The Visibility–Voronoi Complex and its Applications*

Ron Wein† Jur P. van den Berg‡ Dan Halperin§

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. This diagram can be used for planning natural-looking paths for a robot translating amidst polygonal obstacles in the plane. 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. 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 (a) the path should be *short* — that is, it should not contain long detours when significantly shorter routes are possible; (b) it should have a guaranteed amount of *clearance* — that is, the distance of any point on the path to the closest obstacle should not be lower than some prescribed value; and (c) it should be *smooth*, not containing any sharp turns. Requirements (b) and (c) may conflict with requirement (a) in case it is possible to considerably shorten the path by taking shortcuts through narrow passages. In such cases we may prefer a path with less clearance (and perhaps containing sharp turns).

The *visibility graph* [2, Chap. 15] is a well-known data structure for computing the shortest collision-free path between a start and a goal configuration. However, shortest paths are in general tangent to obstacles, so a path computed from a visibility graph usually contains semi-free configurations 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 [4] yields a path with maximal clearance, but this path may be significantly longer than the shortest path possible, and may also contain sharp turns.

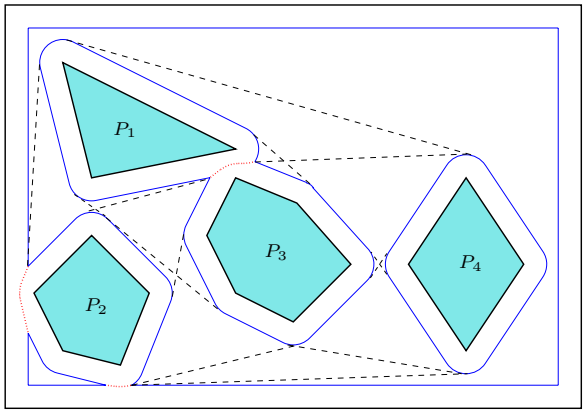


Figure 1: The $VV^{(c)}$-diagram for four convex obstacles located in a rectangular room. The boundary of the union of the dilated obstacles is drawn in a solid line, the relevant portion of the Voronoi diagram is dotted. The visibility edges are drawn using a dashed line.

We suggest a hybrid of these two approaches, 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. Beside the straightforward algorithm for constructing the $VV^{(c)}$-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*. 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 $VV^{(c)}$-diagram for that c-value.

2 The $VV^{(c)}$-Diagram

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 obsta-

*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.

†Tel-Aviv University, wein@tau.ac.il
‡Utrecht University, berg@cs.uu.nl
§Tel-Aviv University, danha@tau.ac.il

cles. Given a start configuration, a goal configuration and a preferred clearance value $c > 0$, we wish to find a shortest path between the query configurations, keeping a clearance of at least c from the obstacles where possible, but allowing to get closer to the obstacles in narrow passages when it is possible to make considerable shortcuts.

We begin by dilating each obstacle by c, by computing the Minkowski sum of each polygon with a disc of radius c. The visibility graph of the dilated obstacles contains all shortest paths with a clearance of at least c from the obstacles. Moreover, as each convex polygon vertex becomes a circular arc of radius c, the valid visibility edges are bitangents to two circular arcs (note that the dilated polygon edges are also valid visibility edges). This guarantees that a shortest path extracted from such a visibility graph is $\mathcal{C}_1$-smooth, containing 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 1 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 relevant portions of the Voronoi diagram are connected to the reflex vertices in the union of the dilated boundaries, which we refer to as *chain points*. The resulting structure is called the $\mathrm{VV}^{(c)}$-diagram, and it is easy to show that it can be constructed in $O(n^2 \log n)$ time.

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

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. 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.

We have implemented an extension package of CGAL [1] that robustly computes the $\mathrm{VV}^{(c)}$-diagram of a set of rational polygons, utilizing — among other CGAL packages — the segment Voronoi diagram package by Karavelas [3]. 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 1 takes about 10 seconds (running a Pentium IV 2 GHz machine with 512 MB of RAM). However, once the $\mathrm{VV}^{(c)}$-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.

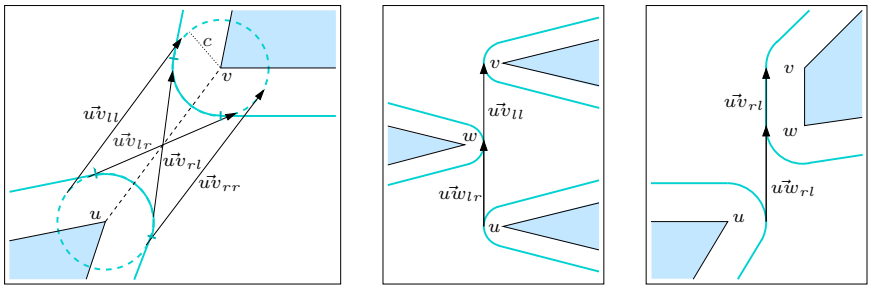
3 The VV-Complex

The construction of the $\mathrm{VV}^{(c)}$-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 the full version of this paper [5] for the 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 $\mathrm{VV}^{(c)}$-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 $\mathrm{VV}^{(c)}$-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, so as c grows visibility edges disappear from $\mathrm{VV}^{(c)}$ and make way to Voronoi chains. We start with a set of visibility edges containing all pairs of the polygonal obstacle vertices that are mutually visible, regardless whether these edges are bitangents of the obstacles.[1] 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 2(a)). The two bitangents $\vec{uv}_{ll}$

[1] 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.



(a) (b) (c)

Figure 2: (a) 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.
Visibility events involving u, v and w: (b) The dilated vertex w blocks the visibility of u and v. (c) As $\vec{uw}_{rl}$ becomes equally sloped with $\vec{uv}_{rl}$ (where vw is an obstacle edge), it becomes a valid visibility edge.

and $\vec{uv}_{rr}$ retain the same slope, while the slopes of the other two bitangents change for increasing c-values.

Note that for a given c-value, it is impossible that all four edges are valid. 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 $c \in R(e)$. These validity ranges will be stored in an interval tree, so it is easy to obtain all valid edges for any given c-value. 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: (a) The tangency point of e to either $B_c(u)$ or to $B_c(v)$ leaves one of the respective circular arcs; (b) The tangency point of e to either $B_c(u)$ or to $B_c(v)$ enters one of the respective circular arcs; (c) The visibility edge becomes blocked by the interior of a dilated obstacle.

The important observation is that at the moment that a visibility edge $\vec{uv}$ gets blocked, it becomes tangent to another dilated obstacle vertex w, so essentially one of the edges associated with $\vec{uv}$ becomes equally sloped with one of the edges associated with $\vec{uw}$ (see Figure 2(b)). The first two cases mentioned above can 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 2(c).

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 $\vec{uw}_{rl}$ or $\vec{uw}_{rr}$. Hence, we associate two circular lists $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$ of the left and right edges of the vertex u, respectively, both sorted by the slopes of the edges. Two edges participate in an event at some c-value only if they are neighbors in the list for infinitesimally smaller c. At these event points, we should update the validity range of the edges involved, and also update the adjacencies in their appropriate lists, resulting in new events.

As mentioned in Section 2, 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. We therefore compute the Voronoi diagram of the polygonal obstacles, which is comprised of *Voronoi chains* that separate between neighboring Voronoi cells. The chains are sequences of *Voronoi arcs*, which are either line segments or circular arcs, and their endpoints are called *Voronoi vertices*. 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).

3.1 The Preprocessing Stage

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

After this initialization step, we proceed to the event-handling step: 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 events always come in pairs — that is, if $\vec{uv}$ becomes equally sloped with $\vec{uw}$, we will either have an event for the opposite edges $\vec{vu}$ and $\vec{vw}$, or for the opposite edges $\vec{wu}$ and $\vec{wv}$. We therefore handle a pair of visibility events as a single event. Let us assume that the edges $\vec{uv}$ and $\vec{uw}$ become equally sloped for a clearance value c', and at the same time the edges $\vec{vu}$ and $\vec{vw}$ become equally sloped (see Figure 2(b) and (c)). As the edges $\vec{uv}$ and $\vec{vu}$ now become blocked, we assign c' to be the maximal c-value of the validity range of $\vec{uv}$ and $\vec{vu}$. We also remove the other event involving $\vec{uv}$ (based on its other adjacency in $\mathcal{L}(u)$) from $\mathcal{Q}$, and delete this edge from $\mathcal{L}(u)$. We examine the new adjacency created in $\mathcal{L}(u)$ and insert its visibility event into the event queue $\mathcal{Q}$. We repeat this procedure for the opposite edge $\vec{vu}$. If the edge $\vec{uv}$ was valid before it was deleted and the edges $\vec{uw}$ and $\vec{vw}$ do not have a minimal validity value yet, assign c' to it, because these edges have become bitangent for this c-value (see Figure 2(c) for an illustration).

A *chain event* occurs when 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. 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. For all edges e incident to u, we compute the c-value c' for which e becomes incident to one of the chain points $p_i(\chi_a)$, and insert a *tangency event* associated with c' and e to the event queue. If a is equidistant from u and from another obstacle vertex v, we do the same for the edges incident to v. We also insert two *endpoint events* to the queue, associated with the clearance values obtained at z_1 and z_2, respectively.

When dealing with a chain event, we introduced two additional types of events: tangency events and endpoint events. A *tangency event* occurs when 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, so we have to replace e by $p(\vec{\chi_a})x$ associated with the chain point $p(\chi_a)$ and update the relevant adjacency lists and the priority queue accordingly. An *endpoint event* occurs when a chain point $p(\chi_a)$ reaches the endpoint z of the Voronoi arc a — in this case the edges associated with $p(\chi_a)$ should be transferred to the next arc in the chain (or possibly to the next chain, if z is a Voronoi vertex). We skip the fine technical details involved in handling these events, which can be found in the full version of this paper [5]. The total number of events is $O(n^2)$ and the time complexity of the algorithm is $O(n^2 \log n)$. A proof of correctness is provided in [5] as well.

3.2 The Query Stage

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 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 is done using a radial sweep-line algorithm. We now start searching the graph we have implicitly constructed using a Dijkstra-like search to find the "shortest" path between s and g. Whenever we reach a vertex (a dilated polygon vertex or a chain point) we query the VV-complex with the given c-value $\hat{c}$ to obtain a list of its valid incident edges, and add g to this list if necessary. We proceed until the goal configuration g is reached.

The query time is dominated by the running time of Dijkstra's algorithm, which 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 and we can therefore switch to machine-precision floating-point arithmetic.

References

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

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

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

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

[5] R. Wein, J. P. van den Berg, and D. Halperin. The Visibility–Voronoi complex and its applications, 2004. www.cs.tau.ac.il/∼wein/publications/pdfs/vvc_TR.pdf.