Published in proceedings of 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), doi: 10.1109/COMPSAC54236.2022.00021.

Reliable Visibility Algorithms for Emergency Stop Systems in Smart Industries

Gabriele Capannini
Mälardalen University
Västerås, Sweden
gabriele.capannini@mdu.se

Jan Carlson
Mälardalen University
Västerås, Sweden
jan.carlson@mdu.se

Roger Mellander
ABB Robotics
Västerås, Sweden
roger.mellander@se.abb.com

***Abstract*—Automated machinery and robots working with humans are the norm in modern smart industries. A previous work in this area proposed a tool for improving the safety of such work places: an emergency system which halts those machines that are visible from an emergency stop button when it is pressed [1]. The solution presented in this paper improves the reliability of the aforementioned one at the expense of a higher computational complexity. Furthermore, two algorithmic optimizations are presented to mitigate the extra computational cost as it is shown by the results collected from the set of experiments conducted.**

***Index Terms*—I.2.1.g Industrial Automation; I.3.5 Computational Geometry and Object Modeling; K.4.1.d Human safety.**

I. Introduction

In modern highly automated factories, humans work closely to autonomous vehicles and stationary machinery as sketched in Figure 1. Such environments pose several challenging problems concerning the safety, e.g., autonomous robot path planning and collision avoidance. Complementing the automated solutions, we intend to equip human operators with emergency stop buttons that allow them to manually stop the machinery and robots' activity if a dangerous situation is discovered.

A viable solution for factories which only contains stationary machinery is to strategically place a number of emergency devices and to statically map such devices to machinery. However, in more dynamic contexts (as the one we are aiming at) a fixed mapping is not sufficient due to the presence of automated moving robots and machinery. A simplistic solution would be to stop all of them within a given area, but this could be prohibitively costly in large scenarios.

Capannini *et al.* proposed a solution such that when an emergency button is pressed, only those machines that are visible from the button's location are stopped [1].

As explained in Section III, however, this solution has two main drawbacks mostly related to the assumptions made by the authors and the approximation with which the real-world objects are represented in the proposed model thus the limited ability of their algorithm to discover all visible objects in some circumstances.

The main contribution of this paper is proposing a more accurate model representing the real-world objects in order to guarantee the correctness of the results in all contexts. The paper also analyses the extra computational cost derived from using a more accurate object representation and proposes an algorithmic optimization to reduce it.

The rest of this paper is structured as follows. Section II formally defines the visibility problem and describes the assumptions at the base of the solution proposed in [1]. Section III takes up the formal definition to point out the limitation of the existing approach. Section V introduces our solution to the drawbacks detected and the computational complexity of the related solution. It also presents a viable optimization which, once applied to the original visibility algorithm, is able to empirically double its performance (according to the test results shown in Section VI). Conclusions and future works are presented in Section VII.

II. Formal Definition of the Visibility Problem

The visibility problem is a relevant topic used for modeling and solving many problems in the computational geometry field [2]. In computer graphics, for example, scenes are made of meshes that includes millions of polygons placed in different positions and efficient visibility algorithms can help to reduce the complexity of rendering such scenes. Again, the robot path planning problem is frequently reduced to a planar visibility problem and visibility algorithms can be effectively employed for calculating a collision-free path.

In what follows, the formal definitions of polygon and visible polygon from a given point are presented as reported in [2], [3].

Definition 1: A polygon P is defined as two-dimensional closed shape limited by a finite number of line segments, the so-called *edges*, of which endpoints are also called the *vertices* of P. The set of edges define a circuit such that they do not intersect each other.

Definition 2: Given a set S of disjoint polygons and a query point q, define the subset $V(S, q) \subseteq S$ made of all those polygons that are visible from q, where a polygon p is visible from q if it exists an open line segment from q to any point of p that does not intersect any other polygon in S.

Unlikely, Definition 2 does not perfectly fit into the case where polygons in S are used to represent the moving robots and machinery described in Section I. As observed in [1], reducing the real-world objects to their top-view orthographic projections inevitably creates overlaps among the resulting polygons which, in turn, contradicts the initial assumption of having an set of disjoint polygons as input. Moreover, as a

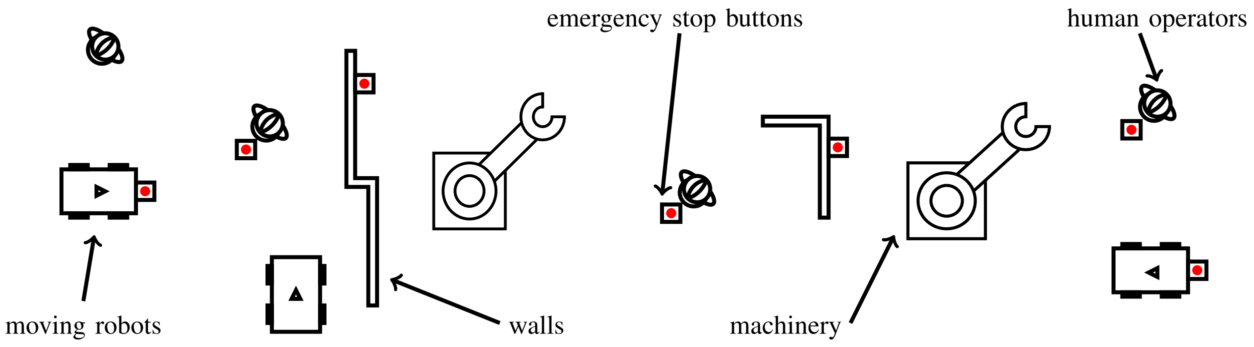

Fig. 1. Possible scenarios made of human operators, moving robots, and machinery sharing the work space. Walls are an example of static obstacles delimiting the different areas. Emergency stop buttons can be static or attached to moving entities.

consequence of their size and conformation, some objects let the hypothetical human operator placed in q see beyond them so that an object is possibly visible even if an second object stands between it and q. These two facts break Definition 2 since we cannot assume that the set of input polygons are disjoint and it is not required a unbroken line from q to an polygon to make it visible. For such a reason, the original visibility algorithms cannot be effectively used in this case study.

III. Background and Related Work

To the best of our knowledge, the solution proposed in [1] is the first work to address the visibility problem when in presence of overlapping polygons as introduced in Section II. This section illustrates, among other things, such a solution and how the authors partially overcome the issues presented in Section II.

The work presented in [1] assumes having every real-world object represented with (at least) one bounding volume enclosing the top-view orthographic projection of the object. No assumption is made on the type of bounding volumes in use which can vary from a simple circle to a more complex convex hull as shown in Figure 2. For sake of simplicity, the authors adopted AABBs. Since our work aims to improve even the accuracy of that method, it is worth noting that simpler bounding volumes require lower memory for their representation and imply lower computational costs at the expense of a lower accuracy since they are less tight fitting to the object projection.

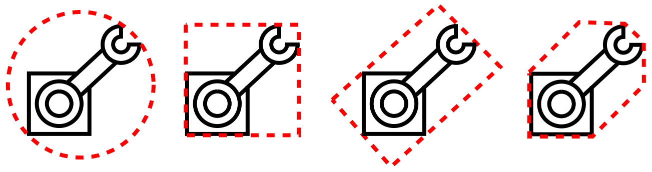

Fig. 2. Some of the most popular bounding volumes available in literature [4]. From left to right: circle, axis-aligned bounding box (AABB), oriented bounding box (OBB), and k-direction discrete orientation polytope (k-DOP).

Let the set of all bounding volumes be the set S of polygons of Definition 2 and let q denote the query point, the first step of the algorithm is to transform each bounding volume $v \in S$ into a segment $s(v, q)$ which defines the portion of the visibility field of q occupied by v, as depicted in Figure 3.

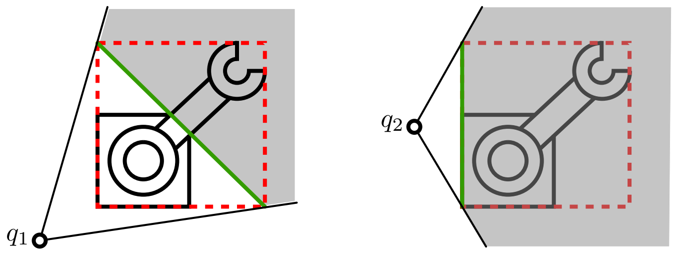

Fig. 3. Example of how the same bounding volume v (i.e., the red dashed box) can originate different segments $s(v, q)$ (i.e., the green lines) depending on the query point $q \in \{q_1, q_2\}$.

Similarly to the solution presented by Asano *et al.* [3], the algorithm then sorts the set of endpoints which define the set of segments according to their polar angle calculated in the polar coordinate system where the query point q is the reference point as shown in Figure 4.

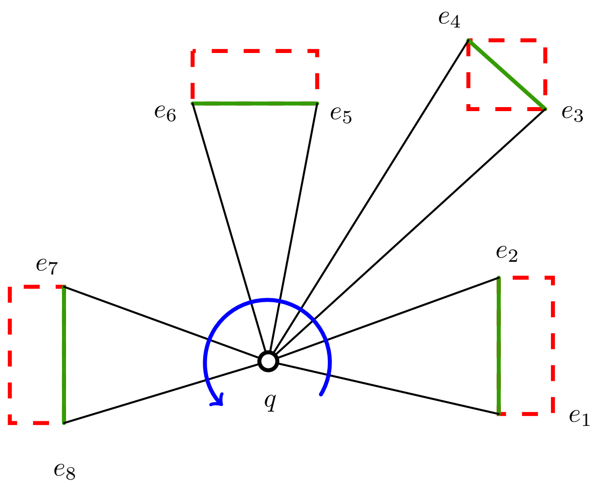

Fig. 4. Given a query point q and a set S of n polygons, the visibility algorithm firstly sorts the endpoints e_i of all segments in the set $\{s(q, v) | v \in S\}$ with $1 \le i \le 2n$.

Successively, the algorithm goes through the set of sorted endpoints and, for each endpoint e, tests the subset of segments intersecting the ray r, with $r = \overrightarrow{qe}$, to determine which of such segments are visible thus the related set of visible objects (so-called *sweeping phase*). To avoid useless tests, during the sweep, the algorithm keeps track of those segments that actually intersect r, so-called *active segments*, by adding and removing them from a data structure T according to the endpoint currently visited.

Let $|S|$ equal n and let T be a generic binary search tree, since $|T| = O(n)$ and T is visited once for each endpoint, the worst-case time complexity of the algorithm is $O(n^2)$ even if the tests show that its performance are almost linear in n.

IV. Correctness and Accuracy Analysis

This section points out two types of issues encountered during the analysis of the solution proposed in [1]. As first, we analyse the *correctness* of the algorithm from the perspective of the problem definitions given in Section II, then we discuss the *accuracy* of the overall model and the possible drawbacks due to approximating real-world object by means of two-dimensional bounding volumes.

A. Correctness

Contrary to the original solution presented by Asano *et al.* [3], the approach presented by Capannini *et al.* [1] is able to work correctly even when a subset of the segments representing the polygons in S are overlapping – at the expense of a higher computational complexity. This feature, however, can not guarantee the correctness of the method in the presence of overlapping polygons in S. In fact, representing polygons with just a segment may lead to false negatives apart from the fact that their segments overlap or not. As an example, let us consider the case depicted in Figure 5. Here, three AABBs are represented by means of their segment as suggested in [1] but, even if the bottom-left vertex of B is visible, its segment is hidden by those corresponding to A and C so that B turns out to be hidden as well.

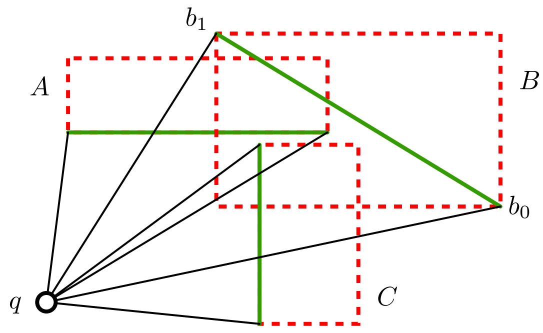

Fig. 5. Example of three AABBs with the related segments and the rays cast from the query point q to each endpoint. Even if B is visible, no rays hit directly the segment of B which is not reported as a visible polygon.

Another class of situations where the algorithm does not produce the correct result occurs when, in a cluster of polygons, part of the segment defining one of them is visible from the query point q, but actually no rays cast toward the corresponding set of endpoints hits it, as shown in Figure 6.

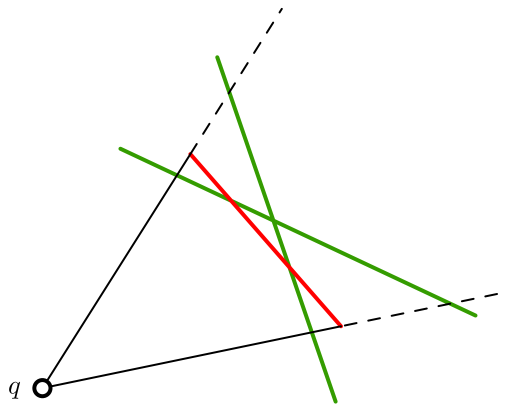

Fig. 6. Example of three segments where none of the rays cast from the query point q to the six endpoints does not hit any part of the red segment.

B. Accuracy

Depending on the accuracy we want to achieve and the type of bounding volumes utilized, the computed segments can lead to harmful over-approximations of the field of view that an object actually hides from q which, in turn, may produce wrong results. As shown in Figure 7, the bounding volume A and the related segment does not accurately represent the space hidden by the enclosed object where B is placed so that B is not considered visible although should be.

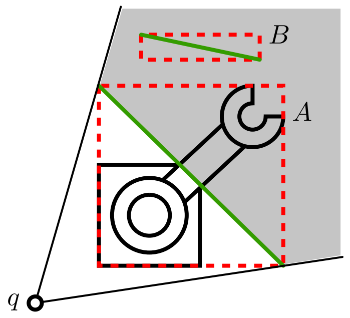

Fig. 7. Simple bounding volumes can lead to over-approximate the area hidden from a human operator as in this example where B is not considered visible from q due to the low accuracy applied for representing A.

Figure 8 shows two other scenarios where even a tighter bounding shape can be inaccurate. An object at position X is considered hidden behind the green segment although in reality it is visible. Moreover, an object at position Y is hidden in this 2D representation, but could very well be visible under the arm in the real-world 3D scenario.

V. Our Proposed Solution

This section describes a further variation to the solution presented by Asano *et al.* [3] that solves the issues discussed in Section IV.

To address the visibility inaccuracy caused by approximative bounding shapes, we first separate the objects into two disjoint

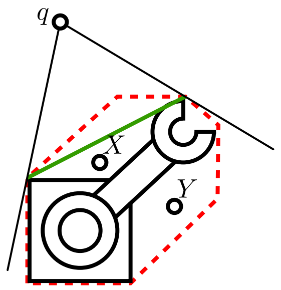

Fig. 8. Segment representation and difference between the real-world object and the projected representation can also cause incorrect visibility results.

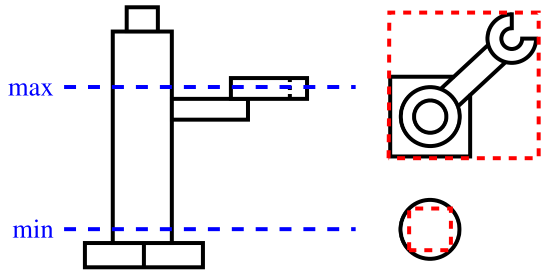

Fig. 9. Example of the difference representing covers and stoppables.

sets: *covers* and *stoppables*. Covers are objects that can impact the visibility of other objects, such as walls or large machines. Stoppables are the objects that can be stopped and that we thus want to determine if they are visible or not. If some real-world object is stoppable but also large enough to potentially hide other objects, it would be represented as two objects: one cover and one stoppable.

Keeping the two subsets disjoint means that we can treat them differently when constructing their representation. The basic idea is that we want stoppables to be over-approximated and covers under-approximated. As a result, if the visibility algorithm returns that a particular stoppable is not visible and thus does not have to be stopped, we know for sure that the actual object is also not visible in the real-world scenario.

The first difference between the two categories is the construction of a 2D representation of the real-world object. For stoppables, we use a simple top-view orthographic projection. For covers, however, we first define a minimum and maximum visibility height (corresponding to what we would consider as the lowest and highest reasonable eye-level of a human operator). Then, the projection of a cover only considers those parts of the object that are solid between the minimum and maximum visibility height.

The second difference is that for stoppables we use *bounding* volumes such as the ones exemplified in Section III, ensuring that if the stoppable object is visible, so is the bounding volume. However, for covers we want to avoid over-approximation and thus they are instead represented by *bounded* volumes, i.e., a volume enclosed in the projection of the object.

Figure 9 exemplifies the difference. If the object on the left is a cover, given the min and max visibility heights, only the large cylindrical part is used for the projection, as shown in the bottom right together with the enclosed *bounded* volume. If the object is instead a stoppable, its projection is shown in the top right together with the enclosing *bounding* volume.

In the visibility algorithm, covers are treated as non-transparent, non-stoppable objects and stoppable ones are treated as transparent, stoppable objects, as defined in [1].

As we observed in Section IV-A, the segmentation-based approach is subject to some drawbacks managing scenarios with overlapping bounding volumes. A viable solution to is to directly use all edges of each polygons instead of computing the segments.

This has twofold benefits: overlapping polygons are correctly identified regardless of their placement and, since there is no middle phase which transforms the bounding volumes into segments, the approximation of the results can meet the required accuracy by increasing (or decreasing) the tightness of the polygons used to represent real-objects.

This, however, affects the performance of the sweeping phase by increasing the computational complexity proportionally to the average number of the polygons' edges. The original solution has a two-to-one ratio between the number of endpoints to sweep and number of polygons, n, so that the final computational complexity of the algorithm is $O(n^2)$ as stated in Section III. While, for iterating the set of edges defining the bounding volumes, the computational complexity is quadratic in the number of vertices. For example, if instead of AABBs we employ a k-DOP (see Figure 2) the average computational complexity becomes k^2-times higher.

Furthermore, to address cases like the ones in Figure 6, it is required to cast additional rays toward the intersection points between different segments, or edges once we adopt them instead of segments. To define the direction of these extra-rays requires to preventively discover any *cluster* of overlapping polygons in S (referred as *broad phase*) by means, for example, of a collision detection method that theoretically is solvable in $O(n \log n + c)$ where c denotes the number of pairwise collisions [5]. On each cluster, we successively calculate the intersection points for the edges of the pertaining polygons (referred as *narrow phase*). Dividing the process in two phases (i.e., broad and narrow) reduces the overhead that we would pay by applying the collision detection algorithm to the whole set of polygon edges at once [6].

A. Algorithmic Optimizations

Let us consider the bounding volumes in Figure 5 when using all polygon edges instead of calculating the segments. Actually, to discover that B is visible from q, it is not necessary to test the all edges of B, but only those facing q, namely those lying within the triangle with vertices: q, b_0, and b_1 that are the endpoints of the segment $s(B, q)$. In general, hence, the edges of a generic bounding volume that are hidden by other edges of the same polygon can be safely skipped during the sweep phase.

As a consequence, given a query point q, for each bounding volume $v \in S$, we calculate $s(v,q)$ then we represent v only by means of the *significant* edges belonging to one of the two possible paths between the endpoints of $s(v,q)$ (i.e., the pair of vertices with minimum and the maximum polar angle calculated in the polar coordinate system where q is the reference point). Let e_0 and e_1 denote such endpoints for a given segment, we call $P_{\circlearrowleft}$ the path traversing the edges of v from e_0 to e_1 in the counterclockwise direction while $P_{\circlearrowright}$ identifies complement $P_{\circlearrowleft}$ i.e., the path containing the edges of v from e_0 to e_1 in clockwise direction. To select the path, since the edges of v do not intersect each other according to Definition 1, we simply choose the one passing closer to q, as shown in the example in Figure 10.

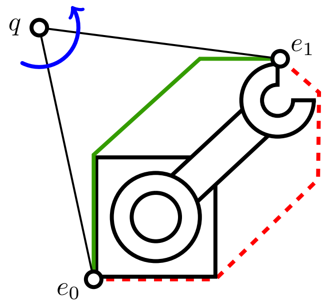

Fig. 10. Only the green edges of the given polygon (i.e., the bounding volume) are visible from q and correspond to the path $P_{\circlearrowleft}$ so that the others can be skipped during the sweep phase.

A second viable optimization is to restrict the calculation of the significant edges to the subset of the input polygons that overlap each other while the remaining disjoint ones can be represented by means of the segment $s(v,q)$ as the original approach does. Furthermore, the subset to which a polygon belongs to, can be easily defined by reusing the result of the *broad* collision-detection phase so that the overall computational cost of the algorithm will not be further penalized.

Figure 11 recaps how we combined all the proposed techniques to manage a cluster of polygons like the one depicted in that figure: the *broad* collision-detection phase discovers the cluster then, for each involved polygon, we define the set of *significant* edges (denoted by the solid edges in the figure) representing the polygon during the sweep phase and that are used by the *narrow* collision-detection phase to calculate the edges' intersection points toward which the sweep phase will cast the additional rays (the three ones having a square arrowhead in the figure).

VI. Experimental Results

This section shows the results of the experiment conducted to compare the performance of four solutions: the one proposed by Capannini *et al.* [1] which is considered as baseline, the naïve approach introduced in Section V, and the two solutions based on the optimizations discussed in Section V-A. In what follows, these four methods are referred as *baseline*, *naïve*, *path*, and *path&coll*, respectively. For our three pro-

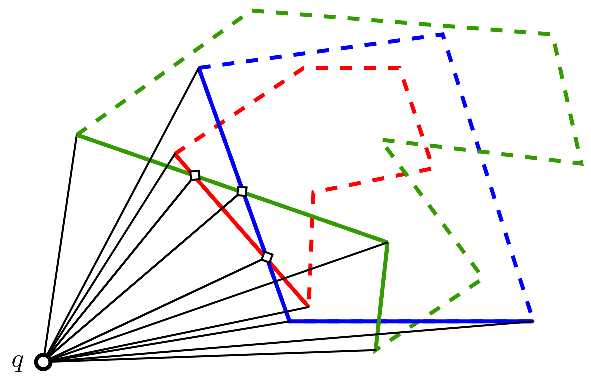

Fig. 11. Sketch of a possible cluster made of three polygons: red, green, and blue. Every polygon is represented by its the edges facing q. The intersection points between different paths define the direction of additional rays.

posed methods, the collision detection algorithm applied was the solution proposed by Larsson *et al.* [7].

All the experiments presented in this section were conducted on an Intel® Core™ i3-10110U CPU equipped with 16 GB of RAM and GCC 9.3.0 installed.

As first, we run a set of synthetic benchmarks (since it was not yet possible to conduct tests in real world scenarios) to measure the general performance of all methods. We generated different sets of scenes with a varying number, n, of irregular polygons. For each input size, 200 different scenes have been created by generating and placing randomly the n polygons according to the standard uniform distribution around a given query point q. Test results are shown in Table I where m denotes the average number of edges generated for any n-size set of scenes.

TABLE I
Elapsed times of the synthetic benchmark run for all approaches varying the problem size (in milliseconds).

n	m	*baseline*	*naïve*	*path*	*path&coll*
125	939	0.01	0.07	0.04	0.03
250	1879	0.02	0.15	0.09	0.06
500	3751	0.05	0.30	0.17	0.13
1000	7497	0.10	0.62	0.35	0.23
2000	14987	0.19	1.25	0.71	0.45
4000	30005	0.38	2.51	1.43	0.93
8000	60007	0.77	5.21	2.88	1.82
16000	120001	1.50	9.98	5.60	3.65
32000	240005	2.93	20.74	11.34	7.58
64000	480009	6.51	44.72	24.50	17.32
128000	960082	16.46	107.52	59.10	42.92

All the three methods presented run slower than the *baseline* which, as discussed in Section IV, possibly produces false negatives depending on the input scenario (we observed an average error in the number of visible polygons discovered equal to $\sim$1.5%). In fact, as discussed in Section III, this approach does not preform any preliminary collision detection phase and, during the sweep phase, each polygon is represented by a single segment so as to keep as low as possible the number of endpoints to iterate. On the other hand, the *naïve* approach is the slowest one by sweeping the largest number of elements, i.e., m polygon edges. Results also show the effectiveness of two optimizations proposed in Section V-A that are able to lighten the computational load of sweep phase with respect to the *naïve* approach by preserving the correctness of the results. On the average, the *path*-based solution was able to skip about one half of the edges while such a ratio rose up to 82% for the *path&coll*-based solution. As a consequence, the speedup of the *path* solution over the *naïve* one is $\sim 1.8\times$ on the largest problem instances while, for the *path&coll*-based solution, the measured speedup achieved $\sim 2.5\times$.

Since all phases of any visibility algorithm are affected by the clustering degree of the input set S, we conducted a second set of tests by varying the polygon *density* of the scenes. The polygon density, denoted by d, is calculated as the ratio of the sum of all polygon areas over the area of the entire scene. In particular, the density value d calculated on the scenes reported in Table I was 5% while, for this second test, we tested only the *path&coll*-based solution with $d \in \{15\%, 25\%, 35\%\}$. We expect that, by increasing d, even the overall computational work increases for each of our three proposed methods. In particular, higher density means higher chance that a generic pair of polygons overlaps. This increases the computational complexity of the collision detection algorithm (as stated in [5], [7]) as well as the number of edges to be processed during the sweep and the other intermediary phases (e.g., sorting). Results are shown in Table II.

TABLE II
ELAPSED TIMES OF THE SYNTHETIC BENCHMARK RUN FOR THE PATH&COLL APPROACH VARYING THE PROBLEM SIZE AND DENSITY (IN MILLISECONDS).

n	$d = 15\%$	$d = 25\%$	$d = 35\%$
125	0.04	0.04	0.05
250	0.08	0.09	0.10
500	0.15	0.17	0.20
1000	0.30	0.35	0.39
2000	0.60	0.70	0.83
4000	1.20	1.46	1.70
8000	2.45	2.99	3.62
16000	4.83	6.17	7.46
32000	10.51	13.32	16.60
64000	24.00	31.04	38.76
128000	59.79	79.06	98.67

To better interpret the results in Table II, we restricted our analysis to the largest instances of the problem and measured how the different phases of the visibility computation are affected by the density variation in term of effectiveness and performance. It turned out that comparing the results obtained for $d = 5\%$ (see Table I) and $d = 35\%$, the number of skipped-edges dropped from 82% to 65% while the number of fired rays almost doubled. Furthermore, as shown in Figure 12, the performances of the collision detection phase and the other intermediary phases were more slightly affected by the increased density than the sweep phase of which elapsed time rose $\sim 7\times$ due to the greater number of fired rays and, even more, the higher average amount of concurrently active elements to test for every ray.

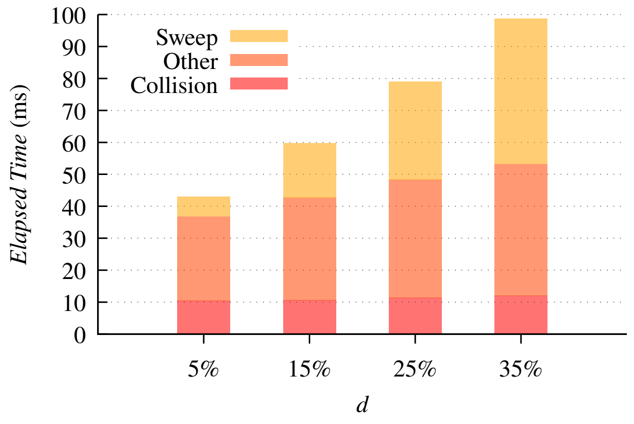

Fig. 12. Elapsed time breakdown by phases testing the *path&coll* solution by varying the density, d, with n equal to 128000.

VII. Conclusions and Future Works

We presented a more reliable version of the visibility algorithm proposed in [1] which, at the expense of a higher computational cost, is able to correctly solve scenarios where the input set of polygons can possibly overlap. To address the drop of performance, we proposed two algorithmic optimizations that we merged together in one approach.

As a future work, we would like to address the drop of performance pointed out in the last test. To this end we have a chance to reduce the average-case time complexity through the parallelization of the algorithm phases. Finally, we are also trying to solve the visibility problem by means of a three-dimensional model to address the accuracy issues of the overall emergency stop system.

Acknowledgment

This work was supported by the Knowledge Foundation in Sweden through the ACICS project (20190038).

References

[1] G. Capannini, J. Carlson, and R. Mellander, “Thou shalt not move: A visibility-based emergency stop system for smart industries,” in *7th Conference on the Engineering of Computer Based Systems*, ser. ECBS 2021. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3459960.3459966

[2] S. K. Ghosh, *Visibility Algorithms in the Plane*. Cambridge University Press, 2007.

[3] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, “Visibility of disjoint polygons,” *Algorithmica*, vol. 1, no. 1, pp. 49–63, Jan. 1986.

[4] C. Ericson, *Real-Time Collision Detection*. USA: CRC Press, Inc., 2004.

[5] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Berlin, Heidelberg: Springer-Verlag, 1985.

[6] P. Hubbard, “Interactive collision detection,” in *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*, 1993, pp. 24–31.

[7] T. Larsson and G. Capannini, “Adaptive collision culling for massive simulations by a parallel and context-aware sweep and prune algorithm,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 7, pp. 2064–2077, May 2017.