The Visibility–Voronoi Complex and Its Applications

Ron Wein
Tel-Aviv University
wein@tau.ac.il

Jur P. van den Berg
Utrecht University
berg@cs.uu.nl

Dan Halperin
Tel-Aviv University
danha@tau.ac.il

Abstract

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

1 Introduction

In this report we study the problem of planning a natural-looking collision-free path for a robot with two degrees of motion freedom moving in the plane among polygonal obstacles. By "natural-looking" we mean that the robot should select a path that will be as close as possible to the path a human would take in the same scene to reach the goal configuration from the start configuration. This essentially means the following: (a) the path should be *short* — that is, it should not contain long detours when significantly shorter routes are possible; (b) it should have a guaranteed amount of *clearance* — that is, the distance of any point on the path from 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 the first 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 *visibility graph* is a well-known data structure for computing the shortest collision-free path between a start and a goal configuration (see, e.g., [6, Chapter 15]). However, shortest paths are in general tangent to obstacles, so a path computed from a visibility graph usually contains semi-free configurations (the robot is in contact with an obstacle, but their interiors do not intersect) and therefore does not have any clearance. This not only looks unnatural, it is also unacceptable for many motion-planning applications. On the other hand, planning motion paths using the

Voronoi diagram of the obstacles [20] yields a path with maximal clearance, but this path may be significantly longer than the shortest path possible, and may also contain sharp turns.

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

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

We implemented our algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram and applied it to the problem of motion planning for coherent groups of entities. The paths 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 discuss this application in detail in Section 3.2. 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 report.

The rest of this report 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 discuss the algebraic details of the computations needed for the two previous sections. We finally show some experimental results in Section 6.

2 Preliminaries

We next give a short review of the basic ingredients of the data structure we propose — visibility graphs, Voronoi diagrams and Minkowski sums — all basic and standard structures in computational geometry.

[1] Despite the similarity in names, our structure is not related to the *visibility complex* introduced in [21].

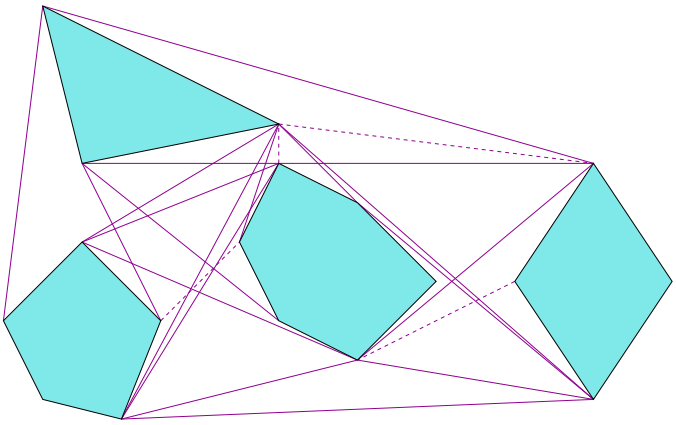


Figure 1: The visibility graph of a set of four convex polygons. The valid visibility edges are drawn with solid lines, while some invalid edges are also shown, drawn with dashed lines. Notice that all obstacle edges are also valid visibility edges.

2.1 Visibility Graphs

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of pairwise interior-disjoint polygons having n vertices in total, representing configuration-space obstacles. The *visibility graph* of $\mathcal{P}$ is an undirected graph whose set of vertices is the set of polygon vertices, and whose set of edges consists of those pairs of vertices that are mutually visible. Two vertices are *mutually visible* if the straight line segment connecting them does not intersect the *interior* of any of the polygons in $\mathcal{P}$ — in this case, we call this segment a *visibility edge*. 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.

The visibility graph can straightforwardly be computed in $O(n^2 \log n)$ time [16], by separately identifying the vertices visible from each vertex. Ghosh and Mount [10] were the first to give an output-sensitive algorithm for computing the visibility graph in optimal $O(n \log n + s)$ time, where s is the number of visibility edges in the output visibility graph. For more information on shortest paths, see [18].

It should be noted that it is not necessary to consider all visibility edges in order to compute shortest motion paths. It is sufficient to store only the edges that are bitangent to the polygons they connect: Assume that two mutually visible vertices u and v belong to the polygons $P(u)$ and $P(v)$, respectively — then the visibility edge is bitangent to both polygons only if its supporting line does not penetrate either $P(u)$ or $P(v)$. In this case we say that uv is a *valid* visibility edge (see Figure 1 for an illustration). Note that invalid visibility edges will never be part of a shortest path.

2.2 Voronoi Diagrams of Polygons

Given a set S of geometric entities in $\mathbb{R}^m$ and a distance metric $\|\cdot\|$, the *Voronoi diagram* of S, denoted $\mathrm{Vor}(S)$, is the subdivision of $\mathbb{R}^m$ 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.

The are many variants of Voronoi diagrams (see [4, 8]), here we focus on the Voronoi diagram of

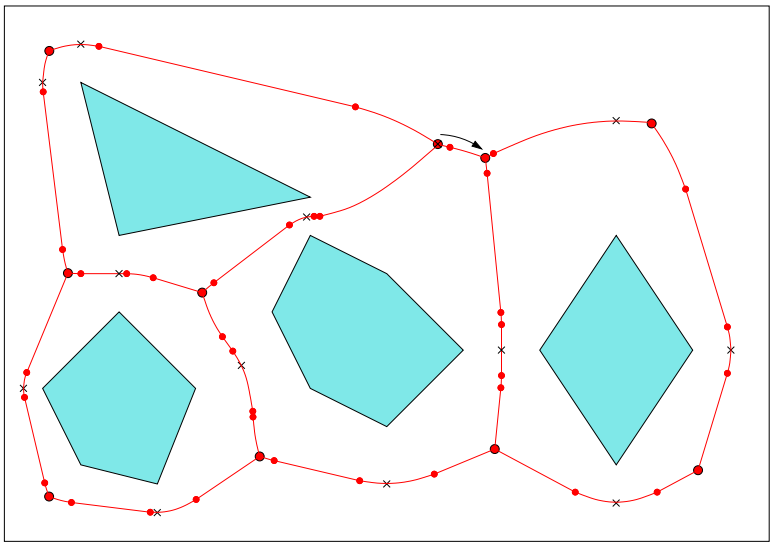


Figure 2: The Voronoi diagram of four convex polygons contained inside a rectangle. Small dots mark the endpoints of each Voronoi arc, while the Voronoi vertices are marked by larger dots. The point of minimum clearance along each chain is marked by $\times$. Notice that the chain marked by an arrow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

a set of pairwise interior-disjoint planar polygons, under the Euclidean distance metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$, which can be regarded as a special case of a Voronoi diagram of line segments [17]. 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 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).

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. For more details on the connection between Voronoi diagrams and motion planning see [19, 20, 22].

2.3 Minkowski Sums

Given two sets $A, B \in \mathbb{R}^m$, their *Minkowski sum*, denoted $A \oplus B$, is defined as:

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

We will not describe all properties of Minkowski sums. We just mention that given a polygon

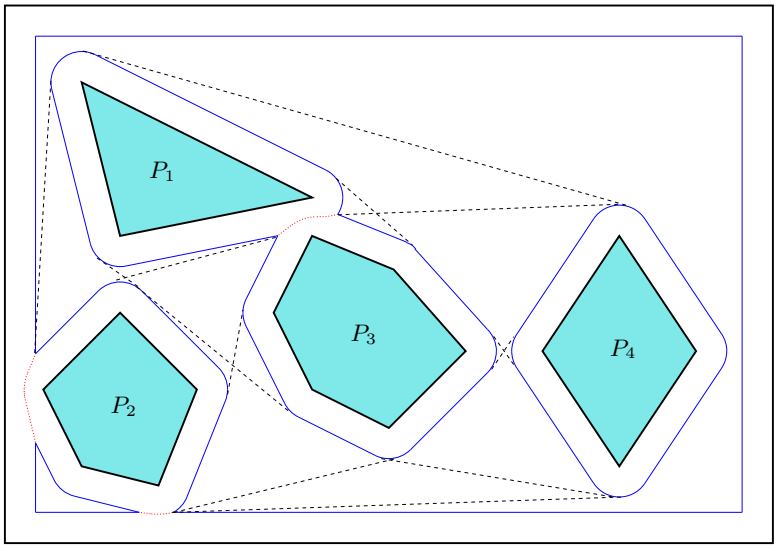


Figure 3: The $\mathrm{VV}^{(c)}$-diagram for four convex obstacles located in a rectangular room. The boundary of the union of the dilated obstacles is drawn in a solid line, the relevant portion of the Voronoi diagram is dotted. The visibility edges are drawn using a dashed line. Notice that an endpoint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

P, the set of points whose distance from P is less than d is the Minkowski sum $P \oplus B_d$, where B_d is a circle of radius d. This Minkowski sum of a set $\mathcal{P}$ 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 [15], where n is the number of polygon vertices. It is also possible to use an incremental randomized algorithm that achieves a running time of $O(n \log n)$ [5]. See [6, Chapter 13] and [11] for further discussions and more references.

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

3.1 Description

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

We begin by dilating each obstacle by c, by computing the Minkowski sum of each polygon with a disc of radius c. As it is the case with the visibility graph of the original polygons (for $c = 0$), 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. 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.

Given a start and a goal configuration we just have to connect them to our $\mathrm{VV}^{(c)}$-diagram and compute the shortest path between them using Dijkstra's algorithm. To this end, we have to associate a weight with each diagram edge. The weight of a visibility edge can simply be equal to its length, while for Voronoi edges we may add some penalty to the edge length, taking into account its clearance value, which is below the preferred c-value. Below we discuss natural weight functions related to the flow of the moving entities.

An interesting observation is that for $c = 0$, the $\mathrm{VV}^{(c)}$-diagram equals the visibility graph, while for $c = \infty$, it equals the Voronoi diagram. Thus, our diagram interpolates between the visibility graph and the Voronoi diagram.

3.2 Motivation

A straightforward application of the $\mathrm{VV}^{(c)}$-diagram is planning natural motion paths for a polygonal robot among polygonal obstacles. A robust implementation of a complete algorithm for this problem was given by [3] (generalizing the algorithm of Kedem et al. [15]), where they first compute the Minkowski sum of each obstacle with the robot to obtain a set of configuration-space obstacles, then plan a path that lies outside the union of these Minkowski sums. The main drawback of the algorithm is that it creates a piecewise linear path that may be significantly longer that the shortest path. If we construct the $\mathrm{VV}^{(c)}$-diagram of the configuration-space obstacles we can easily achieve more naturally-looking paths.

Another interesting application is motion planning for a group of entities in a two-dimensional environment cluttered with polygonal obstacles. Kamphuis and Overmars [12] solve this problem by planning a collision-free path π from the start location to the goal for a single entity (a small disc). The entities are constrained to a group shape, which has a constant area A during traversal of the path. The group shape is constructed symmetrically about the path π and has a "pear"-like shape. It is created by sweeping a disc (with a diameter equal to the preferred group width w) along the path from the current position of the group. The disc becomes smaller only when it collides with obstacles. That is, its radius becomes equal to the amount of clearance along the path. This sweep extends along the path to the point where the total swept area by the disk becomes equal to the desired group area A. The motions of the individual entities inside the "pear"-like shape are governed by a potential field.

The method described above yields very natural group behavior if the path along which the group is moving is smooth. In [12] a Probabilistic Roadmap (PRM — see, e.g., [14]) is used to find a path between start and goal position. However, PRMs do not contain smooth paths, so a path computed using a PRM has to be smoothed in a post-processing stage to serve as a convenient backbone for the motion of the group. This smoothing is done by shortcutting the path, but it must obey a clearance of $\frac{w}{2}$ if possible. At places where this is not possible, the path is approximately retracted to the Voronoi diagram of the obstacles, using the technique described in [9].

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. Moreover, computing the shortest path from the PRM and then smoothing it does not guarantee that we get the shortest path possible, as smoothing may considerably deform the original path. 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.

Fortunately, all the paths that can possibly be obtained after the smoothing step are contained in the $\mathrm{VV}^{(c)}$-diagram of the environment where $c = \frac{w}{2}$. So, we can select a path from the $\mathrm{VV}^{(c)}$-diagram based on its true length and amount of clearance. It is guaranteed that the $\mathrm{VV}^{(c)}$-diagram contains all possible paths between the start and goal locations (this is usually not the case when using a PRM). The $\mathrm{VV}^{(c)}$-diagram also does not require any smoothing step, which saves a precious amount of time.

When we compute a "shortest" path between two configurations using the $\mathrm{VV}^{(c)}$-diagram we have to give a weight to each edge. Naturally, we wish to use the length of the visibility edges (as well as circular arcs), which have a clearance of the preferred value c at least. In case of Voronoi arcs, we should add some penalty to the arc length. For example, if the clearance of an 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 a moving group of entities is to estimate the time it takes the group to traverse each edge: For edges with a clearance of $c = \frac{w}{2}$, 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.

3.3 Diagram Construction

Given a collection of disjoint convex (we will later discuss non-convex obstacles as well) obstacles $P_1, \dots P_m$ and a preferred clearance c, we proceed as follows:

- Compute the Voronoi diagram $\mathcal{V}$ of the obstacles $P_1, \dots P_m$. This is independent of the preferred clearance c.
- Dilate every obstacle P_i with a disc of radius c, i.e. 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.
- Compute the union $\mathcal{M}^{(c)}$ of all $M_i^{(c)}$. The boundary of $\mathcal{M}^{(c)}$ consists of circular arcs, straight line segments and reflex vertices. The reflex vertices are the intersection of the boundary arcs of two dilated obstacles, hence they lie on the Voronoi diagram $\mathcal{V}$ as their distance from both relevant polygons is exactly c.
- Compute a modified visibility graph $\mathcal{G}^{(c)}$ of $\mathcal{M}^{(c)}$. This graph contains every valid 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 reflex vertices of $\mathcal{M}^{(c)}$, and every free line segment from a reflex vertex tangent to a circular arc.

- Compute the intersection $\mathcal{V} \cap \mathcal{M}^{(c)}$, that is the portion of the Voronoi diagram that is contained within the union of the dilated obstacles. Combine the corresponding Voronoi arcs (and sub-arcs, as we consider only portions of some Voronoi arcs) with $\mathcal{G}^{(c)}$ to connect the chain points via narrow passages and form the final $\mathrm{VV}^{(c)}$-diagram.

In case our polygons are not convex, we decompose them to obtain a set of convex polygons and compute $\mathcal{M}^{(c)}$ for this set. Note that not every reflex vertex of $\mathcal{M}^{(c)}$ lies on the Voronoi diagram $\mathcal{V}$ in this case, since reflex vertices can also be induced by reflex vertices of the original polygons. However, these reflex vertices can be easily identified and ignored.

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 may also be possible to improve the running time to be $O(n \log n + s)$, where s is the number of visibility edges, based on the work of Pocchiola and Vegter [21]. The main difficulty here is that we are interested in the visibility among dilated obstacles, which may *not* be disjoint.

4 The VV-Complex

The construction of the $\mathrm{VV}^{(c)}$-diagram for a given c-value is straightforward, yet it requires some non-trivial algebraic operations that should be computed in a robust manner — namely the construction of the Voronoi diagram of the obstacles (see, e.g., [13]) and of the arrangement of dilated obstacle boundaries and Voronoi arcs, which require handling segments of algebraic curves of degree 2 (see, e.g., [23]). Moreover, if we wish to plan motion paths for different c-values and select the best one, we must construct the $\mathrm{VV}^{(c)}$-diagram for each c-value from scratch. In this section we explain how to efficiently preprocess an input set of 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 $\mathrm{VV}^{(c)}$-diagram as c continuously changes from zero to infinity. For simplicity, we consider only convex obstacles in this section. As we mentioned before, $\mathrm{VV}^{(0)}$ is the visibility graph of the original obstacles, while $\mathrm{VV}^{(\infty)}$ is their Voronoi diagram.

We start with a set of visibility edges containing all pairs of the polygonal obstacle vertices that are mutually visible. Note that these edges need not be bitangents of the obstacles.[2] We also include the original obstacle edges in this set, and treat them as visibility edges between two neighboring polygon vertices. Furthermore, we treat our visibility edges as directed, such that if the vertex u "sees" the vertex v, we will have two directed visibility edges $\vec{uv}$ and $\vec{vu}$.

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

Let α_{uv} be the angle between the vector $\vec{uv}$ and the x-axis, and $d(u, v)$ the Euclidean distance

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

[3] Recall that all our edges are directed. According to our notation, $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ are equivalent to $\vec{vu}_{rr}$ and $\vec{vu}_{ll}$, respectively, while $\vec{uv}_{lr}$ and $\vec{uv}_{rl}$ are equivalent to $\vec{vu}_{lr}$ and $\vec{vu}_{rl}$, respectively.

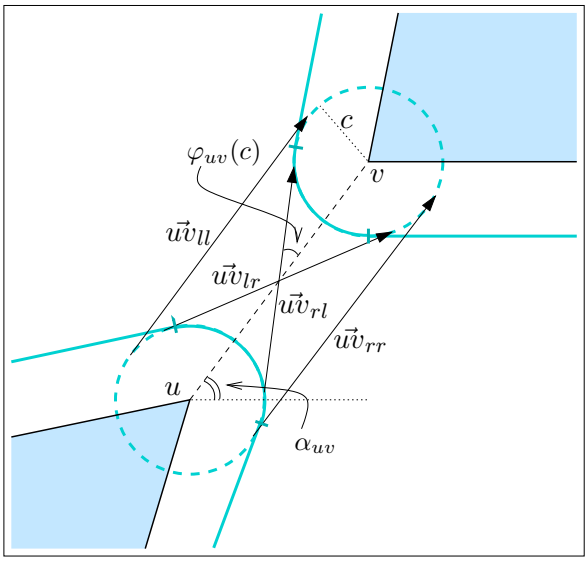


Figure 4: The four possible bitangents to the circles $B_c(u)$ and $B_c(v)$ of radius c centered at two obstacle vertices u and v. Notice that in this specific scenario only the bitangent $\vec{uv}_{rl}$ is a valid visibility edge.

between u and v, then it is easy to see that the two edges $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ retain the same slope α_{uv} for increasing c-values. The slope of the other two edges 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 $\varphi_{uv}(c) = \arcsin(\frac{2c}{d(u,v)})$.

Note that for a given c-value, it is impossible that all four edges are valid (at most three can be valid, and the ll- and rr-edges can never be valid simultaneously). Our goal is to compute a *validity range* $R(e) = [c_{\min}(e), c_{\max}(e)]$ for each edge e, such that e is part of the $\mathrm{VV}^{(c)}$-diagram for each $c \in R(e)$. In Section 4.3 we prove that such a validity range exists for each edge.

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.

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 neighboring 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. We note that the edge $\vec{uv}_{ll}$ (or $\vec{uv}_{lr}$) can only have events with arcs of the form $\vec{uw}_{ll}$ or $\vec{uw}_{lr}$, while the edge $\vec{uv}_{rl}$ (or $\vec{uv}_{rr}$) can only have events with arcs of the form

$u\vec{w}_{rl}$ or $u\vec{w}_{rr}$. 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 can have an event at some c-value only if they are neighbors in the list for infinitesimally smaller c. At these event points, we should update the validity range of the edges involved, and also update the adjacencies in their appropriate lists, resulting in new events.

In the rest of this section, we will 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 $u\vec{v}_{rr}$. Moreover, we will use $\mathcal{L}(u)$ to denote either $\mathcal{L}_l(u)$ or $\mathcal{L}_r(u)$ (whether we choose the "left" or the "right" list depends on the type of edge involved).

As mentioned in Section 3, an endpoint of a visibility edge in the $\mathrm{VV}^{(c)}$-diagram may also be an intersection point of dilated obstacle boundaries, which by definition also lies on the Voronoi diagram (see Figure 3 for an illustration). We call such an endpoint that lies on the Voronoi diagram a *chain point*, as it can be associated with a Voronoi chain — in fact, as a Voronoi chain is either monotone or has a single point with minimal clearance, we can associate at most two chain points with every Voronoi chain. Our algorithm will also have to compute the validity range for edges connecting a chain point with a dilated vertex or with another chain point. For that purpose, we will have a list $\mathcal{L}(p)$ of the outgoing edges of each chain point p, sorted by their slopes (notice that we do not have to separate the "left" edges from the "right" edges in this case).

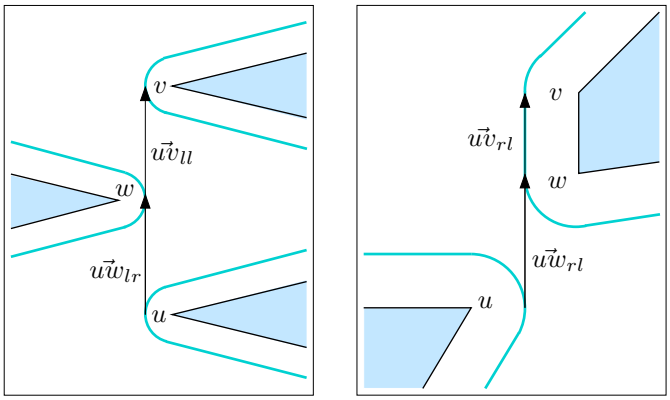
In the next section we review the algorithmic details of the preprocessing stage for constructing the VV-complex, and describe how to query this data structure in Section 4.2. We prove the correctness of our algorithm in Section 4.3, and conclude the presentation of the algorithm by a complexity analysis in Section 4.4. We finally show how to generalize the algorithm to work properly with non-convex obstacles as well in Section 4.5.

4.1 The Preprocessing Stage

4.1.1 Initialization

Our algorithm starts as follows:

1. Compute the visibility graph of the polygonal obstacles.
2. Examine each bitangent edge in the visibility graph: For an infinitesimally small c only one of the four edges it spawns is valid — assign 0 to be the minimal value of the validity range of this edge.
3. Initialize an empty event queue $\mathcal{Q}$, storing events by their increasing c-order.
4. For each obstacle vertex u:
   (a) Construct $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$, based on the edges obtained in step 2.
   (b) Examine each pair of neighboring 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. Insert the *visibility event* $\langle c, e_1, e_2 \rangle$ to $\mathcal{Q}$.
5. Compute the Voronoi diagram of the polygonal obstacles.



(a) (b)

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

6. For each *non-monotone* Voronoi chain, locate the arc a that contains the minimal clearance value $c_{\min}$ of the chain in its interior, and insert the *chain event* $\langle c_{\min}, a \rangle$ to $\mathcal{Q}$.[4]

4.1.2 Event Handling

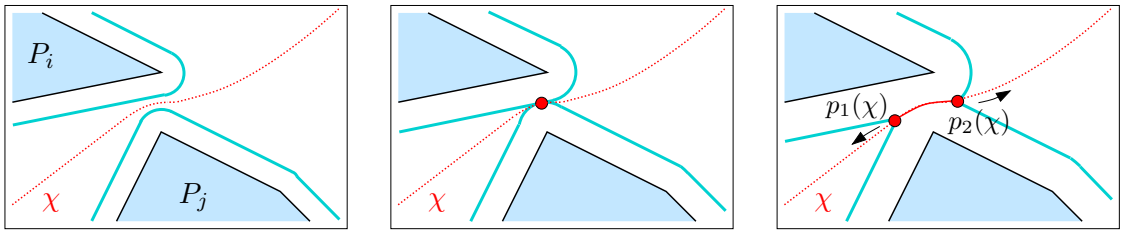
While the event queue is not empty, we proceed by extracting the event in the front of $\mathcal{Q}$, associated with minimal c-value, and handle it according to its type. We note that the visibility events (created, for example, by step 4b of the initialization stage) always come in pairs — that is, if $\vec{uv}$ becomes equally sloped with $\vec{uw}$, we will either have an event for the opposite edges $\vec{vu}$ and $\vec{vw}$, or for the opposite edges $\vec{wu}$ and $\vec{wv}$. We therefore handle a pair of visibility events as a single event:

Visibility event: The edges $\vec{uv}$ and $\vec{uw}$ become equally sloped for a clearance value c', and at the same time the edges $\vec{vu}$ and $\vec{vw}$ become equally sloped (see Figure 5).

1. Assign c' to be the maximal c-value of the validity range of $\vec{uv}$ and $\vec{vu}$.
2. Remove the other event involving $\vec{uv}$ (based on its other adjacency in $\mathcal{L}(u)$) from $\mathcal{Q}$, and delete this edge from $\mathcal{L}(u)$. Examine the new adjacency created in $\mathcal{L}(u)$ and insert its visibility event into the event queue $\mathcal{Q}$.
3. Repeat step 2 for the opposite edge $\vec{vu}$.
4. If the edge $\vec{uv}$ used to be valid before it was deleted and the edges $\vec{uw}$ and $\vec{vw}$ do not have a minimal validity value yet, assign c' to it, because these edges have become bitangent for this c-value (see Figure 5(b) for an illustration).

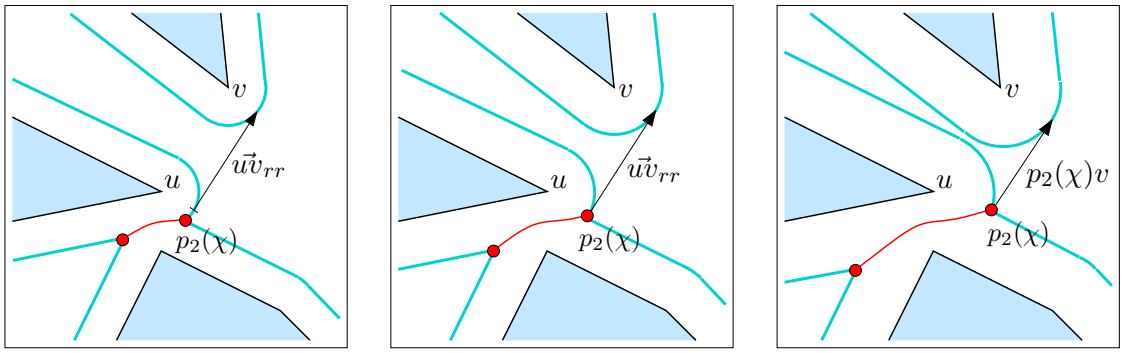
Chain event: The value c equals the minimal clearance of a Voronoi chain χ_a, obtained on the arc a, which is equidistant from an obstacle vertex u and another obstacle feature (see Figure 6). Let z_1 and z_2 be a's endpoints.

[4]Note that we do not create chain events for monotone Voronoi chains, whose minimum is obtained at one of their end-vertices (see Section 2.2).



(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: (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 toward the end-vertices of χ.



(a) (b) (c)

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.

1. 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.

2. For all edges $e = \vec{ux}$ incident to u, compute the c-value c' for which e becomes incident to one of the chain points $p_i(\chi_a)$ of a. If c' is within the range of the Voronoi arc a, then insert the *tangency event* $\langle c', e, p_i(\chi_a)\rangle$ to the event queue.

3. If a is equidistant to u and to another obstacle vertex v, repeat the last step for the edges incident to v.

4. Let c_1 and c_2 be the clearance values of z_1 and z_2, respectively. Insert the *endpoint events* $\langle c_1, p_1(\chi_a), z_1\rangle$ and $\langle c_2, p_2(\chi_a), z_2\rangle$ to the event queue.

When dealing with a chain event, we introduced two additional types of events: *tangency events* and *endpoint events*. We next explain how we deal with these events.

Tangency event: An edge $e = \vec{ux}$ (the endpoint x may either represent a dilated vertex or a chain point) becomes tangent to $B_c(u)$ at a chain point $p(\chi_a)$ associated with the Voronoi arc a (see Figure 7).

1. Remove all events involving the edge e from $\mathcal{Q}$.

2. Assign c to be the maximal validity value of the edge e, and remove this edge from $\mathcal{L}(u)$. Note that it is possible to disregard the new adjacency created in u's list.

3. Insert a reincarnate of e to $\mathcal{L}(p(\chi_a))$, and assign c as its minimal validity value. Examine the new adjacencies in $\mathcal{L}(p(\chi_a))$ and insert new visibility events into $\mathcal{Q}$.

4. Replace the edge $x\vec{u}$ in $\mathcal{L}(x)$ by $xp(\vec{\chi_a})$ and recompute the critical c-values of the visibility events of this edge with its neighbors.[5] Modify the corresponding visibility events in $\mathcal{Q}$.

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

Endpoint event: A chain point $p(\chi_a)$ reaches the endpoint z of a. If z is a local maximum of the clearance function, there are multiple event points associated with it, so we should just assign a maximal validity value to all edges in the edge lists of all chain points coinciding with z. If z is not a local maximum, we have to deal with one of the following two cases:

- z is incident only to two Voronoi arcs a and a' belonging to the same chain ($\chi_a = \chi_{a'}$). In this case the chain point $p(\chi_a)$ is transferred from a to a', and we only have to examine the adjacencies in $\mathcal{L}(p(\chi_{a'}))$ and modify the corresponding visibility events in the queue (as the slopes of these arc become a different function of c from now on). We also have to deal with the opposite edges, as we did in step 4 of the tangency-event procedure.

  If one of the polygon features associated with the new arc a' is a vertex u, iterate over all edges incident to u and check whether each edge has a tangency event in the range of the new Voronoi arc a' — if so, add this event to the queue $\mathcal{Q}$.[6] If a' is associated with two vertices u and v, repeat the procedure above for v as well.

- 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 synchronous endpoint event, and a single monotone chain $\hat{\chi}$ beginning at z (see for example the marked vertex in Figure 2):

  1. Create a new chain point $p(\hat{\chi})$ associated with the monotone chain.
  2. Assign the 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. Remove all visibility events associated with these edges from $\mathcal{Q}$.
  3. Insert reincarnates of all edges from $\mathcal{L}(p(\chi_1))$ into $\mathcal{L}(p(\hat{\chi}))$, and assign c as their minimal validity value. 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, as we did in step 4 of the chain-event procedure.

[5] Notice that the slope of $xp(\vec{a})$ becomes a different function of c from now on.

[6] 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.

Note that in the last step all edge lists of the chain points ending at z should be equal ($\mathcal{L}(p(\chi_1)) = \mathcal{L}(p(\chi_2)) = \ldots$), thus we consider only one of these lists. This event should be dealt with before any visibility event occurring at the same c-value, in order to avoid handling visibility events involving duplicate edges. In fact, when we have several events occurring at the same c-value, we deal with endpoint events first, then with visibility events, then chain events and finally tangency events.

4.2 The Query Stage

In order to support queries in an efficient manner, we should store a VV-complex, containing:

- The Voronoi diagram $\mathcal{V}$ of the polygonal obstacles. We 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 $c_{\min}(\chi)$.
- An interval tree $\mathcal{T}_u$ for each obstacle vertex u, containing its incident edges and and their validity ranges — namely, the intervals are the valid c-ranges.
- An interval tree $\mathcal{T}_{\chi,i}$ for each chain point associated with the Voronoi chain χ (at most two chain points are associated with each Voronoi chain, so $i \in \{1, 2\}$), containing its incident edges and incident Voronoi arcs 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 proceed as follows:

1. First, we compute the coordinates of all chain points. For each Voronoi chain χ, connecting the Voronoi vertices z_1 and z_2:
   - If $c(z_1), c(z_2) \leq \hat{c}$, the entire chain χ should be in the graph (it should be incident to both z_1 and z_2).
   - Otherwise, if χ is a monotone chain, assume without loss of generality that $c_{\min}(\chi) = c(z_1)$. If $c(z_1) < \hat{c} < c(z_2)$, compute a chain point p with clearance $\hat{c}$ on χ.
   - Otherwise, if $c_{\min}(\chi) < \hat{c}$ we compute two points $p_1, p_2 \in \chi$ such that either $p_i = z_i$ or $c(p_i) = \hat{c}$. Notice that for at least one of these points p_i does not equal z_i and is a chain point.
2. 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.
3. Now we can start searching the graph we have implicitly constructed using a Dijkstra-like search. We should devise a distance measure for the edges, which we discuss below.
   (a) Initialize a Fibonacci heap $\mathcal{H}$, containing vertices sorted by their distance from s. At first $\mathcal{H} \leftarrow \mathcal{L}(s)$.
   (b) Extract the heap minimum, associated with an obstacle vertex or a chain point x. Stop when the goal is reached (that is, if $x = g$).
   (c) Obtain a list $\mathcal{E}$ of edges incident to x:

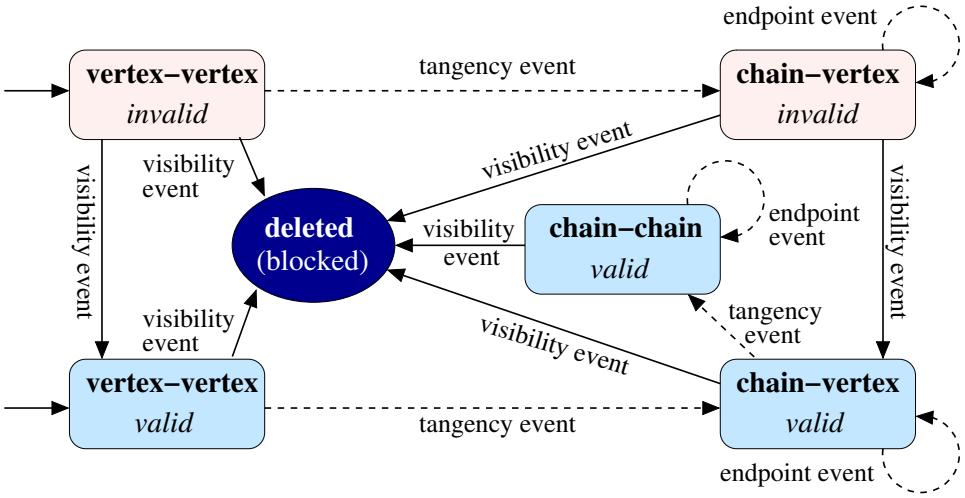


Figure 8: The schematic "life-cycle" of a visibility edge during the execution of the preprocessing stage. The solid arrows denote a change in the validity of the edge while the dashed arrows denote a reincarnation of the edge.

- If x is a chain point, and we arrived on x over a visibility edge, consider only the Voronoi arc incident to x. If we arrived on x over a Voronoi arc, consider only the visibility edges incident to x (an exception to this rule is that x is a Voronoi vertex — in this case, we arrived on x over a Voronoi arc and we will leave it over another Voronoi arc).
- Otherwise (x represents a dilated vertex), query $\mathcal{T}_x$ with the given c-value $\hat{c}$ to obtain the valid edges incident to x and consider only the edges that emanate from x at the same side (left or right) as we entered it.

In addition, add g to the list of x's neighbors if $x \in \mathcal{L}(g)$.

(d) For each edge $\vec{xy} \in \mathcal{E}$ obtained in the previous step, check whether $D(y) > D(x) + w(\vec{xy})$, where $D(x)$ is the currently calculated distance of x from s, and $w(\vec{xy})$ is the weight of the current edge — if x is an obstacle vertex, we should keep in mind to add the length of the portion of the corresponding circular arc to the overall weight. If so, we update the record of y in $\mathcal{H}$.

(e) While $\mathcal{H} \neq \emptyset$ goto step 3b.

The way we select the distance measure for the graph edges may depend on the path-planning strategy we employ. All visibility edges (and portions of the circular arcs which need to be traversed) have a clearance of at least $\hat{c}$, so their distance measure depends only on their length. For the portions of the Voronoi diagram, the limited amount of clearance may add extra weight (see the discussion in Section 3.2).

4.3 Proof of Correctness

To prove that the algorithm described in Section 4.1 for constructing the VV-complex is indeed correct, we need to show that every visibility edge has only one continuous range $[c_{\min}, c_{\max}]$ of c-values for which it is valid, so once it has been deleted it should not become valid again for a higher c-value. As we see in Figure 8, which describes the schematic "life-cycle" of a visibility edge,

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

Consider a visibility edge $\vec{uv}$ (it may either be invalid or valid) tangent to the supporting circles of the dilated vertices u and v, and a dilated vertex w whose supporting circle will start blocking the edge $\vec{uv}$ for some c-value c'. At the moment $c = c'$, the edge $\vec{uw}$ has the same slope as $\vec{uv}$, and $\vec{uv}$ is deleted. Assume without loss of generality that the slope of $\vec{uw}$ was *smaller* than the slope of $\vec{uv}$ for $c < c'$, then for c-values infinitesimally larger than c' the slope of $\vec{uw}$ is larger than the slope of uv (assuming that we do not delete $\vec{uv}$).

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

It is easily proved that such a value c'' does not exist. In Section 5.2.1 we solve a quadratic equation to compute for which c-values two edges are equally sloped. This equation has two solutions for c, one of which one is negative and therefore invalid. Hence, there is only one c-value between 0 and ∞ for which the two bitangent edges are equally sloped. This means that the edge $\vec{uv}$ cannot become unblocked once it has been deleted.

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

4.4 Complexity Analysis

Proposition 1 *The preprocessing stage takes $O(n^2 \log n)$ in total, where n is the total number of obstacle vertices.*

Proof: In the initialization of the preprocessing stage we first have to compute the visibility graph, which can be performed in $O(n^2 \log n)$ time — this also accounts for the time needed to construct the initial edge lists $\mathcal{L}(u)$ for each obstacle vertex u (we need $O(n \log n)$ time to construct each of the n 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.

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

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

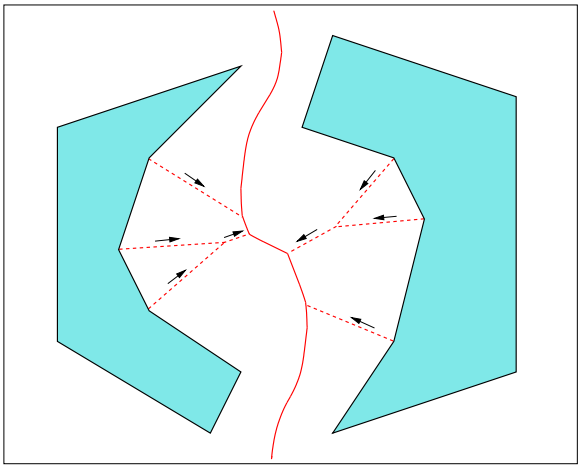


Figure 9: A portion of the Voronoi diagram of two non-convex polygons. The Voronoi chain separating the two obstacles is drawn with a solid line, while the Voronoi chains induces by features of the same polygon are drawn with a dashed line.

As the preprocessing algorithm proceeds, it starts handling events: In total we have $O(n^2)$ visibility events, each of them can be handled in $O(\log n)$ time. 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. □

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 $O(n \log n + k)$ where k is the number of edges encountered during the search. In practice, Dijkstra's algorithm turns out to be very fast, because hardly any geometric operations have to be performed anymore. In particular the A*-variant of Dijkstra may be the method of choice here, as it biases the search toward the goal configuration, which keeps the number k low.

As we noted in Section 3.3, the $\mathrm{VV}^{(c)}$-diagramfor a fixed c-value may be constructed in $O(n \log n + s)$ time, where s is the number of visibility edges, 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. 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. Moreover, after preprocessing our set of input obstacles, we may switch to machine-precision arithmetic in the query stage (see more details in Section 6).

4.5 Non-Convex Obstacles

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

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

As c grows, the reflex vertices will be treated as chain points. These chain points move over monotone Voronoi chains originating in the reflex vertices themselves (see Figure 9). 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 Voronoi diagram of line segments, so this change is easily carried through.

The rest of the algorithm remains unchanged. Also, the complexity analysis is still valid, since the construction time and the complexity of both the visibility graph and the Voronoi diagram is not affected by the non-convexity of its input obstacles.

5 Implementation Details

CGAL, the computational geometry algorithms' library [1] offers the infrastructure we need for developing a robust software that constructs the $\mathrm{VV}^{(c)}$-diagram. First, we made use of software components by Hirsch and Leiserowitz [11], who implemented a motion-planning algorithm for a disc robot among polygonal obstacles. For this purpose they decompose each obstacle into convex sub-polygons, compute the Minkowski sums of the convex sub-polygons with the disc robot, and finally compute the union of the Minkowski sums in order to obtain an exact representation of the forbidden configuration space for the robot.

Karavelas [13] has recently implemented a CGAL package for constructing the Voronoi diagram of line segments. By adding a label to each segment (polygon edge) that identifies its source polygon, it is straightforward to compute the Voronoi diagram of the polygon edges and remove Voronoi chains induced by features of the same polygon. Moreover, as we assume that there is one obstacle that contains all other obstacles in its interior (typically a bounding box), we can ignore infinite Voronoi edges. We should note here that the fact that our input segments are pairwise disjoint in their interior makes the construction of the Voronoi diagram very efficient.

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 (see below). The intersection between the dilated obstacles and the Voronoi edges can therefore be robustly computed using the conic-arc traits [23] of CGAL's arrangement package [7]. In Section 5.1 we prove the following theorem, which guarantees that we can compute the $\mathrm{VV}^{(c)}$-diagram in a robust manner:

Theorem 2 *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of pairwise interior-disjoint polygons whose vertices all have* rational coordinates. *Given a preferred clearance value c such that c^2 is* rational, *the*

boundaries of the dilated obstacles and the edges of the Voronoi diagram can be represented as segments of algebraic curves of degree 2 at most with rational coefficients.

For the construction of the VV-complex, we are mainly interested in the computation of the critical c-values for our events. In Section 5.2 we prove the following theorem:

Theorem 3 *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of pairwise interior-disjoint polygons whose vertices all have* rational coordinates. *All critical c-values needed for the computation of the VV-complex of $\mathcal{P}$ can be obtained as solutions to algebraic equations with rational coefficients.*

5.1 Representation of Edges

We next show that under the assumption that the polygon vertices all have rational coordinates, then all Voronoi arcs have supporting algebraic curves of degree 2 at most with rational coefficients and that all chain minima are also points with rational coordinates. Moreover, we show that for a clearance value c such that c^2 is rational, then the dilated obstacle boundaries are also supported by algebraic curves of degree 2 with rational coefficients. We can therefore use the conic-arc traits of CGAL's arrangement package to robustly compute the $\mathrm{VV}^{(c)}$-diagram for such c-values. We also show that the visibility edges between two dilated obstacle vertices can also be formulated as curves of degree 2 with rational coefficients in this case.

5.1.1 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:[8]

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

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 $\ell^\perp:\ By - Ax + (Ay_u - Bx_u) = 0$,

[8] Throughout this section we use the squared distance between two vertices, or between a vertex and an edge, in order to avoid the square-root operation.

and the point with minimal clearance is the midpoint between u and the intersection point of ℓ and $\ell^{\perp}$:

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

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

Edge–edge arc: The arc is equidistant from two polygon edges, whose supporting lines will be denoted by $\ell_1 : \ A_1 x + B_1 y + C_1 = 0$ and $\ell_2 : \ A_2 x + B_2 y + C_2 = 0$, respectively. The supporting curve of this edge is a line, but in general this line cannot be represented as a linear curve with rational coefficients.[9] Instead, we represent the edge as a segment of a pair of perpendicular lines, which form the two angle bisectors of ℓ_1 and ℓ_2:

$$\frac{(A_1 x + B_1 y + C_1)^2}{A_1^2 + B_1^2} = \frac{(A_2 x + B_2 y + C_2)^2}{A_2^2 + B_2^2} . \tag{3}$$

As we mentioned before, such an arc is always monotone — that is, if we traverse it from its end-vertex that with smaller clearance value toward the other end-vertex, we keep getting further away from the obstacles.

We showed that under the assumption that the polygon vertices all have rational coordinates, then all Voronoi arcs have supporting curves with rational coefficients and that all chain minima are also points with rational coordinates.

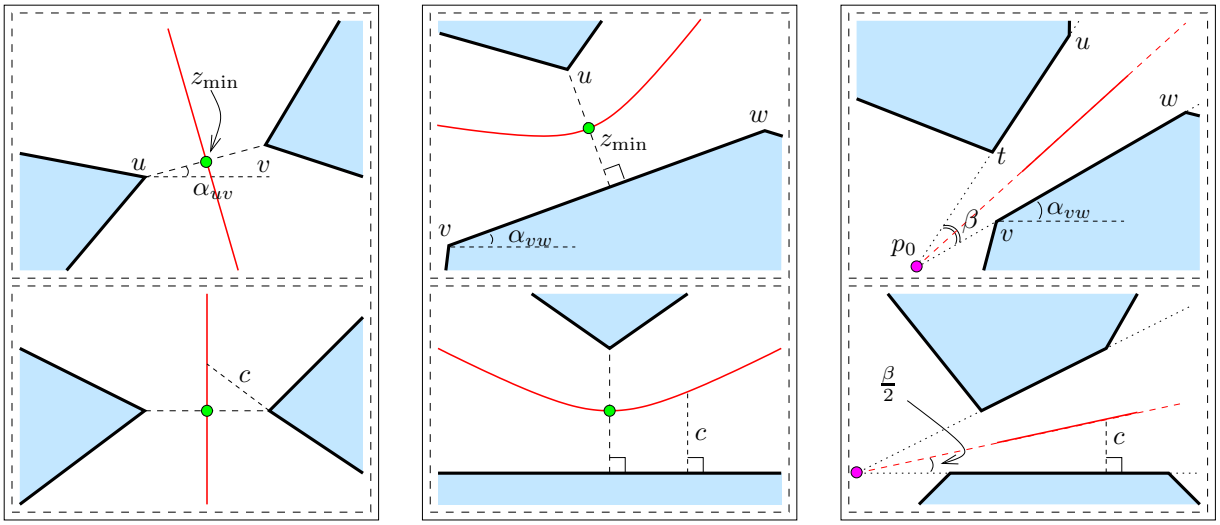
In order to construct and to query the VV-complex, we need to carry out the following computation: Given a Voronoi arc and a clearance value c, find a point z on the arc whose clearance is exactly c. Let us examine how we handle this computation for the various arc types. Notice that in all cases it is sufficient to locate a point whose distance from the two polygon features that define the arc is exactly c, as any other polygon feature is obviously further away:

Vertex–vertex arc: If the arc is defined by two polygon vertices u and v and $\frac{1}{2}d(u,v) \le c$, the point we are looking for is $\sqrt{c^2 - \frac{1}{4}d^2(u,v)}$ away from the midpoint $z_{\min}$. 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 10(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.

[9] 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 ($y = \frac{1}{1+\sqrt{2}}$) cannot be represented using rational coefficients. Note however that the line $y = \frac{1}{1-\sqrt{2}}$ is also an angle bisector in this case ...



(a) (b) (c)

Figure 10: 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.

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 $z_{\min}$ is located on the origin, so $\ell : y = -\frac{\delta}{2}$ and the arc is supported by the parabola $y = \frac{x^2}{2\delta}$ (see Figure 10(b) for an illustration). Note that given a point (x, y) on this hyperbola, 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: If the arc is defined by two polygon edges tu and vw, 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 iff $c \in [c(z_1), c(z_2)]$. Let $p_0 = (x_0, y_0)$ be the intersection point of the two supporting lines of these edges.[10] 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

[10] In the degenerate case of a Voronoi arc defined by two *parallel* polygon edges, all points on the arc have the same clearance, so we do not consider this case here. We also note that such arcs do not pose any problem when implementing the algorithm for the VV-complex construction.

between the lines supporting tu and uv, we can apply the Cosine Theorem on the triangle $\triangle up_0w$ and obtain:

$$\cos\beta = \frac{d^2(u,p_0) + d^2(w,p_0) - d^2(u,w)}{2d(u,p_0)d(w,p_0)} \ , \tag{4}$$

and thus:

$$\begin{aligned} \sin\beta &= \sqrt{1-\cos^2\beta} = \\ &= \frac{\sqrt{\left(2d^2(u,p_0) + 2d^2(w,p_0) - d^2(u,w)\right) d^2(u,w) - \left(d^2(u,p_0) + d^2(w,p_0)\right)^2}}{2d(u,p_0)d(w,p_0)} \ . \end{aligned} \tag{5}$$

Let us transform our coordinate system such that vw lies on the x-axis and p_0 is the origin (see Figure 10(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 (4) and (5) we get:

$$\begin{aligned} \cot\frac{\beta}{2} &= \frac{1+\cos\beta}{\sin\beta} = \\ &= \frac{\left(d(u,p_0) + d(w,p_0)\right)^2 - d^2(u,w)}{\sqrt{\left(2d^2(u,p_0) + 2d^2(w,p_0) - d^2(u,w)\right) d^2(u,w) - \left(d^2(u,p_0) + d^2(w,p_0)\right)^2}} \ . \end{aligned} \tag{6}$$

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:

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

Note that in the first two cases, where the arc can contain a minimum point, $z_{\min}$ has rational coordinates, and in the latter case p_0 is a point with rational coordinates. At any case, we showed that given a Voronoi arc and a clearance value c, we can compute a point (or two points) on the arc whose clearance is exactly c using the square-root operator (in addition to the basic arithmetic operations) on the input coordinates and the given c-value.

5.1.2 Dilated Obstacle Boundaries

The boundaries of the dilated obstacles are formed by arcs of the two following types:

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:

$$(x - x_u)^2 + (y - y_u)^2 = c^2 \ . \tag{7}$$

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

Dilated edge: The edges of the dilated obstacles are formed by offsetting the polygon edges parallel to themselves. However, it is impossible to represent a dilated edge as a linear curve

with rational coefficients.[11] Instead, we represent it as a segment of a pair of parallel lines, representing the locus of all points whose distance from the line $\ell: \; Ax + By + C = 0$ supporting the original polygon edge equals c:

$$\frac{(Ax + By + C)^2}{A^2 + B^2} = c^2 \; . \tag{8}$$

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.

5.1.3 Edges in the Visibility Graph

As we construct the $\mathrm{VV}^{(c)}$-diagram for clearance values such that c^2 is rational, we can show that the bitangents to two dilated obstacle vertices can also be formulated as curves of degree 2 with rational coefficients. This fact helps us to achieve a convenient representation for the visibility edges as well.

Given two obstacle vertices u and v with a clearance value c, there are four possible bitangents to $B_c(u)$ and $B_c(v)$ (see Figure 4). First notice that the two bitangents $\vec{uv}_{rr}$ and $\vec{uv}_{ll}$ are both parallel to the line ℓ connecting u and v and at a distance c from it. We can therefore represent this pair of bitangents by a line-pair of the form (8). Let us examine the other two bitangents. We know that they intersect at the midpoint $(x_0, y_0) = (\frac{x_u+x_v}{2}, \frac{y_u+y_v}{2})$, and that the slope of $\vec{uv}_{rl}$ is $\tan(\alpha_{uv} + \varphi_{uv}(c))$, while the slope of $\vec{uv}_{lr}$ is $\tan(\alpha_{uv} - \varphi_{uv}(c))$. We can therefore write:

$$\begin{cases} \vec{uv}_{rl}\text{:} & \cos(\alpha_{uv} + \varphi_{uv}(c))(y - y_0) - \sin(\alpha_{uv} + \varphi_{uv}(c))(x - x_0) = 0 \\ \vec{uv}_{lr}\text{:} & \cos(\alpha_{uv} - \varphi_{uv}(c))(y - y_0) - \sin(\alpha_{uv} - \varphi_{uv}(c))(x - x_0) = 0 \end{cases}$$

We can multiply the two equations to obtain a line-pair whose equation is given by:

$$\underbrace{\cos(\alpha_{uv} + \varphi_{uv}(c))\cos(\alpha_{uv} - \varphi_{uv}(c))}_{\hat{X}}(y - y_0)^2 + \underbrace{\sin(\alpha_{uv} + \varphi_{uv}(c))\sin(\alpha_{uv} - \varphi_{uv}(c))}_{\hat{Y}}(x - x_0)^2 \; -$$
$$\underbrace{(\cos(\alpha_{uv} + \varphi_{uv}(c))\sin(\alpha_{uv} - \varphi_{uv}(c)) + \sin(\alpha_{uv} + \varphi_{uv}(c))\cos(\alpha_{uv} - \varphi_{uv}(c)))}_{\hat{Z}}(x - x_0)(y - y_0) \;\; = \;\; 0$$

However, we have:

$$\begin{aligned} \hat{X} &= \cos(\alpha_{uv} + \varphi_{uv}(c))\cos(\alpha_{uv} - \varphi_{uv}(c)) = \\ &= \cos^2(\alpha_{uv})\cos^2(\varphi_{uv}(c)) - \sin^2(\alpha_{uv})\sin^2(\varphi_{uv}(c)) \; , \end{aligned}$$

$$\begin{aligned} \hat{Y} &= \sin(\alpha_{uv} + \varphi_{uv}(c))\sin(\alpha_{uv} - \varphi_{uv}(c)) = \\ &= \sin^2(\alpha_{uv})\cos^2(\varphi_{uv}(c)) - \cos^2(\alpha_{uv})\sin^2(\varphi_{uv}(c)) \; , \end{aligned}$$

$$\begin{aligned} \hat{Z} &= \sin(\alpha_{uv} + \varphi_{uv}(c) + \alpha_{uv} - \varphi_{uv}(c)) = \\ &= \sin(2\alpha_{uv}) = 2\sin\alpha_{uv}\cos\alpha_{uv} = 2\frac{(x_v - x_u)(y_v - y_u)}{d^2(u, v)} \; . \end{aligned}$$

As one can see in equations (9) and (10), $\sin^2(\alpha_{uv})$ and $\cos^2(\alpha_{uv})$ are always rational, while $\sin^2(\varphi_{uv}(c))$ and $\cos^2(\varphi_{uv}(c))$ are rational if c^2 is rational. We conclude that the two bitangents $\vec{uv}_{lr}$ and $\vec{uv}_{rl}$ form a line-pair with rational coefficients.

[11] 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 $y = x - \sqrt{2}$ also has the same distance property ...

5.2 Detecting Critical c-Values

We next explain how do we compute the critical c-values for visibility events and tangency events. We note that the critical c-values for chain events are actually the $c_{\min}$ values we discussed in Section 5.1.1, while endpoint events are easily detected by taking the clearance values of the endpoints of a Voronoi arc.

5.2.1 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:

$$\sin\alpha_{uv} = \frac{y_v - y_u}{d(u,v)} \ , \qquad \cos\alpha_{uv} = \frac{x_v - x_u}{d(u,v)} \ . \tag{9}$$

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):

$$\sin\varphi_{uv}(c) = \frac{2c}{d(u,v)} \ , \qquad \cos\varphi_{uv}(c) = \frac{\sqrt{d^2(u,v) - 4c^2}}{d(u,v)} \ . \tag{10}$$

The slope of this bitangent is therefore:

$$\begin{aligned} \tan(\alpha_{uv} + \varphi_{uv}(c)) &= \frac{\sin\alpha_{uv}\cos\varphi_{uv}(c) + \cos\alpha_{uv}\sin\varphi_{uv}(c)}{\cos\alpha_{uv}\cos\varphi_{uv}(c) - \sin\alpha_{uv}\sin\varphi_{uv}(c)} = \\ &= \frac{(y_v - y_u)\sqrt{d^2(u,v) - 4c^2} + 2(x_v - x_u)c}{(x_v - x_u)\sqrt{d^2(u,v) - 4c^2} - 2(y_v - y_u)c} \ . \end{aligned} \tag{11}$$

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:

$$\tan(\alpha_{uv} + \varphi_{uv}(c)) = \tan(\alpha_{uw})$$

$$\frac{(y_v - y_u)\sqrt{d^2(u,v) - 4c^2} + 2(x_v - x_u)c}{(x_v - x_u)\sqrt{d^2(u,v) - 4c^2} - 2(y_v - y_u)c} = \frac{y_w - y_u}{x_w - x_u}$$

$$\underbrace{((y_v - y_u)(x_w - x_u) - (x_v - x_u)(y_w - y_u))}_{\Phi_{uvw}} \cdot \sqrt{d^2(u,v) - 4c^2} = -2\underbrace{((x_v - x_u)(x_w - x_u) + (y_v - y_u)(y_w - y_u))}_{\Psi_{uvw}} \cdot c \ .$$

Squaring the equation above we get (note that $\Psi_{uvw} = u\vec{v} \cdot u\vec{w}$ and $\Phi_{uvw} = u\vec{v}^{\perp} \cdot u\vec{w}$):

$$\begin{aligned} \Phi_{uvw}^2 \left(d^2(u,v) - 4c^2\right) &= 4\Psi_{uvw}^2 \cdot c^2 \\ c^2 &= \frac{\Phi_{uvw}^2 d^2(u,v)}{4(\Phi_{uvw}^2 + \Psi_{uvw}^2)} \ . \end{aligned}$$

But it is easy to show that $\Phi_{uvw}^2 + \Psi_{uvw}^2 = d^2(u,v)d^2(u,w)$, so we have:

$$\begin{aligned} c^2 &= \frac{\Phi_{uvw}^2 d^2(u,v)}{4(d^2(u,v)d^2(u,w))} = \frac{\Phi_{uvw}^2}{4d^2(u,w)} \\ c &= \frac{(y_v - y_u)(x_w - x_u) - (x_v - x_u)(y_w - y_u)}{2\sqrt{(x_w - x_u)^2 + (y_w - y_u)^2}} \ . \end{aligned} \tag{12}$$

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

If we assume that all our input vertices have rational coordinates, the critical c-values, given in Equation (12), only involve taking the square root of a rational number. Moreover, c^2 is a rational number, so the equations of the supporting circle $B_c(u)$ of each circular arc, which are of the form $(x - x_u)^2 + (y - y_u)^2 = c^2$, have rational coefficients.

5.2.2 Detecting Tangency Events

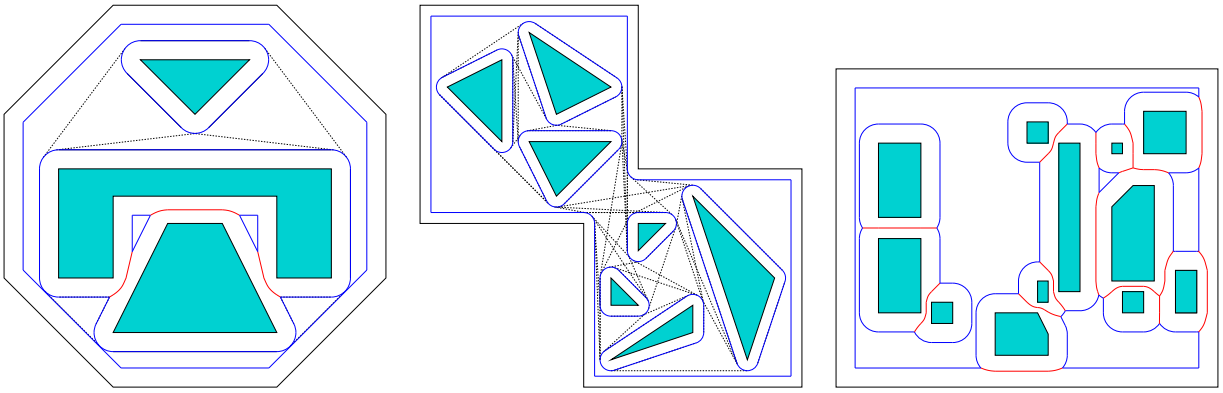
In a tangency event, a visibility edge between a chain point p and a dilated obstacle vertex v becomes equally sloped with a bitangent $u\vec{v}$ (see Figure 7 for an illustration). In Section 5.1.1 we explained in detail how to compute a chain point, and we now formulate the slope of $\vec{pv}$ as a function of c.

Consider the tangent to the obstacle vertex v dilated by c, supported by the circle $B_c(v)$ emanating from a point $p = (x_0, y_0)$. In order to compute the slope of this tangent, we have to compute the tangency point $q = (x, y)$. We do this by defining the following system of equations:

$$\begin{array}{ll} \text{I} & (x - x_v)^2 + (y - y_v)^2 = c^2 \\ \text{II} & (x - x_0)(x - x_v) + (y - y_0)(y - y_v) = 0 \end{array} \ . \tag{13}$$

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

$$(x_0 - x_v)(x - x_v) + (y_0 - y_v)(y - y_v) = c^2 \ . \tag{14}$$



(a) (b) (c)

Figure 11: The $\mathrm{VV}^{(c)}$-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) `scene14` with $c = \frac{9}{10}$ (visibility edges are not shown in this case).

Thus we can plug the expression for $(y - y_v)$ into Equation (13-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 $(y - y_v)$). In general, there are two tangency points emanating from p, given by:

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

Having computed the tangency point(s) q in terms of c, we can express the slope of the corresponding tangent as a function of c (note that the coordinates of p are functions of c themselves), thus we can compute for which c-value does such a tangent become equally sloped with another visibility edge. This gives rise to an algebraic equation of degree 4 in c, but only one solution of the equation may be valid and give the critical c-value.

We note that the critical clearance value at with the visibility edge $\vec{pv}$ is blocked by a dilated obstacle vertex w can also be obtained from an equation of the same form. Detecting the visibility event blocking an edge $\vec{xy}$ that connects two chain points is slightly more complicated, as it involves extracting the roots of a polynomial of degree 8. However, only one of the positive solutions is relevant in each case.

6 Experimental Results

Our software is implemented using CGAL 3.1 and the exact number types are supplied by CORE 1.7 [2]. As we wish to obtain an exact representation of the $\mathrm{VV}^{(c)}$-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 $\mathrm{VV}^{(c)}$-diagram depicted in Figure 3 (the `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 the same scene, such that no chain points appear in the diagram, the construction time drops to 2.5 seconds (see Table tab:times). In more involved scenes, the construction of the diagram may take 15–20 seconds (see Figure 11 and Table 1).

However, once the $\mathrm{VV}^{(c)}$-diagram is constructed, it is possible to use a floating-point approxi-

mation of the edge lengths to speed up the time needed for answering motion-planning queries,[12] so that the average query time is only a few milliseconds.

We also used the $VV^{(c)}$-diagram to generate convincing group motions in a more complex scene (the `scene14` input file). The construction of the diagram took about 15 seconds (for a clearance value that causes 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.

Input file	c	Construction time (sec.)	Average query time (sec.)
`shapes`	$^1/_5$	2.3	0.01
`shapes`	$^2/_5$	9.7	0.01
`octagon`	$^3/_{10}$	4.9	0.01
`octagon`	$^7/_{10}$	15.2	0.01
`two_rooms`	$^2/_5$	2.8	0.02
`scene14`	$^9/_{10}$	15.4	0.02

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

References

[1] The CGAL project homepage. `http://www.cgal.org/`.

[2] The CORE library homepage. `http://www.cs.nyu.edu/exact/core/`.

[3] 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. Special Issue, selected papers from the European Workshop on Computational Geometry, Eilat, 2000.

[4] 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.

[5] M. de Berg, J. Matoušek, and O. Schwarzkopf. Piecewise linear paths among convex obstacles. *Discrete and Computational Geometry*, 14:9–29, 1995.

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

[7] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. In *Proc. 12th Europ. Symp. Alg.(ESA 2004)*, pages 664–676. Springer-Verlag, 2004.

[8] 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.

[12] 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.

[9] R. Geraerts and M. H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA'04)*, pages 2386–2392, 2004.

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

[11] S. Hirsch and E. Leiserowitz. Exact construction of Minkowski sums of polygons and a disc with application to motion planning. Technical Report ECG-TR-181205-01, Tel-Aviv University, 2002.

[12] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In R. Boulic and D. K. Pai, editors, *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 1–10, 2004.

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

[14] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12:566–580, 1996.

[15] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete and Computational Geometry*, 1:59–70, 1986.

[16] D.-T. Lee. *Proximity and Reachability in the Plane*. PhD thesis, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1978.

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

[18] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 27, pages 607–642. Chapman & Hall/CRC, 2nd edition, 2004.

[19] 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.

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

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

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

[23] R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. 10th Europ. Symp. Alg.(ESA 2002)*, pages 884–895. Springer-Verlag, 2002.