A Planar Subdivision Method Beyond Straight Lines

Saeed Gholami Shahbandi

September 13, 2016

1 Introduction

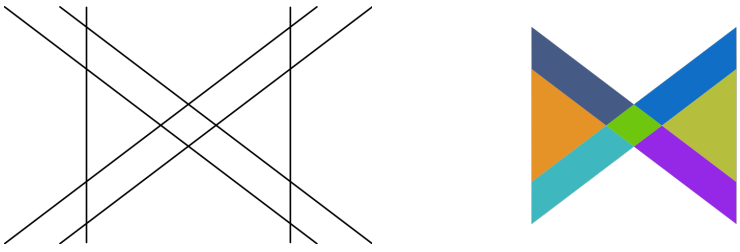
Subdivision is the process of partitioning a space $\mathcal{S}$, according to a provided set of level-curves $\mathcal{C}$ in that space, into a set of faces $\mathcal{F}$.

$$subdivision(\mathcal{C}) : \mathcal{S} \xrightarrow{decompose} \mathcal{F}(\mathcal{C})$$

Figure 1 shows an example of a planar subdivision (where $\mathcal{S}$ is two dimensional), according to a set of curves containing six straight lines. The process of subdivision provides an abstraction of the space. Such an abstraction potentially facilitates some geometric processes required on that space.

1.1 Problem Statement

Here we define some of the terminologies, and briefly explain some of the challenges and requirements.



(a) curves (b) partitionaing

Figure 1: A planar subdivision, over a set of six straight lines. Figure 1a shows the set of curves, and the "faces" resulting from partitioning are color coded in Figure 1b.

(a) curves (b) separate faces (c) unified face

Figure 2: Two approaches for dealing with the unbounded exterior region. In this work we take the approach in Figure 2c, that is to say, treating the whole exterior region as one unbounded face.

Curves A curve set $\mathcal{C}$ contains the level-curves of some multivariate functions $f(X)$ defined over a space $\mathcal{S}$.

$$\mathcal{C} = \{c_i \mid c_i : f_i(X) = 0, i \in I, I\text{: index set of } \mathcal{C}\}$$

Faces A face is the "interior" region of a "Jodan Curve". The Jordan curve in our problem setting could be a compilation of curves. The set of faces $\mathcal{F}$ satisfies following conditions:

$$\nexists face_i, face_j \in \mathcal{F}, \quad face_i \cap face_j \neq \emptyset \quad (\cap\text{: geometric intersection})$$

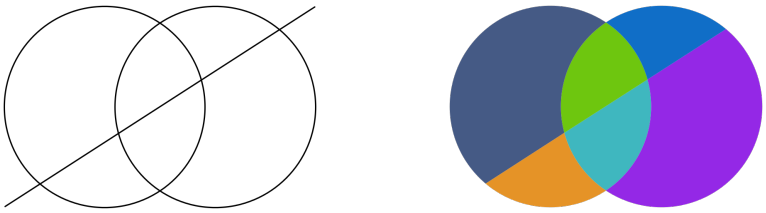
$$\bigcup_{i \in \mathcal{I}} face_i \subset \mathcal{S} \quad (I\text{: index set of } \mathcal{F}\text{, and } \cup\text{: geometric union})$$

Membership and neighborhood of faces Two important method that the face data structure must support are the *membership* and *neighborhood* functions. Membership is a function of points which returns the face that encompass a given point. That is to say, membership function identifies the "interior" region of the faces. Neighborhood is a function of faces and returns all other faces that are neighboring the input face by the mean of [at least] one shared edge in their boundaries.

$$member\,(point) = face_i \quad , face_i \text{ encompasses the } point$$
$$neighbor\,(face_i) = \{face_j \mid \exists edge_k, edge_k \in face_i \wedge edge_k \in face_j\}$$

Unbounded faces The fact that $\bigcup face_i$ is a proper subset of and not equal to $\mathcal{S}$, raise a question about the exterior region which does not belong to any of the normal faces. The exterior region could be treated either as separate unbounded faces (as in Figure 2b), or a unified single unbounded face (as in Figure 2c). Following the latter approach, we treat the union of all the exterior region as a single unbounded face.

Node



(a) curves (b) partitionaing

Figure 3: A planar subdivision, over a set of curves containing a straight line and two circles. Figure 3a shows the set of curves, and the "faces" resulting from partitioning are color coded in Figure 3b.

Half-edge

1.2 Background

I still can't believe this problem has not been solved! :(

1.3 This work

This work aims at studying the subdivision problem in the presence of circles. After identifying the limitation of the existing solutions in handling the more generic cases, challanges are identified and addressed. Figure 3 illustrates an example of a planar subdivision in the presence of circles. This work relies on the following assumptions:

Dimension the space is a two dimensional plane.

Curves the set $\mathcal{C}$ contains levels curves resulted from either linear functions (straight lines as an example of an unbounded class) or conic sections (circles as an example of a bounded class).

Redundancy if two curves were identical, their intersection would be the same curve which is beyond a finite set of points. In order to prevents the intersection procedure yielding a result other than a finite set of points, redundant curves are rejected so that;

$$\nexists c_i, c_j \in \mathcal{C}, \quad c_i = c_j$$

1.3.1 Challenges

Introducing circles raises challenges from the meta-algorithm of subdivision, all the way to sub-procedures within. Most of these challenges, as we'll see, are

based on three essential differences between lines and circles; *i*) *boundness*: a circle could be enclosed within a closed curve (i.e. a Jordan Curve), while a line stretches to infinity; *ii*) *closed curve*: unlike a line, a circle itself is a closed curve and alone could result in bounded faces; and *iii*) *segment representation*: while a segment of a line could be represented by a vector, a segment of a circle is represented by an arc.

None intersecting circles If a line does not intersect with any other curves in the curve set $\mathcal{C}$, it could be safely discarded as it would not result in any bounded region. On the other hand, a circle which does not have any intersection point with other curves would result in a bounded region that should be counted and identified as a face. That is to say, in the absence of circles it is safely assumed that the intersection points are well representing all the edges, and consequently all the faces. We tackle this problem by introducing dummy intersection points over non-intersecting circles.

Sorting nodes over curves In order to identify the edges resulting from a curve, it is essential to be able to geometrically sort the intersection points over the curve. For lines it suffices to sort the points according to their x (or y) values. This does not hold for circles. To this end, we employ a univariate represetation for curves (presented in Appendix A).

Edges are no longer vectors A key information in face identification is the ability to follow a geometric path of edges that bound a face. This is an iterative process of finding the correct successor to an edge, until the sequence of edges form a closed curve, which represents the face. The correct successor is identified by its departing angle with respect to the arriving angle of the last edge. In the case of only lines, the arriving and departing angles of edges are equivalent to the slope of the line. But in the case of arc shaped edges, we have to employ the derivatives of the curve and the intersection point to find the correct successor.

Holes A new feature that appears in the presence of circles is the holes. That is to say, a face could have a non-continuous sequence of edges that define its boundary. This is due to the face that a circle could be enclosed in the closed curve of a face, and creating a "hole" in it. On the subdivision level, it causes the subdivision to become topologically disconnect (i.e. have separate connected components). We approach this problem by partitioning the space according to each connected components of the subdivsion separately, and merging their results at the end.

Membership function Membership function for the subdivision of lines could be implemented through a set of tests relying on cross products, as described in [1]. The tests are to check whether if a point is located on the same side of all the bounding edges of a face. This would only work if the edges

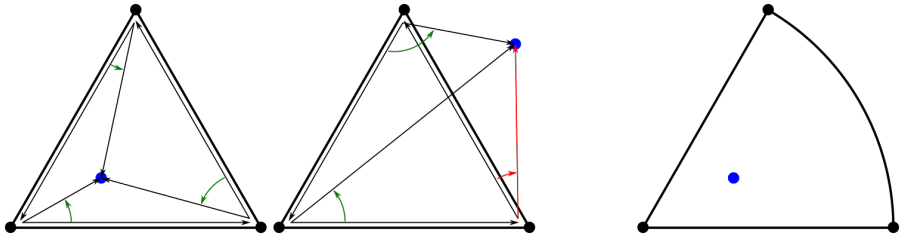


Figure 4: Membership function for the case of only straight lines could be based on tests relying on a cross product. However, as the edges are no longer vectors in the presence of circles, the conventional membership procedure requires revision.

could be represented by vectors, and if the face is convex. As can be seen in Figure 4, edges turn to arcs in the presence of circles, and potentially a face could be concave. As an alternative we suggest the use of "point-in-polygon" method, which is based upon the Jordan Curve theorem.

2 Subdivision

The process of our method for constructing the subdivision is explained in this section. The process of subdivision is the decomposition of the space into faces, according to a provided set of curves. That is to say, identifying every irreducible face. On the other hand, to satisfy the required functionality of the resulting subdivision(such as membership function and neighborhood function), proper descriptions of intermediate and underlying data structures are essential. Such descriptions will be provided in Appendix **??**.

Algorithm 1 presents a restricted version of the subdivision construction algorithm. While explaining the sub-procedures mentioned in Algorithm 1, the restrictions and their remedy will be described in this same chapter. A more comprehensive version of Algorithm 1 is given afterward.

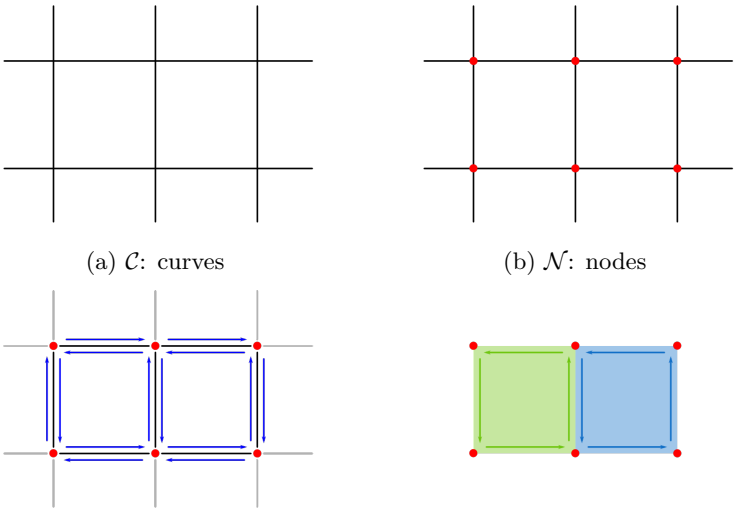
Algorithm 1 Subdivision (restricted version)

INPUT $\mathcal{C} : curves$
OUTPUT $\mathcal{F} : faces$

$\mathcal{N} : nodes = intersect\,(\mathcal{C})$
$\mathcal{E} : half\text{-}edges = segment\,(\mathcal{C}, \mathcal{N})$
$\mathcal{F} : faces = partition\,(\mathcal{C}, \mathcal{E})$

Algorithm 1, which is similar to the subdivision algorithm presented in [1], is demonstrated in Figure 5. It should be reiterated that the focus of this work is to revise the meta-algorithm and its sub-procedures, so that they can handle the more general cases including curves of circle class. Regardless, we first try to provide an intuitive and basic understanding of the restricted version, and in the following sections we will provide details on the computation and required extensions to each procedure. The objective of the restricted version of the subdivision algorithm (presented in Algorithm 1 and demonstrated in Figure 5) is to identify the irreducible components of the partitioned space (denoted by $\mathcal{S}$) as a set of faces (denoted by $\mathcal{F}$.) The first step is to find the set all the intersection points between all the curves (Figure 5b). This is the set $\mathcal{N}$ of *nodes*[1]. Every node in this set is also assigned to the curves that yielded it. For the next step, that is, segmenting curves into half-edges based on their nodes, it is important that the nodes' assignments to the curves should be geometrically sorted over curves (as illustrated in Figure 6.) In the case of only straight lines, the sorting process is performed by looking at the x (or y) values of the nodes. A more comprehensive approach for sorting is presented in Section 2.2. After sorting the nodes over each curve, the curves are segmented into edges, as illustrated in Figure 5c. Note that each simple edge is composed of two "twin"

[1]The reason for the name is the important role that nodes play in constructing a graph in the extended version of the algorithm. Even though we'll come to this aspect later, we use the same term for consistency.



(a) $\mathcal{C}$: curves (b) $\mathcal{N}$: nodes

(c) $\mathcal{E}$: half-edges (d) $\mathcal{F}$: faces

Figure 5: A simplified demonstration of the subdivision algorithm.

directed half-edges, represented by blue vectors in Figure 5c. For more detail on half-edge data structure see [1], or Appendix **??**. In the final stage, the set of faces $\mathcal{F}$ is constructed, through a process of path following. An *open list* is generated, identical to the set $\mathcal{E}$ of half-edges. The process starts by fetching (and removing) a half-edge from an open list, moving it to a *boundary list* representing the boundary of a face. The next member of the boundary list of the face, is a successor half-edge that follows the last half-edge in the boundary list. The potential candidates for the successor are those half-edges that start from the node to which the last half-edge of the boundary list arrives. The correct successor is identified among the candidates by measuring their angles to the last half-edge. The one with widest Counter Clock-Wise (CCW) angle with respect to the last half-edge is identified as the new member of the face's boundary list. As we will see later, this conventional method for following the boundary of a face would fail in the presence of circles, for which we another appraoch based on the first and second derivatives of the curves.

2.1 Intersection procedure

This procedure, as presented in Algorithm 2, intersects all curves with each other. Each intersection point (node) will be assigned to the curves that contains it. Furthermore, different curves might intersect at the same location, resulting in duplication of the points. The procedure overcomes this problem by removing redundant points while updating the assignment of points to curves.

Algorithm 2 Intersect: curve intersection procedure

INPUT $\mathcal{C} : curves$
OUTPUT $\mathcal{N} : nodes$

$\mathcal{N} = \emptyset$
for all $curve_i, curve_j \in \mathcal{C} \mid i > j$ **do**
  $nodes = intersect\,(curve_i, curve_j)$
  for all $node_k \in nodes$ **do**
    append $node_k$ to $\mathcal{N}$
    set $node_k$ to $curve_i$ and $curve_j$
  end for
end for

for all $node_i, node_j \in \mathcal{N} \mid i \neq j, node_i = node_j$ **do**
  assign $node_i$ to all curves that $node_j$ is assigned to
  remove $node_j$ from $\mathcal{N}$
end for

Non-intersecting Circles The result of the intersection procedure is the ground for identifying half-edges (as detailed in the next section). Each curve is segmented into half-edges, according to the nodes that the curve contains. In the presence of circles, there is a chance that a circle might not intersect any other curve. Consequently, no half-edges would be identified on that circle. We tackle this problem by placing a *pseudo* intersection point on the circle's curve at an arbitrary location, resulting in a node that is only assigned to one curve (i.e. the non-intersecting circle.)

2.2 Half-Edge identification

Following the Algorithm 2, every intersection point (node) laying on every curve is provided. Algorithm 3 describes the process of segmenting each curve into half-edges that connect nodes to each other. Such information is essential for the next step of the Algorithm 1, i.e. face identification (and graph construction in the complete version of the algorithm.)

This step of the algorithm requires the ability to sort nodes over each curve. As mentioned earlier, in the case of only straight lines, the sorting process is performed by looking at the x (or y) values of the nodes. This would fail in the presence of circles, because a circle might contain two points for a given x value. In another word, the points on a circle are ordered in a polar coordinate, not Cartisian. To this end, we employ an alternative representation of the level-curves (denoted by $\dot{f}$) as a function of a single variable θ that traverses the level-curve of the multivariate function[2]. Given the alternative representation

[2]The equations are presented in Appendix A.

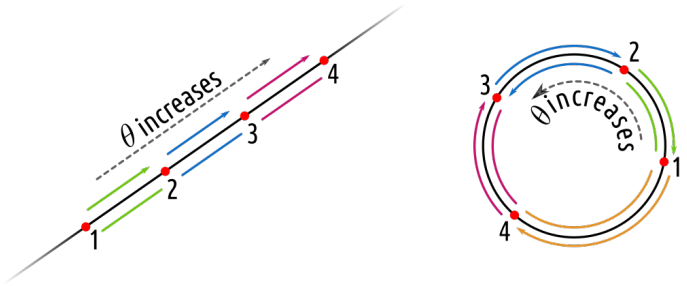


Figure 6: Examples of sorting nodes over a line and a circle. Numbers next to the nodes shows their order after sorting. Half-edge construction is demonstrated by colored arrows. In each case, half-edges with the same color are twins.

of a level-curve as a function of a single variable θ, we use its inverse (denoted by $\dot{f}^{-1}$) to sort nodes over the curve ($\theta = \dot{f}^{-1}(x, y)$). Provided that it is possible to sort nodes over the curves, we can proceed with the Algorithm 3 that describes the construction of half-edges. Figure 6 demonstrates two examples of half-edge construction over a line and a circle. Algorithm 3 takes segments of the curve under process and turns each segment into two twin half-edges. The segments are identified by the consecutive nodes over the curve.

Algorithm 3 Segment: half-edge construction

INPUT $\mathcal{C}$: $curves$, $\mathcal{N}$: $nodes$
OUTPUT $\mathcal{E}$: $half\text{-}edges$

$\mathcal{E} = \emptyset$
for all $curve_i \in \mathcal{C}$ **do**
  $list\_n = \{node_j \mid node_j \in \mathcal{N}, node_j \text{ set to } curve_i\}$
  sort the $list\_n$ according to $\theta_j = \dot{f}_i^{-1}(node_j)$
  $I =$ index set of the $list\_n$
  $list\_he = \{(curve_i, node_j, node_{j+1}), (curve_i, node_{j+1}, node_j) \mid \forall j \in I\}$
  if $curve_i$ is a circle **then**
    $j = max(I)$
    append $\{(curve_i, node_j, node_0), (curve_i, node_0, node_j)\}$ to $list\_he$
  end if
  append $list\_he$ to $\mathcal{E}$
end for

2.3 Partitioning procedure: face identification

Identification of faces is performed by a procedure of path following. Given that each half-edge belongs to only one face, the process starts by picking an arbitrary

half-edge and detecting the boundary of the face it belongs to, by picking the correct successor to the last half-edge. This iterative process terminates when the last half-edge arrives at the node which the first half-edge started from. This procedure is represented by Algorithm 4 and demonstrated in Figure 7.

Algorithm 4 Partition: face identification

INPUT $\mathcal{C} : curves, \quad \mathcal{N} : nodes, \quad \mathcal{E} : half\text{-}edges$
OUTPUT $\mathcal{F} : faces$

$\mathcal{F} = \emptyset$
$open\_list = \mathcal{E}$
while $open\_list \neq \emptyset$ **do**
  fetch a random $half\_edge_i$ and remove it from $open\_list$
  $face\_list = \{half\_edge_i\}$
  $start\_node = node_s$, where $half\_edge_i$ departs from $node_s$
  $end\_node = node_e$, where $half\_edge_i$ arrives at $node_e$
  while $start\_node \neq end\_node$ **do**
    $half\_edge_j = find\_successor$ (last $half\_edge$ in $face\_list$)
    remove $half\_edge_j$ from $open\_list$ and add it to $face\_list$
    update end_node according to $half\_edge_j$
  end while
  construct a face instance from $face\_list$ and append it to $\mathcal{F}$
end while

2.3.1 Finding the correct successor half-edge

Half-edges departing from the node, which the last half-edge of the face arrived at, are flagged as candidates. Among the candidates, the twin of the last half-edge is rejected. The winning candidate is the one with widest departing angle (CCW) with respect to the departing angle of the last half-edge's twin. That is to say, the half-edge that is closest to the twin's departure. Some examples of correct successor selections are demonstrated in Figure 8, where the arriving half-edges and the correct successors are colored green, and all the rejected candidates are colored red.

In cases of only straight lines (e.g. Figure 8a), finding the successor would be simply to look at the angles of the candidate half-edges, as half-edges are vectors. However this does not work in the presence of circles, where half-edges are arcs (see Figures 8b and 8c). To overcome this problem, we rely on the angle of the tangent vector to the level-curves[3]. These vectors provide the departing and arriving angles of half-edges at the nodes' location.

[3]The equations for the derivatives are available in Appendix A.

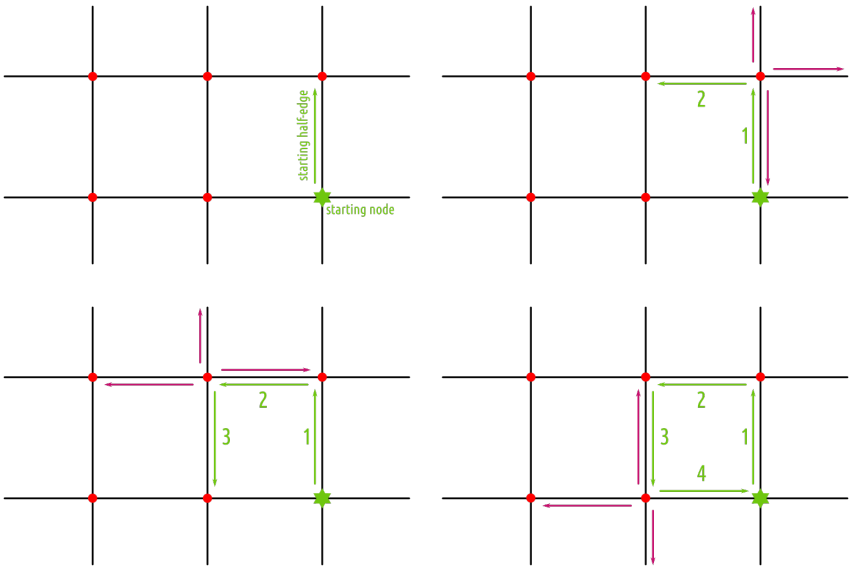
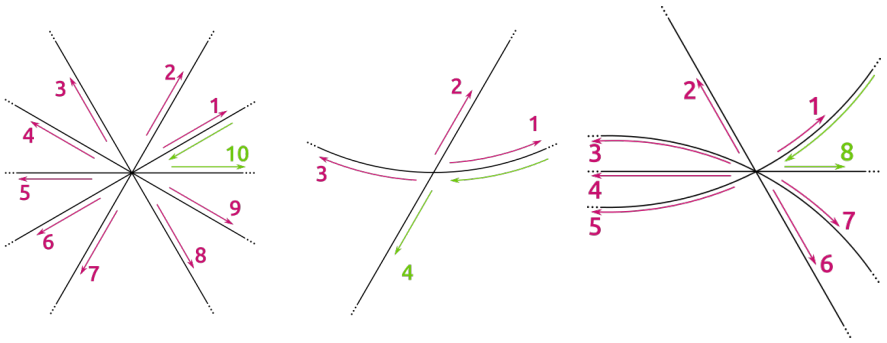
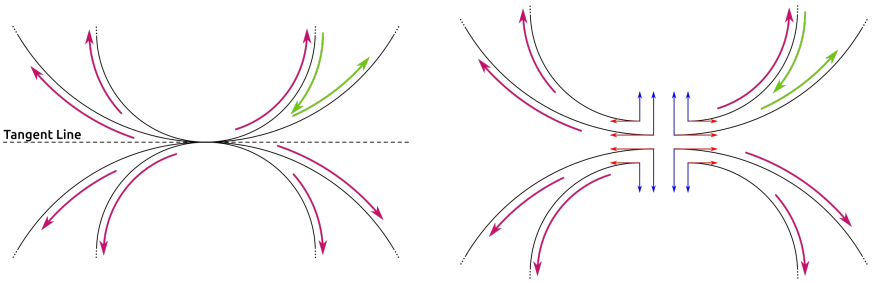


Figure 7: The process of face identification towards partitioning. First, a half-edge is picked, which initiates the identification of a new face. Then, the correct successor, colored green, is identified among candidates. Rejected candidates at each step are colored red. The process terminates and the face is completely identified when the last half-edge arrives at the starting node of the first half-edge.



(a) only straight lines (b) a line and circles (c) a line and circles

Figure 8: Few examples of correct successor selections. Arriving half-edges and the correct successors are colored green; all rejected candidates are colored red.



(a) a case of tangency (b) derivatives of *half-edges*

Figure 9: Curves that intersect, but do not intercept. When curves intersect while being tangent, the measure of first derivatives' angle at the intersection point is not sufficient to detect the correct successor. A second measure for sorting the candidates half-edges are provided based on both the first and second derivatives.

2.3.2 Insufficiency of the first derivative in case of tangency

When curves intersect while being tangent, the measure of the first derivative at the intersection point is not sufficient to detect the correct successor. This challenge is demonstrated in Figure 9a where four circles intersect at a tangent point. While the correct successor is colored green after the arriving half-edge, one can see that there are two other candidates (neglecting the twin of the arriving half-edge) that have the same departing angle at the intersection point as the correct successor.

We tackle this problem by adding a secondary sorting measure that relies on both first and second derivatives of the curve at the intersection point[4].

$$key_1 = \measuredangle \overrightarrow{\frac{d\dot{f}(\theta)}{d\theta}}$$

$$key_2 = \overrightarrow{\frac{d^2\dot{f}(\theta)}{d\theta^2}} \cdot \frac{\vec{V}}{\|\vec{V}\|} \quad \text{where: } \vec{V} = (-y, x) \quad , (x, y) = \overrightarrow{\frac{d\dot{f}(\theta)}{d\theta}}$$

If the key_1 is not sufficient to detect the correct successor, that is to say in the tangent case, the second derivative tells us in which direction, the curve is changing its *course*. Along with a measure of the first derivative, this change of "course" that is delivered by the second derivative in the key_2.

2.4 Resolving the restriction

So far we have presented modified versions of the sub-procedures in Algorithm 1, that can handle curves beyond straight lines (i.e. circles). However, this does not suffice, since Algorithm 1 itself would fail in the presence of circles. Figure 10

[4]The equations for the derivatives are available in Appendix A.

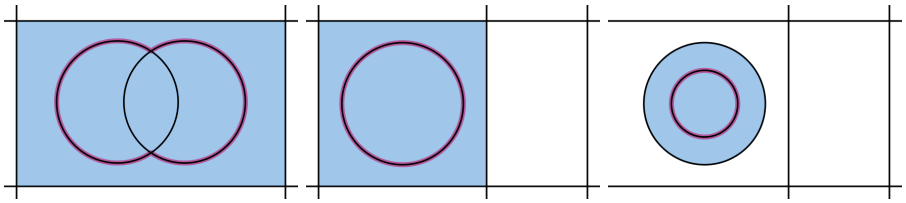
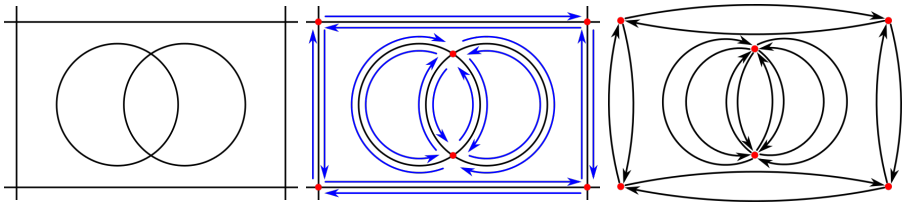


Figure 10: The restricted Algorithm 1 fails when some curves are enclosed inside other faces without intersecting with each other. Blue areas show the enclosing faces in each case, and the regions that should be substracted from those faces are marked red (what we call super-faces) .



(a) curves (b) nodes and half-edges (c) *MDG*

Figure 11: Construction of a multi-directional graph *MDG* of nodes and half-edges. It is used to identify connected components which result in disconnected partitioning.

show examples of where the "restricted" algorithm would fail. These failures occur when some curves are located inside other faces without intersecting with them. As a consequence, those insider curves are not accounted for in the encompassing faces as there is no connected path between them.

To understand the cause of this failure, let us construct a multi-directional graph (*MDG*) from all the nodes $\mathcal{N}$ and half-edges $\mathcal{E}$, as demonstrated in Figure 11. The restricted algorithm would work under the assumption that the *MDG* is connected. This assumption would not hold in the cases of Figure 10.

Sub-graphs (connected components of the *MDG*) being disconnected implies that they are topologically disjoint. However it does not imply that they are geometrically disjoint. If there are multiple disconnected sub-graphs, one can be certain that they are either: *i*) geometrically disjoint and non-overlapping; or *ii*) one sub-graph is completely contained in only *one* face of the other subgraph(s). Relying on this, we resolve this problem by partitioning each sub-graph of the *MDG* independently, and perform a check afterward to evaluate the relative position of the sub-graphs with respect to each other. This check relies on the membership operation on the subdivision's faces that is described in Section 2.5. It involves picking an arbitrary point (node) from each sub-graph and check whether it is entailed by the faces of any other sub-graph. If the test implies that two sub-graphs overlap, basically a hole in the shape of inside

sub-graph is punched in the face that entails it[5]. The sub-graphs' outer shape that are colored red in Figure 10 are called super-face, and will be described next. After punching holes, the last step to have a whole subdivision would be to append all the faces of each sub-graph to one final list. This whole process is summarized in Algorithm 5 as a complete version of the subdivision algorithm.

Algorithm 5 Subdivision (complete version)

INPUT $\mathcal{C} : curves$
OUTPUT $\mathcal{F} : faces$

$\mathcal{N} : nodes = intersect\,(\mathcal{C})$
$\mathcal{E} : half\text{-}edges = segment\,(\mathcal{C}, \mathcal{N})$
$MDG = multi\_directional\_graph\,(\mathcal{E}, \mathcal{N})$
$\mathcal{SG} : sub\_graphs = connected\_components(MDG)$

for all $sub\_graph_i \in \mathcal{SG}$ **do**
  $\hat{\mathcal{N}}_i = \{node_j \mid node_j \in sub\_graph_i\}$
  $\hat{\mathcal{E}}_i = \{edge_j \mid edge_j \in sub\_graph_i\}$
  $\hat{\mathcal{F}}_i = partition\left(\hat{\mathcal{N}}_i, \hat{\mathcal{E}}_i\right)$
end for

for all $sub\_graph_i, sub\_graph_j \in \mathcal{SG}$ **do**
  if $sub\_graph_i \in face_k \mid face_k$ belongs sub_graph_j **then**
    $face_k \backslash super\text{-}face\,(sub\_graph_i)$
  end if
end for

$\mathcal{F} = \bigcup_{i \in I} \hat{\mathcal{F}}_i$, where I is the index set of $\mathcal{SG}$

Super-faces Incidentally and interesting, one of the useful outcomes of the subdivision algorithm presented in Algorithm 5 is the detection of what we call a "super-face". For every connected component in the *MDG*, the subdivision algorithm finds one super-face that entails all the other faces of that connected component of the *MDG* (see Figure 12.) The algorithm returns the super-face among other faces. We identify the superface by the size of its surface, which is the biggest face in each sub-graph of the *MDG*.

2.5 Neighbourhood and membership functions

Neighbourhood functions of a face is handled by the twin concept of half-edges, as described in [1]. That is to say, two faces are neighbors if they each have one

[5]The face data structure is curated with an attribute of a *hole*.

(a) curves (b) partitioning faces (c) super-face

Figure 12: Each subdivision process yields a set of partitioning faces, plus a "super-face" that covers the whole partitioned area, and represents its outer shape.

half-edge with a twin belonging to the other face.

$$neighbour\,(face_i) = \{face_j \mid \exists e_i, e_i \in face_i, twin(e_i) \in face_j\}$$

For cases consisting of only straight lines, membership function could be implemented through a set of tests relying on cross products, as described in [1]. The tests are to check whether if a point is located on the same side of all the bounding edges of a face. This would only work if the edge could be represented by a vector. As can be seen in Figure 4, edges turn to arcs in the presence of circles. As an alternative we suggest the use of "point-in-polygon" method, which is based upon the Jordan Curve theorem [6]. Regardless of the method for checking whether if a point is inside a closed curve, the membership function behaves as:

$$member\,(face, p) = \begin{cases} 1 & \text{if } (p \in face) \wedge (p \notin hole_i \mid hole_i \in face) \\ 0 & \text{otherwise} \end{cases}$$

Another important note regarding the membership function is that the holes are represented with the same data structure as faces. It is impotant to ignore nested holes, when checking the second condition of the membership function $(point \notin hole_i \mid hole_i \in face)$. That is to say checking the $point \notin hole_i$ is itself another evaluation of the membership function for the $hole_i$, in which any nested holes inside $hole_i$ should be discarded. Figure 13 demonstrates an example where missing this point might result ina wrong membership estimation for the outer face.

[6] In fact, as will be described in Section **??**, in our implementation each face will be represented by "path" class of the *matplotlib* library. This class, based on bézier curves, provides a method for detecting whether if a given point is located in a closed path. As a consequence the modification of point-in-polygon method is skipped here.

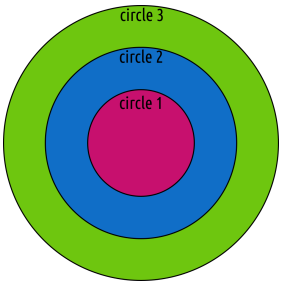


Figure 13: xxx

Appendices

A Univariate representation of the curves

Line The alternative representation of the level-curve of a linear function $f(x, y) = ax + by + c$ could be written as:

$$\dot{f}(\theta) = (x, y) = \begin{cases} (x_l + \theta, y_l) & \text{if } s = 0 \\ (x_l, y_l + \theta) & \text{if } s = \pm\infty \\ \left(x_l + \dot{a}\theta, y_l + \dot{b}\theta\right) & \text{otherwise} \end{cases}$$

where (x_l, y_l) is an arbitrary (but fixed as a reference) point on the level-curve of the $f(x, y)$, s is the slope of the level-curve of the $f(x, y)$, and $\dot{a}, \dot{b}$ are given by:

$$\dot{a} = \frac{\sqrt{s^2 + 1}}{s^2 + 1}, \quad \dot{b} = \dot{a}s$$

The inverse function ($\theta = \dot{f}^{-1}(x, y)$) is given by:

$$\theta = \dot{f}^{-1}(x, y) = \begin{cases} x & \text{if } s = 0 \\ y & \text{if } s = \pm\infty \\ \frac{x - x_l}{\dot{a}} & \text{if } \dot{a} \neq 0 \\ \frac{y - y_l}{\dot{b}} & \text{otherwise} \end{cases}$$

While the second derivative of a line is a null vector, the first derivative of a line in this representation would be:

$$\frac{d\dot{f}(\theta)}{d\theta} = (\cos(\theta), \sin(\theta)), \text{ where } \theta = \arctan(s)$$

Circle Similarly, for the level-curve of a conic function $f(x, y) = (x - x_c)^2 + (y - y_c)^2 - r_c$ (hence the level-curve would be a conic section, in this case a circle) the alternative representation could be written as:

$$\dot{f}(\theta) = (x, y) = (x_c + r_c \cos(\theta), y_c + r_c \sin(\theta))$$

where (x_c, y_c) is the center of the $f(x, y)$ function, and radius r_c defines the desired level-curve of the $f(x, y)$.

The inverse function ($\theta = \dot{f}^{-1}(x, y)$) is given by:

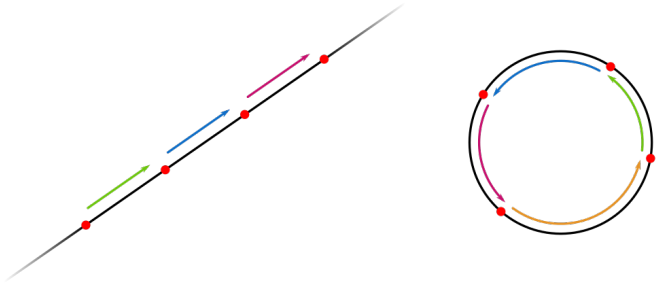
$$\theta = \dot{f}^{-1}(x, y) = \arctan2(y - y_c, x - x_c)$$

The first and second derivatives of a circle in this representation would be:

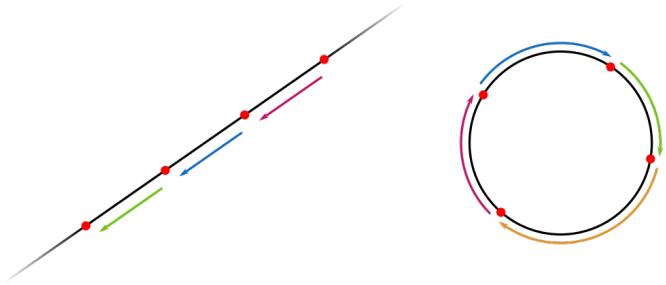
$$\frac{d\dot{f}(\theta)}{d\theta} = (-r_c \sin(\theta), r_c \cos(\theta))$$

$$\frac{d^2\dot{f}(\theta)}{d\theta^2} = (-r_c \cos(\theta), -r_c \sin(\theta))$$

Direction of half-edges and the derivatives half-edges are directional, as the same segment of a curve turns into two half-edges with opposite directions. Direction of half-edges is very important in the correct calculation of the first derivative vectors. If a half-edge has a negative direction, as demonstrated in figure 14, its first derivative vectors are rotated 180^o degrees to face in the oppposite direction.



(a) half-edges with positive direction



(b) half-edges with negative direction

Figure 14: Direction of half-edges and its effect on the derivative vectors.

References

[1] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*. Springer, 2000. (Cited on pages 4, 6, 7, 14, and 15.)