Adaptive Cell Decomposition
A Space Subdivision Method Beyond Lines

September 7, 2016

1 Subdivision

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

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 would be described. A more comprehensive version of algorithm 1 will be presented 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 quite identical to the subdivision algorithm presented in [1], is demonstrated in figure 1. It should be reiterate that the focus of this work is to revise the meta-algorithm and its sub-procedures, so that it 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'll provides 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 1) is to identify those partitioned areas of the space (denoted by $\mathcal{S}$) into a set of

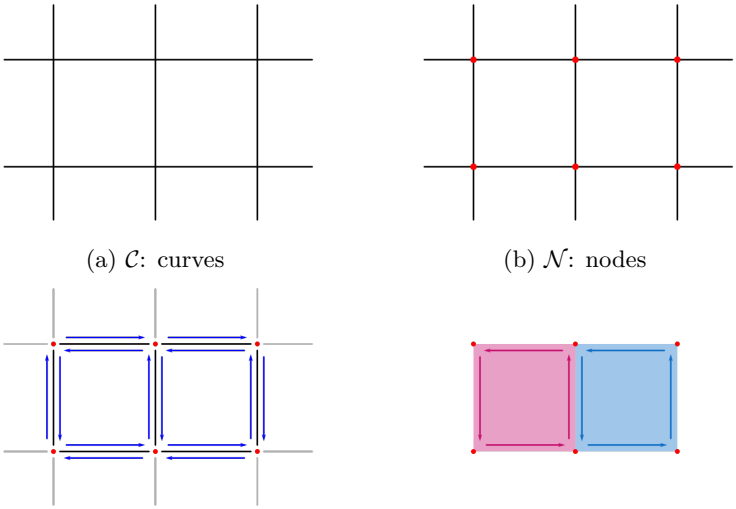
faces (denoted by $\mathcal{F}$.) The first step is to find the set all the intersection points bewteen all the curves (figure 1b). This set is denoted by $\mathcal{N}$, standing for *nodes* [1]. Every node resulting from this process is also assigned to the curves that yielded it. For the next step, that is segmenting curves into half-edge based on their nodes, it is important that the nodes' assigments to the curves should be geometrically sorted over curves. In the case of only straight line, the sorting process is performed by looking at the x (or y) values of the nodes. A more comprehensive approach for sorting will be presented later in section 1.2. After sorting the nodes over each curve, the curves are segmented to edges as illustrated in figure 1c. Note that each simple edge is composed of two "twin" directed half-edges, represented by blue vectors in figure 1c. For more detail on half-edge data structure please see [1], or the appendix B. In the final stage, we attend to construct the set of faces $\mathcal{F}$, through a process of path following. The process starts by fetching (and removing) a half-edge from an open list, moving it to the list of a face's boundary. The next member of the face's boundary list is the half-edge that follows the last last in the list. The potential candidates for the next member are those half-edge who start from the node which the last half-edge of the list is arriving at. The next element 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 curves from circle classes, for which we provide other measures from first and second derivatives of the curves.

1.1 Intersection procedure

This procedure, as presented in algorithm 2, intersects all curves with each other. Each intersection point (node as we call them) will be assigned to curves which it is yielded from. 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.

Non-intersecting Circles The result of intersection procedure is the ground for identifying half-edges (as detailed in the next section). Each curve is segmented into half-edges, according to those nodes that are placed on them. In the presence of circles, there is a chance that a circle might not be intersecting with 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.)

[1] The reason for the name is the important role that they play in constructing a graph in the extended version of the algorithm. Eventhough 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 1: A simplified demonstration of the subdivision algorithm.

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 $\mathcal{N}$
    assign $node_k$ to $curve_i$ and $curve_j$
  end for
end for

for all $node_i, node_j \in \mathcal{N} \mid i \neq j, distance(node_i, node_j) = 0$ **do**
  assign $node_i$ to all curves that $node_j$ is assigned to
  dump $node_j$ from $\mathcal{N}$
end for

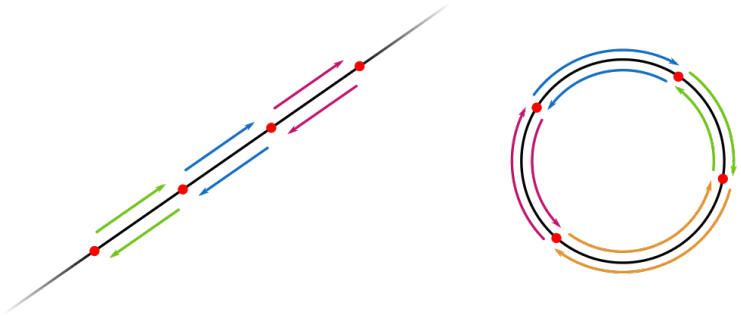


Figure 2: Examples of half-edge construction over a line and a circle. In each case, half-edges with the same color are twins.

1.2 Half-Edge identification

Following the algorithm 2, every intersection point (node) lying 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 are 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 line, the sorting process is performed by looking at the x (or y) values of the nodes. This would clearly fail in the presence of circles. 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 over the level-curve of the multivariable function[2]. Given the "alternative" representation of a level-curve as a function of a single variable θ, we use the inverse function of it (denoted by $\dot{f}^{-1}$), to sort nodes over the curve ($\theta = \dot{f}^{-1}(x, y)$). Provided that it's possible to sort nodes over the curves, we can proceed with the algorithm 3 which describes the construction of half-edges. Figure 2 demonstrates two examples of half-edge construction over a line and a circle. The 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.

1.3 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 start 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 is an iterative process that terminats 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 3.

[2]The equations are presented in appendix A.

Algorithm 3 Segment: half-edge construction

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

$\mathcal{E} = \emptyset$
for all $curve_i \in \mathcal{C}$ **do**
  $list\_n = \{node_j \mid node_j \in \mathcal{N}, node_j$ is assigned to $curve_i\}$
  sort the $list\_n$ according to $\theta_j = f_i^{-1}(node_j)$
  $I =$ 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

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

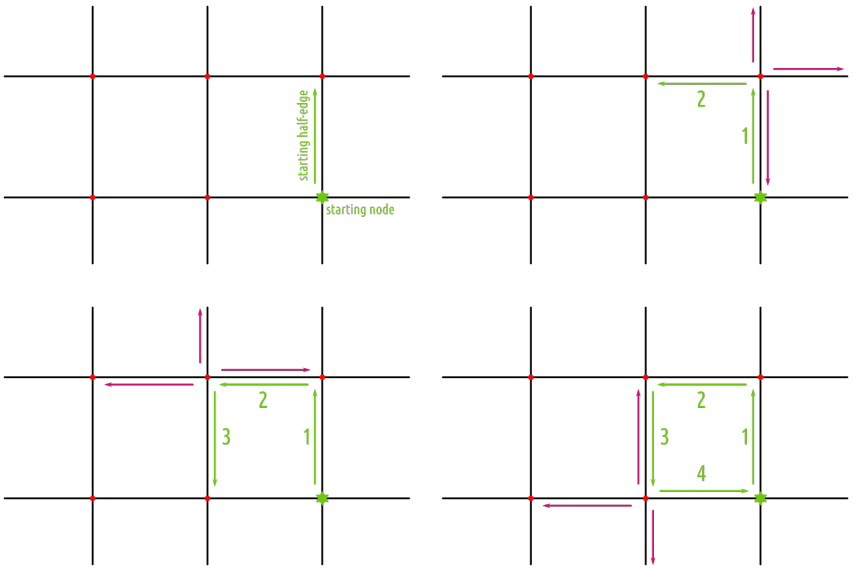
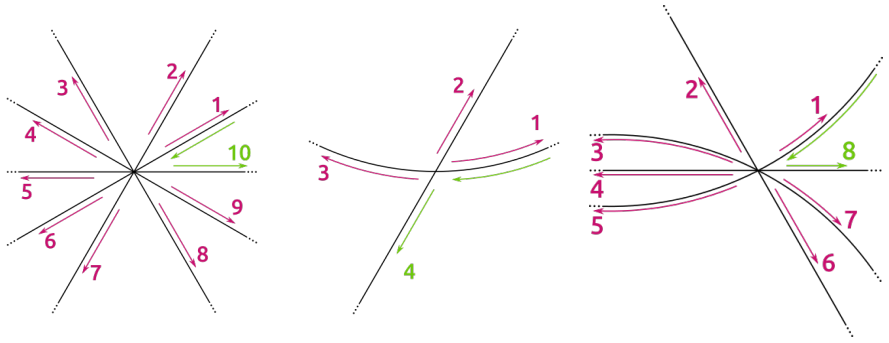


Figure 3: The process of face identification towards partitioning. First a half-edge is picked, which initiats the identification of a new face. Then the correct successor (colored green) is identified among candidates. Rejected candidates at each step are colored red. The process terminates, and the 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 4: Few examples of correct successor selections. Arriving half-edges and the correct successors are colored green, and all the rejected candidates are colored red.

1.3.1 Finding the correct successor half-edge

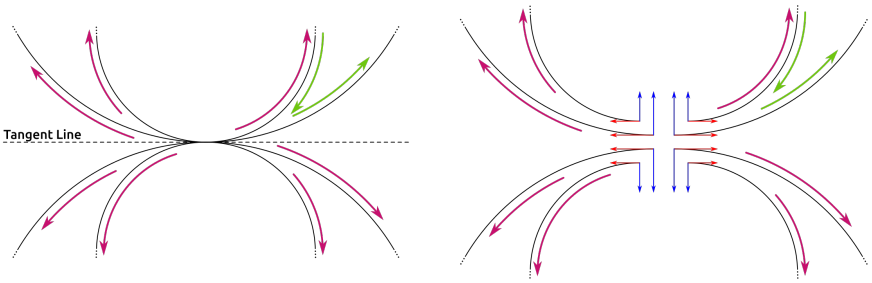
Half-edges departing from the node, which last half-edge of the face arrived at, are flagged as candidates. Among the candidates, the twin of the last half-edge is rejected. The winning candidate is the one with widest departing angle (CCW) with respect to the departing angle of the last half-edge's twin. That is to say, the half-edge that is closest to the twin's departure. Few examples of correct successor selections are demonstrated in figure 4, 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. figures 4a), finding the next half-edge would be simply to look at candidate half-edges' angles, as half-edges are vectors. However this does not hold in the presence of circles where half-edges are arcs (see figures 4b and 4c.) 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.

1.3.2 Insufficiency of the first derivative in case of tangency

When curves intersect while being tangent, the measure of first derivatives' angle at the intersection point is not sufficient to detect the correct successor. This challenge is demonstrated in figure 5a 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 arriving half-edge) that have the same departing angle at the intersection point as the correct successor.

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



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

Figure 5: 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 cadidates half-edges are provided based on both the first and second derivatives.

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

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

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

If the key_1 is not sufficient to detect the correct successor, that is to say in the tangent case, the second derivative brings a measure of 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 makes up the key_2.

1.4 Resolving the restriction

So far we have presented modified versions of the sub-procedures in algorithm 1, so that they can handle curves beyond straight lines (i.e. circles.) However this does not suffice, since the algorithm 1 itself would fail in the presence of circles. Figure 6 show examples of where the "restricted" algorithm would fail. These failures occure when some curves are located inside other faces without intersecting with them. As a consequence, those insider curves are not accounted for in the entailing faces as there is not connected path between the two.

To understand the root of this failure, let's construct a multi-directional graph (MDG) from all the nodes $\mathcal{N}$ and half-edges $\mathcal{E}$ as demostrated in figure 7. The restricted algorithm would work under the assumption that MDG is connected. This assumption would not hold in the cases of figure 6.

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

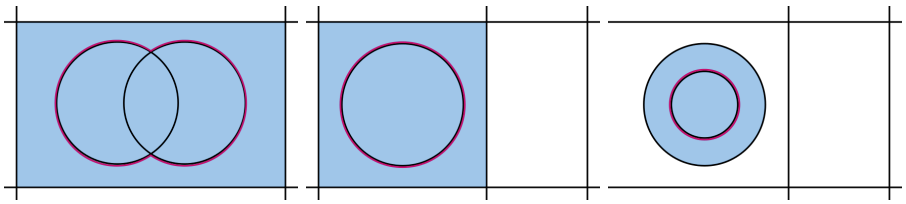
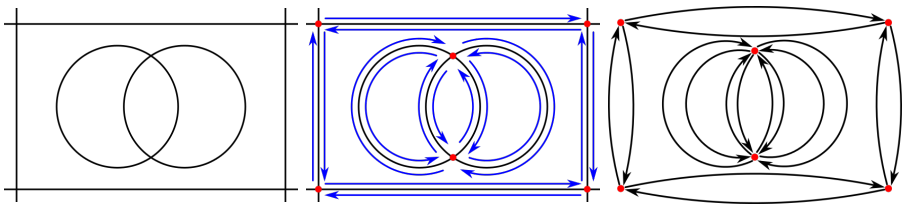


Figure 6: The restricted algorithm 1 fails when some curves are enclosed inside other faces without intersecting with each other. Blue areas show the entailing faces in each case, and the regions that should be substracted from those faces are marked red (what we call super-faces) .



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

Figure 7: Construction of a multi-directional graph *MDG* from nodes and half-edges. It is used to identify connected components which result in disconnected partitioning.

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 disconnected and non-overlapping; or *ii*) on subgraph is completely contained in only *one* face of the other subgraph(s). Relying on this, we resolve this problem by partitioning each subg-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" functionality of the subdivision's faces that will be described in section 1.5. It involves picking an arbitrary point (node) from each sub-graph and check whether if it is entailed in any other sub-graphs' faces. If the test implies the overlapping of two sub-graphs, figuratively speaking a hole in the shape of inside sub-graph is punched in the face entailing it[5]. The sub-graphs' outer shape that are colored red in figure 6 are called super-face, and will be described next. After punching holes, 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 summerize in the 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**
    punch a hole (super-face of sub_graph_i) in $face_k$ of sub_graph_j
  end if
end for

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

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

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

Figure 8: Each subdivision process yields a set of partitioning faces, plus a "super-face" that covers the whole partitioned area, and represets its outer shape.

Super-faces Incidintally 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 coonected component of the *MDG* (see figure 8.) The algorthim returns the super-face amongst all other faces. We identify the superface by the size of its surface, which is the biggest face in each subg-graph of the *MDG*.

1.5 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 neighbours if they each have one 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 9, 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, the membership function behaves as:

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

[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.

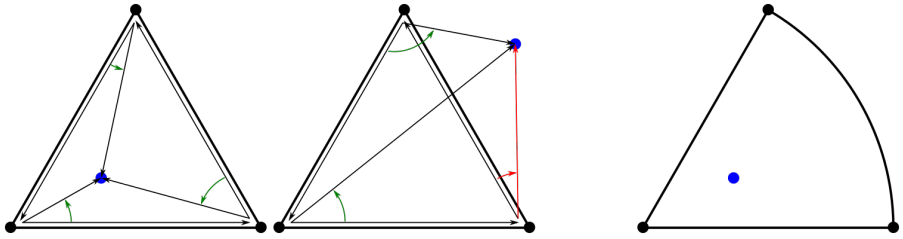


Figure 9: Memebrship function for the case of only straight lines could be based on tests relying on a cross product. However, as the edges are no longer vectors in the presence of circles, the conventional membership procedure requires revision.

Appendices

A Single variable representation of line and circle

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 represetation 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

B Data Structures

Curve

Node

Half-Edge

Face

Decomposition

Subdivision

References

[1] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*. Springer, 2000. (Cited on pages 1, 2, and 11.)