Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

A New Method for Mapping the Configuration Space Obstacles of Polygons

Evan Behar ebehar@gmu.edu
Jyh-Ming Lien jmlien@cs.gmu.edu

Technical Report GMU-CS-TR-2010-11

Abstract

Configuration space (C-space) plays an important role not only in motion planning but also in geometric modeling, shape and kinematic reasoning, and is fundamental to several basic geometric operations, such as continuous collision detection and generalized penetration depth estimation, that also find their applications in motion planning, animation and simulation. In this paper, we developed a new method for constructing the boundary of the C-space obstacles (C-obst) of polygons. This method is simpler to implement and often more efficient than the existing techniques. These main advantages are provided by a new algorithm that allows us to extract the Minkowski sum from the *reduced convolution* of the input polygons. We also developed a method for estimating the generalized penetration depth by computing the distance between the query point and the C-obst surface.

Keywords: *Configuration space (3-d), Minkowski sum (2-d), Geometric convolution (2-d), Generalized penetration depth (2-d)*

1 Introduction

The Configuration space (C-space) of a movable object P is the enumeration of all configurations of P, and a closed subset of the C-space that causes P to collide with obstacles Q is called C-space obstacle (C-obst). It is the mapping from the workspace obstacles to the C-obst that has interested researchers since late 1970s.

It is well known that computing an explicit geometric representation of C-space is intractable for objects with high degree of freedom [8], and researchers have been successfully solving difficult problems without computing the C-obst, e.g., using probabilistic motion planners (see [18]). However, an explicit representation of C-space remains important to many problems, including problems that require complete motion planners (e.g., assembly/disassembly), CAD (e.g., Caine's design of shape [7]), virtual prototyping, object placement [1] and containment [3]. In addition, C-space mapping is fundamental to basic geometric operations, such as continuous collision detection and generalized penetration depth estimation. Since the early 1980s to the mid-1990s, many researchers have proposed several methods to compute and approximate various types of representation of the C-obst. However, not until more recently have newer developments (e.g. the idea of *configuration products* by Nelaturi and Shapiro [21]) been made toward improving and generalizing these methods, partly because the need for an explicit representation of C-obst diminished after the development of sampling-based motion planners. See the survey done by Wise and Bowyer [26] for a complete review on these earlier works and see Section 2 for a brief overview of the related and recent works.

In this paper, we propose a new method for mapping 2-d polygons to their 3-d C-obst. Our method represents the boundary (∂C-obst) of C-obst as a set of *ruled surfaces*. The proposed method is simpler to implement than the existing methods in the literature [1, 6] and is often more efficient. These main advantages are provided by a new algorithm that allows us to extract the Minkowski sum (M-sum) boundary from the *reduced convolution* (defined in Section 4.2) of the input polygons. As a warm-up, we will first show that the C-obst of convex polygons can be computed using the idea of

convolution at *critical orientations* (in Section 4.1). We then show that the M-sum of two simple non-convex polygons can be computed efficiently using the filtering-based approach from the reduced convolution. Finally, the ∂C-obst of non-convex polygons is constructed by updating the M-sum at the critical orientations (a superset of those of convex polygons). The time complexity of our method is $O(m^3n^3 + bT_{cd})$ for polygons with m and n vertices, where b is the number of boundaries of C-obst, and T_{cd} is the time for a single collision query. We also show that the resulting C-obst can be used for efficiently estimating the generalized penetration depth by computing the closest feature between the query point and the ruled surfaces (Section 5.2).

2 Related Work

The idea of C-space mapping is first proposed by Lozano-Pérez [20]. His original idea was to construct the C-obst by slicing it at a predefined resolution along the rotational axis, and each slice can be computed as the Minkowski sum (M-sum)

$$-P \oplus Q = \{-p + q \mid p \in P, q \in Q\} \qquad (1)$$

of the robot P and the workspace obstacles Q. To connect two consecutive slices at θ and θ', Lozano-Pérez proposed to use the swept volume (area) of P rotating from θ to θ' to replace P in Eq. 1.

Since then, more techniques and representations (including grids [17], bounding shapes [27], analytical functions [1, 6], and semi-algebraic set [10, 8]) have been proposed, mostly in the context of solving motion planning problems. Note that the mapping techniques for free-flying robots and fixed-base articulated (manipulator) robots are very different. Since this paper focuses on free-flying robots, for readers interested in the manipulator, please refer to the work by Branicky and Newman [4], by Hwang [16], and by Ward and Katupitiya [24] for more recent work on this topic.

Among all these techniques, the slicing-based strategy remains quite popular due to its simplicity. Notably, Zhu and Latombe [27] generalized the idea to use the outer and inner swept areas (which are the union and the intersection of the areas swept out by the robot) to bound ∂C-obst. Kavraki [17] proposed a method to construct C-obst by computing the convolution of two polygons (represented as pixels) using the Fast Fourier Transform algorithm. Curto and Moreno [9] extended Kavraki's method to handle both free-flying and articulated robots. Later, Sacks and Bajaj [22] proposed a method to generate the slices for curved 2-d objects at fixed intervals. There are two major drawbacks of the slicing-based approaches. First, computation is wasted. There are significant performance improvements that can be gained by exploiting the temporal and spatial coherence. Second, the M-sums are normally separated by a *fixed rotational resolution*, which is defined empirically. A better approach is to identify "events" where the structure of the M-sum changes.

Thus, another line of research focused on the exact representation either the boundary using analytic functions or the volume using semi-algebraic set. For example, Donald [10] dealt with motion planning problem with a 3-d free-flying robot amongst polyhedral obstacles. Both robot and obstacles are composed of a set of convex shapes; the C-obst are therefore a set of 6-d *contact surfaces*. The intersections of these contact surfaces are computed to help the robot move along the intersection or slide from one surface to another. Another example is the work done by Halperin et al. [14]. They studied the motion planning problem of a L-shaped robot (composed of two line segments) among point obstacles. In this setting, the C-obst *is* a set of rule-surfaces. The simplicity of the problem allows them to construct the complete C-obst by identifying all critical orientations to determine the changes of the line segment arrangements. In these techniques, the methods proposed by Avnaim et al. [1] and Brost [6] are the ones closely related to our work.

Avnaim et al. [1] proposed to compute ∂C-obst using contact regions. A contact region is computed between a vertex of P and an edge of Q or vice versa. Their method computes sets of translations that result in contact between the polygons, excluding regions which represent non-free contacts (formed by the rotating parallelogram regions) by means of *set-theoretic differences*. This rotation/intersection space is computed as analytic functions that are put into one-to-one correspondence with the actual configuration space to compute the general contact regions for a given contact. Their method pre-computes a discrete set of contacts, and then computes the contact regions for each contact, which become the facets of the boundary. Though M-sums are not used in this method, a similar configuration space is produced. Their algorithm has time complexity $O(n^3m^3 \log nm)$ for polygons with n and m vertices.

Similar to Avnaim et al. [1], Brost [6] also considered all possible contacts, and used local information between the contact vetex/edge pair to compute the contact regions. Each contact region is a ruled surface. For non-convex polygons, part of the contact region may belong to the interior of the C-obst. Therefore, all contact regions are tested for intersection. Finally, each contact region is trimmed around the boundary created by the intersections and the remaining area is part of the ∂C-obst.

Note that in both methods a contact region can have zero area on C-obst. This means that the entire contact

region is trimmed. In fact, even for simple shapes (for example the star shape shown in Fig. 2), many contact regions will not make to the surface of C-obst. As a result, significant computation is wasted on finding the intersections between contact regions (which is a computational expensive operation). Our method avoids this problem and accelerates the mapping process (1) by reducing the number of contact regions and (2) by detecting and resolving the intersection in the two dimensional space. Another disadvantage of these approaches is that no clear means for computing the distance (e.g. for penetration depth estimation) is presented or easily derived from the representations.

Recently, Varadhan and Manocha [23] also proposed an approach that generates polygonal meshes to approximate the ∂C-obst using a marching cube technique with adaptive cell to extract the iso-surface from a signed distance field. The construction of the non-directional backprojection for solving compliant motion planning problems under uncertainty is also similar to the C-space mapping. Donald [11] and Briggs [5] have proposed ways to accelerate the construction by identifying the critical points of topological changes of the backprojections with respect to the visibility graph of the workspace. The main difference is that rays, instead of line segments, are used in these computations.

3 Preliminaries

In this section, we define the notations that will be used throughout the paper. We assume that P is movable while Q is stationary. Both P and Q are simple polygons composed of n and m (counterclockwise) ordered vertices, respectively. Our approach is based on computing and updating the M-sums using convolution. The convolution of two shapes P and Q, denoted as $P \times Q$, is a set of line segments in 2-d that is generated by "combining" the segments of P and Q [13]. One can think of the convolution as the M-sum that involves only the boundary, i.e., $P \otimes Q = \partial P \oplus \partial Q$. It is known that the convolution forms a superset of their M-sum boundary [12], i.e., $\partial(P \oplus Q) \subset P \otimes Q$. If both P and Q are convex, $\partial(P \oplus Q) = P \otimes Q$. Otherwise, it is necessary to trim the line segments or the facets of the convolution to obtain the M-sum boundary. Recently, Wein [25] shows a robust and exact method based on convolution for non-convex polygons. To obtain the M-sum boundary from the convolution, his method computes the arrangement induced by the line segments of the convolution and keeps the cells with non-zero winding numbers. A more detailed review on the M-sum can be found in our previous work [19].

An edge $\overline{p_i p_{i+1}}$ of P and a vertex q_j of Q (or vice versa) form a segment of $P \otimes Q$ if $\overrightarrow{p_i p_{i+1}} \in [\overrightarrow{q_{j-1} q_j}, \overrightarrow{q_j q_{j+1}})$, and we say that $\overline{p_i p_{i+1}}$ and q_j are *compatible*. Equivalently, $\overline{p_i p_{i+1}}$ and q_j are compatible if the outward normal of $\overline{p_i p_{i+1}}$ lies between the normals of the incident edges of q_j. For example, in Fig. 1(a), $\overline{p_3 p_1}$ and q_1 are compatible. When rotation is considered, the edges of P are rotated about a fixed center, c, and an edge/vertex pair can become *alive* if they are compatible and *dead* otherwise. Without loss of generality, we assume that P rotates counterclockwise about c, and c is the world origin, so that $c_x = c_y = 0$.

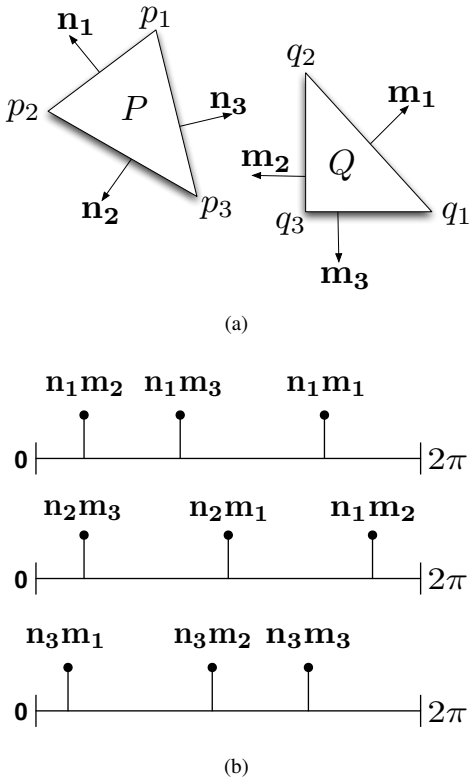


Figure 1: (a) Two convex polygons P and Q shown with the edges outward normals. (b) Events for $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$ (from top to bottom) when P rotates counterclockwise from 0 to 2π. For example, when P rotates $\pi/4$, $\vec{n}_1$ and $\vec{m}_2$ (and $\vec{n}_2$ and $\vec{m}_3$) become aligned and two events are issued.

4 Our Methods

We will first discuss the case of convex polygons in Section 4.1 and then extend the ideas the handle non-convex polygons in Section 4.2.

4.1 C-obst of Convex Polygons

Given two convex polygons P and Q (see Fig. 1(a)), an edge of their convolution is the sum of an edge $\overline{p_i p_{i+1}}$ of P and a vertex q_j of Q or vice versa. We let θ_0 be the orientation when an edge/vertex pair is alive until its death at θ_1. Then, each edge/vertex pair forms a *parameterizable ruled surface* (i.e., a contact region) between θ_0 and θ_1. Let $p_i = (x_0, y_0)$ and $p_{i+1} = (x_1, y_1)$. We take a vector $\vec{v} = \overrightarrow{p_i p_{i+1}}$ and a vector $\vec{t} = \overrightarrow{O q_j}$, where O is the world origin. Then the surface defined by the pair ($\overline{p_i p_{i+1}}$ and q_j) is parameterized as:

$$S_R(r,\theta) = \begin{bmatrix} (y_0 + rv_y)\cos\theta - (x_0 + rv_x)\sin\theta + t_x \\ (x_0 + rv_x)\cos\theta + (y_0 + rv_y)\sin\theta + t_y \\ w\theta \end{bmatrix}, \quad (2)$$

where $r \in [0,1]$, $\theta \in [\theta_0, \theta_1)$ and w is the weighting factor of the rotation.

Surfaces are also formed by the edges of Q as P rotates. Similarly, we let $q_j = (x_0, y_0)$ and $q_{j+1} = (x_1, y_1)$. We take the vector $\vec{v} = \overrightarrow{q_j q_{j+1}}$ and a vector $\vec{t} = \overrightarrow{O p_i}$. The surface for the pair p_i and $\overline{q_j q_{j+1}}$ is parameterized as:

$$S_N(r,\theta) = \begin{bmatrix} y_0\cos\theta - x_0\sin\theta + t_x + rv_x \\ x_0\cos\theta + y_0\sin\theta + t_y + rv_y \\ w\theta \end{bmatrix}. \quad (3)$$

In order to support operations like distance query and line intersection, each surface is stored as a tuple $(p, q, \theta_0, \theta_1)$, where p and q are the indices to the vertices and edges of P and Q, and θ_0 and θ_1 define the *birth and death* orientations of the surface. To construct the surfaces in this representation, we use a sweeping algorithm that updates the convolution at critical orientations (events). Fig. 1(b) shows all the events for each edge of P. To handle each event, we delete two segments from the convolution and create two new segments. For example, at event $n_3\vec{m}_1$ in Fig. 1(b), the pair $\overline{p_3 p_1}$ and q_1 and the pair p_1 and $\overline{q_1 q_2}$ are both dead and the pair $\overline{p_3 p_1}$ and q_2 and the pair p_3 and $\overline{q_1 q_2}$ both become alive. Note that these changes are local, therefore each event can be handled in a constant time, and there can be at most mn events. Moreover the events for each edge of P is simply an offset copy of the normals of Q, so all (sorted) events can be built in linear time. Therefore, the entire computation takes only $\Theta(nm)$ time. This is much more efficient than computing each convolution separately, which takes $\Theta(n^2 m + nm^2)$ time.

For two convex polygons, the surfaces trivially form ∂C-obst, as the surfaces will never penetrate into the interior of the C-obst. However, in the case where one or both of the inputs are non-convex, this is not guaranteed to be the case. This poses fundamental problems in computing the penetration depth on such a solid; for example, the closest point on a non-manifold hull to a query point inside the solid may consequently still be on the interior of the solid.

4.2 C-obst of General Simple Polygons

We will discuss how to compute the M-sum for polygons without rotation first in Section 4.2.1 and then generalize the approach to consider rotation in Section 4.2.2.

4.2.1 The M-sum of General Simple Polygons

We propose a simple filtering-based method to compute the M-sum of simple polygons. Our method can also be considered as a type of convolution-based approach. However, unlike Wein's method [25], the proposed method avoids computing (1) the complete convolution, (2) the arrangement of the segments of the convolution, and (3) the winding number for each arrangement cell.

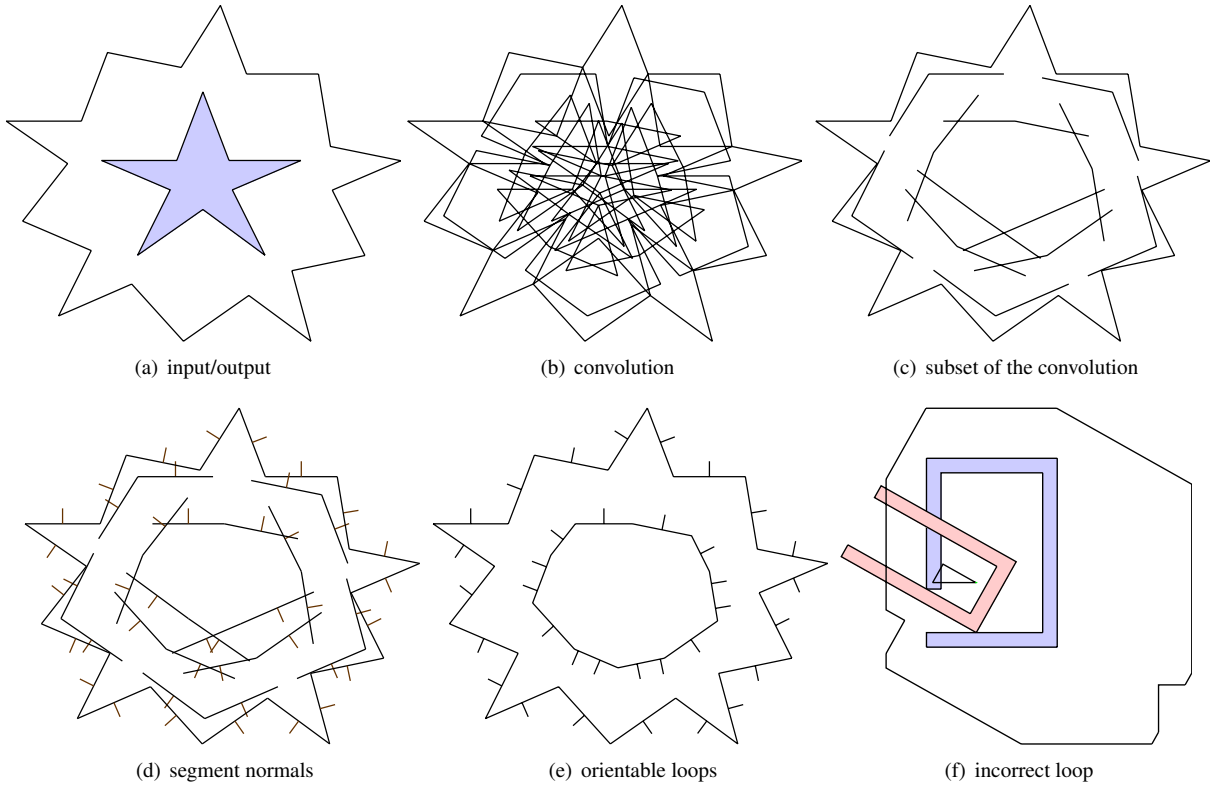
Briefly, our method first computes a subset of the segments that is from the convolution of the inputs, and identifies closed loops that are non-overlapping and *orientable*. These loops form potential boundaries of the M-sum and are further filtered by analyzing their nesting relationship. Finally, the remaining boundaries are filtered by checking the intersections between the input polygons placed at the configurations along these loops. Fig. 2 illustrates these steps.

In the first step of the algorithm, we compute a subset of the segments of the convolution based on the following simple observation.

Observation 4.1. *Given a convolution segment $s = e_P \oplus v_Q$ of an edge e_P and a vertex v_Q, if v_Q is a reflex vertex, s must not be a boundary of the M-sum of P and Q. This observation remains true if $s = v_P \oplus e_Q$ and v_P is reflex.*

Proof. Sketch. Because s must incident to the segments $\mathscr{S}$ formed by the end points of e_P and the edges incident to v_Q, the incident vertex of s and $\mathscr{S}$ is locally non-manifold. Moreover, by definition of convolution, s must be enclosed by the turning range of $\mathscr{S}$. Therefore, s cannot be on the boundary of the M-sum. □ □

Figs. 2(b) and 2(c) show an example of the difference between the complete convolution and the reduced convolution. By definition, the number of edges that are compatible with a reflex vertex must be greater than that of a convex vertex. Due to this, the number of segments filtered by Observation 4.1 is significant, and the size of the problem that we have to consider later is greatly reduced, in particular when the number of the reflex vertices is large. See the more detailed analysis in Section 5.



(a) input/output (b) convolution (c) subset of the convolution

(d) segment normals (e) orientable loops (f) incorrect loop

Figure 2: Steps for computing the M-sum of two simple polygons. In (a), the boundary of the M-sum of a star and a slightly rotated copy of it is shown.

Now, since the segments that we will be working with are no longer a complete convolution, we cannot apply the idea of computing the winding number for each arrangement cell to extract the M-sum boundary as done in [25]. Alternatively, we proceed by defining two additional filters.

Observation 4.2. *we observe that the boundary of the Minkowksi sum must be an orientable loop (if it encloses an area, either positive or negative).*

We say that a loop is orientable if all the normal directions of the edges in the loop are all either pointing inward or outward. Note that the segments we considered are edges from P and Q, therefore, they are directional (as vertices in P and Q are ordered) and include normal directions pointing outward (to P or Q). Fig. 2(d) shows the normals of the segments. Therefore, given two adjacent segments $s = \{u, v\}$ and $s' = \{v, u'\}$ sharing an end point v, we can check if s and s' belong to an orientable loop if

$$\overrightarrow{uv} \times \vec{n}_s = \overrightarrow{vu'} \times \vec{n}_{s'} \ , \tag{4}$$

where $\vec{n}_x$ is the normal vector of segment x, and $\times$ is the cross product. If s and s' satisfy Eq. 4, we say they are compatible segments.

To extract all orientable loops, we compute the intersections of the segments and split all segments at the intersections. A loop is then traced by starting at an arbitrary segment s that has not been considered and then iteratively including compatible segments adjacent to s. Note that there can be multiple compatible segments adjacent to s and all are incident to a single point v. This problem is in fact easy to handle since all M-sum boundaries must be manifold. Thus, we simply pick the segment that makes the largest clockwise turn from s among all the incident segments. Fig. 2(e) shows the loops generated by this step.

Observation 4.3. *The loops must obey the nesting property, i.e., the loops that are* directly *enclosed by the external loop must be holes and will have negative areas, and the loops that are directly enclosed by the holes must have positive areas.*

This is because all loops we generated are all separated (i.e., they don't intersect or touch) due to the manifold properties. The nesting property can be determined efficiently using a plane sweep algorithm, e.g., [2], in $O(n \log n)$ time for n segments. This filter removes the inner loop in Fig. 2(e) because it has a positive area.

So far, we have introduce three quite efficient filters

based on Observations 1 to 3. Unfortunately, not all the remaining loops are the M-sum boundaries. For example, the hole in Fig. 2(e) is a false loop. Therefore, we will have to resort to collision detection to remove all the false loops. That is, we will use the close relationship between the M-sum boundary and the concept of "contact space" in robotics. Every point in the contact space represents a configuration that places the robot in contact with (but without colliding with) the obstacles. Given a translational robot P and obstacles Q, the contact space of P and Q can be represented as $\partial((-P) \oplus Q)$, where $-P = \{-p \mid p \in P\}$. In other words, if a point x is on the boundary of the M-sum of two polygons P and Q, then the following condition must be true:

$$(-P^\circ + x) \cap Q^\circ = \emptyset \,,$$

where Q° is the open set of Q and $(P+x)$ denotes translating P to x. Fig. 2(f) shows a hole loop that passes all the filters except the last filter.

Although there are many methods to optimize the computation time for collision detection, collision detection is more time consuming than the previous filters. Fortunately, it is easy to show that only a single collision detection is needed to reject or accept a loop based on the following lemma.

Lemma 4.4. *All the points on a false hole loop must make P collide with Q.*

Proof. Sketch. Since each loop must belong a cell from the arrangement of the segments in the complete convolution, and all vertices in each cell must to have the same winding number according to [25]. □ □

We will leave the discussion of time complexity until Section 5, and talk about using this method to compute C-obst next.

4.2.2 Handling Rotation

Similar to the algorithm that we proposed for computing the ∂C-obst for convex shapes, the algorithm for the ∂C-obst for non-convex shapes also consists of $\Theta(mn)$ events for creating and deleting each contact patch. Each patch is generated by a segment (with varying length) in the reduced convolution. In addition to these events, the intersection of line segments (from the reduced convolution) also changes during the rotation of P, and these changes can affect the topological structure of the M-sum. Therefore, the second type of event for a given segment s is a list of rotations $\{\theta_i\}$ where the intersection status of s changes (e.g., s starts to or stops intersecting with a segment) when P rotates from 0 to 2π. There can be $O(C^2)$ such events, where C is the size of the reduced convolution.

The data structure that we use for representing the surface is also a tuple $(p, q, \theta_0, \theta_1, s_1, s_2)$, where p, q, θ_0, and θ_1 are the same as the convex case, and s_1 and s_2 are indices to the convolution segments intersecting with the segment between θ_0, and θ_1. We use the same sweeping algorithm to construct this data structure. To handle the events where a segment s is created (or deleted), we simply add (or remove) s to the reduced convolution and add (or remove) the intersections due to s. To handle the second type of event, intersections due to the events are updated. For both events, we check if the event site (i.e., the new or dead intersection) is locally orientable (i.e., Eq. 4) and manifold to decide if a loop (of constant size) should be created or deleted as described above. Note that we will skip the last two filters (i.e., the polygon nesting and the collision detection filters) during sweeping. Both filters will only be needed at the end of the sweep to reject the false 3-d hole boundaries. Similarly, we can show that only one (2-d) point is needed to verify each 3-d boundary; the proof is similar to Lemma 4.4. An example of the results generated by our method is shown in Fig. 4.

In the rest of this section, we will briefly discuss how to detect the second type of event. Although each of these events can be found in constant time, the computation is quite involved and requires us to classify the types of edges and surfaces, i.e., S_R and S_N in Eqs. 2 and 3, since they move (and rotate) in different ways.

Consider two rotating edges in the convolution e_1 and e_2 that may intersect at some θ. We let θ_0 be the first value of θ for which e_1 and e_2 are both alive. An edge e_i lays along a line L_i whose equation is $y_i(\theta) = m_i(\theta)x_i(\theta) + b_i(\theta)$. The intersection $(x(\theta), y(\theta))$ of L_i can be computed so that

$$x(\theta) = \frac{b_2(\theta_0) - b_1(\theta)}{m_1(\theta) - m_2(\theta_0)} \,,$$

where $b_i(\theta) = x_0 \cos\theta + y_0 \sin\theta + t_y - m_i(\theta)(y_0 \cos\theta - x_0 \sin\theta + t_x)$ and $m_i(\theta) = \frac{1+m_i \tan\theta}{m_i - \tan\theta}$, and m_i is the initial slope of L_i. It is trivial to compute $y(\theta)$ from $x(\theta)$. Then from the intersection of the lines, we solve for θ such that the intersection $(x(\theta), y(\theta))$ will fall into the range of the line segments e_1 and e_2. This is done by classifying the segments into three cases that involving *rotating* and *non-rotating* edges. We say that the edges that create S_R surfaces are rotating edges and the edges that create S_N surfaces are non-rotating edges. Therefore e_1 and e_2 can be either (1) both rotating edges, (2) both non-rotating edges or (3) a rotating and non-rotating pair. In certain cases, the segments can be checked quickly to determine if they ever intersect. The details are shown in the Appendix.

5 Experimental Results, Application and Discussion

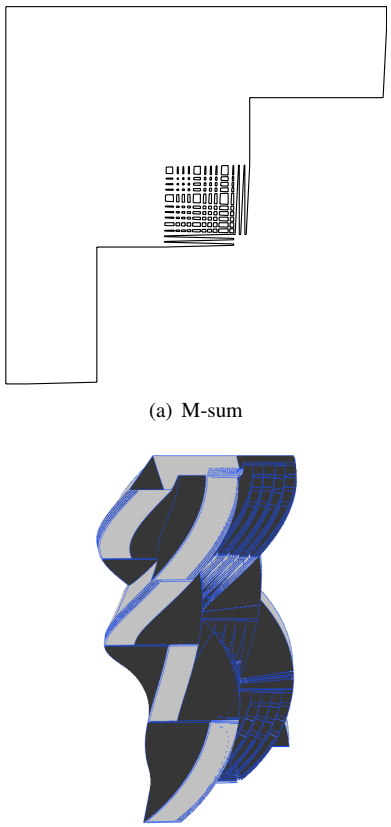
5.1 Results

We have implemented the proposed method in C++. In this section, we will show some of the results that we obtained from this implementation using the examples shown in Fig. 3. In these examples, there are two convex polygons and 9 non-convex polygons. The number of the vertices of each polygon is also shown. Some of these models are inspired by those in [15, 25]. All the experiments are performed on a PC with Intel CPUs at 2.13 GHz with 4 GB RAM.

In Table 1, we show the computation time for constructing ∂C-obst using the proposed method. The running times range from a fraction of a second to close to an hour. Since we have no other implementation to compare to and our implementation is highly unoptimized (for example, our collision detection takes $\Theta(mn)$ for each collision check), it is important to look at these running times relatively. Therefore, we list the the number of ruled surfaces before trimming (N_s), the number of ruled surfaces on the final ∂C-obst (n_s), and the number (external and hole) of C-obst boundaries (n_b). From the values of N_s, n_s, and n_b, it is clear all of them can affect the computation time. For example, both "star/star" and "grate 1/grate 2" take about the same time to compute, but the number of ruled surface patches in "grate 1/grate 2" is half of that in "star/star." Therefore, it is the large n_b in "grate 1/grate 2" that increases the computation time. Moreover, it is clear that the reason that the "dog/bird" takes nearly an hour to finish is because of N_s, which is about 40 times the N_s of "grate 1/grate 2" and "grate 3/grate 4." One single example that we cannot explain from Table 1 is the time difference between "grate 1/grate 2" and "grate 3/grate 4." Both N_s and n_s are smaller and n_b is larger in "grate 1/grate 2."

Fortunately, we can explain this in Table 2. In Table 2, we show the number of segments and the number of intersections in both complete convolution ($N_\otimes$ and $I_\otimes$, resp.) and reduced convolution ($n_\otimes$ and $i_\otimes$, resp.) at the orientation shown in Fig. 3. The reason that "grate 1/grate 2" takes less time to compute than "grate 3/grate 4" does is because "grate 1/grate 2" tends to have smaller $i_\otimes$.

An important observation from Table 2 is the significant difference between $N_\otimes$ and $n_\otimes$. As we have mentioned earlier, when the full convolution is used, a large number of ruled surfaces will be generated, and many of these surfaces will never make to the surface of ∂C-obst. As a result, much computation is wasted on computing the intersections between these surfaces. To make the problem worse, the values for $I_\otimes$ and $i_\otimes$ show that these unnecessary surfaces produce drastically more intersections that those left in the reduced convolution. This difference can also be observed in Fig. 2. This property distinguishes our method from the existing methods [1, 6], which consider all contact regions (surfaces). Therefore, we believe that our method is more efficient.



(a) M-sum

(b) C-obst

Figure 4: The M-sum and C-obst of grate 1 and grate 2 in Fig. 3. The darker (lighter) patches in (b) are S_R (S_N) surfaces. More results can be found in the Appendix.

5.2 Application: Generalized Penetration Depth Estimation

The parameterizations in Eqs. 2 and 3 also yield distance functions in r and θ which can be used to find the minimum distance to a given facet relatively easily. Let p be a query point and let $f(r) = (x_0 + rv_x)$, $g(r) = (y_0 + rv_y)$, $F(r) = g(r)\cos\theta - f(r)\sin\theta$, and $G(r) = f(r)\cos\theta + g(r)\sin\theta$, then the square distance $d(r,\theta,p)$ for the rotating edges:

$$d(r,\theta,p) = (F(r)+t_x-p_x)^2+(G(r)+t_y-p_y)^2+w^2(\theta-p_z)^2$$

If we fix r, then d is a very well-behaved sinusoid, and while there does not seem to be a closed-form solution

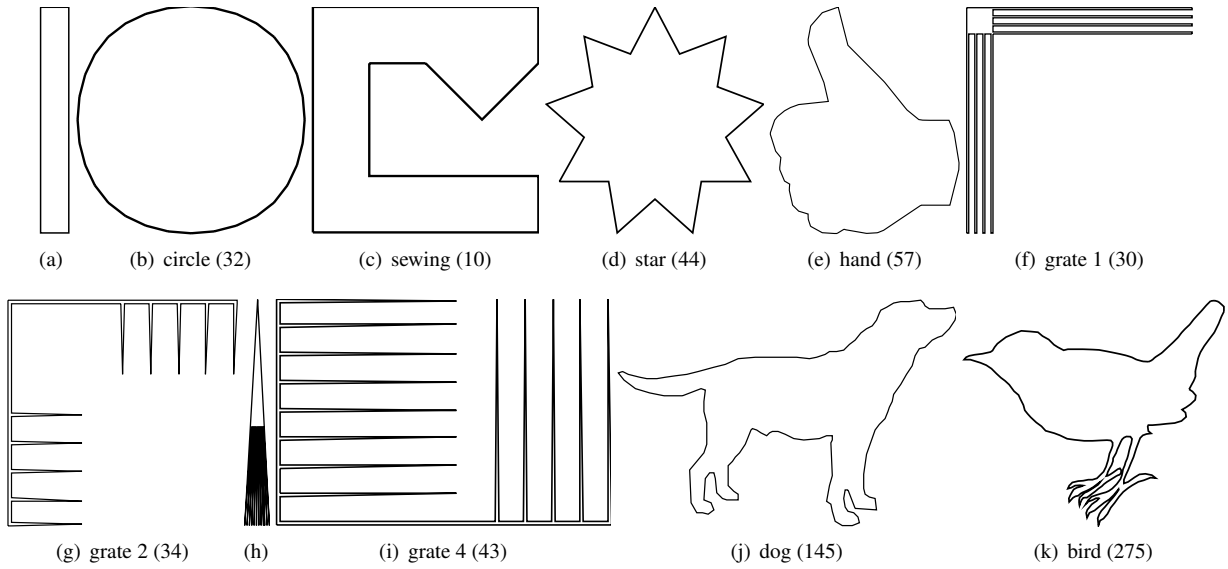


Figure 3: Examples used in the experiments. The numbers in the parentheses are the size of the polygon. (a) bar (4), (h) grate 3 (24). Some of these models are inspired by those in [15, 25].

Table 1: Experimental results for C-space mapping. Here, N_s is the number of ruled surfaces before trimming, n_s is the number of ruled surfaces on ∂C-obst, n_s is the number (both external and hole) C-obst boundaries, and t is the total computation time in seconds.

P/Q	bar/circle	bar/sewing	star/star	star/hand	grate 1/grate 2	bar/grate4	grate 3/grate 4	dog/bird
t	0.1	0.05	4.9	6.8	4.9	0.3	21.7	3350.6
N_s	256	68	2288	3286	1028	244	991	39145
n_s	256	82	3499	3034	4097	1027	1947	18500
n_b	1	1	1	1	126	17	39	1

for the global minimum, it is easy to find the minimum using simple gradient descent. If by contrast we fix θ, then d is simply quadratic in r, and finding the global minimum on $[0,1]$ is also quite easy.

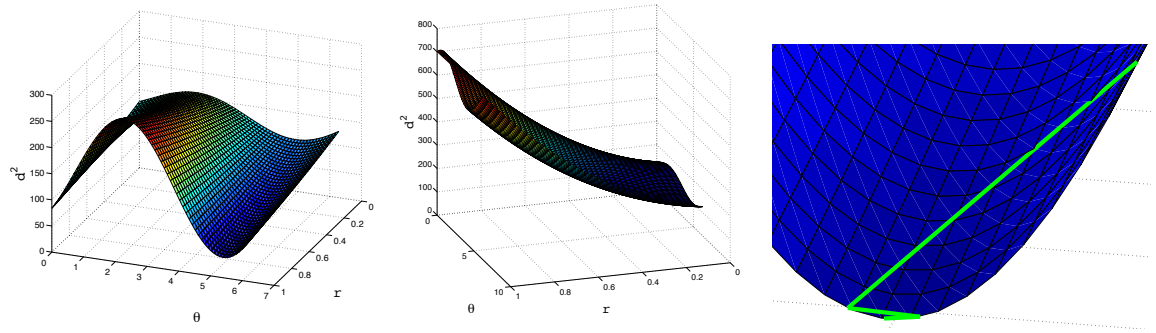
Computing $d(r,\theta)$. In the case of S_R (see Fig. 5(a)), the regularity of the surface of the distance function allows us to easily calculate a global minimum by finding θ values for the global minimums at $r=0$ and $r=1$ by gradient descent, then finding the global minimums for r when we fix θ at the values found by fixing r initially. Picking the minimum of the yielded values gives us the global minimum of the distance function, as well as yield r and θ values which explicitly give us the closest point on the facet (see Fig. 5(c)). The distance function follows similarly for S_N (see Fig. 5(b)), except that because the r term is independent of the rotation, the surface is somewhat more regular. We still end up with no clear closed-form solution for the sinusoid however, so we must solve for the minimum using gradient descent as above.

In the case of non-convex polygons, a surface may have a left-r-bound function $r_{min}(\theta)$ and a right-r-bound function $r_{max}(\theta)$ that describe how its non-manifold intersections move as θ changes, so that its associated facet is r-bounded at a given θ by $[\max\{0, r_{min}(\theta)\}, \min\{1, r_{max}(\theta)\}]$. These same r-bounds apply to the distance function. As a consequence, finding seed values for r and θ in the general case is more complicated. To deal with this issue, we choose to seed at regular intervals. Let segment e have its birth at θ_0 and death at θ_1, then seed values are taken for $\theta \in \{k\frac{|(\theta_0-\theta_1)|}{8} : k \in \mathbb{Z}, 0 \le k \le 8\}$. For each of these θ values, we seed at $\max\{0, r_{min}(\theta)\}$, $\min\{1, r_{max}(\theta)\}$, and $(\max\{0, r_{min}(\theta)\} + \min\{1, r_{max}(\theta)\})/2 \pm \varepsilon$, just to the left and right of the medial axis of the r-bounds.

This gives us a total of 36 seeds per surface. We use so many seeds largely because the r-bounds are irregular enough that some descents may get caught along the boundary. This spread however provides good coverage. Because of the regularity of the surface itself, a particular iteration of the gradient descent tends to converge in a small number of iterations and so the total cost of the gradient descent is relatively low in any case.

Table 2: Experimental results for M-sum computation. Here, $N_{\otimes}$ is the number of segments in the convolution, $n_{\otimes}$ is the number of segments in the reduced convolution, $I_{\otimes}$ is the number intersections in the convolution and $i_{\otimes}$ is the number intersections in the reduced convolution.

P/Q	bar/circle	bar/sewing	star/star	star/hand	grate 1/grate 2	bar/grate4	grate 3/grate 4	dog/bird
$N_{\otimes}$	36	42	1608	1689	1394	191	1162	38342
$I_{\otimes}$	36	33	1300	2758	5243	131	10136	255635
$n_{\otimes}$	36	30	382	281	469	92	400	2921
$i_{\otimes}$	36	22	257	297	1204	77	1544	3742



(a) (b) (c)

Figure 5: (a) An example distance function for S_R, $c = (0,0), p = (10,-5,\pi)$, $v = (1,4)$, $x_0 = 1, y_0 = 1$. (b) An example distance function for S_N, same parameters as (a) except that v is elongated. (c) An example gradient descent for the sinusoid portion zoomed in on the gradient descent. The descent converges in just 5 iterations (two iterations in which only step-size is adjusted), and in this case finds not only the local minimum for the sinusoid at r_{max}, but also the global minimum.

Computing penetration depth. Given a configuration p of P, we would like to find the closest feature on ∂C-obst. This problem can be decomposed into two steps: (1) find the closest surface f to p and (2) find the closest point on f to p. We have already proposed a method for the second step. For finding the closest surface, ideally, we can precompute the *Voronoi tessellation* of the space using each surface as a site, and then find which cell q is in. However, both computing the tessellation and finding the enclosing cell seem to be difficult. The only properties that we know are that the boundaries of the tessellation are also ruled surfaces, and each cell forms a single connected component. Based on these properties, we propose a sampling-based approach. Initially, a set of uniformly distributed samples are taken, and the closest surface for each sample point is computed offline using a brute-force search (through all surfaces). Each query point is then categorized by its k nearest neighbors, and only the $n \leq k$ surfaces associated with those neighbors are checked. For the results in Table 3, we set $k = 10$ experimentally. For convex polygons, this approach yields a very high rate (98.3% for bar/circle) of identifying the actual closest facet and low average error values ($< 10^{-5}$) when a facet other than the closest is chosen for distance comparison. For non-convex polygons, this approach still yields very high rate ($> 95.7\%$) of identifying the actual closest facet and low average error values ($< 10^{-3}$). The accuracy and error are estimated by comparing to the results of the brute force approach.

5.3 Discussion

We will analyze the time complexity of the proposed method and discuss the extension of our method to handle 3-d convex polyhedra.

Complexity Analysis. When P and Q have n and m vertices which include n' and m' reflex vertices, respectively, there will be $2mn$ segments in the complete convolution; in the reduced convolution there are at most $(m-m')n + (n-n')m$ segments. That is, the arrangement of the reduced convolution is at least 4 times less complex than that of the complete convolution when $n' = 1/2n$ and $m' = 1/2m$. Thus, the reduction will further reduce the complexity of the arrangement of 3-d rule-surface patches by at least 8 times. Note that this analysis is based on the assumption that a convex vertex is compatible with $\Theta(n)$ edges and in the worst

Table 3: Results for penetration depth estimation. Here ε is the avg. distance error, t is the avg. query time over 1000 queries, and T is the time to pre-compute the distances for all samples.

P/Q	bar/circle	bar/sewing	star/star	grate 1/grate 2
ε	0.000004	0.00067	0.0004	0.0001
t	7.8ms	6.5ms	8.5ms	12.0ms
T	50.1s	80.1s	398.2s	862.2s

case that each segment will intersect all the other segments. In the examples that we have above, the difference between the reduced and complete convolutions is more significant (e.g., "star/star" and "dog/bird"). The time complexity for computing the M-sum of P and Q is $O((mn+I)\log(mn+I)+\ell T_{cd})$, where $I = O(m^2n^2)$ is the complexity of the arrangement of the reduced convolution, ℓ is the number of loops, and $T_{cd} = O(mn)$ is the collision detection time in our implementation. The time complexity for computing the C-obst of P and Q is $O\left((mn+I)\log(mn+I)+m^2n^2T_e+bT_{cd}\right)$, where b is the number of (hole) boundaries in the C-obst, and $T_e = O(mn)$ is the time for handling each event (i.e., finding all new/dead intersections and update the M-sum locally near the intersections).

C-obst of Convex Polytopes. Computing the C-obst of *3-d polytopes* is similar but involves more steps in creating and handling events. An important observation is that the event list of each edge of P (see Fig. 1(b)) is simply an offset of Q's Gaussian map! Similarly, given polytopes P and Q, the events for each facet of P can be constructed by rotating Q's Gaussian map. In 3-d, the facets of the M-sum can only come from two sources: fv-facets, generated from a facet of P and a vertex of Q or vice versa, and ee-facets, generated from a pair of edges from P and Q, respectively. Similar to polygons, a M-sum facet (f_M) is valid if the orientations of f_M's primitives (i.e., a vertex-facet pair or an edge-edge pair) from P and Q match each other, and the events occur at these boundary conditions, where the M-sum structure changes. Finally, all the events can be enumerated by overlaying all the rotated Gaussian maps (one for each P's facet).

The resulting data structure for representing the C-obst of polytopes is a list of facets. Each facet f_M is a tuple (p,q,r), where p and q are the indices to the vertices, edges or facets of P and Q, and r is a convex region (in the Gaussian map) in which f_M remains valid. One can show that the complexity of this data structure is $O(n^2m+nm^2)$, where n and m are the complexities of P and Q. If a brute force method is used to enumerate all possible M-sums, it will take $O(n^3m^2+n^2m^3)$.

6 Conclusion and Future Work

We proposed a new method for constructing ∂C-obst. Our methods takes $O(m^3n^3+bT_{cd})$ for polygons with m and n vertices and C-obst with b boundaries, where T_{cd} is the collision detection time. We believe that this method is easier to implement and more efficient than the existing methods. The main step in our method is the computation of the Minkowski sum (M-sum) which is based on the ideas of reduced convolution, orientable and manifold loops, and polygon nesting and collision detection filters (discussed in Section 4.2). Then the M-sum is updated at each critical orientation to construct ∂C-obst. The main efficiency gain is that there are significantly fewer segments and surfaces produced compared to the existing methods. The evidence supporting this observation was shown and discussed in Section 5.

We consider this work as the first step toward a more interesting and challenging problem: computing the C-obst of 3-d polyhedra. Donald [10] and several others have studied the problems, but we believe that there is still room for significant improvement. For example, most of these methods depends on convex decomposition. As we have discussed above, computing C-obst for convex polyhedra is not difficult. However, it is unclear how to deal with non-convex polyhedra without using 3-d convex decomposition, which is notoriously slow. We hope to provide an answer to this by using the ideas from the recent development in M-sum and the ideas from this paper.

References

[1] F. Avnaim and J. Boissonnat. Polygon placement under translation and rotation. *STACS 88*, pages 322–333, 1988.

[2] C. Bajaj and T. K. Dey. Polygon nesting and robustness. *Inform. Process. Lett.*, 35:23–32, 1990.

[3] B. S. Baker, S. J. Fortune, and S. R. Mahaney. Polygon containment under translation. *J. Algorithms*, 7:532–548, 1986.

[4] M. Branicky and W. Newman. Rapid computation of configuration space obstacles. In *Proc. IEEE Int. Conf. Rob. and Aut*, pages 304–310, 1990.

[5] A. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. *Algorithmica*, 8(1):195–208, 1992.

[6] R. Brost. Computing metric and topological properties of configuration-space obstacles. In *1989 IEEE International Conference on Robotics and Automation, 1989. Proceedings.*, pages 170–176, 1989.

[7] M. Caine. The design of shape interactions using motion constraints. In *1994 IEEE International Conference on Robotics and Automation, 1994. Proceedings.*, pages 366–371, 1994.

[8] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[9] B. Curto and V. Moreno. Mathematical formalism for the fast evaluation of the configuration space. In *cira*, page 194. Published by the IEEE Computer Society, 1997.

[10] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Art. Intell.*, 31(3):295–353, 1987.

[11] B. R. Donald. The complexity of planar compliant motion planning under uncertainty. *Algorithmica*, 5:353–382, 1990.

[12] P. K. Ghosh. A unified computational framework for Minkowski operations. *Computers and Graphics*, 17(4):357–378, 1993.

[13] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[14] D. Halperin, M. H. Overmars, and M. Sharir. Efficient motion planning for an L-shaped object. *SIAM J. Comput.*, 21(1):1–23, 1992.

[15] D. Halperin and M. Sharir. Arrangements and their applications in robotics: Recent developments. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 495–511, 1995.

[16] Y. Hwang. Boundary equations of configuration obstacles for manipulators. In *1990 IEEE International Conference on Robotics and Automation, 1990. Proceedings.*, pages 298–303, 1990.

[17] L. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. *IEEE Trans. Robot. & Autom*, 11:255–261, 1995.

[18] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 6th edition, 2006.

[19] J.-M. Lien. A simple method for computing Minkowski sum boundary in 3d using collision detection. In *The Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, Mexico, Dec 2008.

[20] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[21] S. Nelaturi and V. Shapiro. Configuration products in geometric modeling. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 247–258, 2009.

[22] E. Sacks and C. Bajaj. Sliced configuration spaces for curved planar bodies. *The International Journal of Robotics Research*, 17(6):639, 1998.

[23] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graph. Models*, 68(4):343–355, 2006.

[24] J. Ward and J. Katupitiya. Free space mapping and motion planning in configuration space for mobile manipulators. In *2007 IEEE International Conference on Robotics and Automation*, pages 4981–4986, 2007.

[25] R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *Proc. 14th Annual European Symposium on Algorithms*, pages 829–840, 2006.

[26] K. Wise and A. Bowyer. A survey of global configuration-space mapping techniques for a single robot in a static environment. *The International Journal of Robotics Research*, 19(8):762–779, 2000.

[27] D. Zhu and J. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7(1):9–20, 1991.

Appendix

Compute the intersection of two rotating line segments. Consider two edges in the convolution e_1 and e_2 that may intersect at some θ. We let θ_0 be the first value of θ for which e_1 and e_2 are both alive. An edge e_i lays along a line L_i whose equation is $y_i(\theta) = m_i(\theta)x_i(\theta) + b_i(\theta)$. To compute the change in intersection as e_1 changes with θ but e_2 remains stationary, we compute:

$y_1(\theta_0) = y_2(\theta) = m_1(\theta)x_1(\theta) + b_1(\theta) = m_2(\theta_0)x_2(\theta_0) + b_2(\theta_0)$
for $x_1(\theta) = x_2(\theta_0) = x(\theta) = \frac{b_2(\theta_0) - b_1(\theta)}{m_1(\theta) - m_2(\theta_0)}$.

If we then hold the transformation of e_1 stationary and compute the intersection for the transformation of e_2, by similar computation we yield the composite computation: $x(\theta) = \frac{b_1(\theta) - b_2(\theta)}{m_2(\theta) - m_1(\theta)}$. It is clear from this that the order of transformations is inconsequential, as it should be. In order to compute $x(\theta)$ however, we need to compute $b(\theta)$ and $m(\theta)$ for any edge e. To compute $b(\theta)$, we use the point-slope form of the line.

$$y(\theta) - S_y(0,\theta) = m(\theta)(x(\theta) - S_x(0,\theta))$$

$$y(\theta) = m(\theta)x(\theta) - m(\theta)S_x(0,\theta) + S_y(0,\theta) = m(\theta)x(\theta) + b(\theta)$$

$$b(\theta) = S_y(0,\theta) - m(\theta)S_x(0,\theta)$$

When e is a non-rotating edge, $m(\theta)$ is a constant m, and $S_y(0,\theta) = S_{Ny}(0,\theta)$, $S_x(0,\theta) = S_{Nx}(0,\theta)$:

$$b(\theta) = x_0\cos\theta + y_0\sin\theta + t_y - m(y_0\cos\theta - x_0\sin\theta + t_x)$$

The derivation follows similarly when e is a rotating edge, except that $m(\theta)$ is not constant, and so:
$m(\theta) = \frac{S_{Ry}(1,\theta) - S_{Ry}(0,\theta)}{S_{Rx}(1,\theta) - S_{Rx}(0,\theta)} = \frac{1 + m_0\tan(\theta)}{m_0 - \tan(\theta)}$

$$b(\theta) = x_0\cos\theta + y_0\sin\theta + t_y - m(\theta)(y_0\cos\theta - x_0\sin\theta + t_x)$$

$$b(\theta) = x_0\cos\theta + y_0\sin\theta + t_y - \left(\frac{1 + m_0\tan(\theta)}{m_0 - \tan(\theta)}\right)(y_0\cos\theta - x_0\sin\theta + t_x)$$

Now that we have forms for $m(\theta)$ and $b(\theta)$ for both types of edges, we can compute $x(\theta)$ for any pair of edges, and the boundaries of a facet can be defined by it. At each event point, the segment is split into parts according to its intersections. We compute the θ boundaries for each intersection point, and these must become a second type of event: when an intersection is born, the facet must be split and culled again–when an intersection dies, parts of the facet must be merged and restored.

Computing these θ boundaries however is non-trivial. We cannot easily solve for them from $x(\theta)$, as the x and y ranges of the segments vary depending on the segment and its orientation, so for each type of edge, we look to compute the function $r(\theta)$, which gives us the r-value for a given intersection at θ. The birth and death of intersections between two segments on $\theta \in [0, 2\pi)$ can only occur at endpoints of at least one of the two segments unless those two segments become colinear, in which case the intersection will still occur at the overlapping endpoints at birth and disappear at death at the same event.

Endpoints are at $r = 0$ and $r = 1$. If we solve $r(\theta)$ for θ at the end-points for both edges, the minimum θ-value we find is the birth of the intersection, and each subsequent θ-value alternates between death and birth, respectively. We cull these according to the birth and death events of the segments, leaving us with a list of intersection events for these two segments. Note that it is easier to solve for $r = 0$ than it is to solve for $r = 1$. However, we can solve for $r = 0$ for both endpoints by using $-v$ instead of v, and picking the opposite endpoint to use as the reference point (x_0, y_0) when we wish to solve for $r = 1$ in the initial frame of reference.

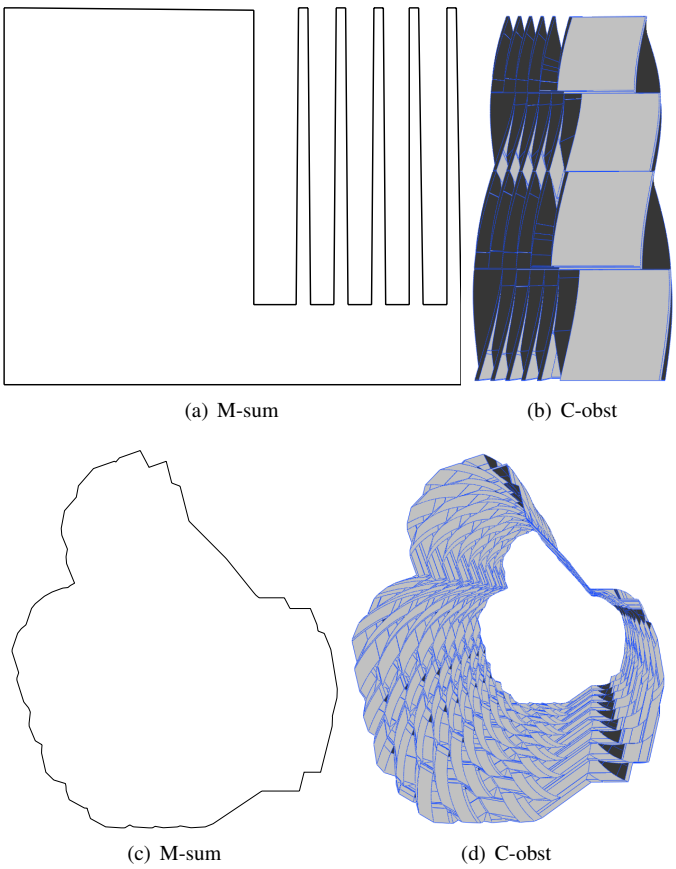
To solve for $r(\theta) = 0$, we set the parameterization of the line's x-coordinate equal to the intersection function $x(\theta)$ when $r = 0$ and solve for θ:

$$x(\theta) - y_0\cos(\theta) - x_0\sin(\theta) + t_x = 0 .$$

When both edges are non-rotating, then the equation simplifies to the form: $\alpha\cos(\theta) + \beta\sin(\theta) + \gamma = 0$. Using that $\alpha\cos(\theta) + \beta\sin(\theta) = \sqrt{\alpha^2 + \beta^2}\sin(\theta + \delta)$ where $\delta = \arcsin(\frac{\beta}{\sqrt{\alpha^2+\beta^2}})$, we get that: $\theta = \arcsin\left(\frac{-\gamma}{\sqrt{\alpha^2+\beta^2}}\right) - \delta$. Because of this, it is easy to check whether lines will ever intersect: if $|\frac{-\gamma}{\sqrt{\alpha^2+\beta^2}}| > 1$ then θ will have an imaginary component and the segments will never intersect.

Unfortunately, in all other cases the resulting form is quite complex and has no readily-apparent closed-form solution. In these cases we must perform gradient descent on $|r(\theta)|$. Proper seeds are sufficient to find the minima of the function. There is no faster test at present to determine if the segments intersect than to check if any of the minima are 0.

Additional results. We show more examples generated by the proposed method.



(a) M-sum (b) C-obst

(c) M-sum (d) C-obst

Figure 6: (a) and (b) are generated from bar and grate 3 in Fig. 3. (c) and (d) are generated from star and hand in Fig. 3.