New Algorithm for Medial Axis Transform of Plane Domain

Hyong In Choi, Sung Woo Choi, and Hwan Pyo Moon

Department of Mathematics, Seoul National University, Seoul 151-742, Korea

and

Nam-Sook Wee

Department of Industrial Engineering, Hansung University, Korea

Received April 23, 1996; revised April 23, 1997; accepted July 23, 1997

In this paper, we present a new approximate algorithm for medial axis transform of a plane domain. The underlying philosophy of our approach is the localization idea based on the Domain Decomposition Lemma, which enables us to break up the complicated domain into smaller and simpler pieces. We then develop tree data structure and various operations on it to keep track of the information produced by the domain decomposition procedure. This strategy enables us to isolate various important points such as branch points and terminal points. Because our data structure guarantees the existence of such important points—in fact, our data structure is devised with this in mind—we can zoom in on those points. This makes our algorithm efficient. Our algorithm is a “from within” approach, whereas traditional methods use a “from-the-boundary” approach. This “from within” nature of our algorithm and the localization scheme help mitigate various instability phenomena, thereby making our algorithm reasonably robust. © 1997 by Academic Press

1. INTRODUCTION

The medial axis transform, which was originally proposed in [2], is a very important concept which is widely used in many different contexts: for example, besides being useful in computer aided geometric design as a means of generating the offset curves in NC machine tool path planning, it is used to represent information of a shape compactly in vision and pattern recognition. (Historically, before Blum, Dirichlet [5] and Voronoi [22] studied a discrete point version of the medial axis and this is still called a Dirichlet tessellation or a Voronoi diagram.)

Although the medial axis transform is a very useful de-
thereby localizing the problem. We then develop tree data structure and various operations on it to keep track of the domain decomposition information. This strategy enables us to isolate various important points such as branch points and terminal 1-prong points. Because our data structure guarantees the existence of such important points—in fact, our data structure is devised with this in mind—we can zoom into those points. This makes our algorithm efficient.

It should be mentioned that our method differs from most known approaches. Our method is based on interior point arguments, whereas most methods known so far use the “from-the-boundary” approach. Namely, one shoots a wave from the boundary and studies how the wave-front set meets each other. This is also the basic approach of the skeleton algorithm in mathematical morphology, although it is a discrete version [8]. The “from-the-boundary” approach, however, has a serious drawback because the perpendicular direction depends very sensitively on the \( C^1 \) perturbation of the boundary, which makes the “from-the-boundary” methods very unstable with respect to the boundary perturbation. Of course, this instability phenomenon is there in the background even in our method, but our approach mitigates this instability quite a bit.

The study of algorithms for medial axis transform has a rich history, and there are literally hundreds of articles written on various aspects of medial axis transform. Its early efforts were directed at finding the medial axis (transform) of a simply connected polygonal domain. Kirkpatrick [12] and Lee [13] obtained an \( O(n \log n) \) algorithm for such domains. It was soon generalized by many people to eventually cover multiply connected domains with boundaries consisting of line segments and circular arcs. (See [9, 13, 14, 15, 16, 20, 24].) When the boundary curves are more general than lines and arcs, there has been very little progress. Recently Chou [4] describes an interesting method to handle this general case, and Alt and Schwarzkopf [1] also discuss a method to handle the medial axis of curved objects. There also has been many interesting results on higher dimensional medial axis transform. Notably, [17, 18] provide new algorithms for 3D medial axis transform for regions with a polyhedral boundary, and [6, 10] present interesting discussions on how to construct the medial axis transform of CSG objects. Conversely, one can reconstruct the boundary from the medial axis transform. Refer to [21] for interesting results on this aspect.

While the above research were mostly directed to finding workable algorithms, there has been some recent works dealing with more theoretical aspects of the medial axis transform. Wolter [23] proved that the medial axis transform is a strong deformation retract of the domain, and later Sherbrooke, Patrikalakis, and Wolter [19] gave another proof of the same result. These results work for higher dimension too. Hoffmann and Chiang [11] studied the domain decomposition lemma and obtained the same result when the domain is simply connected. They also discuss the domain decomposition idea when the domain has holes. Their approach is similar to ours in spirit, but theirs is not exactly in the form that we want for our algorithm to work. The above mentioned works study other various interesting aspects of medial axis transform, and in particular, they all proved the medial axis transform is path connected. The results in [3], which was carried out without prior knowledge of the above mentioned authors, deal with the theoretical issues systematically. Among other things, they proved that the medial axis transform is a \textit{finite} geometric graph together with the regularity results. But from the algorithmic point of view, the most important facts from this theory are various finiteness results: Namely, under the assumption that the boundary curve consists of finitely many real analytic curves, there are only finitely many branch circles and inscribed osculating circles. (See Definition 2.1 for the definition of branch circle.) This finiteness is crucial to guarantee that our algorithm terminates in finite time.

Our own interest stems from our work on Hangeul/Hanja font technology. In the Korean and Chinese languages, there are tens of thousands of characters. And each character is composed of simpler elements such as Jamos or strokes. Each Jamo or stroke is then transformed into different sizes while the thickness or the stem is kept fixed, or the thickness has to be increased or decreased depending on the context prescribed by the font designer. Our research grew out of this necessity. In our font application, the medial axis represents the path of the tip of the brush or pen, and the radius of the circle in the medial axis transform represents the size or the thickness of the brush. Based on this algorithm, we have currently developed a Hangeul/Hanja font design and generation system which is being put to commercial use by Hyundai Media Systems Co., Ltd. in Seoul, Korea.

2. MATHEMATICAL FACTS ABOUT MEDIAL AXIS TRANSFORM

In this section, we summarize some mathematical results in [3] about the medial axis transform of plain domain. For more details, see [3] in which mathematically rigorous definitions and proofs are given. We also follow the notations and the terminologies in [3]. But for the convenience of the reader, we list some frequently used ones here.

2.1. Assumptions on Domain

A domain \( \Omega \) in our algorithm is always assumed to satisfy the standing assumption in [3], which we list below.

Standing Assumption. We will assume that the domain
\[ \Omega \] is a noncircular domain which satisfies the following two conditions.

1. \( \Omega \) is the closure of a connected bounded open subset in \( \mathbb{R}^2 \) bounded by a finite number of mutually disjoint simple closed curves. (Here a simple closed curve means an embedding of the unit circle in \( \mathbb{R}^2 \).)

2. Each simple closed curve in \( \partial \Omega \) consists of a finite number of pieces of real analytic curves. (We use the notation that for a set \( A \) in \( \mathbb{R}^2 \), \( \partial A \) denotes its boundary.)

We exclude the circular domain, i.e., the disk, since the disk poses an exception to many of our technical results, while everything is trivial in this case. The simple closed curve bounding the unbounded region of \( \mathbb{R}^2 \setminus \Omega \) is called the outer boundary (curve), and the rest are called the inner boundary (curves). The number of the inner boundary curves in \( \Omega \) is called the genus of \( \Omega \). A domain has an inner boundary curve if and only if it is not simply connected (i.e., multiply connected). We also describe this situation by the words such as “\( \Omega \) has a hole (holes),” or “\( \Omega \) has homology.”

Of all the conditions in the above standing assumption, the real analyticity condition is the most important and needs some explanation. We say that a simple closed curve \( \gamma : [a, b] \rightarrow \mathbb{R}^2 (\gamma(a) = \gamma(b)) \) consists of finite number of pieces of real analytic curves, if there are numbers \( a = t_0 < \cdots < t_n = b \), such that \( \gamma|_{[t_i, t_{i+1}]} \) is a real analytic curve for each \( i = 1, \ldots, n \). As mentioned above, we can easily construct some pathological examples without the real analyticity conditions (see Examples 2.1 and 2.2 in [3]).

2.2. Terminologies

Let \( B_r(p) \) denote the closed disk of radius \( r \) centered at \( p \). We define the ordered set \( D(\Omega) \) by \( D(\Omega) = \{ B_r(p) \mid B_r(p) \subset \Omega \} \). That is, \( D(\Omega) \) is the set of all disks contained in \( \Omega \).

The core, \( \text{CORE}(\Omega) \), of a domain \( \Omega \) is the set of all maximal disks in \( \Omega \); i.e., it consists of \( B_r(p) \in D(\Omega) \) such that if \( B_s(q) \in D(\Omega) \) contains \( B_r(p) \), then \( B_s(q) = B_r(p) \).

A disk \( B_r(p) \) in \( \text{CORE}(\Omega) \) is called a maximal disk, and its boundary circle \( \partial B_r(p) \) is called a maximal (inscribed) circle or a contact circle. The medial axis of a domain \( \Omega \) is the set of all centers of disks in \( \text{CORE}(\Omega) \). That is, \( \text{MA}(\Omega) = \{ p \in \Omega \mid B_r(p) \in \text{CORE}(\Omega) \} \).

The medial axis transform of a domain \( \Omega \) is the set of all ordered pairs of centers and radii of disks in \( \text{CORE}(\Omega) \). That is, \( \text{MAT}(\Omega) = \{ (p, r) \in \Omega \times \mathbb{R} \mid B_r(p) \in \text{CORE}(\Omega) \} \). Here, we allow \( r = 0 \) and regard \( B_r(p) \) as \( \{ p \} \) in this case. Such cases occur exactly at the sharp corners of \( \partial \Omega \).

A boundary point is a corner (point) if the unit tangent vector field is discontinuous at that point. It is called a sharp (resp., dull) corner if the interior angle is strictly less (resp., greater) than \( \pi \).

Let \( \gamma(t) \) be a piecewise real analytic curve which is a part of the boundary of \( \Omega \). Let us assume that \( \gamma \) is oriented in such a way that the interior of \( \Omega \) is always on the left of \( \gamma \). Let \( q = \gamma(0) \) and suppose \( q \) is not a sharp corner. We define the inward unit cone \( \text{IC}(q) \) at \( q \) to be a set of vectors \( \vec{\tau} \) such that \( |\vec{\tau}| = 1 \) and \( \vec{\tau} \) is an inward pointing vector at \( q \) in the sense that \( \langle \vec{\tau}, \gamma'(0+) \rangle \leq 0 \) and \( \langle \vec{\tau}, \gamma'(0-) \rangle \geq 0 \). Note that if \( \gamma \) is differentiable at \( q \), \( \text{IC}(q) \) consists of the single inward unit normal vector.

For a point \( p \in \text{MA}(\Omega) \), we denote by \( B_r(p) \) the disk \( B_r(p) \) in \( \text{CORE}(\Omega) \) with the center \( p \). Let \( B_r(p) \) be a disk in \( \text{CORE}(\Omega) \). Then we define the contact set of \( p \) (or of \( B(p) \) or of \( \partial B(p) \)), denoted by \( \text{C}(p) \), to be the set \( \partial B(p) \cap \partial \Omega \).

A point in \( \text{C}(p) \) is called a contact point of \( p \) (or of \( B(p) \) or of \( \partial B(p) \)). A connected component of \( \text{C}(p) \) is called a contact component of \( p \) (or of \( B(p) \) or of \( \partial B(p) \)). A contact component is called an isolated contact point if it is a point and a contact arc if it is an arc containing its two end points. Finally, \( \partial B(p) \) is called a contact circle.

In fact, it is a mathematically trivial fact that a contact component is either an isolated contact point or a contact arc. Now we can characterize the medial axis points by the number of their contact components.

Definition 2.1. A point \( p \) in \( \text{MA}(\Omega) \), which is not a sharp corner point, is called an n-prong point \( (n \geq 1) \), if \( \text{C}(p) \) has \( n \) contact components. For reasons that become clear when we deal with the graph structure of the medial axis, we classify the sharp corner points as 1-prong points. An n-prong point \( p \) for \( n \geq 3 \) is called a branch point or a bifurcation point. A 1-prong point \( p \) is also called a terminal point.

Let \( (p, r) \) be in \( \text{MAT}(\Omega) \). We call \( B_r(p) \) a branch disk or a bifurcation disk if \( p \) is a branch point. In this case, \( \partial B_r(p) \) is called a branch circle or a bifurcation circle. A disk \( B_r(p) \) in \( \text{CORE}(\Omega) \) is called an osculating disk at \( q \in \partial \Omega \) if \( \partial B_r(p) \) is an inscribed circle which osculates \( \partial \Omega \) at \( q \). In fact, around an n-prong point \( p \) \( (n \geq 1) \), the medial axis \( \text{MA}(\Omega) \) has exactly \( n \) “prongs” emanating from \( p \). (See [3] for the proof.)

It is a fact that a terminal (i.e., 1-prong) point which is not a sharp corner is the center of an inscribed osculating circle, where an inscribed osculating circle of \( \Omega \) is a circle contained in \( \Omega \) which osculates \( \partial \Omega \) at some point of \( \partial \Omega \). Also, we can easily see, with some geometric argument, that the curvature of \( \partial \Omega \) takes a local maximum at an osculating point of an inscribed osculating circle. (See Theorem 3.1 in [3] for the proof of these facts.)

2.3. Mathematical Results

Definition 2.2. A 2-prong point \( p \in \text{MA}(\Omega) \) is a generic 2-prong (point), if the following conditions are satisfied:
(1) The two contact components of \( p \) are isolated contact points (denoted by \( q_1 \) and \( q_2 \)).

(2) If \( q_i \) (\( i = 1, 2 \)) is not a dull corner, then \( \partial \Omega \) near \( q_i \) is real analytic, and \( p \) is within the focal locus of a small piece of \( \partial \Omega \) near \( q_i \). (See [3] for the definition of being within the focal locus.)

(3) If \( q_i \) (\( i = 1, 2 \)) is a dull corner, then \( q_i \) is in a purely interior direction of \( q_i \).

It is proved in [3] that each of the following is finite in \( MA(\Omega) \):

(1) The number of contact components of a point.

(2) The number of 1-prongs.

(3) The number of branch points.

(4) The number of 2-prongs which are not generic.

Another important mathematical result in [3] is that \( MA(\Omega) \) and \( MAT(\Omega) \) have the structure of the geometric graph. We call a set in \( \mathbb{R}^2 \) (or in \( \mathbb{R}^3 \)) a geometric graph if it is topologically a usual connected graph with a finite number of nodes and edges, where a node is a point in \( \mathbb{R}^2 \) (or in \( \mathbb{R}^3 \)) and an edge is a real analytic curve with a finite length whose limits of tangents at the end points exist.

In fact, \( MA(\Omega) \) and \( MAT(\Omega) \) are isomorphic as graphs, and besides the above results, we can get the following correspondence between the node degree of a point in \( MA(\Omega) \), \( MAT(\Omega) \), and the geometric property of that point [3]. Let \( (p, r) \) be a point in \( MAT(\Omega) \).

(1) If \( p \) (resp., \( (p, r) \)) is in an edge of \( MA(\Omega) \) (resp., \( MAT(\Omega) \)), then \( p \) is a generic 2-prong, and thus \( MA(\Omega) \) (resp., \( MAT(\Omega) \)) is real analytic at \( p \) (resp., \( (p, r) \)).

(2) If \( p \) (resp., \( (p, r) \)) is a node of degree 1 in \( MA(\Omega) \) (resp., \( MAT(\Omega) \)), then \( p \) is either a sharp corner or the center of an inscribed osculating circle with one contact component.

(3) If \( p \) (resp., \( (p, r) \)) is a node of degree 3 or higher in \( MA(\Omega) \) (resp., \( MAT(\Omega) \)), then \( p \) is a branch point.

(4) If \( p \) (resp., \( (p, r) \)) is a node of degree 2 in \( MA(\Omega) \) (resp., \( MAT(\Omega) \)), then \( p \) is a 2-prong which is not generic.

It should be noted that the graph structure of \( MAT(\Omega) \) as described above is different from the tree data structure which represents \( MAT(\Omega) \) in Section 4. The main difference is that each single edge in the geometric graph of \( MAT(\Omega) \) may be subdivided into many smaller edges in the tree data structure to facilitate the various algorithmic steps.

2.4. Domain Decomposition Lemma

In this section, we introduce a fundamental tool for our algorithm, called the Domain Decomposition Lemma, which enables us to localize the argument (Fig. 1).

**Theorem 2.1** (Domain Decomposition Lemma). For any fixed medial axis point \( p \in MA(\Omega) \), let \( B(p) = B_r(p) \) be the corresponding maximal disk, i.e., \( B(p) \in CORE(\Omega) \).

Suppose \( A_1, \ldots, A_n \) are the connected components of \( \Omega \setminus B(p) \). Denote \( \Omega_i = A_i \cup B(p) \) for \( i = 1, \ldots, n \). Then

\[
MA(\Omega) = \bigcup_{i=1}^{n} MA(\Omega_i) \quad \text{and} \quad MAT(\Omega) = \bigcup_{i=1}^{n} MAT(\Omega_i).
\]

Moreover, we have

\[
MA(\Omega_i) \cap MA(\Omega_j) = \{p\} \quad \text{and} \quad MAT(\Omega_i) \cap MAT(\Omega_j) = \{(p, r)\},
\]

for every distinct \( i \) and \( j \).

See [3] for the proof of this lemma. Our MAIN ALGORITHM describes how to reduce the domain into the following simple building blocks.

**Definition 2.3** (Fundamental Domain). A domain \( \Omega \) is a fundamental domain, if \( MA(\Omega) \) has no bifurcation points.

3. BOUNDARY TRAVERSING SCHEME OF SUBDOMAINS

Our basic strategy to find the medial axis (transform) is to decompose the domain into smaller and simpler subdomains by using the Domain Decomposition Lemma. Thus the main algorithmic part of this paper is how to organize this information by devising a relevant data structure and operations on it. In this section, we explain in more intuitive terms the basic ingredients of our subdomain representation. The important point in this section is how each contact point of a contact circle represents a subdomain by specifying its boundary through our boundary traversing scheme.
3.1. Orientation Convention

We follow the general rule of determining the orientation as the usual one, i.e., if a person is walking along the \( \partial \Omega \) with this orientation, then the domain should always be on his or her left. So when the domain \( \Omega \) has homology, the outer boundary has the counterclockwise orientation and the inner boundaries have the clockwise orientation. Also, we will fix the orientation of the contact circle. For any contact circle \( B(x) \), the orientation of \( \partial B(x) \) is defined to be the counterclockwise orientation, that is, positive orientation with respect to \( B(x) \). Then this orientation is consistent with the orientation of \( \partial \Omega \).

3.2. Boundary Traversing Scheme

As we explained above, we will decompose a given domain into several subdomains by core disks. In this process, each contact point determines a unique subdomain. (But several contact points may determine the same subdomain.) Now let us consider the relations between the contact points and the subdomains.

A contact circle may have arcs, but for simplicity, we will assume without loss of generality throughout this paper that there are no contact components other than isolated contact points. We now describe how such a contact point determines a unique subdomain through our boundary traversing scheme.

**Boundary Traversing Scheme**

*Step 1.* Start with a given contact point \( S \) of a given contact circle \( C_1 \).

*Step 2.* Traverse along the boundary \( \partial \Omega \) of the original domain following the orientation of \( \partial \Omega \) until it first reaches a contact point \( E \) of a contact circle, say \( C_2 \).

*Step 3.* Enter the circle \( C_2 \) at \( E \) and traverse along the boundary of \( C_2 \) following the orientation of \( C_2 \) until it reaches the exit point \( X \). The exit point \( X \) is the last contact point before it reaches the entering point \( E \). If \( E \) is the only contact point of \( C_2 \) (as is the case with a 1-prong inscribed osculating circle), then exit immediately at \( E \) without ever traversing along the boundary of \( C_2 \). In this case, however, we regard \( C_2 \) as visited.

*Step 4.* Exit \( C_2 \) at \( X \). If \( X \) is the starting point \( S \) of \( C_1 \), stop. Otherwise, go to Step 2 with \( X \) as a new starting point.

Figure 2 illustrates the convention for entering and exiting a contact circle. One should note that we do not exclude the possibility that \( C_2 \) may be the same as \( C_1 \) as in Fig. 3. In Step 2 above, there may be no portion of \( \partial \Omega \) between \( S \) and \( E \) as shown in Fig. 4. In that case, \( S = E \) and it enters \( C_2 \) at \( E \) immediately after it leaves \( C_1 \) at \( S \).

Even though the original domain has holes, our traversing scheme works. In this case, a closed curve generated by the boundary traversing scheme may not enclose a genuine subdomain of \( \Omega \). But our algorithm is devised to work properly as if it were a genuine subdomain (after appropriate initialization). See Section 4.4 for more details on the initialization. We illustrate this phenomenon in Fig. 5.

Our domain has one hole inside. If we have only one contact circle, we get a closed curve with self intersection which is schematically represented as

\[
1 \rightarrow a \rightarrow 1 \rightarrow b \rightarrow 2 \rightarrow c \rightarrow 2 \rightarrow d \rightarrow 1.
\]

In that case, it is as if we have a domain on the right, and...
the original domain is obtained by identifying the two disks in the domain on the right.

We call such process of opening up a domain a split, and the reverse process a merge process. But one must note that these split and merge processes do not actually take place. The use of the term “split” and “merge” is for the convenience of the reader’s imagination. And our algorithm automatically handles both without an actual “split” or “merge.” This process in our algorithm is called homology killing. See Section 4.4.1.

Remark 3.1. We emphasize again that the contact points may very well be contact arcs. We use contact points for the sake of simplicity of presentation. The required modifications are obvious, so they are left to the reader.

3.3. Subdomain Labeling Convention

The boundary traversing scheme described above assigns to a contact point of a contact circle a subdomain bounded by the curve traversed. For example, in Fig. 3, the contact point S denotes (labels) the shaded subdomain. Figure 6 illustrates the convention for a more complex domain. We have four subdomains, \( \Omega_1, \Omega_2, \Omega_3, \) and \( \Omega_4. \) The numbers assigned to each contact point represent the corresponding subdomain. Thus the number 3 at the rightmost contact circle indicates that the corresponding contact point labels (determines) subdomain \( \Omega_3 \) and so on.

![FIG. 6. Example of labeling convention.](image)

4. TREE DATA STRUCTURE AND OPERATIONS ON TREE

Many people have attempted to obtain formulas that describe the medial axis (transform). But our approach is different. We successively find contact circles and organize the data in such a way as to reveal the geometric and topological structure of the medial axis. In this section we do not concern ourselves with how to find such contact circles. It is the topic of Section 5. In this section, we are assuming the contact circles are successively found, and we describe the data structure and the algorithm for organizing the information.

4.1. Data Structure

We organize the data in a tree. Each node represents the contact circle chosen. It also contains pertinent information such as the center, the radius, the contact points, and which boundary curve of \( \partial \Omega \) each contact point is on.

Finding the next contact circle subdivides the previously constructed subdomain (via our boundary traversing scheme). That is codified into our tree data by introducing new nodes.

4.2. Tree Data and Basic Subdomain

4.2.1. Edge. Two adjacent two-prong contact circles determine a subdomain as in Fig. 7, and the corresponding information is codified in the tree as an edge connecting two nodes. To illustrate, the contact point \( a \) in Fig. 7 represents a subdomain which is bounded by the curve traversed in the order

\[
 a \rightarrow b \rightarrow \alpha \rightarrow c \rightarrow d \rightarrow \beta \rightarrow a.
\]

In completing this traverse, Circle 2 encounters Circle 1 but no other contact circles, and similarly the traverse from the contact point \( c \) also encounters Circles 1 and 2 only. Thus Circles 1 and 2 bound a common subdomain, and we connect nodes 1 and 2 with an edge. Traversing from \( b \) or from \( d \) goes to somewhere else; thus the nodes 1 and 2 have edges connected to some other nodes.

![FIG. 7. Subdomain and tree data structure.](image)
4.2.2. Virtual Node. Three or more circles may share a common subdomain as in Fig. 8. For example, by traversing the domain from \( a \) as

\[ a \rightarrow b \rightarrow \alpha \rightarrow c \rightarrow d \rightarrow \beta \rightarrow e \rightarrow f \rightarrow \gamma \rightarrow a, \]

we encounter Circles 1, 2, and 3. This means that Circles 1, 2, and 3 bound a common domain.

This situation is codified in the tree as having one fictitious node, called a virtual node and marked as “V,” which connects nodes each of which bounds the same subdomain. The reason for introducing a virtual node is that the mathematical result in [3] guarantees the existence of at least one branch circle. Thus this virtual node represents the existence of the not-yet-found branch circle. And the edge connecting node 1 and the virtual node \( V \) is expected to represent the not-yet-found subdomain bounded by the Circle 1 and the not-yet-found branch circle.

4.2.3. Terminal Node. A terminal node in our tree may represent two different situations. The first is the case when the terminal node represents a 1-prong inscribed osculating circle or a sharp corner. This is illustrated in Fig. 9, where we use node 2 as a terminal node.

In the second case, Circle 2 is not a 1-prong oscillating circle, and it bounds a subdomain by itself. Then this subdomain has a 1-prong inscribed osculating circle in it. This inscribed osculating circle is represented as a terminal virtual node as in Fig. 10. In this case, this virtual node has to be resolved by the algorithm explained in Section 5.5. Or it may have to be subdivided by inserting more circles. The boundary traversing scheme to detect the cases in Fig. 9 or Fig. 10 is very similar to that explained in the above subsections. The reason for introducing a terminal virtual node is similar to that given in Section 4.2.2, i.e., the mathematical result in [3] guarantees the existence of a 1-prong inscribed osculating circle. (See Lemma 6.1 in [3] and remarks thereafter.)

4.3. Operations on Tree

Our strategy is to add a circle at each step and reorganize this new information in our tree data structure by operating on the tree. Before we present the full procedure, let us look at a few examples to help understand this procedure.

The first case is when a new circle, denoted by \( C \), is found, and suppose \( a \) and \( b \) are its contact points. Start from \( a \) and run the boundary traversing scheme as described in the previous section. And do the same starting from \( b \). Figure 11 illustrates this case.

The boundary traversing from \( a \) is

\[ a \rightarrow c \rightarrow d \rightarrow b \rightarrow a, \]

and this visits Circle 2 besides \( C \). Similarly the boundary traversing from \( b \) terminates at \( b \) after visiting Circles 1 and \( C \) itself. This traverse is

\[ b \rightarrow e \rightarrow f \rightarrow a \rightarrow b. \]

Clearly Circles 1 and 2 were joined by an edge before the insertion of \( C \). Disconnect this edge in the tree and insert a new node representing \( C \), denoted also by \( C \), and then

Before the insertion of \( C \)

\[ \begin{array}{c}
\begin{array}{c}
1 \\
2
\end{array}
\end{array} \]

After the insertion of \( C \)

\[ \begin{array}{c}
\begin{array}{c}
1 \\
\begin{array}{c}
C \\
2
\end{array}
\end{array}
\end{array} \]
connect $C$ with node 2 with an edge and also connect $C$
with node 1 with an edge. So the tree operation looks like
Fig. 12.

Another possibility is illustrated in Fig. 13. The boundary traversing starting from $b$ terminates at $b$ after visiting
Circles 1, 2, and $C$, and the boundary traversing from $a$ terminates at $a$ after visiting no circles other than $C$ itself.
Thus the tree operation looks like Fig. 14.

A more complicated example involving the virtual node is given in Fig. 15. After insertion of $C$ and traversal of
the boundary from $a$ and $b$, it is obviously found that $C$ is
adjacent to Circles 1 to 5, which are connected to each other via a virtual node $V$. Now destroy the original virtual
node and the edges connected to it. The boundary traversing from $a$ visits Circles 1, 2, 3, and $C$ and the boundary traversing from $b$ visits Circles 4, 5, and $C$. Thus nodes 1, 2, 3, and $C$ are joined via a virtual node, and nodes 4, 5, and $C$ are joined via another virtual node.

Another operation on the tree data structure is the resolu-
tion of the virtual node (or the terminal virtual node). This procedure is illustrated in Fig. 16. Namely, the new actual contact circle $C$ replaces the virtual node $V$.

We now organize the procedures in this section as a
pseudo code. In what follows, a subdomain is represented
by a contact point as explained in Section 3.3, and $B_r(x)$
is the contact circle in the given subdomain, which will be
inserted in the data structure. And the array, contact_point[], means the array of contact points of $B_r(x)$.

![Figure 12](image1.png) **FIG. 12.** Insertion operation on tree.

![Figure 13](image2.png) **FIG. 13.** New circle contacts on the same boundary.

![Figure 14](image3.png) **FIG. 14.** Insert the new node and introduce two virtual nodes.

![Figure 15](image4.png) **FIG. 15.** A complicated example of tree operation.

![Figure 16](image5.png) **FIG. 16.** Resolution of a virtual node.
**procedure** INSERT_CIRCLE (subdomain Ω, $B_i(x)$, contact_points[ ])

traverse the boundary of Ω;

if the number of the visited circles are not 2 then
    delete the common virtual node connected to the nodes
    bounding the traversed subdomain;

for $i := 1$ to number of the contact points do

begin
    traverse the boundary starting from contact_points[$i$];
    if visit no other circle then
        attach a terminal virtual node;
    else if visit only one other circle then
        connect this circle to $B_i(x)$;
    else
        create a virtual node;
        connect all circles traversed and $B_i(x)$
        through this virtual node;

end;
end INSERT_CIRCLE

**Remark 4.1.** In our tree operations, it is tacitly assumed that the circles traversed from one contact point of $B_i(x)$ is different from those traversed from the other points of $B_i(x)$. Otherwise, we may have trouble with our tree structure because some traversals may not bound a domain if the domain is not simply connected. In reality, we are avoiding this problem by our initialization procedure of killing homology in Section 4.4.1. In Section 4.4.1, when the domain has $n$ holes, the initialization procedure finds $n$ special contact circles each of which is duplicated as two nodes in the tree data structure. (This is morally like the split process of Fig. 5 of Section 3.2.) Two nodes generated from the same special circle are distinguished by marking the corresponding contact point on each. The boundary traversing procedure detects those duplicate circles (or nodes) by observing the convention in Section 4.4.1. Thus when we refer to “Circles” in our tree data structure, what we really mean are the nodes with special marking in care of the special nodes generated by the killing homology procedure. This procedure produces a “simply connected” domain for the purpose of the algorithm.

### 4.4. Initialization Procedures

**4.4.1. Killing Homology.** Killing homology is one of the key initialization steps, which drastically simplifies the data structure compared with that without this step. After this initialization, the domain Ω behaves like a simply connected domain in our algorithm.

If Ω has $n$ inner boundary curves, i.e., $n$ holes, the boundary $\partial \Omega$ consists of $n + 1$ simple closed curves. We fix the coordinate system and choose a special reference direction. In the following, we choose the positive $y$-direction as the reference direction. First, compute the maximum values of the $y$-coordinate of each boundary curve, and order the boundary curves $\gamma_0, \gamma_1, \ldots, \gamma_n$ according to these maximum values. Let $q_i$ be the point which has the maximum $y$-coordinate of $\gamma_i$ for $i \geq 1$. With this indexing scheme, $\gamma_0$ has the highest $y$-coordinate, so it is the outer boundary curve. We now find the contact circle $B(p_i)$, which contacts $\gamma_i$ at $q_i$ for $i \geq 1$. Note that the vector $\overrightarrow{q_i p_i}$ is pointing in the positive $y$-direction. Thus since any points on $\gamma_j$ for $j > i$ cannot lie below $q_i$, the contact circle $B(p_i)$ does not contact $\gamma_j$ for $j > i$, and also $B(p_i)$ does not contact $\gamma_i$ except at the contact point $q_i$. We illustrate this in Fig. 17.

Let us now show by induction that these contact circles $B(p_i)$ join all boundary curves to the outer boundary curve $\gamma_0$. After finding $i - 1$ contact circles $B(p_1), \ldots, B(p_{i-1})$, by the induction hypothesis, one can assume the boundary curves $\gamma_1, \ldots, \gamma_{i-1}$ are joined to $\gamma_0$. And then the $i$th contact circle $B(p_i)$ joins the boundary curve $\gamma_i$ to one (or more) of the previous boundary curves $\gamma_0, \ldots, \gamma_{i-1}$. So $\gamma_i$
is also joined to the outer boundary curve $\gamma_0$. With this process, each $B(p_i)$ does not contact $B(p_k)$ for $k > i$. Therefore, this process of connecting the boundary curves stops in $n$ steps, where $n$ is the number of inner boundary curves (holes).

Now we summarize this as a pseudo code below.

```
procedure KILL_HOMOLOGY(boundary curves $\gamma[ ]$)
  if these are no inner boundary curves then exit;
  create a virtual node;
  compute the highest point of each boundary;
  sort the boundary curves according to the $y$-coordinate of the highest point;
  for $i:=1$ to the number of the inner boundary curves do
    find the contact circle $B(p_i)$ at $q_i$;
    attach $B(p_i)$ to the virtual node;
  end;
end;
```

4.4.2. **Circular Arc.** In case there are circular arcs on the boundary curve, we draw a full circle from each circular arc, and check if this circle is inside $\Omega$. If so, we add it to our tree data. Otherwise, we do nothing.

4.4.3. **Corner.** Corners occupy a special place. The medial axis always goes to the sharp corners; but the behavior of the medial axis near the sharp corner is well understood, so they do not require any special treatment, except that it is advantageous to mark them as sharp corners. But dull corners require special attention. Let $\gamma(t)$ be a (part of) boundary curve near $q = \gamma(0)$, where $q$ is a dull corner, and let $IC(q)$ be the inward unit cone at $q$.

Let $v_1$ and $v_2$, as in Fig. 18, be the unit inward pointing vectors at $q$ such that $(v_1, \gamma'(0-)) = 0$ and $(v_2, \gamma'(0+)) = 0$, respectively. Then $IC(q)$ lies between $v_1$ and $v_2$ in the unit circle.

Draw contact circles $C_i$ at $q$ which are normal to $v_i$ for $i = 1, 2$. Add these circles, $C_1$ and $C_2$, to the tree data structure, and repeat this at each dull corner. The geometric algorithm for finding these circles, $C_1$ and $C_2$, will be given in Section 5.2.

4.4.4. **Inscribed Osculating Circle.** The inscribed osculating circles are important for detecting terminal points in $MA(\Omega)$. Since the curvature is a local maximum at the point where an inscribed osculating circle osculates, one may find those points, and check if the osculating circle at each of those points is contained in $\Omega$. If so, one can add those circles to the tree data structure. However, since the curvature involves the second derivatives, it may not be numerically stable to find such maximum points. In that case, one can skip this initialization procedure altogether. But our MAIN_ALGORITHM takes care of these points by designating them as terminal virtual nodes in due course and then resolving them by the procedure in Section 5.5.

5. GEOMETRIC ALGORITHM

In Section 4, we illustrated how to represent the data and how to manipulate it without explaining how to find such contact circles in practice. In this section, we present various geometric algorithms to find such contact circles.

5.1. **2-Prong Contact Circle**

The workhorse of our algorithm is the 2-prong contact circle. As it is important to place a 2-prong contact circle as near a designated spot as possible, we have developed here a geometric algorithm to place a 2-prong contact circle inside a given (larger) circle. The starting point of this algorithm is a choice of a circle $B_r(x)$ centered at an interior point $x$ which contains two boundary portions $\gamma$ and $\delta$ of $\partial \Omega$ as in Fig. 19. We select any point $y$ on $\gamma$ such that $d(x, y) = d(x, \gamma)$, and any point $z$ on $\delta$ such that $d(x, z) = d(x, \delta)$. If $d(x, y) = d(x, z) = r$, then $B_r(x)$ is the one we are looking for. Otherwise, we choose, among $y$ and $z$, one that is farther from $x$, and call it $y$ and the other $z$. We will look for the circle containing $y$ with the center on the line segment $\overline{xy}$. Note that any disk $B_a(x')$ such that $x' \in \overline{xy}$ and $d(x', y) = a' < a$ meets with $\gamma$ only at $y$.

If $\overline{xy} \cap \delta \neq \emptyset$, the situation like in Fig. 20 may occur. The trouble is that the whole segment $\overline{xy}$ may not be

![FIG. 18. Initialization procedure at a dull corner.](image-url)
entirely in the domain. So we choose the point $z'$ on $\overline{xy} \cap \delta$ which is closest to $y$ so that $\overline{yz'}$ is now entirely in the domain $\Omega$. Then we should replace $z$ with $z'$ and choose the middle point of $\overline{yz'}$ as a new $x$. We may now assume that $\overline{xy} \cap \delta = \emptyset$.

Let us draw a circle $B_{a_1}(x_1)$ containing $z$ on its boundary such that $x_1 \in \overline{xy}$. Then $a_1 \leq a$. So we get the situation shown in Fig. 21. And then let us fix a point $z_1$ such that $d(x_1, \delta) = d(x_1, z_1)$.

By applying this algorithm recursively, we get the sequences $\{a_n\}, \{x_n\}$, and $\{z_n\}$. Since $a_n$ is a monotone decreasing sequence and has a lower bound, there exists the limit $a_0$. And $x_n$ also converges to a point $x_0$ in $\overline{xy}$ such that $d(x_0, y) = a_0$. Then the disk $B_{a_1}(x_0)$ contacts $\gamma$ at $y$ and contacts $\delta$ at some point $z_0 = \lim_{n \to \infty} z_n$. So $B_{a_0}(x_0)$ is a 2-prong contact circle. Note that since $y$ (resp., $z_0$) realizes the distance from $x_0$ to $\gamma$ (resp., $\delta$), $B_{a_0}(x_0)$ must be contained in $\Omega$. Note also that the above procedure works even if there are corners in $\gamma$ and $\delta$.

Here we will summarize this algorithm as a pseudo code.

```plaintext
procedure TWO_PRONG(x_0, r,  \delta\Omega); 
determine two boundary portions \gamma and \delta; 
choose the closest points y and z on \gamma and \delta; 
if \ d(x_0, y) = d(x_0, z) \ then 
   return x, y, z, and radius d(x, y); 
else if \ d(x_0, y) < d(x_0, z) \ then 
   begin 
      switch y and z; 
      switch \gamma and \delta; 
   end; 
if \ x_0 \gamma \ meets \ \delta \ then 
   begin 
      let \ z' \ be the closest point from y in \overline{x_0y} \cap \delta; 
      choose the middle point between y and \ z' as a new \ x_0; 
      compute z again; 
   end; 
x := x_0; 
while \ |d(x, y) - d(x, z)| > tolerance \ do 
   begin 
      update x on \overline{x_0y} such that d(x, y) = d(x, z); 
      update z on \delta such that d(x, z) = d(x, \delta); 
   end; 
return x, y, z and radius d(x, y); 
end TWO_PRONG
```
5.2. Contact Circle with Given Normal

Let \( p \in \partial \Omega \), which is not a sharp corner, and let \( N \in IC(p) \). We now present an algorithm to find a contact circle which makes contact at \( p \) and whose center lies in the half line from \( p \) in the direction of \( N \). First, compute \( k_1(p) \) and \( k_2(p) \), and let \( k = \max(k_1(p), k_2(p)) \). If \( k > 0 \), let \( R = 1/k \), draw a circle \( C(r) \) of radius \( r \) which has a contact at \( p \) and whose center lies also in the half line from \( p \) in the direction of \( N \). If \( r \) is sufficiently small, \( C(r) \) meets with \( \mathcal{V} \) only at \( p \). As we increase \( r \), one of the following two things will happen: Either \( C(r) \) first meets with \( \partial \Omega \) at a point other than \( p \) for some \( r < R \), or \( C(r) \) meets with \( \partial \Omega \) only at \( p \) for all \( r < R \). In the later case, \( C(r) \) becomes an inscribed osculating circle. And if \( k \leq 0 \), one may choose \( r \) appropriately so that \( C(r) \) contains two boundary portions. Then we apply the 2-prong algorithm with this circle. Using this idea, we will choose a circle \( C(r) \) on which we can apply the 2-prong algorithm in Section 5.1 as follows.

```plaintext
procedure GIVEN_NORMAL(p, N, \partial \Omega)
    \( \kappa := \max(\kappa'(p), \kappa(p)) \);
    if \( \kappa > 0 \) then
        \( R := 1/\kappa \);
    else
        \( R := \text{diameter of the domain} / 2 \);
        determine \( C(R) \);
        return \( C(R) \);
    end;
    begin
        \( n := 0 \);
        \( t := 1 \);
        \( r := R \);
        repeat
            determine \( C(r) \);
            if \( C(r) \subseteq \Omega \) and \( C(r) \) has contact points other than \( p \) then
                return \( C(r) \);
            end;
            \( n := n + 1 \);
            if \( C(r) \) meets \( \partial \Omega \) only at \( p \) then
                \( t := t + 2^{-n} \);
            else\( t := t - 2^{-n} \);
            end;
            \( r := t \times R \);
        until \( C(r) \) contains only one portion of \( \partial \Omega \) other than \( p \);
    end;
    return TWO_PRONG(\( C(r) \), \( \partial \Omega \));
end GIVEN_NORMAL
```

5.3. Longest Boundary Division

This procedure applies to any subdomain and is called \textsc{LongestBoundaryDivision}. The boundary of a subdomain consists of two kinds of curves: the first kind is the ones which are part of the boundary of the original domain \( \Omega \) we began with in the first place, and the other kind is the ones which are part of the circular arcs of the
inserted circles. LONGEST_BOUNDARY_DIVISION simply means that one chooses the longest curve of the first kind, takes the middle point in terms of the arc length, and then runs GIVEN_NORMAL procedure at that point.

5.4. Inscribed Osculating Circle

As shown in Theorem 3.1 in [3], if there is an inscribed osculating circle at \( p \in \partial \Omega \), then the curvature function must have a positive local maximum at \( p \). So if one can compute the local maximum points of the curvature function of \( \partial \Omega \), then one can find inscribed osculating circles through the following algorithm. Suppose one is given a portion \( \zeta \) of \( \partial \Omega \); then the algorithm is

\textbf{Step 1.} For an analytic piece of \( \zeta \), find positive local maximum points of the curvature function.

\textbf{Step 2.} For each local maximum point \( p \), draw a circle \( C \) of radius \( 1/k(p) \) with the center on the positive half line from \( p \) in the direction of the inward normal vector.

\textbf{Step 3.} If \( C \) is contained in \( \Omega \), then \( C \) is an inscribed osculating circle. Otherwise, skip this point and do this with another local maximum point.

\textbf{Step 4.} Repeat from Step 1 for each analytic piece of \( \zeta \).

In fact, it is not always numerically stable to find the local maximum points of the curvature function, as the curvature function requires the second derivative of the curve. In this case, one may skip this procedure and apply the terminal virtual node resolution algorithm which is described in Section 5.5.

5.5. Resolution of Terminal Virtual Node

If a subdomain \( \Omega \) has a terminal virtual node, then by Lemma 6.1 and remarks thereafter in [3], it is guaranteed to have an inscribed osculating circle. Applying the proof of Lemma 6.1 in [3], one can find an approximate osculating circle as follows. Let \( \zeta \) be the portion of \( \partial \Omega \) which, together with a portion of circular arcs of inserted circles, bounds a subdomain containing the terminal virtual node. Note we may assume \( \zeta \) has no corners and \( \Omega \) has no homologies due to the initialization procedure.

\textbf{Step 1.} Parameterize the boundary \( \zeta \) by the parameter \( t \) in some closed interval.

\textbf{Step 2.} Take the middle parameter value and fix the boundary point \( p \) corresponding to this parameter value.

\textbf{Step 3.} Compute the inward normal vector \( N \) at \( p \).

\textbf{Step 4.} Run procedure GIVEN_NORMAL with this normal vector \( N \), and get a contact circle which decomposes the given subdomain into smaller subdomains. Choose one of them which has a terminal virtual node.

\textbf{Step 5.} If the new subdomain with terminal virtual node has a sufficiently small portion of the original boundary, then take the circle contained in this domain as the inscribed osculating circle. Otherwise, go to Step 1 with this new subdomain.

After applying this procedure, the length of the parameter interval of the boundary of the new domain having a terminal virtual node will be less than the half of the original parametric length. So the circles in this process converge to an inscribed osculating circle. For proof of this fact, see Lemma 6.1 in [3].

5.6. Resolution of Virtual Node of Degree 3 (3-Prong Contact Circle)

Suppose three contact circles bound a subdomain \( \Omega \). Then the mathematical result in [3] guarantees the existence of a branch point. Now let the portion of \( \partial \Omega \) which is not the circular arc parts of the three bounding circles be separated into three pieces \( \alpha \), \( \beta \), and \( \gamma \) as in Fig. 22. Note that we can assume \( \alpha \), \( \beta \), and \( \gamma \) have no corners, since all the corners are processed in the initialization procedure.

In this situation, we allow all kinds of degeneracy. The bounding circle(s) may be one-prong inscribed osculating circle(s), or some of \( \alpha \), \( \beta \), and \( \gamma \) may be degenerated into points as in Fig. 23. In any case, the algorithm below works well even for all these situations. Note that, even if there may be many bifurcation circles in \( \Omega \), there exists a unique bifurcation circle which makes contact with all of \( \alpha \), \( \beta \), and \( \gamma \).

First, for any point \( x \in \Omega \), let \( a(x) \), \( b(x) \), and \( c(x) \), denote...
the distance from $x$ to each boundary pieces $\alpha$, $\beta$, and $\gamma$, respectively; that is,

\[
\begin{align*}
    a(x) &= d(x, \alpha) = d(x, y), \\
    b(x) &= d(x, \beta) = d(x, z), \\
    c(x) &= d(x, \gamma) = d(x, w),
\end{align*}
\]

for some points $y \in \alpha$, $z \in \beta$, and $w \in \gamma$.

Let us define $V(x)$ by

\[
V(x) = (a(x) - b(x))^2 + (b(x) - c(x))^2 + (c(x) - a(x))^2.
\]

So for all $x \in \Omega$, $V(x) \geq 0$ and $V(x_0) = 0$ at the 3-prong point $x_0$. We will use the steepest decent method with $V(x)$ as the potential function.

Now let us consider $\nabla V(x)$. $-\nabla V(x)$ points in the direction in which $V(x)$ decreases the most rapidly. In fact the gradient of $V(x)$ can be computed as follows:

\[
\nabla V(x) = 2(a(x) - b(x))\nabla a(x) - \nabla b(x)) + 2(b(x) - c(x))\nabla b(x) - \nabla c(x)) + 2(c(x) - a(x))\nabla c(x) - \nabla a(x)) = (4a(x) - 2b(x) - 2c(x))\nabla a(x) + (4b(x) - 2a(x) - 2c(x))\nabla b(x) + (4c(x) - 2a(x) - 2b(x))\nabla c(x).
\]

Here, $\nabla a(x)$ means the direction in which $d(x, \gamma)$ increases the most rapidly, and $-\nabla a(x)$ can be computed very easily as $\bar{a}x/|\bar{a}x|$. Also, $-\nabla b(x)$ and $-\nabla c(x)$ can be computed similarly. Therefore, we can compute $-\nabla V(x)$ by the following formula:

\[
-\frac{1}{2} \nabla V(x) = \frac{2(a(x) - b(x) - c(x))}{\bar{a}x}\frac{\bar{x\bar{y}}}{|\bar{x\bar{y}}|} + \frac{2(b(x) - c(x) - a(x))}{\bar{b}x}\frac{\bar{x\bar{z}}}{|\bar{x\bar{z}}|} + \frac{2(c(x) - a(x) - b(x))}{\bar{c}x}\frac{\bar{x\bar{w}}}{|\bar{x\bar{w}}|}.
\]

Let us call the first fixed $x$ by $x_1$ and corresponding points $y$, $z$, and $w$ by $y_1$, $z_1$, and $w_1$. And if we have the points $x_n$, $y_n$, $x_n$, and $w_n$, we will choose the next point $x_{n+1}$ as the following.

\[
x_{n+1} = x_n - \eta \nabla V(x_n).
\]

Here, $\eta$ should be chosen so that $V(x_n - \eta \nabla V(x_n)) < V(x_n)$. Since $V(x)$ decreases infinitesimally along $-\nabla V(x)$ direction, the above condition satisfies for sufficiently small $\eta > 0$. If it is easy to find, $\eta$ can be chosen so that it is the smallest parameter value $\eta_0 > 0$ such that $V(x_n - \eta \nabla V(x_n))$ takes a local minimum at $t = \eta_0$.

Now by applying this process recursively, the sequence $\{x_n\}$ converges to a point $x_0 \in \Omega$ at which $V$ has a local maximum. If $V$ is differentiable at $x_0 \in \Omega$, $x_0$ is the 3-prong point of $\text{MA} (\Omega)$ by the following lemma.

**Lemma 5.1.** Suppose $V$ is differentiable at $x_0 \in \Omega$. If $\nabla V(x_0) = 0$, then $V(x_0) = 0$. Therefore $x_0$ is the 3-prong point of $\text{MA} (\Omega)$.

**Proof.** Let us assume $\nabla V(x_0) = 0$. By letting $A(x) = 2a(x) - b(x) - c(x)$, $B(x) = 2b(x) - c(x) - a(x)$, and $C(x) = 2c(x) - a(x) - b(x)$, we get the following equations from Eq. (1).

\[
A(x_0)\nabla a(x_0) + B(x_0)\nabla b(x_0) + C(x_0)\nabla c(x_0) = 0.
\]

And for all points $x \in \Omega$,

\[
A(x) + B(x) + C(x) = 0.
\]

So,

\[
A(x_0)(\nabla a(x_0) - \nabla c(x_0)) + B(x_0)(\nabla b(x_0) - \nabla c(x_0)) = 0.
\]

Here, $\nabla a(x_0)$, $\nabla b(x_0)$, and $\nabla c(x_0)$ are unit vectors and are different from one another. So $\nabla a(x_0) - \nabla c(x_0)$ and $\nabla b(x_0) - \nabla c(x_0)$ are linearly independent. Therefore, $A(x_0)$ and $B(x_0)$ are both zeros, and hence $C(x_0)$ is also zero. From this fact, we can easily deduce that $a(x_0) = b(x_0) = c(x_0)$. Thus $x_0$ is the 3-prong point.

Now we summarize this algorithm as a pseudo code.

**procedure** THREE_PRONG $(x, r, \partial \Omega)$

1. determine three boundary portions $\alpha$, $\beta$, and $\gamma$;
2. choose the closest points $y$, $z$, $w$ on $\alpha$, $\beta$, $\gamma$;
3. compute $a(x)$, $b(x)$, $c(x)$;
4. while $\max[|a(x) - b(x)|, |b(x) - c(x)|, |c(x) - a(x)|] > \text{tolerance}$ do
   begin
      compute $\nabla V(x)$;
      $\eta := 1$;
      while $V(x - \eta \nabla V(x)) \geq V(x)$ do
begin
  \( \eta := \eta/2; \) \\
  end;
  \( x := x - \eta \nabla V(x); \) \\
  compute \( a(x), b(x), c(x) \) again;
end;
return \( x; \)
end THREE_PRONG

Remark 5.1. Our THREE_PRONG procedure works under the assumption that \( V \) is differentiable at each \( x_n \), and \( x_n \) stays inside \( \Omega \) at each step. But either of these may fail. For example, \( V \) may not be differentiable at some point \( x_n \), or some \( x_n \) may go outside \( \Omega \). This may happen when \( -\nabla V(x_n) \) points outward at a point \( x_n \) near \( \partial \Omega \). We do not yet know a good condition which guarantees the convergence of our procedure. However, in practice, we found our procedure works fairly well for a wide class of domains. In fact, in computer implementation of our algorithm, it almost works unless the domain is too wild. Even in a fairly complicated domain, our procedure works well.

However, we must admit that our procedure occasionally fails, even though it is rare in practice. In fact, one could perhaps cook up some weird examples for which our procedure fails. But, in practice, since we decompose the domain to simpler cases, this problem can be avoided. Anyway, we do not yet know a fail-safe criterion for the convergence of our algorithm. We hope we will be able to address this important issue in our future work.

In most cases, the chance for the convergence can be enhanced by a proper choice of the initial point. Since \( V \) decreases along the trajectory, it should converge if we start with a point \( x \in \Omega \) such that \( V(x) < L \), where \( L \) is the minimum of \( V \) on \( \partial \Omega \), as long as \( V \) is differentiable along the trajectory. Or one can just compute the minimum distances between boundary curves \( \alpha, \beta, \) and \( \gamma \). At a boundary point \( x_0 \) on \( \alpha, a(x_0) = 0 \). Thus \( V(x_0) \) is given by

\[
V(x_0) = b(x_0)^2 + c(x_0)^2 + (b(x_0) - c(x_0))^2
\]

\[
\geq b(x_0)^2 + c(x_0)^2
\]

\[
\geq d(\alpha, \beta)^2 + d(\alpha, \gamma)^2
\]

So \( L' = \min\{d(\alpha, \beta)^2 + d(\alpha, \gamma)^2, d(\beta, \alpha)^2 + d(\beta, \gamma)^2, d(\gamma, \alpha)^2 + d(\gamma, \beta)^2\} \) will be a rough lower bound of \( V(x) \) on \( \partial \Omega \). Thus it is enough to choose an initial point \( x_0 \) such that \( V(x_0) < L' \). However, when \( V \) is nondifferentiable at some point, we may not go further.

In any case, our THREE_PRONG procedure may fail for a variety of reasons. But THREE_PRONG is an important part of our main algorithm, and our main algorithm handles every case when it fails. For details see Section 6.

5.7. General Virtual Node

Resolving a virtual node which has degree 3 or higher may not be done in one step, as it must deal with too numerous possibilities. Because a circle in a plane is determined by three points, and THREE_PRONG works fairly well in practice, it is better to try first to find a 3-prong bifurcation circle. But, then, it may or may not succeed. If it does not succeed, the best one can hope for is to try to divide the given subdomain into smaller ones. The whole procedure is summarized below.

One may wonder why the procedure with various indentations is useful. But it is indeed a crucial step in the bigger scheme of things. This point is well discussed at the end of Section 6.3. As this procedure is used only in connection with MAIN_ALGORITHM, the push operation in the pseudo code below means that in MAIN_ALGORITHM in Section 6.3.

```plaintext
procedure GENERAL_VIRTUAL_NODE(boundary curves \( \gamma[ ] \))
  n := number of the boundary curves;
  reorder \( \gamma[ ] \) so that three consecutive ones are adjacent;
  \( \gamma[n+1] := \gamma[0]; \)
  \( \gamma[n+2] := \gamma[1]; \)
  for \( i := 1 \) to \( n \) do
    begin
      run THREE_PRONG with \( \gamma[i], \gamma[i+1] \) and \( \gamma[i+2] \)
      ignoring the rest of the boundary curves;
      if it converges and the circle is inside \( \Omega \) then
```
6. MAIN ALGORITHM

6.1. Basic Idea of Main Algorithm

We now combine the procedures developed so far into a single coherent procedure. The basic strategy and philosophy behind our main algorithm are as follows:

- Killing homology effectively makes the domain simply connected for the purpose of the algorithm.
- Divide the original domain into smaller and simpler domains and organize the data in such a way that the data structure makes transparent the existence of necessary bifurcation circles, inscribed osculating circles, etc., each of which requires a special algorithm.
- Run the recovery procedures for the medial axis transform and the boundary for each subdomain. If the error is within the tolerance, the subdomain is left alone. Otherwise, decompose further the subdomain where the error is the greatest. This makes the algorithm more efficient.

6.2. Recovery of the Boundary and the Medial Axis Transform

There are two approaches to the recovery of the boundary and the medial axis transform: One is to treat separately each (fundamental) subdomain corresponding to an edge which connects two nonvirtual nodes in the tree data structure, and the other is to treat as a whole a chain of such subdomains. The second approach is more suited to spline type procedures, whereas the first one is more naturally suited for the Bézier–Bernstein type curve representation. Among such Bézier–Bernstein curves, the quadratic one is the simplest and naturally adapted to this situation.

Let us first explain the first approach with the quadratic Bézier–Bernstein curve. Since we have the information in our data structure such as the position of the contact points and the tangent directions at these contact points, we can approximate the boundary of each fundamental subdomain by a quadratic Bézier–Bernstein curve. We then calculate the error of this approximation: One can measure the $L^2$-norm between the curves for a suitable common parameterization. Now, if the error for the subdomain in question exceeds the chosen error bound, find a new contact circle where the error is largest, and insert this in the data structure. Then we have new subdomains in the given subdomain, and one can reapproximate the boundary of these subdomains with quadratic Bézier–Bernstein curves. Continuing this way, we can approximate the boundary of the original subdomain within the desired accuracy. Then the boundary of the whole domain can be approximated by combining the above procedures for each subdomain through a suitable stack operation as in MAIN_ALGORITHM. Or we can decide to use the higher degree Bézier–Bernstein curve. In that case, one has to locate various off points to gain best approximation to the boundary with the given tangent information at the contact points, but the idea is the same.

A similar procedure is used to approximately determine the medial axis transform. But since the medial axis transform is not known a priori, one has to resort to a slightly different procedure to check the accuracy. First, the information at the two contact points gives the tangent direction of the medial axis transform (see Remark 6.4 in [3]), which is enough to determine the quadratic Bézier–Bernstein representation of the medial axis transform. Or one may find several more contact circles which can aid the determination of higher degree Bézier–Bernstein curve. Suppose now that an approximate medial axis transform $(p(t), r(t))$ for $0 \leq t \leq 1$ for a given subdomain is found. We need to check its accuracy. So take prechosen parameter values, $0 < t_1 < t_2 < \cdots < t_N < 1$, and test how closely the circles $\partial B_{t_i}(p(t_i))$ for $i = 1, \ldots, N$ make contact with the boundary. If the error is within tolerance, then this approximation is good enough. Otherwise, decompose the domain where the error is the greatest, and repeat the same procedure.

It is perhaps worthwhile to mention the boundary recovery in relation to the medial axis transform. As described in [3], the boundary curves are given as the envelopes of the circles along the medial axis transform. And it is truer to the word “recovery” to use this envelope procedure. But in practice, it does not improve error more than our procedure described above does. Since the envelope proce-
procedure has to be approximated again to be in the curve representation format, this does not seem to justify the added effort.

The second approach which uses the spline type procedure has essentially the same philosophy. The difference compared with the first approach is that one now uses a chain of subdomains as a single entity to be approximated and then uses appropriate spline. This approach has one advantage in that it is possible to get better approximation for the subdomain between two nodes, if neither one is a generic 2-prong but the nodes in between them are all generic 2-prongs. The approximate determination of the medial axis transform, and further decomposition of the subdomain, is similar.

6.3. Summary of Main Algorithm

In this section, we summarize all of the above to devise a single coherent procedure, and for the clarity of presentation we choose to use the stack oriented algorithm which is best suited for recursion. In MAIN_ALGORITHM, we use the term “subdomains.” In reality, it means that we are given a contact point of a contact circle from which a subdomain can be found by the boundary traversing with the aid of the data in the tree data structure.

Second, when we run THREE_PRONG procedure, the choice of the initial point may be important. We are assuming that we are making the best guess of the initial point. Namely, when the boundary portions of the subdomain in $\partial \Omega$ are small, then one can form a triangle by joining the contact points with line segments, and the initial point is chosen to be the in-center of the triangle. When the subdomain is fairly large, one chooses the initial point at random as long as it is not too close to the boundary.

Finally, it is worthwhile mentioning that INSERT_CIRCLE procedure updates the tree data structure.

Here we summarize our main algorithm as a pseudo code.

```
begin MAIN_ALGORITHM
run KILL_HOMOLOGY;
run procedure for corners;
run procedure for circular arcs;
(run procedure to find inscribed osculating circle;)
run INSERT_CIRCLE successively with all circles
found in the above procedures;
choose any point $p \in \partial \Omega$;
run GIVEN_NORMAL at $p$;
run INSERT_CIRCLE;
(* This last step makes sure the tree data structure
is not empty. *)
push all subdomains obtained by the above procedures;
while the stack is not empty do
begin
pop a subdomain $\Omega$;
if $\Omega$ has a terminal virtual node then
run INSERT_CIRCLE;
push subdomain;
else if $\Omega$ has a virtual node of degree 3 then
run THREE_PRONG;
if succeed then
begin
run INSERT_CIRCLE;
push subdomains;
end;
else
begin
run LONGEST_BOUNDARY_DIVISION;
run INSERT_CIRCLE;
push subdomains;
end;
else if $\Omega$ has a virtual node of degree $n$ ($n \geq 4$) then
```

479
run GENERAL_VIRTUAL_NODE;
(*) This procedure has INSERT_CIRCLE
and LONGEST_BOUNDARY_DIVISION. *)
else (Ω has no virtual node. *)
begin
run boundary recovery procedure;
run MAT recovery procedure;
if the boundary error is greater
than tolerance then
begin
choose a point where the error is maximum;
run GIVEN_NORMAL at that point;
run INSERT_CIRCLE;
push subdomains;
end;
else if the MAT error is greater
than tolerance then
begin
choose the circle on MAT
which has the biggest error;
choose the point on the boundary closest
to the center of the above circle;
run GIVEN_NORMAL at that point;
run INSERT_CIRCLE;
push subdomains;
end;
end;
recover the boundary from the tree data structure;
construct the medial axis (transform)
from the tree data structure;
end;

Remark 6.1. The above pseudo code is written with
the assumption that one can run the domain recovery pro-
cedures for each fundamental subdomain as described in
Section 6.2. However, in some implementation, one may
choose to store an array of such fundamental domains in
a separate place and run a domain recovery procedure by
using things like splines. In that case, one can modify the
above pseudo code accordingly.

It is important to note how our MAIN_ALGORITHM ter-
ninates in finite steps. First, given a subdomain with a
virtual node, we first try to find a 3-prong branch circle
inside. This may or may not succeed. If it does not succeed,
then LONGEST_BOUNDARY_DIVISION procedure halves
the longest boundary curve, and it introduces a new contact
circle. It may even make the subdomain more complicated,
i.e., the virtual degree may increase, if this new contact
circle is a 1-prong inscribed osculating circle. Or, the virtual
degree increases at the same time a new terminal virtual
node is also introduced, if it makes contact as in Fig. 13.

But this kind of step that makes things more complicated
by introducing more virtual nodes stops doing so after a
finite number of steps, since our mathematical result in [3]
guarantees that there are finitely many bifurcation circles
or the inscribed osculating circles. More commonly,
LONGEST_BOUNDARY_DIVISION procedure typically
finds a contact circle that has at least two contact points, in
which case the subdomain is decomposed into two smaller
subdomains each of which has virtual degree not greater
than the parent subdomain. Typically, the virtual degrees
decrease as in Fig. 15, unless the subdomain itself has only
one bifurcation circle. Doing these in finite steps eventually
resolves all virtual nodes, and all subdomains become fun-
damental. (There may be a possibility that there are insig-
ificant branch or 1-prong points such that the boundary
recovered by ignoring these may still be within the error
tolerance. In that case, there is no harm in ignoring these
insignificant points. This flexibility also makes our algo-
ithm more robust.) The fundamental subdomains are de-
composed in finite steps to sufficiently simpler ones for
which the recovered boundary and MAT is within the error
tolerance, which makes our MAIN_ALGORITHM terminate
in a finite number of steps.
7. WORKED OUT EXAMPLE

In this section we illustrate a worked out example along the line of our main algorithm. Our domain has a hole, a dull corner, two sharp corners, and several branch circles and inscribed osculating circles as shown in Fig. 24.

For this example, the boundary curves are represented by the second order Bézier–Bernstein curves. It has some advantages that the recovered boundary curve is uniquely determined by its data at the contact circles. This certainly makes our implementation easier, but we like to emphasize again that one can use splines or other forms of curves and modify our algorithm without altering our philosophy.

Below, we illustrate our algorithm with the corresponding outputs. We begin with performing the initialization procedures which consist of first, killing homology; second, handling the circular arcs; third, processing the corners; fourth, handling the inscribed osculating circles. For better illustration, we selected by hand the initial points of the THREE_PRONG algorithm in Fig. 29 and the starting circles of the TWO_PRONG algorithm in Figs. 31 and 33.

Figure 25 shows the result of homology killing.

Since there are no circular arcs, the second step of the initialization procedure needs not be performed. The third step is to process corners: the result is illustrated in Fig. 26.

Then the final initialization step is to handle the inscribed osculating circles. Since we use second order Bézier–Bernstein curves, we can compute all local maximum points of the curvature function by exactly solving the linear equation derived from setting \( \kappa'(t) = 0 \). However, in some form of representation of the curves, this may not be numerically feasible. In this case, we may skip this process altogether. And proceed to the next step, or add more 2-prong circles. Then the inscribed osculating circles can be pinpointed as the terminal virtual nodes in our tree data structure. In that case, one may use the resolution of terminal virtual node procedure in Section 5.5. In our example, we directly found the inscribed osculating circles by computing the positive local maximum of the curvature function. The result is in Fig. 27.

In Fig. 27, we labeled the circles in the order in which they are entered. Figure 28 shows the tree data structure at this stage.

In Fig. 28, there are two virtual nodes: one of degree 6, the other of degree 3. We now resolve these two virtual nodes into five three-prong points. Note that the degree 6 virtual node generates four three-prong points.

There is one point to merit a mention. In resolving the virtual node, the convergence of the procedure THREE_PRONG is very robust. The relevant fact is also discussed in Remark 5.1. The result of the virtual node resolution is given in Fig. 29.

Figure 30 shows the recovered domain at this stage which is overlapped with the original domain. The domain recovery is done in a very simple way. Namely, as each circle
FIG. 28. Data structure at the stage of Fig. 27.

FIG. 29. Branch circles are found.

FIG. 30. Recovered domain at this stage overlapped with the original domain.

FIG. 31. Adding one more two-prong circle.

FIG. 32. Recovered domain at this stage: Adding one two-prong circle dramatically reduces the error in the domain recovery.

FIG. 33. Adding more two-prong circles to improve accuracy.

FIG. 34. Recovered domain.

FIG. 35. Medial axis with the original domain.
found has the information on the position of the contact point, the recovered curve between two contact points is uniquely determined as a second order Bézier–Bernstein curve by the tangent vector information. We found this technique satisfactory for our purpose. But, of course, one may choose any curve representation and use an approximation procedure such as spline. Notice the large error in some part of the inner boundary curve in Fig. 30.

For the purpose of illustrating the power of our method, we add just one more 2-prong circle in Fig. 31. One should notice the dramatic improvement in Fig. 32.

To improve the accuracy, six more circles are added as in Fig. 33. Then the recovered domain is very close to the original one as in Fig. 34. And the medial axis is depicted in Fig. 35.

8. CONCLUSION

The above worked out example in Section 7 amply illustrates the power of our algorithm. Finding two sharp corners and 18 circles in the strategically chosen locations is enough to find the medial axis and to recover the domain from it with very good accuracy.

This algorithm is currently being used time and time again in the actual industrial situation of font design by Hyundai Media Systems Co., Ltd., and we found its performance excellent. The strength behind our algorithm is the method of reduction to simpler problems of more complicated situations via domain decomposition method and the appropriate data structure and operation best suited for it.

ACKNOWLEDGMENTS

This research, especially the commercial end of it, is being funded by Hyundai Media Systems Co., Ltd. We gratefully acknowledge the generous support of its President, Dr. Jin-Kee Lee, and Directors Joon W. Kim and Tae-Won Kang. We express our sincere thanks to members of the technical staff and font team: Hyun Joo Choi, Myung Joo Baek, Hallan Yang, Shin Haé Tahk, Jin Young Kim, and Jeong Han Kim. We also thank Dr. Sungjin Lee and Kyung Hwan Park for their interest in our work. Special thanks are due to Sung Won Song, and Byungnam Kwon who assisted us in programming our algorithm. Finally, but not least, we thank Sun Gi Hong, Director of R&D of Microsoft Korea, for having introduced us to font technology and for his valuable advice and help on the computer industry in general over the years.

REFERENCES


