Fast Medial Axis Transform for Planar Domains with Curved Boundaries

Francisco de Moura Pinto   Carla Maria Dal Sasso Freitas

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
`fmpinto@inf.ufrgs.br`   `carla@inf.ufrgs.br`

Luiz Henrique de Figueiredo

IMPA – Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil
`lhf@impa.br`

Abstract

We present a new, linear-time approach for computing the medial axis transform of planar regions with curved boundaries given by general parametric curves. The core of our approach is an implicit representation of the medial axis via the radial local feature size, which gives the distance from a boundary point to the internal medial axis measured along the normal direction to the boundary at that point.

Keywords: medial axis, medial axis transform, curved boundaries, offset curves

1. Introduction

The *medial axis* (MA) of a region in the plane is the closure of the locus of the centers of all maximal discs contained in the region (see Fig. 1 for an example). The *medial axis transform* (MAT) is the MA together with its associated radius function, which gives the radius of the maximal disc associated to each point of the MA. The medial axis was introduced in 1967 by Blum [1] for describing shapes. Since then, the MA and the MAT have been useful in applications in several fields.

There are two major approaches to computing medial axes, according to the application: the discrete approach and the continuous approach. The discrete approach extracts the medial axis from binary images and finds use in applications such as pattern recognition and image analysis. The continuous approach, which we follow in this paper, extracts the medial axis from continuous descriptions of region boundaries. The medial axis transform is a powerful tool for reasoning about geometry and shape [2] and is useful in geometric processing tasks such as path planning and mesh generation [3, 4].

The medial axis transform has been investigated for many years and several algorithms have been proposed for computing it [5]. However, very few methods impose no restrictions on the nature of the region's boundary. Many algorithms handle only simple polygons. In this important particular case, the medial axis can be computed exactly [6] and there is a linear-time solution [7] as well as other solutions that are efficient in practice and easier to implement [8, 9]. Despite the importance of regions with mixed straight and curved boundaries in computer-aided geometric design (CAGD), few algorithms can compute the medial axis of such shapes [5, 10] and, to the best of our knowledge, none achieves linear execution time. In this paper, we present a new, linear-time approach for computing the medial axis transform of planar regions with mixed straight and curved boundaries given by general piecewise parametric curves.

The core of our approach is an *implicit representation* of the MA via a scalar function defined on the boundary of the region. For each boundary point, this function gives the distance from the point to the internal medial axis measured along the normal direction to the boundary at that point. In other words, this function gives the radius of the maximal disc that is tangent to the boundary at the given point. Accordingly, we call this function the *radial local feature size* (RLFS) in analogy to the well-known local feature size (LFS) function [11]. The RLFS differs from the LFS in two aspects: while LFS gives the distance to the MA, RLFS gives the distance to the MA in the normal direction; moreover, RLFS neglects the external MA and considers only distances to the internal MA. The RLFS is sufficient to recover the MAT because every point on the MA can be obtained from its footprints on the region's

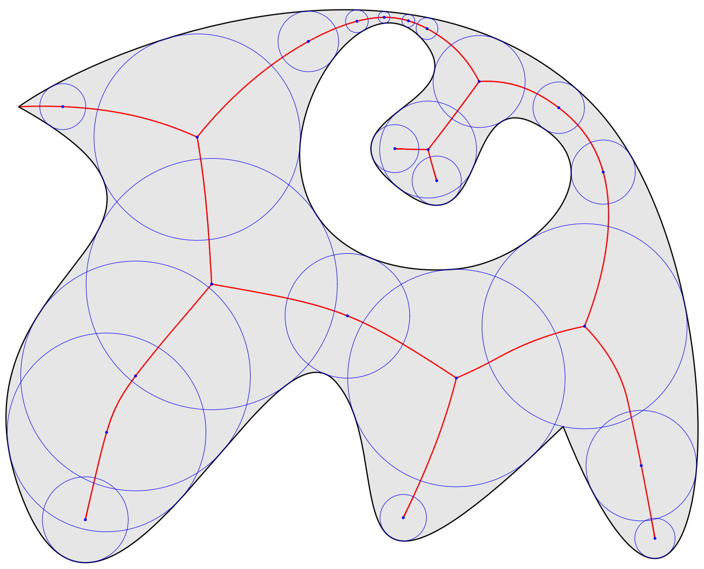


Fig. 1: The medial axis of a region (shown in red) is the locus of the centers of all maximal discs contained in the region. Each maximal disc touches the boundary of the region in at least two points, which are the footprints of the disc.

boundary and the radius of the corresponding maximal disc, which is given by the RLFS function (the *footprints* of an MA point or of its corresponding maximal disc are the points where the disc touches the region's boundary). The proposed representation of the MA is implicit because the topology of the MA is not explicit, even though in principle one can compute as many points on the MA as desired from the RLFS function and a point sample of the boundary.

We compute the RLFS by simulating the *grass-fire* model [1], which is a well-known alternative, but equivalent, definition of the medial axis. In the grass-fire model, the region is a garden covered with grass that starts burning from the boundaries, forming fire fronts that propagate inward isotropically with constant speed. All boundary points are simultaneously and gradually displaced along their normals like a fire front in propagation. The MA is then the locus of all points where a fire front collides with itself or with other fire fronts; the distance from the boundary to the collision point is the radius of the corresponding maximal disc. Evaluating this distance is thus equivalent to evaluating the RLFS.

Preliminary results of our research were presented in the SIBGRAPI 2009 conference [12]. We made the following extensions to that work: improvement of results through adaptive sampling; a robust algorithm to explicit the medial axis that do not suffer from high order MA branching points; a simple algorithm for computing offset curves; a simple parallel implementation of the core of our method; and a careful evaluation of our results regarding correctness and efficiency. Besides, we bring a much richer and easier-to-follow exposition of our research work.

The remainder of this paper is structured as follows. Section 2 reviews the closest related work and the state of the art in algorithms for computing the medial axis transform of continuous planar regions with curved boundaries. Section 3 describes the proposed implicit representation of the medial axis and the basic mechanism to construct it, as well as an efficient strategy to simulate the grass-fire model in order to compute the RLFS function. Section 4 describes an efficient algorithm for extracting a graph-like representation of the MAT from the RLFS function. Section 5 describes a simple algorithm for computing offset curves from the RLFS function. We show and discuss the results obtained with our approach in Section 6, including extensions to compute region skeletons like the medial axis but based on a wider set of structuring elements, besides the disc, without changing the core algorithm. Section 7 contains our final comments and some directions for future work.

2. Related work

Medial axis transform is not a new research topic and giving a wide overview of the vast related literature is beyond the scope of this paper. This section completely neglects algorithms for discrete MAT and focuses instead on continuous MAT, with emphasis on approaches that are closely related to ours. For a short survey on algorithms for continuous MAT, see [5].

Many methods for continuous MAT are restricted to regions enclosed by simple polygons and take advantage of the large background that exists for the computation of Voronoi diagrams. Indeed, medial axes and Voronoi diagrams are closely related: the MA of a simple polygon [8, 7] is a subset of the edges of the Voronoi diagram constructed by taking as sites the polygon's edges and reflex vertices (vertices whose inner angles are greater than π). Lee [8] developed an easy-to-implement algorithm that runs in time $O(n \log n)$, where n is the number of edges in the polygon. Most previous approaches were slower, typically $O(n^2)$; those that also ran in time $O(n \log n)$ were harder to implement. Despite the popularity and relevance of the MAT and the amount of research in finding efficient algorithms for it, a linear-time solution for computing the MAT of a general simple polygon was only found in 1995 by Chin et al. [7]. Their algorithm is still the state of the art in computing the MAT for simple polygonal regions.

The close relation between Voronoi diagrams and MAT inspired methods for computing the medial axis of shapes implicitly defined by point clouds [13]. Such approaches essentially compute the Voronoi of the point cloud and trim the resulting diagram. Geisen et al. [14] provided a simple and robust algorithm based on the Voronoi diagram for approximating the MA of smooth n-dimensional shapes. They proved the correctness of their algorithm when the shape's boundary is sampled according to the local feature size and its inner region is approximated as the union of the inner Voronoi balls of the sample points. The same authors proposed a method for simplification of the MA that is sensitive to the shape's local scale [15]. The MAT of point clouds is especially useful in surface reconstruction [16]. Recent approaches to isosurface extraction employ the MAT for assessment of the surface's local feature size, which defines the required sampling density for accurate reconstruction of isosurfaces of scalar fields [17, 4].

Continuous MAT for general planar regions encounters solid mathematical foundations in the works of Choi et al. [18, 19, 20] and of Ramamurthy and Farouki [21, 22]. Closest to our approach is the work of Ramanathan and Gurumoorthy [23] and Cao et al. [10]. Ramanathan and Gurumoorthy [23] trace the MA by marching along the boundary of the region. They start the march from a convex vertex on the boundary; such a vertex corresponds to an extreme point of the medial axis. From this vertex, they step along the boundary in the two opposite directions simultaneously, tracking pairs of footprints of maximal discs. The MA points are found by intersecting the normal lines at the corresponding footprints. Cao et al. [10] improved this strategy by defining the tracking of MA points as an integration process. Their algorithm integrates the Taylor expansion of a differential equation that defines the MA. The Taylor series coefficients are obtained from the differential properties of the boundary at the footprints of maximal discs. These two algorithms suffer from the high computation cost of finding branching points in the MA, where the tracking process must split. Aichholzer et al. [24] proposed an efficient divide-and-conquer algorithm to compute the MA of complex planar regions with no holes. They used a biarc approximation of the original boundary to simplify the MA extraction process. The same authors later [25] extended their method to compute Voronoi diagrams for arbitrarily-shaped sites and to extract the medial axis of planar regions with holes.

As we explain in detail in the next section, our method is similar to the one by Ramanathan and Gurumoorthy [23] in the sense that we also trace the medial axis from its extreme points by marching along the boundary and tracking the footprints of the maximal discs using normals. Our approach also produces an approximate medial axis of an arbitrarily-shaped planar region. However, our novel implicit representation of the medial axis avoids additional costs in finding the branching points, and we do not need to assume the existence of a convex vertex. While their method has quadratic execution time as a function of the number of boundary sample points, we achieve linear time with ours.

3. Computing the RLFS function

As mentioned in Section 1, we find an implicit representation of the MAT via the RLFS function, which gives the distance from each boundary point to the internal medial axis measured along the normal direction at that point. In this section, we explain how we compute the RLFS function using the grass-fire model.

3.1. Boundary sampling

Computing the exact RLFS function is a very difficult task for regions with general curved boundaries. Like other methods that handle curved boundaries [23, 10, 14], our method works on a discrete approximation of the boundary obtained through point sampling and, accordingly, produces an approximation of the MA. However, unlike

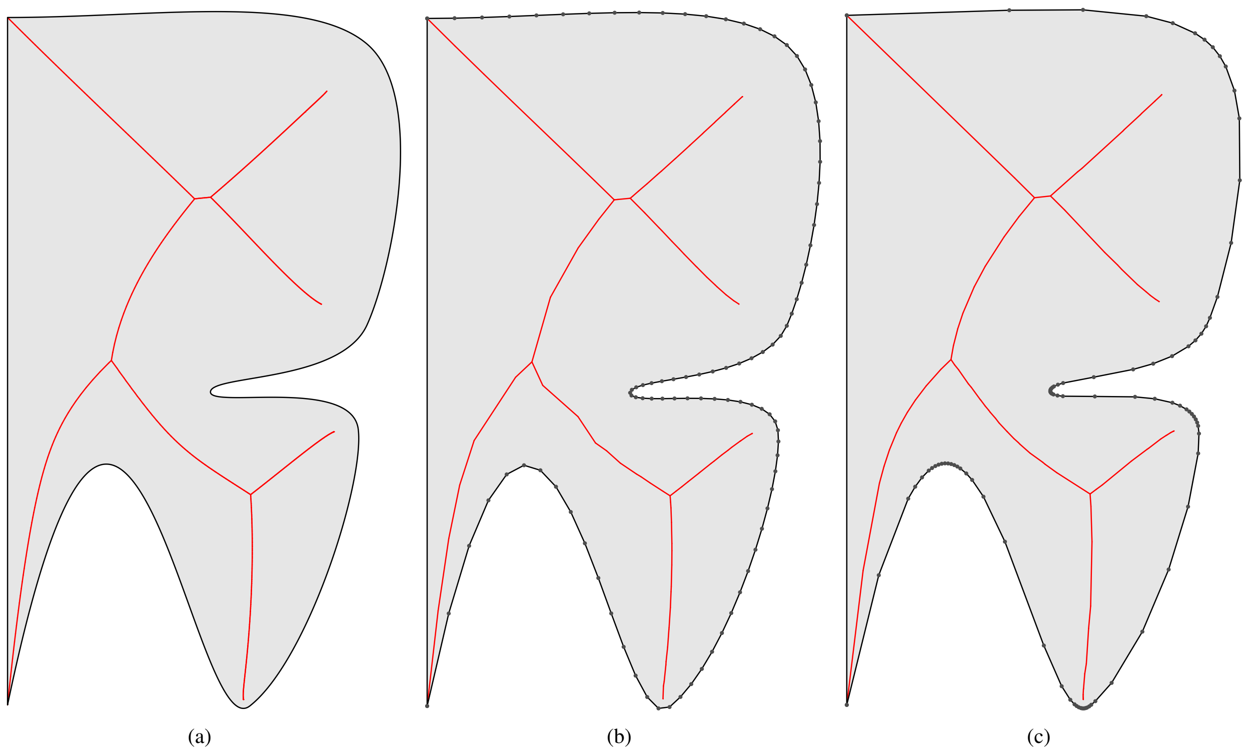


Fig. 2: (a) A region bounded by a straight-line segment and cubic Hermite curves. The boundary was sampled very densely for plotting and for computing a good approximation of the medial axis using our method. (b) A rough uniform sampling of the boundary in parametric space leads to a poor approximation of the MA. (c) Curvature-oriented adaptive sampling using the same number of samples as in (b) leads to a much better approximation of the MA.

methods based on the Voronoi diagram of the samples [14], our method also takes into account the exact normals to the boundary at the sample points. Thus, although the employed discrete representation of the boundary is a set of straight-line segments connecting adjacent samples, our method is not a polygon-based method because the endpoints of the segments preserve the original normals at the corresponding samples.

Like Choi et al. [18, 5], we assume that the region is the closure of a connected bounded open subset in the plane bounded by a finite number of mutually disjoint simple closed curves, and that each boundary curve is a continuous piecewise analytic curve. At a point where the boundary is not smooth, two curves meet and two normals are defined at that point. A simple and very common way to fulfill this assumption is to take each boundary curve as a parametric piecewise polynomial curve (see Fig. 2a). This also makes it easy to sample points on the boundary with the corresponding normals. However, our method is not restricted to piecewise polynomial boundaries.

The input to our algorithm is thus a set of one or more closed chains of straight-line segments. One chain represents the outer boundary, oriented counterclockwise. When the region has holes, the other chains represent the inner boundaries, oriented clockwise. Each segment approximates a piece of a boundary curve. The endpoints of the segments are on the boundary of the region and have associated normals pointing towards the interior of the region. We store the segments of each chain in a dual data structure that allows both random access to the segments through indices and sequential access that gives the *next* and the *previous* segments for a given segment.

The boundary of the region must be sufficiently sampled to be well approximated by line segments. Uniform sampling in parametric space often leads to acceptable results, but better MA approximations are obtained when the sampling density is guided by the curvature of the boundary curves [26]. Polygonal portions of the boundary do not need additional discretization and can be directly provided as input to the algorithm. Fig. 2 illustrates the difference between MA computed based on uniform and on adaptive boundary sampling.

Special treatment must be given to *reflex vertices*, i.e., vertices whose inner angles are greater than π. On curved

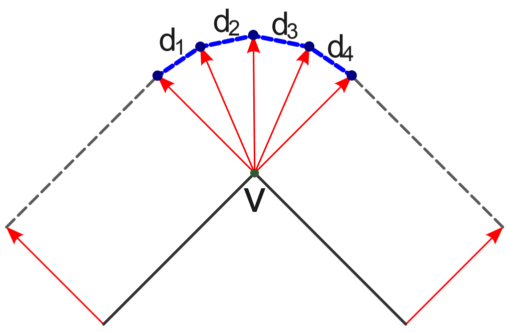


Fig. 3: Reflex vertices, like V, are split into dummy segments with zero length. Smoothly varying normals are assigned to the created vertices. During the simulation of the grass-fire model, when the boundary is displaced along the normals (red), the dummy segments (d_i) are stretched; this ensures that the boundary remains closed.

boundaries, a point of normal discontinuity (having only C^0 continuity) that has an inner angle greater than π must also be considered a reflex vertex. Following Ramanathan and Gurumoorthy [23], we split each reflex vertex into a few zero-length *dummy segments* with normals varying continuously for the created vertices from the normal of the previous incident edge to the normal of the next incident edge (see Fig. 3). The number of dummy segments for a reflex vertex is proportional to the difference between the inner angle and π (we use one dummy segment per $\pi/18$ as default) and must be at least 1. The role of dummy segments is ensuring that the boundary remains closed during the simulation of the grass-fire model.

3.2. Fire-front collision

To simulate the grass-fire model, we simultaneously displace each boundary segment by moving its endpoints along their corresponding normal direction (from the original boundary), until the segment collides with other moving segments. The RLFS at a boundary point is the distance at which the first collision involving that point occurs.

We do not have to perform a temporal simulation of the fire-front propagation because we can test algebraically whether two segments will collide during propagation. Two moving segments collide when an endpoint of one segment collides with a point of the other segment. Thus, an endpoint P with normal N collides with the moving segment P_1P_2 after traveling a distance d when

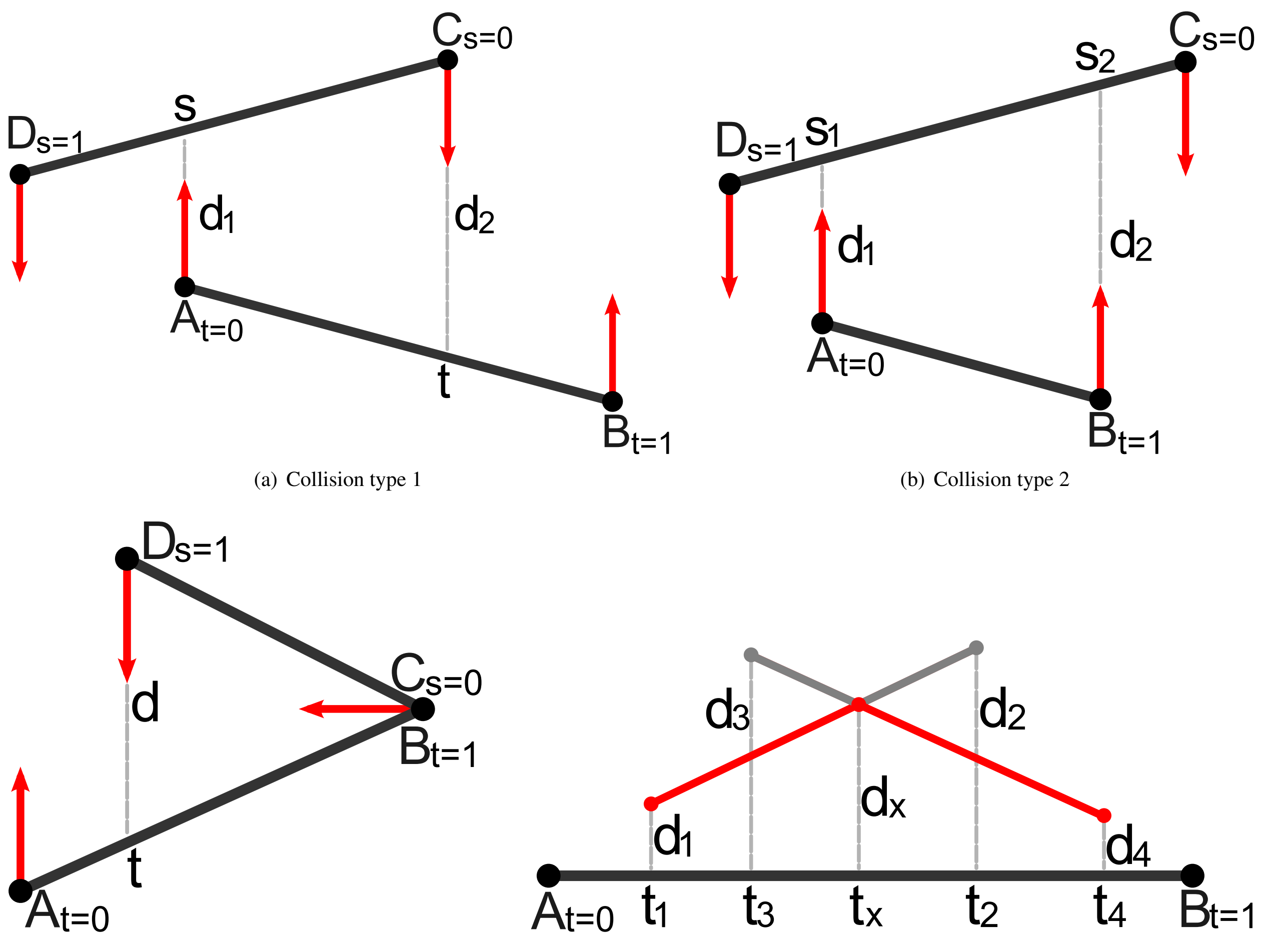
$$P + dN = (1 - t)(P_1 + dN_1) + t(P_2 + dN_2) \tag{1}$$

where N_1 and N_2 are the normal vectors at P_1 and P_2, and $t \in [0, 1]$ parameterizes the moving segment. This equation is actually a non-linear system of two equations in two unknowns (d and t). In the generic case, the system has two solutions, since the substitution method leads to a quadratic equation. In some special cases, the system leads to a linear equation, and thus to a single solution. Such cases must be identified and properly handled. Moreover, we are only interested in solutions having $d > 0$ and $t \in [0, 1]$. Four tests are required to check the collision between two moving segments, since both endpoints of the first segment must be checked for collision against the second segment and vice-versa. However, most such endpoint-segment collision tests produce invalid solutions and usually none or two tests succeed, characterizing non-colliding and colliding segments, respectively.

Fig. 4 illustrates the interactions between colliding segments. In Fig. 4a, first A from AB collides with segment CD and then C from CD collides with AB. In Fig. 4b, first A and then B collide with CD. These are the two most common type of collision. There is a third type that occurs only for adjacent segments that share an endpoint and its normal vector, as shown in Fig. 4c. Such segments come from a smooth boundary region and have a special treatment, since only one valid collision event may occur. In the example, D crosses AB.

3.3. Representing the RLFS function

We represent the RLFS function separately on each segment of the discretized boundary using the [0, 1] parameter interval, with 0 corresponding to the start of the segment and 1 to the end of the segment. The RLFS function is initially undefined for all segments and is updated as new collisions between segments are found. We call *collision test* the



(a) Collision type 1 (b) Collision type 2 (c) Collision type 3 (d) Overlapping RLFS pieces

Fig. 4: The three basic types of segment collisions (a, b, and c). The segment endpoints are being displaced along their normal vectors, which are vertically aligned only to simplify the illustration. t and s parameterize the segments AB and CD, respectively, ranging from zero to one. In (a), A crosses CD at the point defined by s and then C crosses AB at the point defined by t. In (b), A and then B cross CD at the points defined by s_1 and s_2, respectively. The collision between adjacent segments from a curved boundary is shown in (c), where D crosses AB at the point defined by t. Two RLFS pieces, defined in the overlapping intervals $[t_1, t_2]$ and $[t_3, t_4]$, are shown in (d). The *minimum* operator results the piecewise linear RLFS function defined by linking the points (t_1, d_1), (t_x, d_x) and (t_4, d_4). These two linear pieces are obtained by clipping the intervals of the two original pieces and redefining d for the new endpoints through linear interpolation.

process of verifying whether two segments collide, identifying the type of collision if it occurred, and, accordingly, updating the RLFS function for both segments. We approximate the RLFS function on a segment by a piecewise linear function whose pieces are defined in the following way.

In the collision test, we first sort the valid endpoint-segment collision events by the length of the displacement d until collision (see Section 3.2). In the first collision case, illustrated in Fig. 4a, (s, d_1) and (t, d_2) are the parameters (computed from Equation 1) of the first and second endpoint-segment collision events. A piece of the RLFS is defined for the segment AB in the $[0, t]$ interval, linearly ranging from d_1 to d_2. For the segment CD, a piece is defined in the $[0, s]$ interval ranging from d_2 to d_1. In the second collision case, shown in Fig. 4b, (s_1, d_1) and (s_2, d_2) are the parameters of the first and second endpoint-segment collision events. A piece of the RLFS is defined for AB in the $[0, 1]$ interval, linearly ranging from d_1 to d_2. For CD, a piece is defined in the $[s_2, s_1]$ interval ranging from d_2 to d_1. If the point A collides with CD exactly at D, or if B collides with CD exactly at C, redundant collision events occur: A with CD and D with AB in the first case, and B with CD and C with AB in the second case. In such situations, one of the two redundant events must be discarded. If A collides with CD exactly at C, or if B collides with CD

exactly at D, both redundant events must be discarded. Such situations are not considered collisions. Redundant endpoint-segment collision events are removed after sorting the events and before identifying the collision type and defining RLFS pieces. Redundant events are easy to identify because they are adjacent in the list of sorted collision events. One needs only to check whether they have the same d value and occur at the same spatial location, which can be obtained by displacing the endpoint by a distance d. In the third collision case, shown in Fig. 4c, a piece of the RLFS is defined for AB in the $[t, 1]$ interval with constant value set to d, while for CD a piece is defined in the $[0, 1]$ interval with constant value set to d.

A segment may collide with several other segments in our simulation of the grass-fire model. However, for each point of the segment, only the first collision is relevant, because it has the smallest displacement. Overlapping pieces of the RLFS function, generated by multiple collision tests, are thus combined using the *minimum* operator, which may imply clipping the pieces, as illustrated in Fig. 4d. An ordered list of RLFS pieces is maintained for each boundary segment. Each linear piece is represented as an interval of the parameter t with displacement values d defined at the interval endpoints, i.e., the pieces are defined by four parameters: $(t_{start}, d_{start}, t_{end}, d_{end})$.

After all relevant collision tests between pairs of segments are performed, the result is a continuous piecewise linear function defined on the discretized boundary that approximates the RLFS of the original boundary. Our method for efficiently computing the MAT through grass-fire simulation is based on strategies to identify the necessary collision tests, avoiding checking pairs of segments that do not collide or do not contribute to the final RLFS function. Once we have the RLFS function we can sample the medial axis by sampling the discretized boundary, evaluating the RLFS at the sample points, and displacing them in the corresponding normal direction by a distance equal to their RLFS.

Although the representation of the RLFS function we have just described is sufficient to sample the approximate medial axis, the algorithm we are about to present depends on richer information about individual RLFS pieces. The basic representation of the linear pieces was augmented in two ways. First, we added an index to the segment with which a collision produced that piece. We call this index the *peer segment index* (*PSI*); the indexed segment is the *peer segment*. Second, each linear piece of the RLFS function is extended to produce another value, named s, besides the distance d until the first collision: s is the value of the parameter t that defines the point on the peer segment responsible for the collision with the point where the RLFS function is evaluated. This way, by evaluating the RLFS function for a point P, parameterized as t on a segment, we obtain the tuple (d, s, PSI), which tells us that the point that collided with P, after being displaced by a distance d, is the point parameterized as s on the peer segment indexed by PSI. The parameter s, computed in the collision test, is simply added to the RLFS piece at both interval endpoints, and thus it is also piecewise linearly approximated. Clipping overlapping RLFS pieces implies redefining one of their endpoints through linear interpolation, as shown in Fig. 4d. The same interpolation applied to d is applied to s.

3.4. Algorithm

The simplest way to compute the RLFS for all boundary points is the naïve brute-force method that performs collision tests for all possible pairs of segments. This method takes quadratic time in the number of segments. Since the quality of the (approximate) medial axis transform depends directly on the density and quality of the boundary sampling (see Fig. 2), this quadratic method does not scale well with the number of sample points and is only acceptable in applications that can use small sampling rates. That precludes path planning for precision machine cuts, for instance.

In our approach, like Ramanathan and Gurumoorthy [23], we track the locus of the centers of the maximal discs by stepping along the boundary of the region in the two opposite directions simultaneously, tracking pairs of maximal-disc footprints. However, our algorithm is different from theirs in important aspects:

- They perform on-the-fly discretization of the boundary into a discrete set of maximal-disc footprints, whereas we discretize the boundary into straight-line segments in a pre-processing step, without resampling boundary regions already defined by straight edges.
- We compute an implicit representation of the medial axis (the RLFS function) before extracting the actual MA.
- Our implicit representation of the MA can be easily rewritten during its construction as new collision tests between segments reveal smaller values of the RLFS. This feature plays a significant role in achieving good

performance because, during the algorithm execution, we do not need to guarantee that the tracked portion of the (implicitly defined) medial axis is globally correct, since it can be redefined in further steps of the algorithm.

As explained in the next subsections, we have designed our algorithm widely relying on the possibility of rewriting the MA representation to simplify and accelerate the computation of the MAT. Previous equivalent solutions need to ensure the global correctness of the tracked MA at each step of the algorithm, incurring in performance penalty.

3.4.1. Overview

Our algorithm has three main routines: *Find MA Extreme Points*, *Follow MA*, and *Fix MA*. It starts by calling *Find MA Extreme Points*, which may call *Follow MA* several times. When the region has holes, *Fix MA* is called at the end, which again may call *Follow MA* several times.

Before describing each of these routines in detail, we shall describe an example of our algorithm in action. Fig. 5 presents snapshots of consecutive intermediate results produced by our algorithm during the construction of the medial axis of the shape depicted in Fig. 5a. First, *Find MA Extreme Points* marches along the region boundary (counterclockwise in this case, but not necessarily), searching for an MA extreme point generated by a convex vertex or by a region of locally maximal positive curvature (LMPC), where *adjacent* segments collide. Having found such a point, *Find MA Extreme Points* calls *Follow MA* to track an MA branch from this pair of adjacent segments, which is achieved by walking along the boundary in opposite directions tracking new pairs of colliding segments, as indicated by the arrows in Fig. 5b. The tracked branch is shown in red and the LMPC point that started it is marked as a black dot. For illustration purposes, the branch is reconstructed from the already defined part of the RLFS function by densely sampling the boundary and displacing the samples along the normal direction by the distance given by the RLFS function. The displaced samples, representing MA branches, are drawn in red, and straight-line segments representing the sample displacement and also the already defined portions of the RLFS function are shown in black. As can be seen in Fig. 5b, the length of the tracked branch is clearly overestimated since it ends at a point that cannot be the center of a maximal disc.

When *Follow MA* can no longer establish new pairs of colliding segments, *Find MA Extreme Points* resumes the search along the boundary and finds another MA extreme point, generated by the convex vertex of the boundary. Fig. 5c shows the intermediate result of the algorithm after tracking part of the MA branch started by the convex vertex and followed by *Follow MA*. While tracking the newly found branch, its implicit representation partially overwrites the previously found branch until both branches join, as indicated by the circle in Fig. 5d. From the joint, which is identified by a specific pattern on the RLFS function (see Section 3.4.3), a new branch is tracked by *Follow MA*, producing the intermediate result shown in Fig. 5e. Regular points of the medial axis can be recovered from two different points on the boundary (footprints) through displaced boundary samples. However, as the implicit representation of a branch is overwritten, part of the branch may be recoverable from only one boundary portion, where its representation has not been overwritten yet. Such branch parts are drawn in green. Note that parts of the axis computed so far are still invalid, i.e., they are centers of non-maximal discs. These are naturally corrected in further steps of the algorithm.

As *Find MA Extreme Points* continues, another region of LMPC is found. However, this time, it starts the short, invalid MA branch from the intermediate result shown in Fig. 5f. Then, *Find MA Extreme Points* reaches another region of LMPC, which starts another MA branch whose tracking, performed by *Follow MA*, completely overwrites the implicit representation of the previously found, invalid branch (Fig. 5g). Fig. 5g also shows the moment when the newly found branch meets a previously tracked branch after having partially erased invalid portions of the computed axis. From this joint, another branch is started and its tracking further erases more invalid sections of the current axis. A last joining occurs at the moment represented in Fig. 5h, starting the last tracked branch that ends at a region of LMPC, resulting in the complete implicit representation of the MA illustrated in Fig. 5i.

When a hole is introduced into the region shown in Fig. 5a, *Find MA Extreme Points* and *Follow MA* cannot construct the correct MA because the inner and the outer boundaries are treated as independent objects. However, those routines construct the correct implicit representation of the MA corresponding to the outer boundary. They also build an overestimated part of the MA that is induced by the concavity of the inner boundary, where regions of LMCP exist. Fig. 6a shows this intermediate result.

For regions with holes, the *Fix MA* routine is invoked and searches for boundary segments where the RLFS function is not defined. It finds one such segment in the inner boundary of the region. This segment is then tested for collisions against every other segment, which leads to the discovery of its peer segments and starts a new MA branch, as shown in Fig. 6b. Having found the peer segments (marked as black dots), *Fix MA* calls *Follow MA*, which after a few branch tracking and branch joining operations completes the medial axis, as shown in Fig. 6b to 6f.

The next subsections present a detailed description of the three main routines of our algorithm.

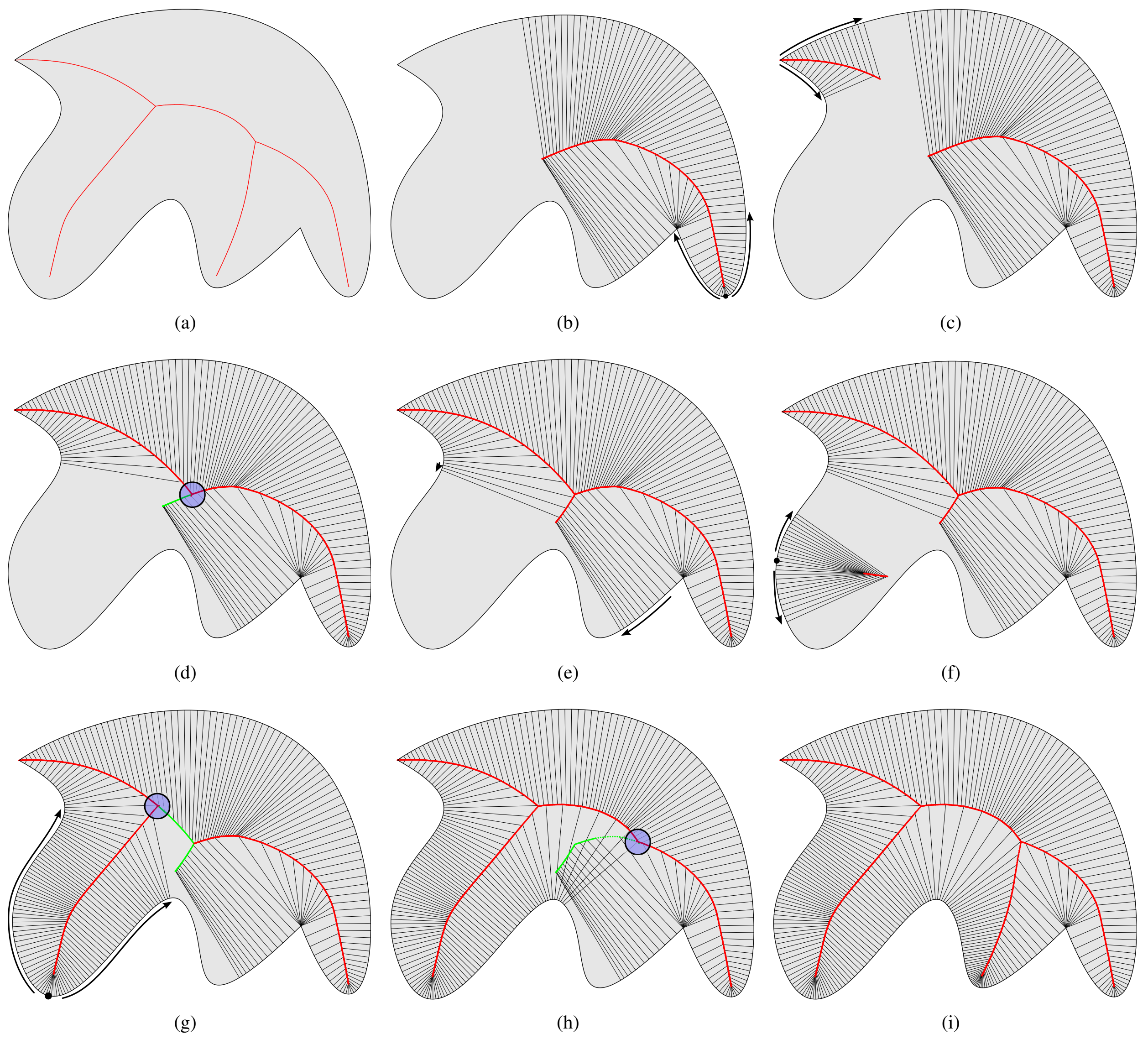

(a) (b) (c) (d) (e) (f) (g) (h) (i)

Fig. 5: A step-by-step illustration of the construction of the medial axis of a planar region with curved boundaries. The region and its explicit medial axis is depicted in (a). Several intermediate results are shown in (b) to (i).

3.4.2. *Find MA Extreme Points*

Find MA Extreme Points searches for (possibly invalid) extreme points of the medial axis along the region boundary. These extreme points start (possibly invalid) MA branches and are associated to convex vertices and points of locally maximal positive curvature (LMPC). *Find MA Extreme Points* visits every segment of all boundaries and performs the collision test against its *next* segment. Fortunately, only adjacent segments that are incident to a convex vertex or that discretize a boundary region of LMPC actually collide. After the collision test, the RLFS is evaluated

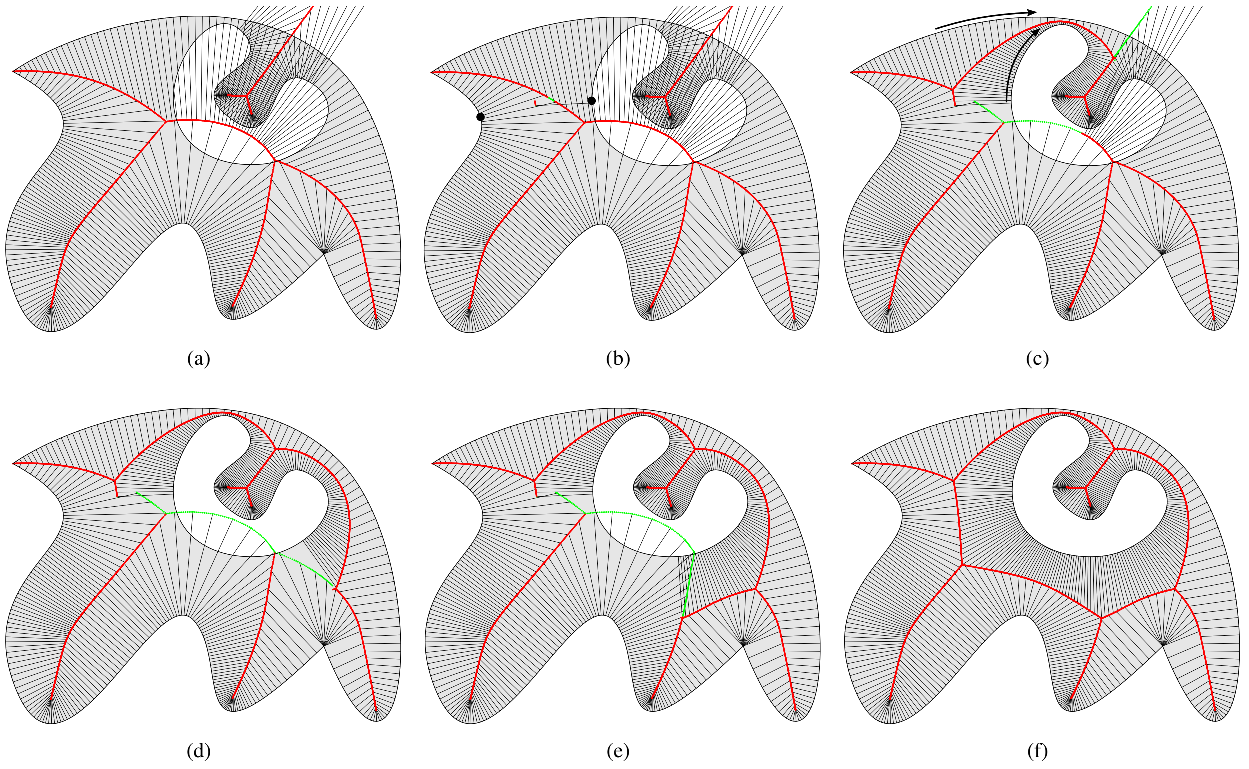


Fig. 6: A step-by-step illustration of the correction of the medial axis performed by the *Fix MA* routine when a hole is introduced in the region depicted in Fig. 5a. A few intermediate results of the correction process are shown from (a) to (f).

at the end of the segment ($t = 1$) to obtain the peer segment (see Section 3.3 for details). If the RLFS function is defined at this point and the peer segment is the next segment, the visited segment and its next segment are candidates for defining a valid extreme point of the MA (this pattern characterizes the footprint of an MA's extreme point). From this pair of segments, a branch of the MA can be tracked by the *Follow MA* routine. Convex vertices always start a valid MA branch. LMPC points start invalid MA branches when the tangent disc having radius equal to the curvature radius is not contained in the region. We cannot identify invalid branches solely through the local boundary analysis performed in this stage of the algorithm. However, invalid MA branches will be later overwritten by valid branches in their implicit representation.

3.4.3. Follow MA

Follow MA follows a (possibly invalid) branch of the medial axis from a known portion of it, implicitly defined by a portion the RLFS function computed from a pair of colliding segments. These segments contain footprints of a set of (possibly) maximal discs and we can follow the footprints of other discs by walking along the boundary from this pair of segments in the two opposite directions simultaneously (refer to Fig. 5 again). This process is somewhat similar to the MA tracking strategy used by Ramanathan and Gurumoorthy [23]. From one segment of that pair the algorithm walks in the *next* segment direction, while from the other segment it walks in the *previous* segment direction, or vice-versa, to follow the footprints in opposite directions. However, from a pair of adjacent segments (a segment S and its *next* segment), like those found by *Find MA Extreme Points*, only one direction can be followed: the *previous* direction for S and the *next* direction for its next segment.

The MA following procedure uses two indices, *left* and *right*, that initially refer to the given pair of colliding segments. *Left* is the segment at the left side of the MA branch to be followed, while *right* is at the right side. First, *left* remains fixed and *right* successively becomes the *next* segment, while it is tested against *left* for collision, until no change in RLFS occurs or the footprint of a medial axis branching point is found (the identification of branching points will be explained later in this subsection). Then, *right* steps back (one *previous* operation) and the iteration process is repeated, but this time keeping *right* fixed and moving *left* using the *previous* operation. Then, *left* steps back (one *next* operation). The sequence of steps described above is repeated while changes are made to the RLFS function of both *left* and *right* segments through the collision tests and no branching point is detected. This way, a (possibly invalid) branch of the MA is followed by constructing its implicit representation on the traveled boundary portion. It is important to understand that the RLFS functions is updated when new pieces of the function are defined for intervals (boundary regions) where the function is not defined yet, or when produced pieces have smaller collision distance values d for an interval where the RLFS function is already defined, causing overwriting branches (as shown in Fig. 5g). *Follow MA* relies only on local information about the boundary and thus may track footprints of discs that are not maximal (i.e., not contained in the region). In other words, tracked MA branches may be invalid (like the short branch in Fig. 5f) or overestimated in length (like the branch in Fig. 5b), but they will be naturally corrected in later steps of the algorithm (Fig. 5g to 5i).

When the implicit representations (RLFS) of two (possibly overestimated or invalid) MA branches overlap, the one that defines smaller collision distances dominates the other (as shown in Fig. 5d), overwriting it due to the minimum operator used to combine overlapping RLFS function pieces, as explained in Section 3.2. While it is tracked, the new branch erases the old one until they join or until the old branch completely disappears from the implicit MA representation. In a joining event, the newly computed RLFS function piece, produced by tracking the new branch, partially overwrites a function piece from the implicit representation of the old branch. This situation is illustrated in Fig. 4d. From a joint, the tracking of the new branch cannot continue because the produced values for the RLFS would be greater than the previously defined ones, which prevents updating the RLFS function. The joint of two MA branches defines an MA branching point and the footprint of the corresponding maximal disc can be easily identified through a local analysis of the RLFS function.

An MA branching point's footprint lying on a segment is characterized by two continuously adjacent RLFS function pieces that satisfy either one of the following conditions:

- They were produced by collisions of the segment in focus against two non-adjacent segments; or
- The parameter s (discussed in Section 3.3) is not 0 and 1 for the end of the first function piece and the start of the second piece, respectively.

The first condition is the general configuration of footprints of discs corresponding to MA branching points. The second condition holds for two adjacent function pieces produced by collisions against two segments that are adjacent to each other and define an MA extreme point, starting a branch. The branching point's footprint is parameterized by the value of t where those adjacent RLFS function pieces join. One can think of a branching point as the point where two MA branches merge into a third branch. Our algorithm tracks the resulting MA branch using *Follow MA* again, once a branching point's footprint is located. The tracking restarts from the two peer segments that produced the adjacent RLFS function pieces that defined the branching point's footprint. Such pair of segments is given by the indices *PSI*, for the peer segments, stored in the representation of the pieces of the RLFS function (see Section 3.3).

At each step of *Follow MA*, our method only requires a local analysis of the RLFS function to identify MA branching points by their footprints. The cost of such operation depends linearly on the number of RLFS pieces stored in the analyzed segment, which is two in average, and thus the expected complexity of finding MA branching points is $O(1)$. The methods by Ramanathan and Gurumoorthy [23] and Cao et al. [10] look for branching points through an iterative search on the boundary for a third footprint of the currently tracked maximal disc. This search occurs in each step of the MA tracking routine and costs $O(n)$, where n is the number of samples required for properly examining the boundary. This explains the higher efficiency of our algorithm.

3.4.4. Fix MA

Fix MA visits all segments searching for those whose RLFS function is undefined or discontinuous, which configures an invalid part of the function. Once such a segment is found, it is tested for collisions against all other segments in order to find its peer segments. This segment and each of the peer segments for its extreme points form new pairs from which a new branch of the MA is followed (as shown in Fig. 6c) in both opposite direction using the *Follow MA* routine. When *Follow MA* finishes, *Fix MA* restarts at the point it was before searching for peer segments and calling *Follow MA*, and continues searching for more segments where the RLFS function is undefined or invalid. Every time

Fix MA finds a new pair of segments to start tracking an MA branch, the *Follow MA* routine sets the RLFS function for a wide set of segments by successively erasing invalid branches and starting new ones at branch joining events. In fact, the goal of *Fix MA* is promoting the interaction between chains of segments that were isolated from each other, and thus treated as independent objects. Before calling *Fix MA* for the first time, the inner boundaries remain isolated from the outer boundary (see Fig. 6a). The RLFS function of isolated inner boundaries cannot be computed for at least a few segments (or even all segments when the holes are convex). For each new pair of colliding segments discovered by *Fix MA* through its global collision tests, two formerly independent boundaries start to interact and *Follow MA* corrects the medial axis based on the information provided by the set of already correlated boundaries. Typically, for each extra (inner) boundary a new pair of colliding segments must be found by *Fix MA* to be the start point of *Follow MA*, which in turn completes the MA for the set of already correlated boundaries (as illustrated in Fig. 6).

3.4.5. Expected execution time

In order to construct the RLFS function we basically execute a series of collision tests between segments, plus some simple local analysis of the RLFS function, while stepping along the boundary. Therefore, the execution time of our algorithm is roughly proportional to the number of collision tests performed, which depends on the number of boundary segments and on the geometry of the region whose medial axis we want to compute. In Section 6 we show through time measurements that the execution time depends weakly on the region's geometry and linearly on the segment count. In this section we draw an explanation for this behavior.

The time complexity of *Find MA Extreme Points* is clearly linear, since it monotonically steps along the boundary segments performing collision tests between adjacent segments. This routine, however, calls *Follow MA*, which has a complex behavior due to the frequent overwriting of the RLFS function caused by the tracking of overestimated and invalid branches. The total number of collision tests performed by *Follow MA* actually varies with the order in which the branches started by MA extreme points are followed. In our implementation, the number of collision tests depends on the segment from which the boundary marching of *Find MA Extreme Points* starts.

The cost of tracking a single branch (*Follow MA* routine), which is a process of marching along the boundary, depends linearly on the length, in number of segments, of the implicit representation of the branch. The cost of tracking all branches therefore depends linearly on the boundary sampling density, which determines the number of boundary segments. However, computational effort is wasted on invalid or overestimated MA branches. The amount of wasted effort depends on the region's geometry, but it is usually small and does not compromise the overall linear time dependence on segment count of our algorithm. The wasted computation is limited because the tracking of invalid or overestimated branches stops as soon as it encounters boundary portions with smaller, already defined RLFS, or when certain boundary configurations are reached: convex vertices, boundary points where the curvature is positive and the curvature radius is smaller than the RLFS corresponding to the currently tracked branch, and pair of segments that diverge with their displacement without colliding. For a constant sample count, as the complexity of the region increases, the number of MA branches also increases, but, in compensation, their implicit representation become shorter, and the boundary configurations that stop the tracking of invalid branch portions become more frequent.

As explained in Section 3.4.4, for each hole in the region, *Fix MA* triggers a global search (through collision tests) for the peer segments of a segment with undefined RLFS. This search is followed by several branch tracking operations. The cost of the global search on the boundary and of the tracking of new branches depend linearly on the segment count (as argued in the previous paragraph). Therefore, we estimate the time complexity of our algorithm as $O((1 + genus) \cdot n)$, where *genus* is the number of holes in the region and n is the number of segments in the boundary. We support this estimate with the time measurements presented in Section 6.

As can be seen in next subsection, the cost of extracting the explicit medial axis clearly depends linearly on the number of segments, since this operation is a boundary marching with no wasted computational effort.

4. Construction of the explicit medial axis

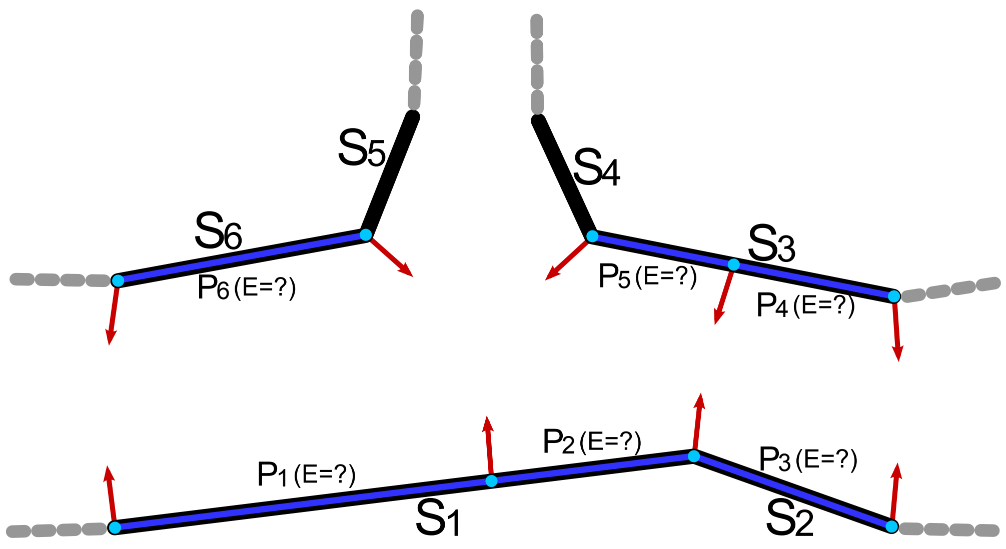
Having the RLFS function, one can approximate the medial axis as a set of unconnected points obtained by densely sampling the region's boundary and displacing the samples along the normal direction, as discussed in Section 3.3. However, besides the lack of connectivity between samples, this simple strategy has the drawback of ambiguity because it leads to at least two estimates of each MA point (except extreme ones) obtained from its footprints on the boundary — these estimates can be slightly different due to the fact that the boundary and the RLFS function are approximations. Without special processing, the resulting sample of the MA looks noisy.

In this section we present a simple algorithm to extract the medial axis as a graph from the RLFS function without ambiguity. The output of the algorithm is a set of nodes and edges that provide a piecewise linear approximation of the medial axis transform. Nodes have two attributes: position and radius of the corresponding maximal disc. Edges connect nodes and represent the set of straight-line segments that approximate the MA. The endpoints of the segment are the positions of the connected nodes, and the radius function is defined for each segment as the linear interpolation between the radius of the two connected nodes. In this representation of the medial axis, each MA segment (edge) corresponds to two pieces of the RLFS function on two different boundary segments, where its footprints lie on (as shown in Fig. 7b). These two boundary segments are peers and generated those two RLFS pieces through the same collision test (see Sections 3.2 and 3.3).

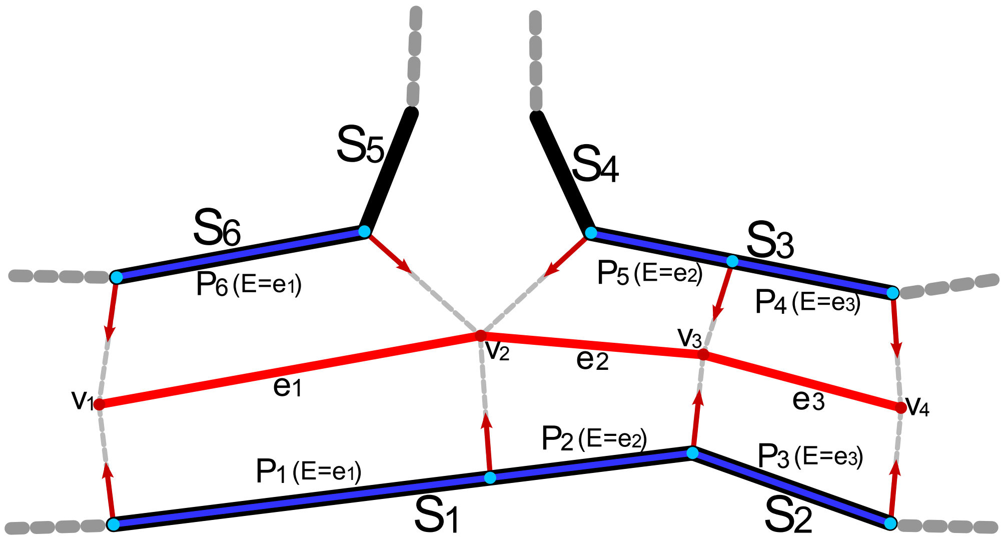
To apply our algorithm, we associate to each linear piece of the RLFS function an index, named E, to the corresponding edge of the medial axis graph. E is initially set to an invalid value, indicating that the corresponding MA edge is not defined yet (as shown in Fig. 7a). The algorithm then iterates over all RLFS pieces of all boundary segments. For each piece, it checks whether E is invalid and, if so, creates an edge of the MA graph. The MA nodes that support this edge are also created if they do not exist yet. The criterion to connect two MA edges through a common node is as follows: because RLFS is a continuous function, if two pieces of the RLFS function are adjacent, the corresponding MA edges should be connected through a common endpoint. RLFS pieces are considered adjacent if they are consecutive on a segment or if one is the last piece on a segment and the other is the first piece on the *next* segment. In Fig. 7b, MA edges e_1, e_2 and e_3 are connected as a chain through MA nodes v_2 and v_3 because the corresponding RLFS function's pieces P_1, P_2 and P_3, on boundary segments S_1 and S_2, are adjacent.

For each RLFS piece P_i, on a boundary segment S, with invalid E an MA edge is created in the following way. We first locate the peer segment of S related to P_i using the index PSI of P_i (Section 3.3). Then, on the peer segment, we search for the RLFS piece P_j whose PSI refers to S. Since they lie in peer segments, P_i and P_j were defined in the same collision test and correspond to footprints of the same MA edge. The endpoints of this edge are computed by sampling their footprints on both peer segments. The first endpoint is obtained by averaging two displaced samples: S parameterized as the start of P_i (t_{start}) and then displaced by the distance d_{start} at the start of P_i; and S's peer segment parameterized as the end of P_j (t_{end}) and displaced by its d_{end} value. The second endpoint is the average of S parameterized by t_{end} and displaced by d_{end} of P_i, and S's peer parameterized by t_{start} and displaced by d_{start} of P_j. The radius of the endpoints, used to define the MA radius function, is the average of the displacement values. Averaging two estimates of the endpoints and their radius from their footprints on different segments attenuate the ambiguity in sampling the medial axis. The normal vector N used to displace a sample on a boundary segment is the linear interpolation between the normals at its endpoints without renormalization. The sample displacement is given by $d \cdot N$. We can use Fig. 7a to exemplify this process. Starting the iteration with P_1 in the segment S_1, we verify that its E index is invalid. Then we locate its corresponding RLFS piece P_6, which lies in the S_1's peer segment S_6, identified by P_1's PSI. The position attributes of MA nodes v_1 and v_2 are then calculated by averaging corresponding displaced boundary samples taken from P_1 and P_6 endpoints.

Having computed the position and radius function of the endpoints, the algorithm creates the MA edge and sets the E value for P_i and P_j to the index of the created edge, as shown in Fig. 7b. The MA graph's nodes corresponding to the endpoints must be created or shared with other already created edge, depending on whether the MA edges corresponding to adjacent RLFS pieces were already created. From P_i, its previous piece ($P_{previous}$) and its next piece (P_{next}) are found. They possibly lie on adjacent segments. If the E value is invalid for both adjacent pieces, two MA nodes, with the attributes of the endpoints, must be created to support the newly created MA edge. If the E value of $P_{previous}$ indexes a valid edge, the node representing its second endpoint is shared with the newly created edge, being its first endpoint. If the E value of P_{next} indexes a valid edge, the node representing its first endpoint is shared with the newly created edge, being its second endpoint. If E is an invalid edge index for either $P_{previous}$ or P_{next}, an MA node must be created to represent the first or second endpoint, respectively, of the newly created edge. When a node is shared, its attributes are updated with the average between them and the attributes of the corresponding endpoint computed for the newly created edge. MA branching nodes are shared by more than two edges and thus their attributes result from the average of more than two edges endpoints. To correctly compute such average during the algorithm execution, for each node we maintain a count of how many edges' endpoints were combined into it. In order to handle the node sharing between edges, the algorithm needs to distinguish the first and second edge's endpoint. Since P_i and



(a) A few RLFS pieces (P_i, in blue), with undefined correspondence to MA edges (E indices), drawn over the boundary segments (S_j, in black) that support them. The arrows are the normal vectors (interpolated) at the boundary points parameterized by the RLFS pieces' interval endpoints.



(b) Correspondences established between RLFS pieces (P_i, blue) and MA graph edges (e_j, in red) through the indices E to MA edges.

Fig. 7: Part of a planar domain. The figure illustrates the construction of its explicit medial axis transform as a graph of nodes (v_i) and straight edges e_j.

P_j correspond to the same MA edge, but running in different directions, a flag is set for P_j to indicate that first and second endpoints must be switched when indexing an MA edge by the index E of P_j.

The described procedure connects the created MA edge to formerly created ones according to the adjacency of the piece P_i on the segment S. This connectivity information is, however, not complete. The adjacency of the piece P_j (on S's peer segment), which also correspond to the same MA edge, must also be taken into account. To cope with this during the iteration over the RLFS pieces, when the algorithm finds one whose edge index E is valid, indicating that the corresponding MA edge was already created, the MA edges corresponding to the two adjacent RLFS pieces are identified. They must share an endpoint with the MA edge in question. If any of those edges exists and does not share an MA node with the edge in question, corresponding nodes are collapsed into a single one and their attributes are averaged, connecting the adjacent edges. The described situation can be exemplified through figure Fig. 7b. If we start the construction of the explicit medial axis by RLFS pieces P_5 and P_6 (a valid possibility), the MA edges e_1 and e_2 would be created and kept unconnected, due to the absence of adjacency information, until the iteration reaches P_1 or P_2. At this point, the E indexes of P_1 and P_2 were already defined (in the early iteration over P_5 and P_6) and point to edges e_1 and e_2. Given that P_1 and P_2 are adjacent, the indexed edges e_1 and e_2 must also be. Once we verify that they are not, we merge the two redundant (and possibly slightly different) instances of node v_2, thus connecting e_1 to e_2.

5. Generation of offset curves

An *offset curve* of a region is a level curve of the distance function to the region, that is, the set of points at a given fixed signed distance to the boundary of the region. Offset curves can be computed by displacing every boundary point along its normal direction (considering vertices as arcs with zero radius), but one needs to handle self intersections of the boundary during the displacement and clip portions that underwent such intersections. Points of self intersection lie on the medial axis, as stated by the grass-fire model. Since the RLFS function gives the length of the normal displacement that a boundary point performs to reach the medial axis, it can be used to clip displaced boundary portions that do not belong to the given offset set. A displaced boundary point belongs to an offset curve if, and only if, the offset of the curve is (in absolute value) less than or equal to the RLFS at the point.

Our algorithm to construct offset curves iterates over all pieces of the RLFS function. Each piece, which is a local linear approximation of the RLFS, is classified according to the values at its extrema: if both values are less than the given offset, the piece is ignored; if both values are greater than or equal to the offset, a straight-line segment of the (approximate) offset curve is created with endpoints computed by displacing the boundary points at the piece's interval endpoints; if one extreme value of the RLFS's piece is greater than and the other is less than the offset, the piece is first converted into a shorter one and then used to produce a segment of the offset curve the same way as in the former case. The piece is shortened by redefining its extremum whose value is less than the offset through linear interpolation. The new value must be exactly the given offset.

The connectivity between the segments of the computed offset curve follows the adjacency between the RLFS pieces that defined them. The only special case regards the pieces that were shortened, as described above. They partially correspond to points of self intersection of the displaced boundary. A segment in the offset curve produced by such a RLFS piece must connect to that produced by the corresponding RLFS piece on the peer segment. Some results are shown in the next section.

6. Results and discussion

In this section we present some results obtained with our algorithms and discuss important aspects of our method for computing the medial axis of arbitrarily-shaped planar regions.

6.1. Generality

Fig. 8 shows a region with polygonal and smooth boundaries and four holes, discretized with 466 samples. It demonstrates the generality of our method. The polygonal portion of the boundary was not further discretized and the smooth portion comes from sampling cubic Hermite curves. Several points of tangent discontinuity exist, including two in the top-left inner boundary. Our method was able to compute a good approximation of the medial axis for that region. The correctness of the result is supported by the set of maximal discs we drawn from the approximate MA and its corresponding radius function.

Fig. 9 shows a region bounded by the analytical smooth polar curve $r = 2 + \sin(5\theta) + \sin(2\theta)$. Boundary portions of very high curvature exist but they were well captured by adaptive boundary sampling, which allowed a good approximation of the MA using 473 samples. This example also shows that our method is not restricted to piecewise polynomial boundary curves; all that it needs is a sample of boundary points and their normals.

The medial axes of some vector fonts are shown in Fig. 10 and 11 to demonstrate the suitability of our algorithm to practical examples. (Vector fonts typically define glyphs using piecewise quadratic and cubic Bézier curves.)

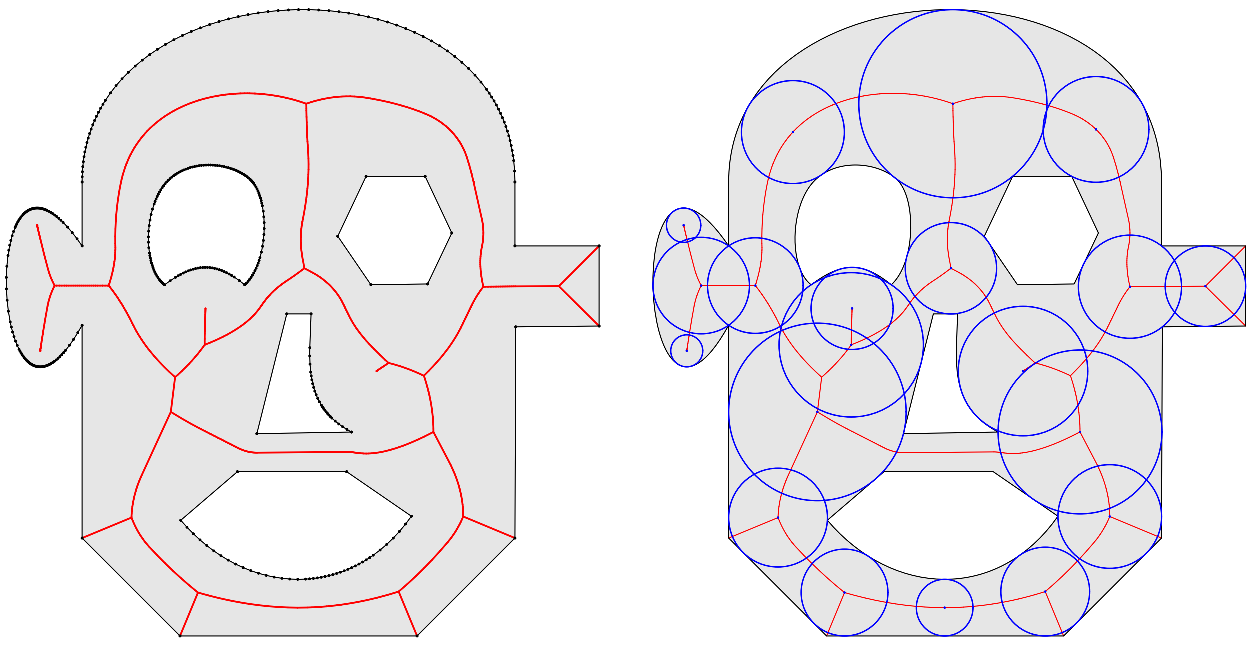


Fig. 8: A region with holes and mixed boundaries. Its medial axis and the boundary discretization are shown on the left and several maximal discs are shown on the right.

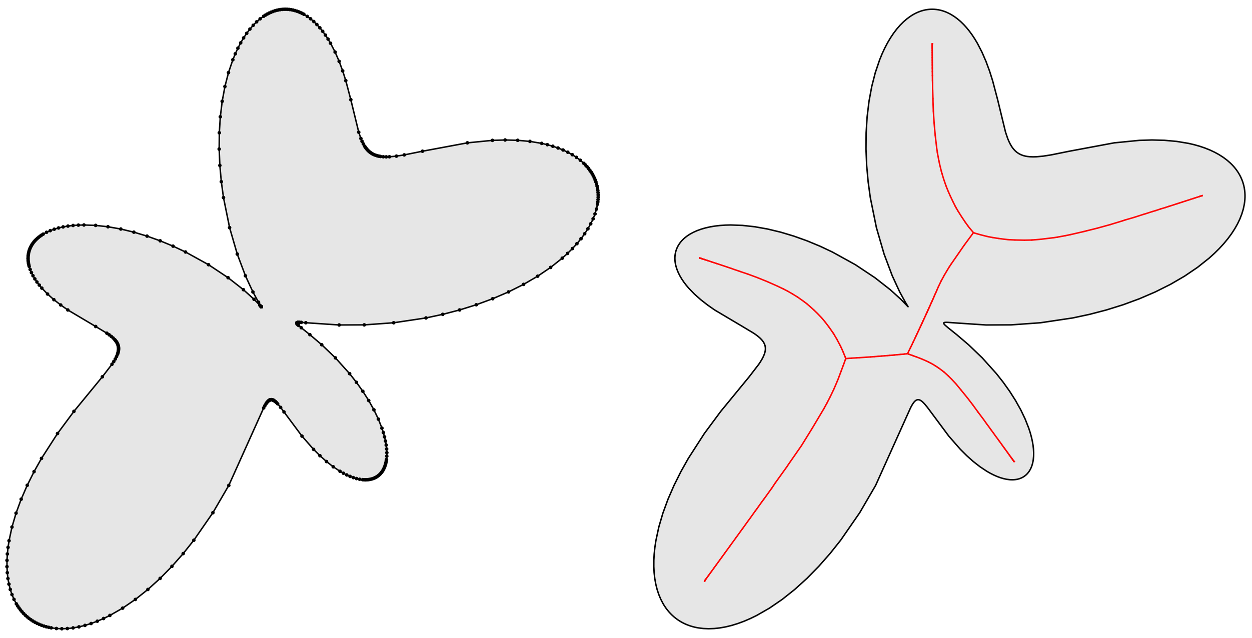


Fig. 9: A region bounded by an analytical smooth polar curve and its medial axis.

6.2. *Offset curves and external MA*

As discussed in Section 5, offset curves can be easily computed once one has the RLFS function. The computation of external offset curves requires the implicit definition (RLFS) of the external medial axis, which can be computed by applying our method considering the given planar region as a hole (inverting its boundary orientation) in a much larger region with a distant external boundary (we used a large square). The explicit external medial axis is extracted

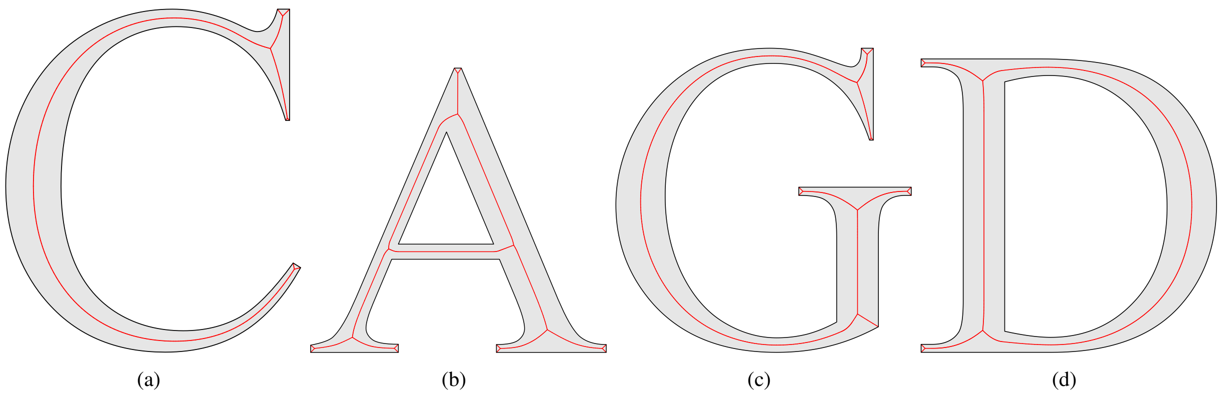


Fig. 10: Medial axes of some capital letters from Times New Roman.

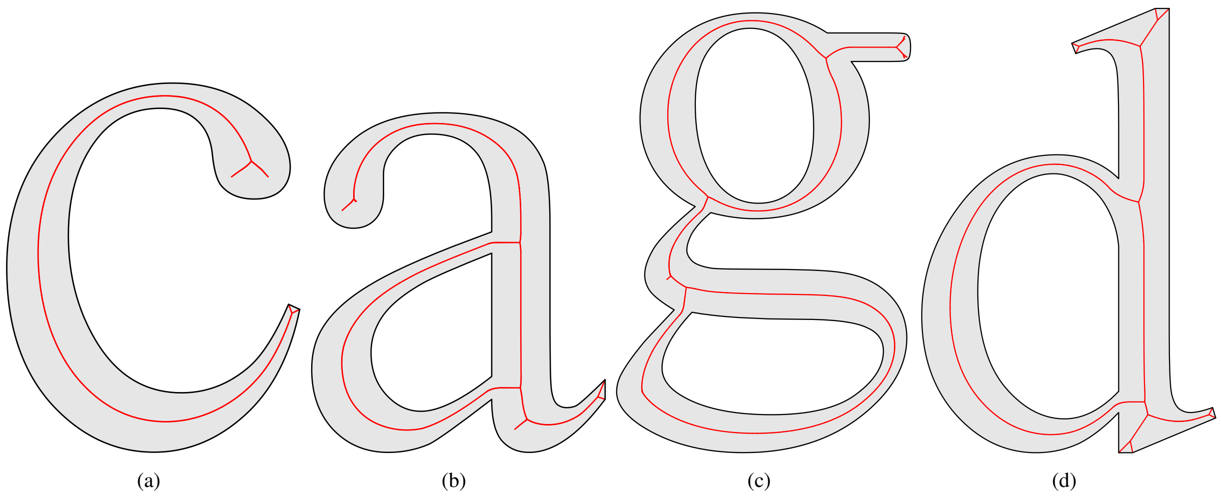


Fig. 11: Medial axes of some minuscule letters from Times New Roman.

with the same method described in Section 4. However, the resulting axis is not complete, because it is bound by a finite region, while the actual external MA extends to infinity. Moreover, the axis computed this way contains extra branches corresponding to the artificial external boundary introduced. These are minor drawbacks that can be mitigated by choosing the artificial boundary sufficiently far away. Fig. 12 shows several internal and external offset curves for the region of Fig. 1, as well as its internal MA and part of the external one.

6.3. Efficiency

Fig. 13 shows a complex smooth region defined by cubic Hermite curves adaptively sampled by curvature. Its approximate MA is drawn in red. For this example, the boundary sampling routine took 57 milliseconds and produced 21398 samples. The construction of the data structure to store the boundary discretized into 21398 segments took 17 milliseconds, the computation of the RLFS function took 149 milliseconds, and the extraction of the explicit MA took 47 milliseconds. (Measurements taken in a laptop computer with an Intel Centrino Core 2 Duo T5450 1.67 GHz processor and 2 GB of RAM, running Windows 7.) As can be seen, the construction of the implicit MA (RLFS function) is the most time consuming step of our method. By varying the refinement threshold in the adaptive sampling one can control the number of produced samples. For the same region, the time taken to compute the MA varies linearly with the number of samples, as shown in the Table 1. Table 2 presents the times taken to compute the MA of other regions without holes and with adaptively sampled boundaries. The ratio between total time and

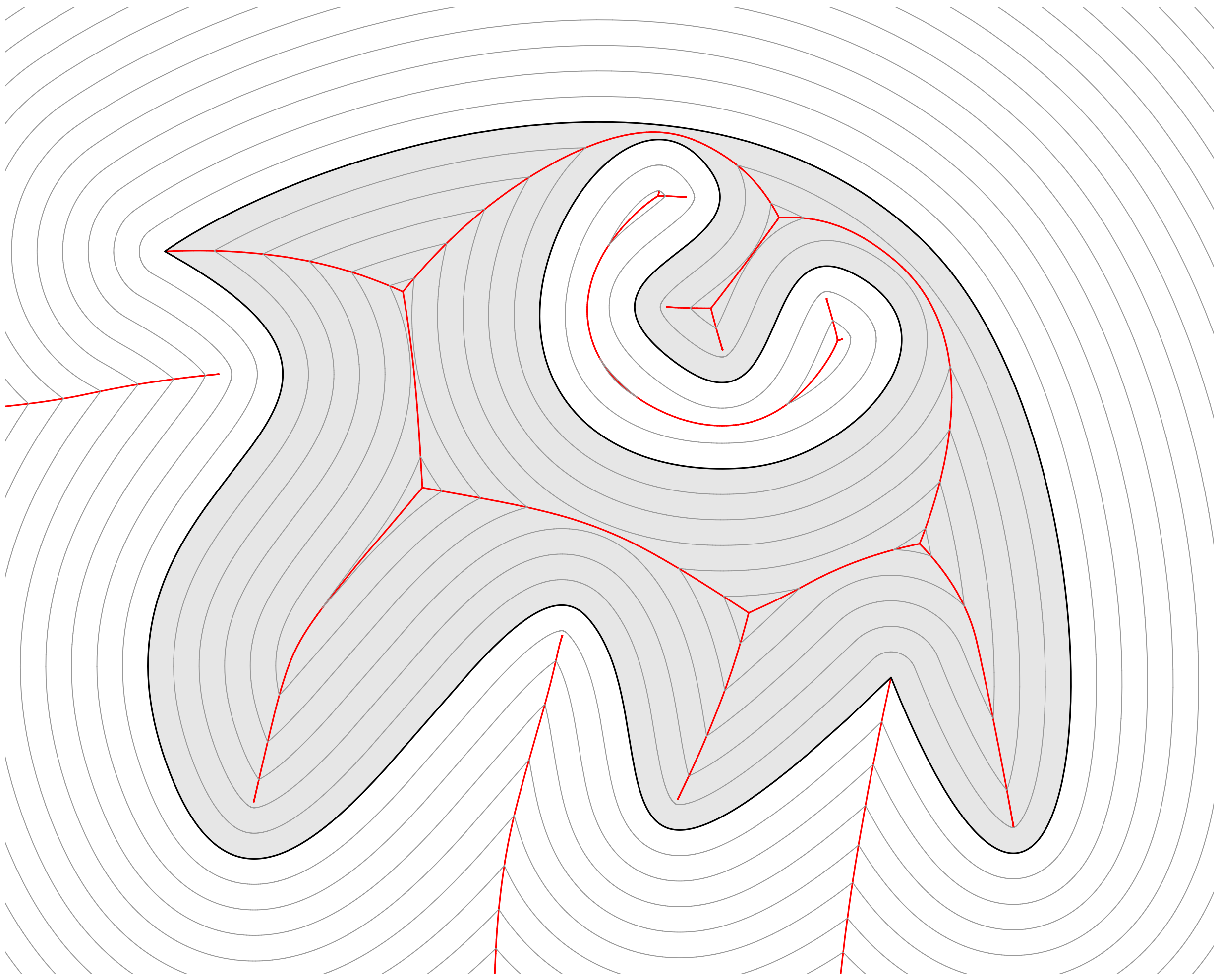


Fig. 12: Offset curves and internal and external medial axes.

segment count (t_{total} / segments) varies with the region shape within the same order of magnitude, confirming that the sensitiveness of our algorithm to shape is limited.

samples	t_s (ms)	t_{ds} (ms)	t_{RLFS} (ms)	t_e (ms)	t_{total} (ms)	t_{RLFS} / samples (μs)	t_{total} / samples (μs)
12086	33	9	87	27	156	7.17	12.93
21398	57	17	149	47	270	6.98	12.60
30634	80	24	214	67	384	6.97	12.55
40855	106	32	280	88	506	6.86	12.39

Table 1: Times for the computation of the medial axis of the region in Fig. 13 under adaptive sampling with varying resolution. The table relates number of samples (same as number of segments in this case) to times in milliseconds required for sampling the boundary(t_s), constructing the data structure (t_{ds}), computing the RLFS function (t_{RLFS}) and extracting the explicit MA (t_e). The total time (t_{total}) is also shown. The ratio in microseconds between t_{RLFS} and the sample count (t_{RLFS} / samples), and between t_{total} and the sample count (t_{total} / samples) is shown to support the linear time behavior of our algorithm. Those ratios are nearly constant with varying number of samples, but they reveal a time behavior slightly below linear.

The performance impact of introducing an inner boundary (hole) into a region is actually smaller than we have predicted through the simple complexity analysis in Section 3.4.5. As explained in Section 3.4.4, when *Fix MA* finds a segment where the RLFS function is undefined, all segments are visited in order to identify its peer segments

shape	segments	t_{total}(ms)	t_{total} / segments (μs)
Outer	3051	37.3	12.22
C letter	977	13.6	13.95
G letter	959	14.3	14.88
Butterfly	2127	21.3	10.03

Table 2: The table relates number of segments to total time in milliseconds for computing the medial axis of four different regions with no holes and boundaries sampled adaptively and densely. The first column refers to the shape name: Outer is the outer boundary in Fig. 14; C letter and G letter are the capital C and G letter from Fig. 10; and Butterfly is the shape from Fig. 9.

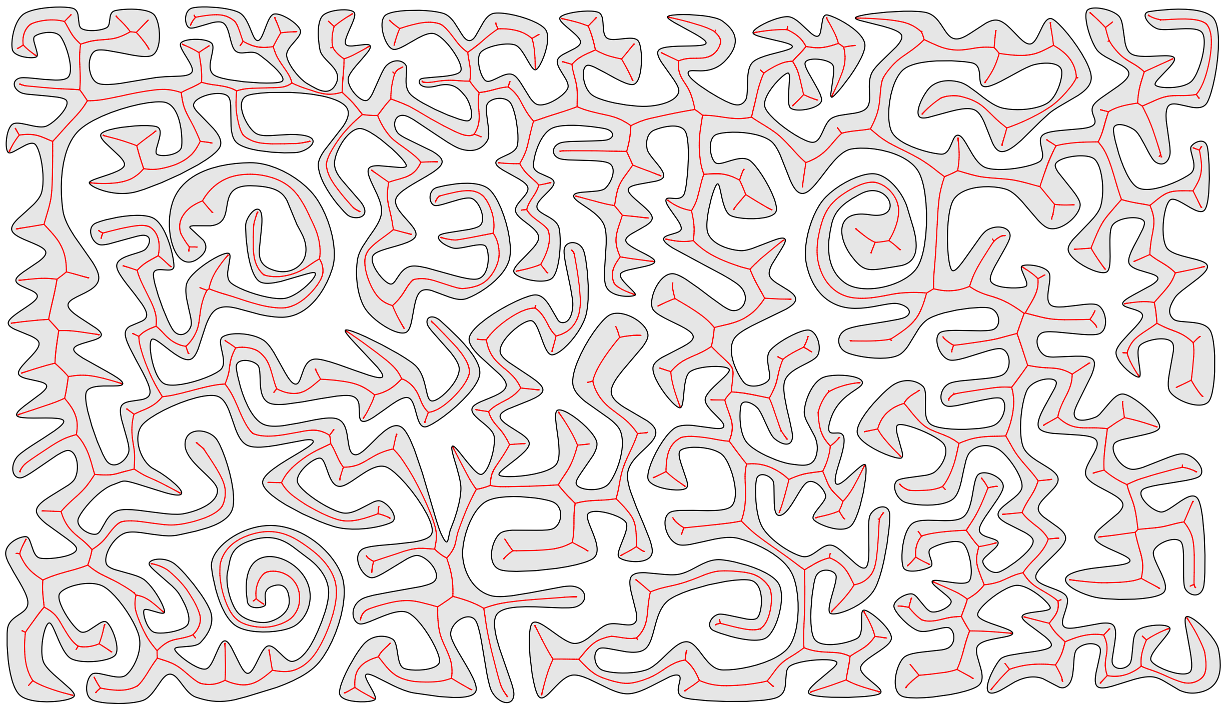


Fig. 13: A complicated smooth region with no holes and its medial axis, drawn in red, computed with our algorithm. The boundary was discretized into 21398 straight-line segments through adaptive sampling.

and then track new MA branches. As consequence, for each hole a complete, linear-time search on the segment set must be performed followed by a few branch tracking operations (*Follow MA*) that rewrite the RLFS function for corresponding boundary portions. However, the collision tests between segments have heterogeneous cost because only those that cause updating the RLFS function require the dynamic memory management (a costly operation) involved in modifying the list of RLFS function pieces. Most collision tests required to search for peer segments of a segment of the inner boundary with undefined RLFS fail, since most segment pairs do not collide. Besides, usually a relatively small portion of the outer boundary has its RLFS function updated by the branch tracking operations that fix the MA, and usually the RLFS function for most segments of the inner boundary is only written, not rewritten, requiring less dynamic memory management operations. Table 3 shows the impact in execution time and required number of collision tests of inserting holes into the region of Fig. 14. The MA of this domain was first computed for the outer boundary only (0 holes), and them for the regions obtained by successively introducing the top triangle (1 hole), the bottom triangle (2 holes), the rectangle (3 holes) and the square (4 holes). Adaptive boundary sampling was used. It can be noticed that the ratio between time and number of collision tests decreases as the genus increases, which occurs because the extra collision tests involved in handling holes are cheaper in average, since most of them do not imply updating the RLFS function pieces.

Regarding the updating of the list of RLFS function pieces and the search for footprints of MA branching points

holes	segments	t_{RLFS} (ms)	collision tests	t_{RLFS} / collisions (μs)
0	1625	13.3	3868	3.45
1	1666	14.7	5725	2.57
2	1709	17.7	7644	2.32
3	1753	21.4	9726	2.20
4	1797	25.4	11856	2.14

Table 3: The table shows the impact in performance of introducing simple holes into the region Fig. 14. The presented data relates the insertion of simple holes to the number of segments used to represent the boundaries (different from sample count due the existence of reflex vertices, in the inner boundaries, which are split into dummy segments), the time in milliseconds to construct the RLFS function (t_{RLFS}), the number of collision tests performed to compute the implicit MA, and the ratio between t_{RLFS} in microseconds and number of collision tests.

(Section 3.4.3), we implemented a simple optimization that employs, per segment, a cursor that points to the position of the last piece modified or inserted into the list. Due to the behavior of our branch tracking method, pieces of the RLFS function for a segment are likely to be inserted (or modified) near the previously inserted (or modified) piece. The search for the insertion point on the list is accelerated if started from the cursor position.

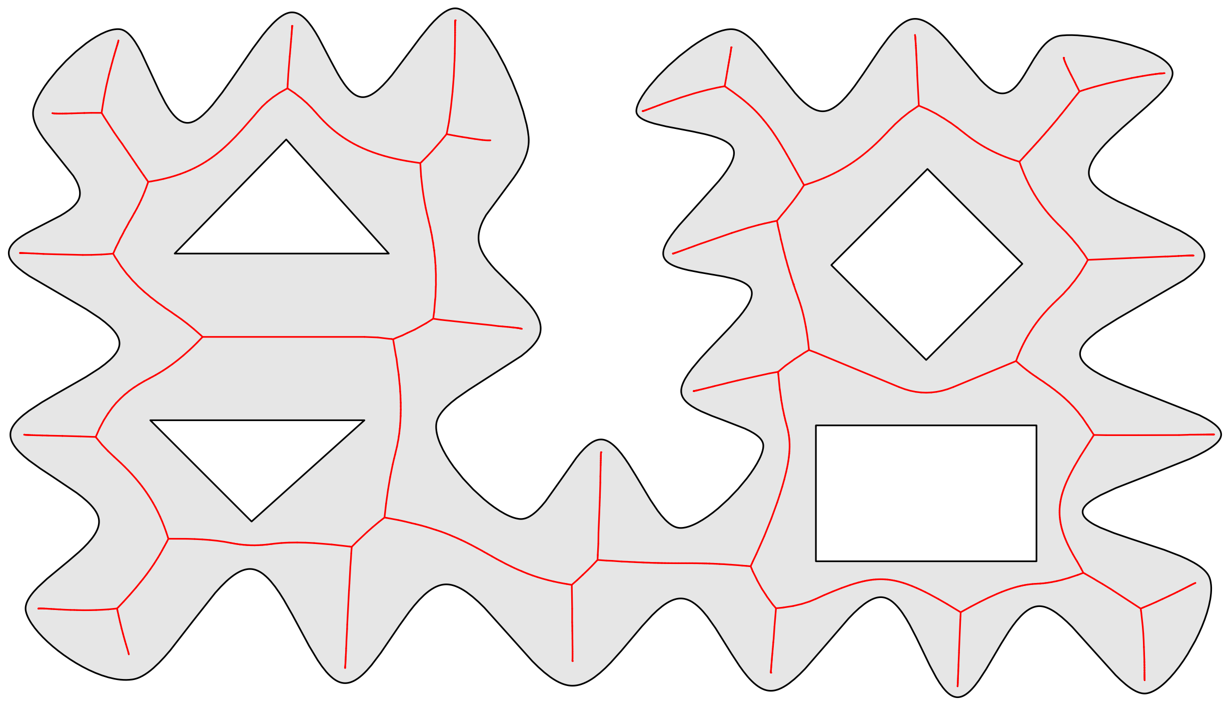


Fig. 14: A region with four simple polygonal holes and its medial axis, drawn in red.

6.4. *Comparison with the state of the art*

The linear time dependence on sample count of our algorithm gives it an important advantage over state-of-art approaches for computing the medial axis of regions with general boundaries. The algorithms by Ramanathan and Gurumoorthy [23] and Cao et al. [10] have quadratic time dependence on the sample count. This performance could be matched even using the naïve version of our technique, which performs collision tests for all possible pairs of segments. The divide-and-conquer approach by Aichholzer et al. [24] has $O(n \log n)$ dependence on the number of elements used to discretize the boundary. However, the biarc approximation of the boundary they employed is better than our boundary approximation by straight-line segments in the sense that less elements are required to properly discretize the region's geometry. Nevertheless, our approach is asymptotically faster and thus will show better performance for complex regions. The method by Geisen et al. [14] approximates planar regions with smooth boundaries by the union of the inner Voronoi balls of a sample set and has $O(n \log n)$ complexity, according to the inherent complexity of computing the planar Voronoi diagram of the boundary samples. However, to ensure the correctness of the solution their algorithm finds, one must provide a sample set taken according to the local feature size of the region's boundary, whose calculation requires the medial axis, which is the region one wants to compute. Our method is asymptotically faster than theirs, can handle non-smooth boundaries, and depends on a less restrictive and easier-to-follow sampling criteria, which is the absolute curvature on the boundary. For instance, in very narrow, but flat, portions of a region, their method requires a very large number of boundary samples, whereas ours is insensitive to such situation.

6.5. Optimizations

For all regions with smooth boundaries in which we tested our algorithm, adaptive sampling of the boundary curves provided better MA approximations than those obtained with uniform sampling. Two main advantages come from curvature-guided refinement of the sampling: the algorithm becomes more robust in finding the correct geometry and topology of the medial axis because the profile of normals along the boundary is better characterized; and the efficiency of the algorithm increases because of the greater concentration of samples, and therefore segments, in boundary portions of high curvature. Regions of high positive curvature more likely define valid MA branches, and thus the concentration of segments in such regions makes the algorithm spend more computational effort in tracking valid branch portions, reducing the number of wasted collision tests.

As further optimization, the construction of the RLFS function, which is the most costly part of our algorithm, can be easily parallelized. As proof of concept, we used the OpenMP library to divide the loop that implements *Find MA Extreme Points* into independent threads that examine different boundary portions and independently call *Follow MA* when a (possibly invalid) MA extreme point is found. We also parallelized the search for peer segments and the branch tracking in opposite directions in *Fix MA*. The only necessary precaution in a parallel implementation of our algorithm is protecting the reading and updating of the RLFS function from concurrent accesses. When the collision test procedure needs to read (and, accordingly, update) the RLFS function for the checked pair of segments, it must ensure consistency by reading and updating the list of RLFS function pieces of both segments at once. This is achieved by associating a semaphore to each segment and preceding the reading and updating of the RLFS function of the segments by a lock operation on the semaphores associated to both segments. The Windows API offers an all-or-none lock operation on a set of semaphores, making the implementation quite simple. After operating on the critical session, the collision test procedure unlocks both semaphores. To avoid overloading the operating system, we actually associated each semaphore to a group of segments (100 segments per semaphore in our implementation). The described parallel implementation can track multiple MA branches simultaneously, depending on the number of threads, which may lead to an early interaction between two branches that causes early termination of the tracking of overestimated branches, slightly reducing the number of collision tests performed. Our parallel implementation, running on the same previously described hardware, with 4 threads running in two cores, reduced the time for computing the RLFS from 149 milliseconds to 122 milliseconds for the region in Fig. 13 discretized into 21398 boundary segments. For the region in Fig. 14, with four holes and discretized into 1797 boundary segments, the time improvement was from 25 milliseconds to 21 milliseconds. The speedups were not impressive, probably because of the bottleneck represented by the centralized dynamic memory management and the overhead of creating and terminating threads.

6.6. Changing the structuring element

By definition, the circle is the structuring element for constructing the medial axis and reconstructing the original region from it. However, other elements may be used for constructing skeletons with different properties. We developed a simple method to allow using any object having C^1-continuous boundary with strictly positive curvature (thus necessarily convex) as structuring element. Such restriction on the shape of the element is required to define the bijective normal transformation described in Fig. 15. Given the boundary of the structuring element, the derived normal transformation is applied to the normals defined for the segments' extreme points obtained by boundary discretization. Then our algorithm for computing the MA takes place with no further modification, resulting in a skeleton similar to the medial axis, but based on a different structuring element. Fig. 16 shows the skeletons obtained using a rounded square and a rounded triangle as structuring elements, respectively, for the same shape depicted in Fig. 8. The original

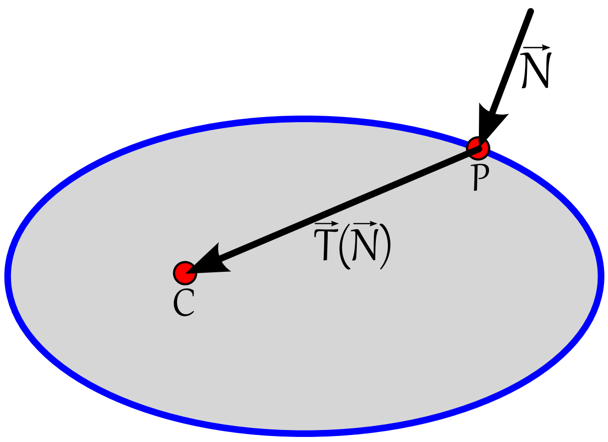


Fig. 15: The normal transformation scheme for using other structuring elements besides the disc — an ellipse in this case. First the point P, on the boundary of the structuring element, where the normalized normal is equal to input normalized normal N is found. The transformed normal $T(N)$ is the non-normalized vector from P to C. Point C is the center of the structuring element in the sense that it belongs to the medial axis when the element is a maximal one. Transformation T can be formalized as a function $T : \{x \in \mathbf{R}^2 : |x| = 1\} \rightarrow \mathbf{R}^2$.

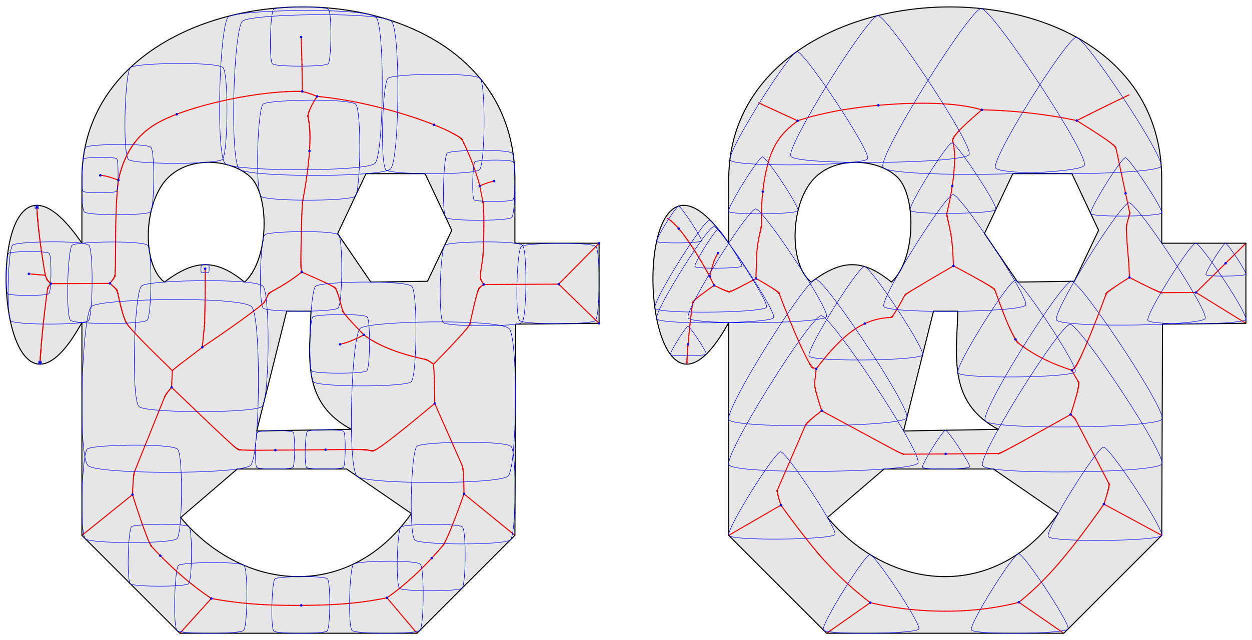


Fig. 16: Medial axes constructed using a rounded square (left) and a rounded triangle (right) as structuring element. Several maximal elements are drawn. Our algorithm was applied without any modification after transforming the normals.

shape can be reconstructed from these skeletons by union of infinite copies of the structuring element centered at all skeleton points and scaled by the radius function. The figures show several maximal elements, confirming the correctness of the result.

6.7. Robustness

Our algorithm to compute the RLFS function (Section 3.4) is fairly robust. The branches are tracked from the extreme points of the medial axis and, therefore, even if a numerical, undersampling, or oversampling problem stops the tracking of a branch, later this branch will likely be tracked as the joining of other two branches. In extreme cases, the *Fix MA* routine (Section 3.4.4) may be necessary to complete the medial axis, even in regions with no holes, incurring in an acceptable performance penalty. Such situations are rare, though. In fact, we always call *Fix MA* as the last step of our algorithm to compute the RLFS function. In regions with no holes, this routine normally only marches along the boundary without causing any change.

The algorithm for extracting the explicit medial axis is also robust due to its simplicity. It only constructs medial axis segments by displacement of boundary samples, and connects the segments by adjacency, which is a robust operation because the RLFS function is continuous and defined for every boundary segment.

Using standard double precision floating point arithmetic we did not experiment any numerical problems in the current implementation of our algorithm. Simple strategies to handle floating point quantization issues were sufficient, such as testing whether the difference between two values is below a threshold to consider them equal, instead of using direct comparison.

7. Conclusion

The method for computation of the medial axis transform of general planar regions that we have presented constitutes one more step toward the development of efficient algorithmic solutions for this still challenging problem. We showed that the execution time of our algorithm linearly depends on the density of the region's boundary sampling, which makes it asymptotically faster than previous solutions. New ideas were proposed, mainly the implicit representation of the medial axis through the Radial Local Feature Size (RLFS) function, which is defined along the region's boundary. Fast and simple algorithms to extract the explicit MA and offset curves from the RLFS function were also proposed. Finally, we presented a detailed analysis of our method in complex domains and under different sampling conditions, demonstrating its efficiency and generality.

Although the introduced implicit representation of the medial axis offers new possibilities of algorithms, we believe that extending and implementing our method for three-dimensional regions would be quite difficult. We aim at developing an approach for 3D MAT using implicit MA representation either by adapting the brute force alternative and using GPU acceleration, or by developing a divide-and-conquer strategy. Both attempts raise interesting challenges. A significant effort was made to find a fast and simple algorithm to construct the explicit MA from its implicit representation, but we think that the RLFS function may be even more useful than the explicit MA in some applications. A deeper investigation of the properties and applications of the RLFS function still needs to be done. We are currently working on the efficient computation of the medial axis of deformable shapes using the implicit MA representation, since it can be easily managed and updated.

References

[1] H. Blum, A transformation for extracting new descriptors of shape, in: W. W. Dunn (Ed.), Models for the Perception of Speech and Visual Form, MIT Press, Cambridge, 1967, pp. 362–380.

[2] D. W. Storti, G. M. Turkiyyah, M. A. Ganter, C. T. Lim, D. M. Stal, Skeleton-based modeling operations on solids, in: SMA'97: Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications, ACM, 1997, pp. 141–154.

[3] V. Srinivasan, L. Nackman, J.-M. Tang, S. Meshkat, Automatic mesh generation using the symmetric axis transformation of polygonal domains, Proceedings of the IEEE 80 (9) (1992) 1485–1501.

[4] M. Meyer, R. Whitaker, R. M. Kirby, C. Ledergerber, H. Pfister, Particle-based sampling and meshing of surfaces in multimaterial volumes, IEEE Transactions on Visualization and Computer Graphics 14 (6) (2008) 1539–1546.

[5] H. I. Choi, C. Y. Han, The medial axis transform, in: G. Farin, J. Hoschek, M.-S. Kim (Eds.), Handbook of Computer Aided Geometric Design, North-Holland, 2002, pp. 451–471, chapter 19.

[6] D.-S. Kim, I.-K. Hwang, B.-J. Park, Representing the Voronoi diagram of a simple polygon using rational quadratic Bézier curves, Computer-Aided Design 27 (8) (1995) 605–614.

[7] F. Y. Chin, J. Snoeyink, C. A. Wang, Finding the medial axis of a simple polygon in linear time, Discrete and Computational Geometry 21 (3) (1999) 405–420.

[8] D. T. Lee, Medial axis transformation of a planar shape, IEEE Transactions on Pattern Analysis and Machine Intelligence 4 (4) (1982) 362–369.

[9] C. Yao, J. Rockne, A straightforward algorithm for computing the medial axis of simple polygon, International Journal of Computer Mathematics 39 (1–2) (1991) 51–60.

[10] L. Cao, Z. Jia, J. Liu, Computation of medial axis and offset curves of curved boundaries in planar domains based on the Cesaro's approach, Computer Aided Geometric Design 26 (4) (2009) 444–454.

[11] N. Amenta, M. Bern, D. Eppstein, The crust and the β-skeleton: combinatorial curve reconstruction, Graphical Models and Image Processing 60 (2) (1998) 125–135.

[12] F. d. M. Pinto, C. M. D. S. Freitas, Fast medial axis transform for planar domains with general boundaries, in: Proceedings of the 2009 XXII Brazilian Symposium on Computer Graphics and Image Processing, IEEE Computer Society, Washington, DC, USA, 2009, pp. 96–103. doi:http://dx.doi.org/10.1109/SIBGRAPI.2009.21.

[13] T. K. Dey, W. Zhao, Approximate medial axis as a Voronoi subcomplex, in: SMA'02: Proceedings of the seventh ACM symposium on Solid modeling and applications, ACM, New York, NY, USA, 2002, pp. 356–366.

[14] J. Giesen, B. Miklos, M. Pauly, Medial axis approximation of planar shapes from union of balls: A simpler and more robust algorithm, in: CCCG, 2007, pp. 105–108.

[15] J. Giesen, B. Miklos, M. Pauly, C. Wormser, The scale axis transform, in: SCG'09: Proceedings of the 25th annual symposium on Computational geometry, ACM, New York, NY, USA, 2009, pp. 106–115.

[16] N. Amenta, M. Bern, Surface reconstruction by Voronoi filtering, in: SCG'98: Proceedings of the fourteenth annual symposium on Computational geometry, ACM, New York, NY, USA, 1998, pp. 39–48.

[17] M. Meyer, R. M. Kirby, R. Whitaker, Topology, accuracy, and quality of isosurface meshes using dynamic particles, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007) 1704–1711.

[18] H. I. Choi, S. W. Choi, H. P. Moon, Mathematical theory of medial axis transform, Pacific Journal of Mathematics 181 (1) (1997) 57–88.

[19] H. I. Choi, S. W. Choi, H. P. Moon, New algorithm for medial axis transform of plane domain, Graphical Models and Image Processing 59 (6) (1997) 463–483.

[20] H. I. Choi, C. Y. Han, H. P. Moona, K. H. Roha, N.-S. Wee, Medial axis transform and offset curves by Minkowski Pythagorean hodograph curves, Computer-Aided Design 31 (1) (1999) 59–72.

[21] R. Ramamurthy, R. T. Farouki, Voronoi diagram and medial axis algorithm for planar domains with curved boundaries – I: Theoretical foundations, Journal of Computational and Applied Mathematics 102 (1) (1999) 119–141.

[22] R. Ramamurthy, R. T. Farouki, Voronoi diagram and medial axis algorithm for planar domains with curved boundaries – II: Detailed algorithm description, Journal of Computational and Applied Mathematics 102 (2) (1999) 253–277.

[23] M. Ramanathan, B. Gurumoorthy, Constructing medial axis transform of planar domains with curved boundaries, Computer-Aided Design 35 (7) (2003) 619–632.

[24] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, M. Rabl, Medial axis computation for planar free-form shapes, Computer-Aided Design 41 (5) (2009) 339–349.

[25] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, E. Pilgerstorfer, M. Rabl, Divide-and-conquer for Voronoi diagrams revisited, in: SCG'09: Proceedings of the 25th annual symposium on Computational geometry, ACM, New York, NY, USA, 2009, pp. 189–197.

[26] L. H. de Figueiredo, Adaptive sampling of parametric curves, in: A. Paeth (Ed.), Graphics Gems V, Academic Press, 1995, pp. 173–178.