Improving the Construction of the Visibility–Voronoi Diagram

Mojtaba Nouri Bygi* Mohammad Ghodsi†

Abstract

Ron Wein et al. [4] introduced the Visibility-Voronoi diagram for clearance c, denoted by $VV^{(c)}$, 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. They also proposed an algorithm that is capable of preprocessing a scene of configuration-space polygonal obstacles and constructs a data structure called the VV-complex.

As [4] used a straightforward approach for constructing $\text{VV}^{(c)}$-diagram, its construction time is $O(n^2 \log n)$, which is not optimum. In this paper we improve this time to $O(k \log n)$, where k is the number of visibility edges, based on the method used for the preprocessing of the VV-complex in [4].

1 Introduction

Ron Wein et al. [4] have studied 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" they 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. They introduced a new data structure, called the $\text{VV}^{(c)}$-diagram, 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 $\text{VV}^{(c)}$-diagram for a given clearance value c, they 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 c.

In this paper we reduce the time needed to construct the $\text{VV}^{(c)}$-diagram for a given c-value to $O(k \log n)$ where k is the number of visibility edges of the polygons, based on the method used for the preprocessing of the VV-complex in [4].

2 Constructing the $\text{VV}^{(c)}$-Diagram

Here we give the outline of our algorithm for constructing the $\text{VV}^{(c)}$-diagram of an input set $\mathcal{P}$ of pairwise interior-disjoint polygons with n vertices in total for a given c-value, say c_m, in $O(k \log n)$ where k is the number of visibility edges of the initial visibility graph of the polygons.

Our approach is to construct the visibility graph of the polygons, grow c while stopping at critical events that would change the visibility graph, until we get to c_m. This approach is very similar to that of [4] in preprocessing of VV-complex, in which they spend $O(n^2 \log n)$ time in computing the validity range of any possible edge of VV-diagram, while we only need the valid edges at c_m.

We start with a set of visibility edges containing all pairs of the polygonal obstacles. We also include the original obstacle edges in this set, and 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.

Let α_{uv} be the angle between the vector $\vec{uv}$ and the x-axis, and $d(u, v)$ the Euclidean distance between u and v, then it is easy to see that the two 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

*Department of Computer Engineering, Sharif University of Technology, P.O. Box 11365-9517, Tehran, Iran, nouribaygi@ce.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science (No. CS1382-2-02), P.O. Box 19395-5746, Tehran, Iran, ghodsi@sharif.edu

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 proceed in growing c while keeping track of valid visibility edges until we reach the final c_m 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:

- 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 $\vec{uw}$. 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 uw_{rl}), when v and w are neighboring vertices in a polygonal obstacle.

This observation stands at the basis of the algorithm we devise for constructing the $\mathrm{VV}^{(c)}$-diagram: We sweep through increasing c-values, stopping at critical visibility events, which occur when two edges become equally sloped, until we reach the goal clearance c_m. We note that the edge $\vec{uv}_{ll}$ (or $\vec{uv}_{lr}$) can only have events with arcs of the form $\vec{uw}_{ll}$ or $\vec{uw}_{lr}$, while the edge $\vec{uv}_{rl}$ (or $\vec{uv}_{rr}$) can only have events with arcs of the form $\vec{uw}_{rl}$ or $\vec{uw}_{rr}$. Hence, we can associate two circular lists $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$ of the left and right-edges of the vertex u, respectively, both sorted by the slopes of the edges. Two edges 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 of the edges involved, and also update the adjacencies in their appropriate lists, resulting in new events.

In the rest of the paper, we will use the notation $\vec{uv}$ to represent any of the four edges $\vec{uv}_{ll}$, $\vec{uv}_{lr}$, $\vec{uv}_{rl}$ or $\vec{uv}_{rr}$. 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 [4], 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. Such an endpoint that lies on the Voronoi diagram is called 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 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).

2.1 Construction

2.1.1 Initialization

Our algorithm start as follows:

1. Compute the visibility graph of the polygonal obstacles. This can be done in $O(k + n \log n)$ [3] where k is the number of visibility edges, but a simpler $O(k \log n)$ algorithm [3] is also sufficient for our algorithm.
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 `true` to be the value of the validity of this edge.
3. Initialize an empty event queue $\mathcal{Q}$, storing events by their increasing c-order. Hereafter, we only add events with c-order less than c_m to the queue.
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. This can be done in total $k \log n$ time.
   (b) Examine each pair of the 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. If the computed c is less than c_m, insert the *visibility event* $\langle c, e_1, e_2 \rangle$ to $\mathcal{Q}$. There are $O(k)$ of such events and updating the queue takes $O(k \log n)$ time.
5. Compute the Voronoi diagram of the polygonal obstacles ($O(n \log n)$)
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 $\mathcal{Q}$.

2.1.2 Event Handling

While the event queue is not empty, we proceed by extracting the event in the front of 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 $u\vec{v}$ becomes equally sloped with $u\vec{w}$, we will either have an event for the opposite edges $v\vec{u}$ and $v\vec{w}$, or for the opposite edges $w\vec{u}$ and $w\vec{v}$. We therefore handle a pair of visibility events as a single event:

Visibility event: The edges $u\vec{v}$ and $u\vec{w}$ become equally sloped for a clearance value c', and at the same time the edges $v\vec{u}$ and $v\vec{w}$ become equally sloped.

1. The edges $u\vec{v}$ and $v\vec{u}$ are blocked. Delete them from the edges of the visibility graph.
2. Remove the other event involving $u\vec{v}$ (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 if is less than c_m into the event queue $\mathcal{Q}$.
3. Repeat step 2 for the opposite edge $v\vec{u}$.
4. If the edge $u\vec{v}$ used to be valid before it was deleted and the edges $u\vec{w}$ and $v\vec{w}$ do not have a true validity value yet, assign it to true, because these edges have become bitangent for this c-value.

The operations above can be done in $O(k \log n)$ because each $O(k)$ edges of the visibility graph can be blocked at most once and deleting such edge would produce $O(1)$ new events.

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. Let z_1 and z_2 be a's endpoints.

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 $e = u\vec{x}$ 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 less than c_m and 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.

As each edge of the visibility graph is became incident to a chain point at most twice (one for each of its vertices), so the total time spent in processing chain events is $O(k \log n)$.

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 $e = u\vec{x}$ (the endpoint x may either represent a dilated vertex or a chain point) becomes tangent to $B_c(u)$ at a chain point $p(\chi_a)$ associated with the Voronoi arc a.

1. Remove all events involving the edge e from $\mathcal{Q}$.
2. The edge e is blocked, so 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 its validity value to true. Examine the new adjacencies in $\mathcal{L}(p(\chi_a))$ and insert new visibility events, if they are smaller than c_m, 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. Modify the corresponding visibility events in 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 Q that is associated with $x\vec{u}$ and recompute the c-value associated with it.

Again, each edge of the visibility graph leaves a $\mathcal{L}(u)$ list and enters a $\mathcal{L}(\chi_a)$ list at most twice, so the total time spent in processing tangency events is $O(k \log n)$.

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 false validity value to all edges in the edge lists of all chain points coinciding with z and delete them from the visibility graph. 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 $\mathcal{Q}$. 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:

  1. Create a new chain point $p(\hat{\chi})$ associated with the monotone chain.
  2. Assign the validity value of each edge in $\mathcal{L}(p(\chi_1)), \mathcal{L}(p(\chi_2)), \ldots$ to false at clearance c, 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 their validity value to true. Examine all adjacencies in $\mathcal{L}(p(\hat{\chi}))$ and add the appropriate visibility event to Q. We also have to deal with the opposite edges, as we did in step 4 of the chain-event procedure.

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.

As we stated earlier, the complexity of each obstacle is $O(1)$, so the complexity of each Voronoi chain is $O(1)$. Therefore the number of chain events is $O(n)$ in total and $O(1)$ for each Voronoi chain, and the number of times an edge would participate in a chain event is $O(1)$, so the time needed to process all the chain events is $O(k \log n)$.

2.2 Complexity Analysis and Proof of Correctness

As we mentioned before, the algorithm described in section 2 is based on the algorithm given in [4] for preprocessing stage of computing VV-complex, so the proof of correctness would be similar.

Proposition 1 *The construction takes $O(k \log n)$ in total, where n is the total number of obstacle vertices and k is the number of visibility edges in the visibility graph.*

Proof: We first have to compute the visibility graph, which can be performed in $O(k + n \log n)$ time [1], though the $O(k \log n)$ is sufficient for us. This also accounts for the time needed to construct the initial edge lists $\mathcal{L}(u)$ for each obstacle vertex u (there are k edges and 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 $O(k)$ visibility edges, and in addition $O(n)$ chain events. Any operation on the event queue thus takes $O(\log n)$. The initialization takes $O(k \log n)$ time in total.

As the construction algorithm proceeds, it starts handling events: In total we have $O(k)$ visibility events, each of them can be handled in $O(\log n)$ time. There are $O(n)$ chain events, and as we said earlier, they can be handled in $O(k \log n)$ time in total. Each visibility edge can participate in a tangency event at most twice , so in total there are $O(k)$ tangency events, each of them can be handled in $O(\log n)$ time. Finally, there are $O(n)$ endpoint events and they will deal with at most $O(k)$ visibility edges, so we need $O(k \log n)$ time to handle them in total.

3 Conclusion

It this paper we studied $\mathrm{VV}^{(c)}$-diagram designed by Wein et al. [4] for finding natural-looking paths amid polygonal obstacles. We presented an algorithm for constructing this data structure that runs in $O(k \log n)$ time, where n is the number of vertices of the polygons and k is the number of visibility edges of the visibility graph of the polygons, which is an improvement to current result that works in $O(n^2 \log n)$ time.

It seems that the $\mathrm{VV}^{(c)}$-diagram for a fixed c-value may be constructed in $O(k + n \log n)$ time, based on the work of [2], 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.

References

[1] S. K. Ghosh and D. M. Mount. *An output sensitive algorithm for computing visibility graphs.* Proc. 28th Annual IEEE Sympos. Found. Comput. Sci., 1987.

[2] M. Pocchiola and G. Vegter. *The visibility complex.* International J. Comput. Geom., 1996.

[3] M. Pocchiola and G. Vegter. *Computing the visibility graph via pseudo-triangulation.* In Proc. Annu. ACM Sympos. Comput. Geom., 1995.

[4] R. Wein, J. P. van den Berg, and D. Halperin. *The Visibility–Voronoi Complex and Its Applications.* In Proc. Annu. ACM Sympos. Comput. Geom., 2005.