minimum clearance x', as the "inducing vertex".

keep in mind to add the length of the portion of the corresponding circular arc to the distance.[11] We proceed until the goal configuration g is reached.

The way we select the weights associated with 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 (see the discussion in Section 3 about the weight we give the graph edges). Note that as the graph edges are implicitly represented, we have to dynamically compute their associated weights, but this can be done in $O(1)$ time per edge and does not incur a significant computational load.

4.3 Proof of Correctness

We begin by stating a lemma that proves that the manner that we move visibility events from dilated vertices to chain points when handling tangency events is indeed correct — i.e., that a chain point cannot start "seeing" an object (a dilated vertex or another chain point) all of a sudden, unless this object is visible from one of the vertices inducing the Voronoi arcs along the chain.

Lemma 1 *If a dilated obstacle vertex $B_c(v)$ is visible from a chain point on a Voronoi arc, then the original vertex v is visible from the vertices inducing this arc. In case of an edge–edge Voronoi arc, we consider the arc endpoint with the minimum clearance value, and refer to the obstacle vertex that lies closest to this point as the "inducing vertex".*

Proof: Consider the example depicted in Figure 8(a), where the dilated vertex $B_c(v)$ is visible from the chain point x, which lies on a vertex–edge Voronoi arc. Let u be the obstacle vertex inducing this arc. If u and v are not mutually visible, then there must exist some polygon blocking

[11] In some cases we will have fictitious visibility edges of length 0, for example when we have a chain point y that lies on a vertex–vertex or a vertex–edge Voronoi edge (see Figure 7(a) for an illustration). In this case, y is connected to the polygon vertices that induce this Voronoi edge with visibility edges of distance 0, and when we examine a path through the relevant Voronoi edge and involving a visibility edge incident to one the vertices inducing y, we should only consider the length of the circular arcs between y and the endpoint of the visibility edge.

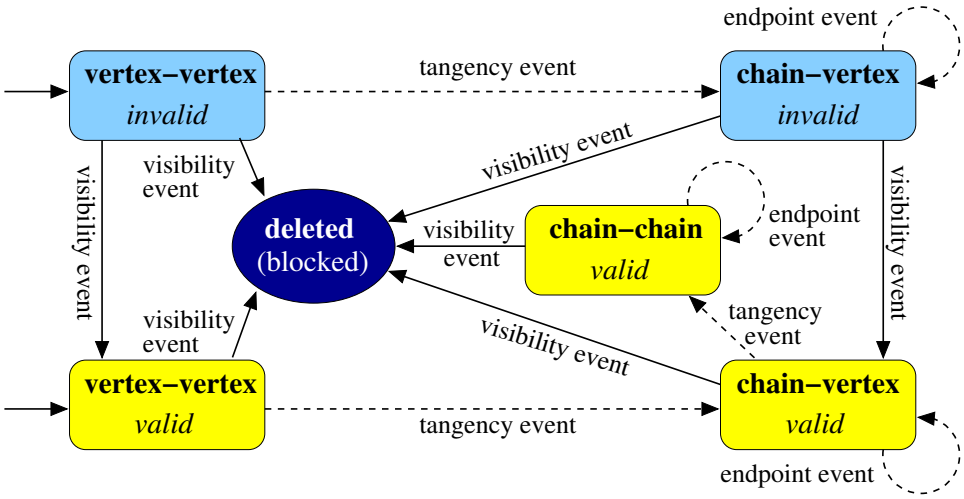


Figure 9: The schematic "life-cycle" of a visibility edge during the execution of the preprocessing stage. The rounded-corner rectangles denote possible visibility edges by the type of their endpoints. The solid arrows denote a change in the validity of the edge while the dashed arrows denote a reincarnation of the edge.

the straight line segment uv — let w be an extreme vertex of this polygon. Let $p_c(v)$ be the tangency point of the visibility edge emanating from x toward $B_c(v)$. It is clear that the distance of v from the line supporting $(x, p_c(v))$ is exactly c, but the distance of u from this line is *less* than c, as it cannot be tangent to $B_c(u)$ and penetrates the interior of this circle. We conclude that the distance of w from this line must also be less than c, thus $B_c(w)$ blocks the visibility of $B_c(v)$ from the chain point x. We have reached a contradiction, so we conclude that the original vertices u and v are mutually visible.

The same arguments hold for a chain point located on a vertex–vertex Voronoi arc, and we conclude that v is visible from *both* vertices inducing the arc. The case of a chain point which lies on an edge–edge Voronoi arc is depicted in Figure 8(b): Once again, if w blocks the visibility edge of v and x, as the distance of u from the supporting line of $(x, p_c(v))$ must be less than c (notice that also in this case this line intersects the interior of $B_c(u)$), the distance of w from this line is also less than c. Again, we have reached a contradiction, as $B_c(w)$ blocks the segment $(x, p_c(v))$. □

Theorem 2 *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 will not become valid again for a higher c-value.*

Proof: As we see in Figure 9, which describes the schematic "life-cycle" of a visibility edge, 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. Note that 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.[12] Here we show that once an edge becomes blocked, it does not become unblocked again for a higher c-value.

[12] When presenting the algorithm we created a new validity range for reincarnated visibility edges instead of treating the validity ranges as a single continuum, as we do in this theorem. This representation simplifies the algorithm without incurring any asymptotic run-time penalty. Our theorem is therefore slightly stronger than what we need for proving the correctness of our algorithm.

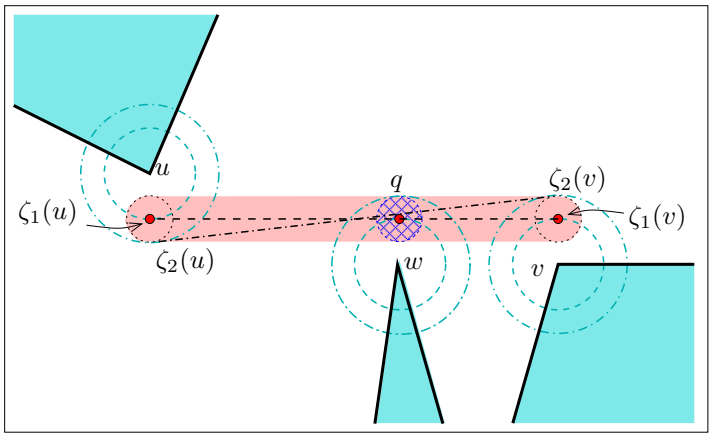


Figure 10: The visibility edges $\vec{uv}_{rl}$ and $\vec{vu}_{rl}$, realized as the segment $(\zeta_1(u), \zeta_1(v))$ (the dashed black line), are blocked at q by the dilated vertex $B_{c_1}(w)$. For $c_2 > c_1$, $(\zeta_2(u), \zeta_2(v))$ (the dash-dotted line segment) is contained in the region $(\zeta_1(u), \zeta_1(v)) \oplus B_{c_2-c_2}$ (lightly shaded), which is divided into two by the disc $B_{c_2-c_2}(q)$.

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 for some clearance value $c_1 > 0$. Let $\zeta_1(u)$ and $\zeta_1(v)$ be the two endpoints of this edge, lying on $B_{c_1}(u)$ and $B_{c_1}(v)$, respectively. As illustrated in Figure 10, if $c_2 > c_1$, the edge $(\zeta_2(u), \zeta_2(v))$ between u and v for clearance c_2 is contained in the Minkowski sum $(\zeta_1(u), \zeta_1(v)) \oplus B_{c_2-c_1}$, as the distance of both $\zeta_2(u)$ and $\zeta_2(v)$ from the line segment $(\zeta_1(u), \zeta_1(v))$ is clearly less than $c_2 - c_1$.

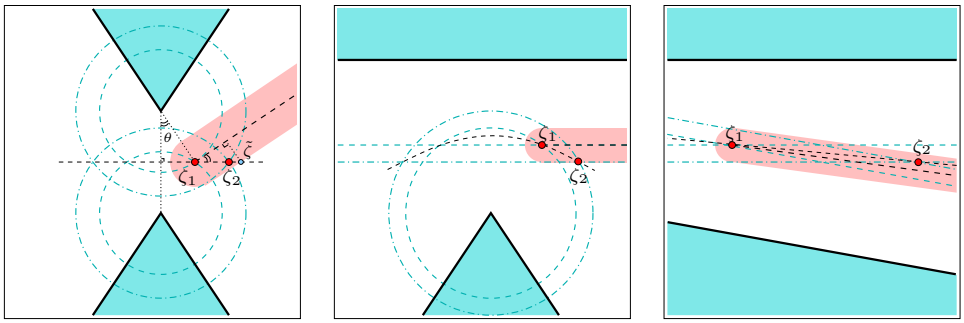
Let us assume that for the clearance value c' the visibility edge $\vec{uv}$ becomes blocked by a dilated obstacle vertex w, which touches $(\zeta'(u), \zeta'(v))$ at some point q — then for each $c'' > c'$ the disc $B_{c''-c'}(q)$ of radius $c'' - c'$ centered at q is fully contained in a dilated obstacle, and no visibility edges can cross it. It is clear that this disc divides the region $(\zeta'(u), \zeta'(v)) \oplus B_{c''-c'}$ into two, making it impossible for the edge $(\zeta''(u), \zeta''(v))$ to be valid.

It is therefore clear that once a visibility edge between two dilated vertices becomes blocked, it can never become unblocked again.[13] Moreover, similar arguments apply if one of the endpoints of the visibility edge (or both its endpoints) is a chain point lying on a Voronoi arc. We begin by showing that the chain point for the clearance value c'' lies inside the cigar-shaped region obtained by taking the Minkowski sum of the original visibility edge with $B_{c''-c'}$:

- The endpoint ζ_1 of a visibility edge for a clearance value c_1 lies on a vertex–vertex Voronoi arc (see Figure 11(a) for an illustration). Without loss of generality, let us assume that the two vertices u and v inducing this Voronoi arc are located at $(0, -\delta)$ and $(0, \delta)$, where $2\delta < c_1$ is the distance between the vertices. In this case the Voronoi arc is supported by the line $y = 0$ and the two chain points for c_i $(i = 1, 2)$ are given by $\zeta_i = (\sqrt{c_i^2 - \delta^2}, 0)$.

  Let us consider the extremal case where the visibility edge is tangent to $B_{c_1}(0, \delta)$ — that is, it is tangent to one of the dilated obstacles and if its slope is increased by $\varepsilon > 0$ it will penetrate this dilated obstacle and become blocked. In this case, the lower part of the "cigar" intersects

[13] In this case, there is also a simple algebraic proof for this fact: The bitangent to $B_{c'}(u)$ and $B_{c'}(v)$ is also tangent to $B_{c'}(w)$ only when c' equals half the distance between u and the line connecting v and w (see Appendix A for the details). For the edge to become unblocked at some $c'' > c'$, the three circles $B_{c''}(u)$, $B_{c''}(v)$ and $B_{c''}(w)$ must have another common tangent, but this is of course impossible.



(a) (b) (c)

Figure 11: The chain points ζ_1 and ζ_2, at clearance values c_1 and c_2, respectively ($c_2 > c_1$). The relevant Voronoi arcs are drawn as thin dashed lines, were the light dashed (dash-dotted) segments and circles correspond to clearance c_1 (c_2, respectively) from the obstacle features inducing these arcs. The visibility edges emanating from ζ_1 are drawn in a thick dashed line, with the Minkowski sum of the edge with $B_{c_2-c_1}$ is lightly shaded. (a) An extreme case were the visibility edge from a chain point lying on a vertex–vertex arc is tangent to one of the dilated obstacles. (b) Another extreme case were the visibility edge from a chain point lying on a vertex–edge arc is parallel to the edge. (c) The case of chain points lying on an edge–edge arc.

$y = 0$ at $\tilde{\zeta}$, where:

$$x_{\tilde{\zeta}} = x_{\zeta_1} + \frac{c_2 - c_1}{\sin\theta} = \sqrt{c_1^2 - \delta^2} + \frac{c_1(c_2 - c_1)}{\sqrt{c_1^2 - \delta^2}} = \frac{c_1 c_2 - \delta^2}{\sqrt{c_1^2 - \delta^2}}$$

It is straightforward to show that $x_{\tilde{\zeta}} > x_{\zeta_2}$, hence ζ_2 is contained in the "cigar".

- The endpoint ζ_1 lies on a vertex–edge Voronoi arc. Without loss of generality, we assume that the obstacle edge inducing the arc is supported by the line $y = \delta$ and the obstacle vertex is given by $(0, -\delta)$ (again, we have $2\delta < c_1$). It is clear that the slope of a visibility edge emanating from ζ_1 is non-positive. In the extremal case, depicted in Figure 11(b), it is a horizontal segment, and since $|y_{\zeta_2} - y_{\zeta_1}| = c_2 - c_1$ then ζ_2 is located on the boundary of the cigar-shaped region around the horizontal visibility edge. It is also clear that in other cases, when the slope of the original visibility is negative, then ζ_2 is located in the interior of the "cigar" around this edge.

- The same arguments also apply if ζ_1 and ζ_2 lie on an edge–edge Voronoi arc. Note that in this case we should consider the slopes of both obstacle edge involved: indeed, $\|\zeta_1 - \zeta_2\|$ may be significantly larger that $c_2 - c_1$, as shown in Figure 11(c), but since the slope of the visibility edge is bounded by the slope of the obstacle edges, it follows that ζ_2 must be contained in the "cigar".

We have showed that a visibility edge $\bar{e}_2$ for c_2 is always contained in the cigar-shaped region, which is the Minkowski sum of the visibility edge $\bar{e}_1$ for $c_1 < c_2$ with $B_{c_2-c_1}$. According to our assumption, $\bar{e}_1$ is blocked at some point q, so $\bar{e}_1 \oplus B_{c_2-c_1}$ is divided into two by the disc $B_{c_2-c_2}(q)$. We argue that each part contains exactly one endpoint of $\bar{e}_2$, which can be easily verified by examining the various cases in Figure 11. If this is not the case, then q must lie between ζ_1 and the projection of ζ_2 onto $\bar{e}_1$ — this is of course impossible, as it implies that there exists another obstacle on the way, other than the ones defining the Voronoi arc. As a consequence, the visibility edge $\bar{e}_2$ must also be blocked.

We conclude that once a visibility edge has been blocked, it will never become valid again. Note that what we have shown so far is that we can associate a single validity range with a visibility edge one of whose endpoints lie on a Voronoi *arc*, while our edges are actually associated with chain points that move along Voronoi *chains*. However, note that when a chain point is created, there are no visibility edges associated with it. By Lemma 1, visibility edges can be associated with a chain point only when it is involved in tangency events, as it traverses a vertex–vertex or a vertex–edge Voronoi arc, and it cannot "see" any object not visible from the relevant vertex. As the chain point moves along the chain, these visibility edges are eventually blocked (note that the chain point can never move from an edge–edge arc to another edge–edge arc, as there should always be a vertex on the way). We conclude that the association of a single validity range with each visibility edge (and with its reincarnates) is indeed correct. □

4.4 Complexity Analysis

Theorem 3 *Constructing the* VV*-complex 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 per visibility edge, of which there are $O(n^2)$ in total, 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.

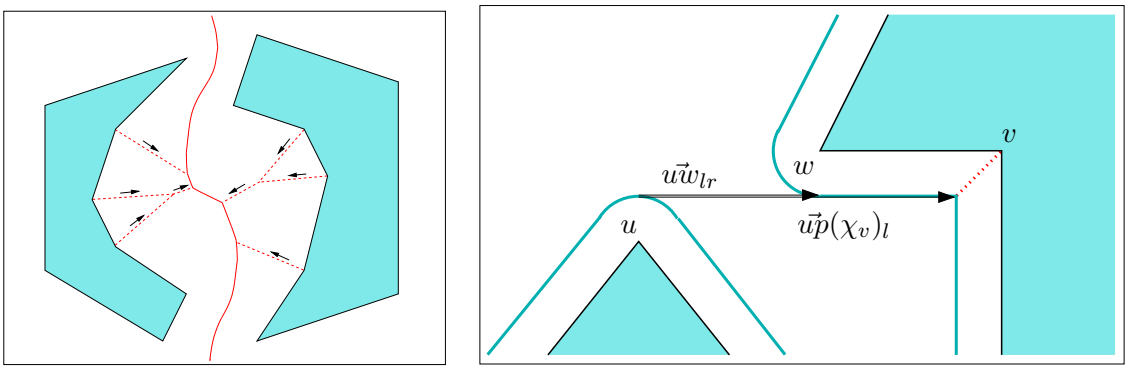
As the preprocessing algorithm proceeds, it starts handling events: In total, Theorem 2 implies that we have $O(n^2)$ visibility events:[14] Every vertex can be involved at most once in a visibility event with another vertex, where a visibility edge between the two vertices (or their dilated version) is created. Each of the visibility events 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.[15] □

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

[14] 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 will be eventually discarded.

[15] It is in fact possible to construct the visibility graph of the input polygons in $O(n \log n + k)$ time, where k is the number of visibility edges in this graph (valid and invalid ones), construct the initial edge lists in $O(k \log n)$ time and then charge each of the $O(k)$ directed visibility edges with $O(\log n)$ operations, to account for all visibility events, chain events and tangency events. Unfortunately, the entire preprocessing stage cannot be completed in $O(k \log n)$ time, as there are cases where $\Theta(n^2 \log n)$ operations are needed to handle the endpoint events.



(a) (b)

Figure 12: (a) A portion of the Voronoi diagram of two non-convex polygons. The Voronoi chain separating the two obstacles is drawn with a solid line, while the Voronoi chains induced by features of the same polygon are drawn with a dashed line. (b) The edge $\vec{uw}_{lr}$ becomes valid after being involved in a visibility event with a visibility edge to the chain point $p(\chi_v)$ that is associated with the reflex obstacle vertex v.

the number of edges encountered during the search (recall that k is the number of visibility edges). 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 $\mathrm{VV}^{(c)}$-diagram for a fixed c-value may be constructed in $O(n \log n + k)$ time, so it may seem we do not need any preprocessing stage, and it is better to construct the $\mathrm{VV}^{(c)}$-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.[16]

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 (see Figure 12(b) for an illustration).

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 (see Figure 12(a)). To this 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

[16] 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.

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 the input obstacles. We should mention that when we query the VV-complex we do not compute the chain points along Voronoi chains induced by reflex vertices, and therefore do not account for these "reflex" chains, as these chains lead to a dead-end (a reflex vertex) and can never be used for making shortcuts in the motion path.

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 [13] 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 [15] for computing Voronoi diagrams of line segments: We simply add a label to each segment (polygon edge) and each segment endpoint (polygon vertex) that identifies the source polygon and the feature index within this polygon. We then can conveniently disregard Voronoi chains induced by features of the same polygon.[17] We note that the Minkowski-sum computation is carried out by decomposing each non-convex obstacle to convex polygons, dilating them by the preferred clearance value and computing the union of the sums, so it is very convenient to compute the diagram of these convex sub-polygons as well, instead of using the non-convex input obstacle. In this case we label each polygon feature with both the convex polygon and the input (non-convex) polygonal obstacle from which it originated. This labeling helps us to determine which Voronoi arcs should be ignored.

The intersection among the dilated obstacles and between the boundary of the union of the dilated obstacles and the Voronoi arcs is robustly computed using the conic-arc traits [26] of CGAL's arrangement package [9]. 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. The endpoints of the line segments, the circular arcs and the parabolic arcs that form our arrangement are in general algebraic numbers of degree 4.

In the rest of this section we give a constructive proof of a lemma that enables us to robustly construct the skeleton of the $\mathrm{VV}^{(c)}$-diagram for rational inputs, based on robust computations with the conic-arc arrangement traits:

Lemma 4 *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.*

[17] The complete Voronoi diagram of the polygon edges contains also the *medial axis* of each polygon, which is redundant in our case.

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):

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

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 will be 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:

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

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 $\ell^{\perp}$:

$$z_{\min}=\frac{1}{2}\left(x_u+\frac{B^2x_u-A(By_u+C)}{A^2+B^2},y_u+\frac{A^2y_u-B(Ax_u+C)}{A^2+B^2}\right)\ . \tag{3}$$

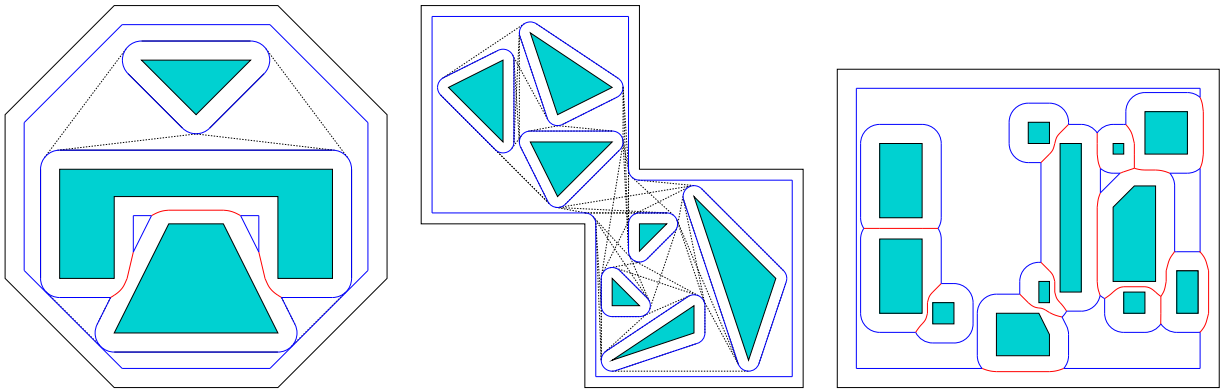
The minimal clearance value, obtained at $z_{\min}$ is half the distance between u and the line ℓ.

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 bisecting the angle formed between ℓ_1 and ℓ_2, but in general this line cannot be represented as a linear curve with rational coefficients.[18] Instead, we represent the edge as a segment of a pair of perpendicular lines (naturally, only one line in this pair supports the relevant segment), which form the two angle bisectors of ℓ_1 and ℓ_2:

$$\frac{(A_1x+B_1y+C_1)^2}{A_1^2+B_1^2}=\frac{(A_2x+B_2y+C_2)^2}{A_2^2+B_2^2}\ . \tag{4}$$

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.

[18] For example, if $\ell_1:\ y=0$ and $\ell_2:\ y=x$, the slope of the line bisecting the angle between ℓ_1 and ℓ_2 is $\tan 22.5^\circ=\frac{1}{1+\sqrt{2}}$, and this line ($y=\frac{1}{1+\sqrt{2}}x$) cannot be represented using rational coefficients. Note however that the perpendicular line $y=\frac{1}{1-\sqrt{2}}$ is also an angle bisector in this case.



(a) (b) (c)

Figure 13: The $VV^{(c)}$-diagrams constructed for several input files and c-values: (a) `octagon` with $c = \frac{7}{10}$, (b) `two_rooms` with $c = \frac{2}{5}$, and (c) `rectangles` with $c = \frac{9}{10}$ (visibility edges are not shown in this case).

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:

$$(x - x_u)^2 + (y - y_u)^2 = c^2 \ . \tag{5}$$

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, in general it is impossible to represent a dilated edge as a linear curve with rational coefficients.[19] Instead, we treat 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:

$$\frac{(Ax + By + C)^2}{A^2 + B^2} = c^2 \ . \tag{6}$$

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.

6 Experimental Results

Our software is implemented using CGAL 3.1, relying on the exact number types supplied by CORE 1.7 [2]. In particular, the `CORE::Expr` number-type class is capable of performing exact computations with polynomial roots. 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 3 (the `four_shapes` scene) 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

[19] For example, if we seek a line lying at a distance 1 from $\ell : \ y = x$, we find the line $y = x + \sqrt{2}$, that cannot be represented using rational coefficients. However, the line $y = x - \sqrt{2}$ is also parallel to ℓ and lies at a distance 1 from it.

the same scene, such that no chain points appear in the diagram, the construction time drops to 2.5 seconds (see Table 1). In more involved scenes, the construction of the diagram may take 15–20 seconds (see Figure 13 and Table 1).

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, so that the average query time is only a few milliseconds.

Table 1: The construction time of the $VV^{(c)}$-diagram for several input scenes and different c-values.

Input file	Bounding-box dimensions	c	Construction time (sec.)	Average query time (sec.)
`four_shapes`	10×7	$^1/_5$	2.3	0.01
`four_shapes`	10×7	$^2/_5$	9.7	0.01
`octagon`	14×14	$^3/_{10}$	4.9	0.01
`octagon`	14×14	$^7/_{10}$	15.2	0.01
`two_rooms`	14×14	$^2/_5$	2.8	0.02
`rectangles`	18×15	$^9/_{10}$	15.4	0.02

We also used the $VV^{(c)}$-diagram to generate convincing group motions in a more complex scene, as the one depicted in Figure 14. 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.

7 Conclusions and Future Work

We introduced a simple, yet powerful, data structure — the $VV^{(c)}$-diagram — which contains all shortest paths for a robot in a planar environment of configuration-space obstacles, given a preferred clearance value and that allows for a trade-off between path length and clearance in the presence of narrow passages. We have implemented a robust software package that maintains this data structure and used it to plan natural-looking paths for coherent groups of moving entities in the plane. Our method, which requires some preprocessing for constructing the diagram, but can answer queries very efficiently, without the need for smoothing or additional post-processing, is especially suitable to real-time applications, such as computer games.

We have also introduced the VV-complex, a data structure that efficiently encodes all $VV^{(c)}$-diagrams for all possible clearance values. We show how to efficiently construct the VV-complex for a given set of obstacles and how to query it given a start and goal configurations and a preferred clearance value.

So far we have used rather simplistic weighting schemes for our diagram edges (see Section 3). In the future we plan to investigate more sophisticated weighting schemes that are more suited to the applications we have. In particular, we plan to use fluid-mechanics techniques in order to estimate the time it takes a group of moving entities to traverse a Voronoi edge that is located inside a narrow passage.

We also plan to investigate generalizations of our constructions for motion-planning problems with more degrees of freedom. The most important task in this category is planning natural-looking

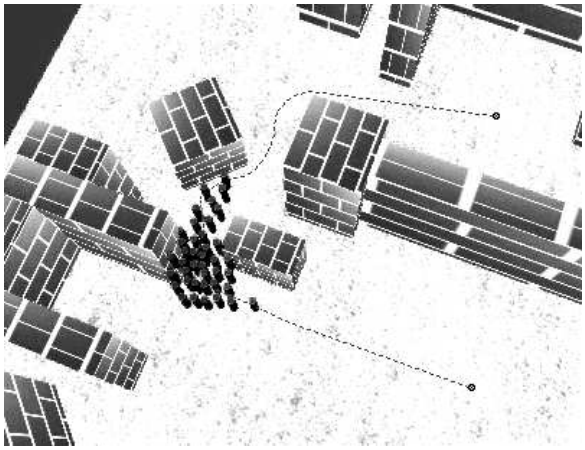


Figure 14: A group of 40 entities moving in a virtual scene along a backbone path, drawn with a dashed line. (Courtesy of Arno Kamphuis.)

paths for a polygonal robot translating and rotating in the plane.

Acknowledgements

We thank Arno Kamphuis for providing us with screen shots from his coherent group-motion demo (Figure 14).

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, J. Matoušek, and O. Schwarzkopf. Piecewise linear paths among convex obstacles. *Discrete and Computational Geometry*, 14:9–29, 1995.

[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 2nd edition, 2000.

[8] 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.

[9] 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.

[10] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 23, pages 513–528. Chapman & Hall/CRC, 2nd edition, 2004.

[11] R. Geraerts and M. H. Overmars. Clearance based path optimization for motion planning. In *IEEE Intern. Conf. Robotics and Automation*, pages 2386–2392, 2004.

[12] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. on Computing*, 20(5):888–910, 1991.

[13] 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.

[14] 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.

[15] M. I. Karavelas. Segment Voronoi diagrams in CGAL, 2004. `http://www.cgal.org/UserWorkshop/2004/svd.pdf`.

[16] 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.

[17] 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.

[18] D.-T. Lee and R. L. Drysdale III. Generalization of Voronoi diagrams in the plane. *SIAM J. on Computing*, 10(1):73–87, 1981.

[19] 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.

[20] 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. Chapman & Hall/CRC, 2nd edition, 2004.

[21] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE Intern. Conf. Robotics and Automation*, pages 446–452, 2004.

[22] C. Ó'Dúnlaing, M. Sharir, and C. K. Yap. Retraction: A new approach to motion-planning. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 207–220, 1983.

[23] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1985.

[24] M. Pocchiola and G. Vegter. The visibility complex. *International J. of Computational Geometry and Applications*, 6(3):279–308, 1996.

[25] H. Rohnert. Moving a disc between polygons. *Algorithmica*, 6:182–191, 1991.

[26] R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. 10th Europ. Sympos. Algorithms*, pages 884–895. Springer-Verlag, 2002.

Appendix

A Detecting Visibility Events

Let u and v be two convex obstacle vertices that are mutually visible (that is, the line segment uv does not intersect the interior of any obstacle). We denote by α_{uv} the angle between the vector $\vec{uv}$ an the x-axis. If $d(u,v)$ is the Euclidean distance between the two vertices, it is clear that:

$$\sin\alpha_{uv} = \frac{y_v - y_u}{d(u,v)} \ , \qquad \cos\alpha_{uv} = \frac{x_v - x_u}{d(u,v)} \ . \tag{7}$$

Let $\varphi_{uv}(c)$ be the angle that the bitangent $\vec{uv}_{rl}$ to the circles $B_c(u)$ and $B_c(v)$ forms with the vector $\vec{uv}$ (see Figure 4). We thus have (note that when $c > \frac{1}{2}d(u,v)$ the two circles intersect and therefore have no rl- or lr-bitangents):

$$\sin\varphi_{uv}(c) = \frac{2c}{d(u,v)} \ , \qquad \cos\varphi_{uv}(c) = \frac{\sqrt{d^2(u,v) - 4c^2}}{d(u,v)} \ . \tag{8}$$

The slope of this bitangent is therefore:

$$\begin{aligned} \tan(\alpha_{uv} + \varphi_{uv}(c)) &= \frac{\sin\alpha_{uv}\cos\varphi_{uv}(c) + \cos\alpha_{uv}\sin\varphi_{uv}(c)}{\cos\alpha_{uv}\cos\varphi_{uv}(c) - \sin\alpha_{uv}\sin\varphi_{uv}(c)} = \\ &= \frac{(y_v - y_u)\sqrt{d^2(u,v) - 4c^2} + 2(x_v - x_u)c}{(x_v - x_u)\sqrt{d^2(u,v) - 4c^2} - 2(y_v - y_u)c} \ . \end{aligned} \tag{9}$$

We mention that the slope of the bitangent $\vec{uv}_{lr}$ is $\tan(\alpha_{uv} - \varphi_{uv}(c))$ and is also an expression of the same form.

Let us examine the three vertices u, v and w and determine the critical clearance values c for which the slope of one of the bitangents of u and v becomes equal to a slope of one of the bitangents of u and w. As a "right" bitangent can never be equally sloped with a "left" bitangent, we should examine the following cases for the right bitangents of uv (the treatment of the left bitangents is symmetrical):

1. The bitangent $\vec{uv}_{rr}$ becomes equally sloped with the bitangent $\vec{uw}_{rr}$. This means that $\tan(\alpha_{uv}) = \tan(\alpha_{uw})$, thus the three vertices u, v and w must be collinear.

2. The bitangent $\vec{uv}_{rl}$ becomes equally sloped with the bitangent $\vec{uw}_{rr}$. Hence:

$$\begin{aligned} \tan(\alpha_{uv} + \varphi_{uv}(c)) &= \tan(\alpha_{uw}) \\ \frac{(y_v - y_u)\sqrt{d^2(u,v) - 4c^2} + 2(x_v - x_u)c}{(x_v - x_u)\sqrt{d^2(u,v) - 4c^2} - 2(y_v - y_u)c} &= \frac{y_w - y_u}{x_w - x_u} \\ \underbrace{\left((y_v - y_u)(x_w - x_u) - (x_v - x_u)(y_w - y_u)\right)}_{\Phi_{uvw}} \cdot \sqrt{d^2(u,v) - 4c^2} &= \\ -2\underbrace{\left((x_v - x_u)(x_w - x_u) + (y_v - y_u)(y_w - y_u)\right)}_{\Psi_{uvw}} \cdot c \ . \end{aligned}$$

Squaring the equation above we get (note that $\Psi_{uvw} = \vec{uv} \cdot \vec{uw}$ and $\Phi_{uvw} = \vec{uv}^{\perp} \cdot \vec{uw}$, and it is easy to show that $\Phi^2_{uvw} + \Psi^2_{uvw} = d^2(u,v)d^2(u,w)$):

$$\begin{aligned} \Phi^2_{uvw}\left(d^2(u,v) - 4c^2\right) &= 4\Psi^2_{uvw} \cdot c^2 \\ c^2 &= \frac{\Phi^2_{uvw} d^2(u,v)}{4(\Phi^2_{uvw} + \Psi^2_{uvw})} = \frac{\Phi^2_{uvw} d^2(u,v)}{4(d^2(u,v)d^2(u,w))} = \frac{\Phi^2_{uvw}}{4d^2(u,w)} \\ c &= \frac{(y_v - y_u)(x_w - x_u) - (x_v - x_u)(y_w - y_u)}{2\sqrt{(x_w - x_u)^2 + (y_w - y_u)^2}} \,. \end{aligned} \tag{10}$$

We note that the geometric interpretation to the equation above is that c should be equal to half the distance between w and the straight line connecting u and v.

3. The bitangent uv_{rr} becomes equally sloped with the bitangent $\vec{uw}_{rl}$. In this case $\tan(\alpha_{uv}) = \tan(\alpha_{uw} + \varphi_{uw}(c))$ and we can compute c in an analogous manner to the previous case.

4. The bitangent $\vec{uv}_{rl}$ becomes equally sloped with the bitangent $\vec{uw}_{rl}$. However, from w's point of view, this means that the bitangent $\vec{wu}_{rl}$ becomes equally sloped with the bitangent $\vec{wv}_{rr}$, so we can compute the critical c-value as we did for case 2.

We conclude that in order to compute the critical c-values for visibility events among dilated obstacle vertices, it is sufficient to perform one square-root operation. Moreover, if we assume that all our input vertices have rational coordinates, the critical c-values, given in Equation (10), only involve taking the square root of a rational number (thus c^2 is a rational number).

B Computing the Chain Points

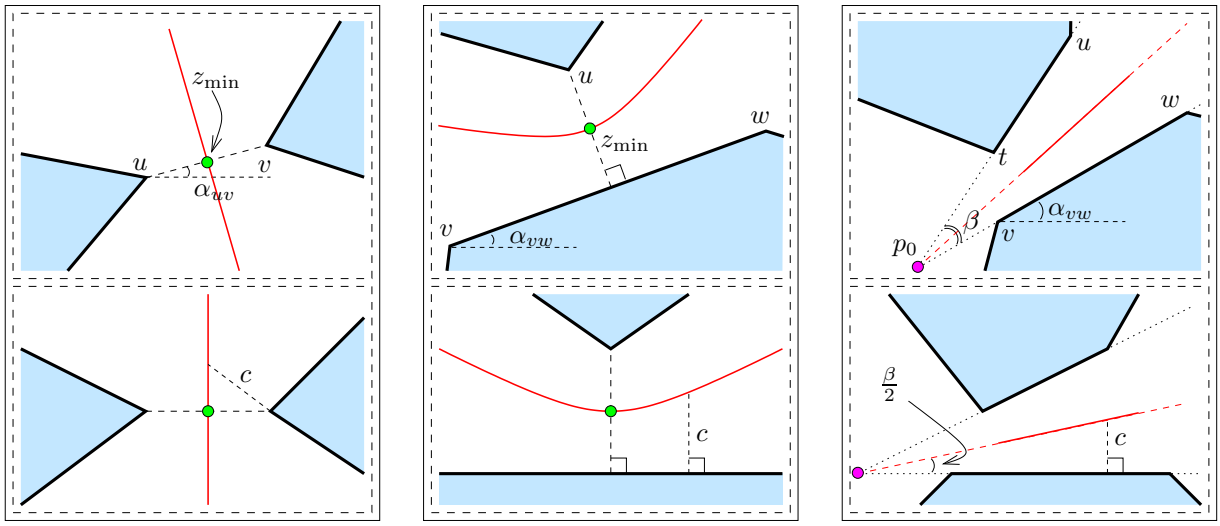
In this section we show how we can compute the chain points. That is, given a Voronoi arc and a clearance value c, we find a point z (or two points) on the arc whose clearance is exactly c. We obviously have to distinguish among the various types of Voronoi arcs:

Vertex–vertex arc: If the arc is defined by two polygon vertices u and v and $\frac{1}{2}d(u,v) \leq c$, the point we are looking for is $\sqrt{c^2 - \frac{1}{4}d^2(u,v)}$ away from the midpoint between u and v, denoted $z_{\min}$ (this is the point of minimum clearance along the arc). Assume that $y_u = y_v$, then there are two candidate points are given by $\left(\frac{1}{2}(x_u + x_v),\ y_u \pm \sqrt{c^2 - \frac{1}{4}d^2(u,v)}\right)$ — but as this is not usually the case, we can simply add the vector $\left(0,\ \pm\sqrt{c^2 - \frac{1}{4}d^2(u,v)}\right)$ rotated by α_{uv} (such that $\sin\alpha_{uv} = \frac{y_v - y_u}{d(u,v)}$ and $\cos\alpha_{uv} = \frac{x_v - x_u}{d(u,v)}$) to $z_{\min}$ (see Figure 15(a)), so we get:

$$\left(\frac{x_u + x_v}{2} \mp (y_v - y_u)\frac{\sqrt{c^2 - \frac{1}{4}d^2(u,v))}}{d(u,v)}\ ,\ \frac{y_u + y_v}{2} \pm (x_v - x_u)\frac{\sqrt{c^2 - \frac{1}{4}d^2(u,v))}}{d(u,v)}\right) .$$

Of course we need to check that each point is indeed between the two endpoints of the arc.

Vertex–edge arc: In case the arc is defined by the polygon vertex u and the polygon edge vw, let δ denote the distance of u from the line ℓ supporting vw (note that δ^2 is rational). Assume that ℓ is parallel to the x-axis and the apex of the parabola $z_{\min}$ (this is the point of minimum clearance along the parabolic arc) is located on the origin, so $\ell : y = -\frac{\delta}{2}$ and the arc is



(a) (b) (c)

Figure 15: Voronoi edges induced by: (a) two vertices u and v of two polygons; (b) a polygon vertex u and an edge vw of another polygon; and (c) two polygon edges tu and vw. The upper frames show the original scenario, while the lower frames show the same scenario transformed to a more convenient coordinate system (the circles mark the origin in each of the lower frames), for computing a point on the arc having a given amount of clearance c.

supported by the parabola $y = \frac{x^2}{2\delta}$ (see Figure 15(b) for an illustration). Note that given a point (x, y) on this parabola, its clearance is simply given by $y + \frac{\delta}{2}$, so we have to find x-values for which:

$$\frac{x^2}{2\delta} = c - \frac{\delta}{2} \ .$$

If $\frac{\delta}{2} \leq c$, the required points are $\left(\pm\sqrt{2\delta c - \delta^2},\ c - \frac{\delta}{2}\right)$, but in order to transform them to the original coordinate system it is necessary to rotate them by α_{vw} around the origin (where $\sin\alpha_{vw} = \frac{y_w - y_v}{d(v,w)}$ and $\cos\alpha_{vw} = \frac{x_w - x_v}{d(v,w)}$) and shift them by $z_{\min}$, so we get:

$$\left(\frac{\pm(x_w - x_v)\sqrt{2\delta c - \delta^2} - (y_w - y_v)(c - \frac{\delta}{2})}{d(v,w)}\ ,\ \frac{\pm(y_w - y_v)\sqrt{2\delta c - \delta^2} + (x_w - x_v)(c - \frac{\delta}{2})}{d(v,w)}\right) \ .$$

Edge–edge arc: Assume that the arc is defined by two polygon edges tu and vw, and let z_1 and z_2 be its endpoints. As such an arc is always monotone, we can find a point with a clearance c on it if and only if $c \in [c(z_1), c(z_2)]$ (where $c(z_1)$ and $c(z_2)$ are the clearance values at the endpoints). Let $p_0 = (x_0, y_0)$ be the intersection point of the two supporting lines of these edges.[20] Assume, without loss of generality, that u and w are the two vertices further away from p_0 (that is, $d(t, p_0) < d(u, p_0)$ and $d(v, p_0) < d(w, p_0)$). If β is the angle between the lines supporting tu and uv, we can apply the Cosine Theorem on the triangle $\triangle up_0w$ and obtain:

$$\cos\beta = \frac{d^2(u,p_0) + d^2(w,p_0) - d^2(u,w)}{2d(u,p_0)d(w,p_0)} \ , \tag{11}$$

[20] We can ignore the degenerate case when the two lines are *parallel*, as the clearance along such an arc is constant and equals half the distance between the lines.

and thus:

$$\begin{aligned}\sin\beta &= \sqrt{1-\cos^2\beta} = \\ &= \frac{\sqrt{\left(2d^2(u,p_0)+2d^2(w,p_0)-d^2(u,w)\right)d^2(u,w)-\left(d^2(u,p_0)+d^2(w,p_0)\right)^2}}{2d(u,p_0)d(w,p_0)} \ . \end{aligned} \quad (12)$$

Let us transform our coordinate system such that vw lies on the x-axis and p_0 is the origin (see Figure 15(c)). The Voronoi arc corresponding to the two edges is now supported by a line that crosses the origin and forms an angle $\frac{\beta}{2}$ with the x-axis. Using the Half-Angle Formula together with Equations (11) and (12) we get:

$$\begin{aligned}\cot\frac{\beta}{2} &= \frac{1+\cos\beta}{\sin\beta} = \\ &= \frac{\left(d(u,p_0)+d(w,p_0)\right)^2-d^2(u,w)}{\sqrt{\left(2d^2(u,p_0)+2d^2(w,p_0)-d^2(u,w)\right)d^2(u,w)-\left(d^2(u,p_0)+d^2(w,p_0)\right)^2}} \ . \end{aligned} \quad (13)$$

Under the new coordinate system, it is clear that there is a single point with clearance c on the arc, given by $(c\cdot\cot\frac{\beta}{2}, c)$. We only have to transform this point to the original coordinate system, by rotating it by α_{vw} and shifting by p_0 to obtain:

$$\left(x_0+\frac{(x_w-x_v)c\cdot\cot\frac{\beta}{2}-(y_w-y_v)c}{d(v,w)}\ ,\ y_0+\frac{(y_w-y_v)c\cdot\cot\frac{\beta}{2}+(x_w-x_v)c}{d(v,w)}\right) \ .$$

C Computing Tangent Slopes

Consider the two tangents to the circle $B_c(v)$ (which represents the obstacle vertex v dilated by c) emanating from a point $p=(x_0,y_0)$. In order to compute the slopes of these tangent, we have to compute the tangency points q_1 and q_2.

Note that if (x,y) is a tangency point to $B_c(v)$, we can write the following system of equations:

$$\begin{array}{ll} \text{I} & (x-x_v)^2+(y-y_v)^2=c^2 \\ \text{II} & (x-x_0)(x-x_v)+(y-y_0)(y-y_v)=0 \end{array} \ . \quad (14)$$

The first equation expresses the fact that q_i lies on $B_c(v)$, while the second condition is that $\angle pq_iv$ is a right angle (that is, $pq\perp vq$, so $\vec{pq}\cdot\vec{vq}=0$). By subtracting the two equations we get:

$$(x_0-x_v)(x-x_v)+(y_0-y_v)(y-y_v)=c^2 \ . \quad (15)$$

Thus we can plug the expression for $(y-y_v)$ into Equation (14-I) and obtain a quadratic equation in $(x-x_v)$ (similarly, if we plug the expression for $(x-x_v)$ we obtain a quadratic equation in $(y-y_v)$). The two tangency points q_1 and q_2 emanating from p, are therefore given by:

$$\left(x_v+\frac{(x_0-x_v)c^2\pm\sqrt{d^2(v,p_0)-c^2}\cdot(y_0-y_v)c}{d^2(v,p_0)}\ ,\ y_v+\frac{(y_0-y_v)c^2\mp\sqrt{d^2(v,p_0)-c^2}\cdot(x_0-x_v)c}{d^2(v,p_0)}\right) \ .$$

As the two tangents are perpendicular to $q_1\vec{v}$ and $q_2\vec{v}$, respectively, their slopes are given by:

$$\alpha_i=-\frac{x_{q_i}-x_v}{y_{q_i}-y_v}=\frac{(x_0-x_v)c\pm\sqrt{d^2(v,p_0)-c^2}\cdot(y_0-y_v)}{(y_0-y_v)c\mp\sqrt{d^2(v,p_0)-c^2}\cdot(x_0-x_v)} \quad , \qquad i\in 1,2 \ . \quad (16)$$