CURDIS: A Template for Incremental Curve Discretization Algorithms and its Application to Conics

LATOUR Philippe [1*] , VAN DROOGENBROECK Marc [1]

1. *Montefiore Institute, University of Liège, Quartier Polytech 1, Allée de la découverte 10, 4000 Liège, Belgium*

*** Corresponding author**, philippe.latour@doct.uliege.be

Received: 20 June 2024 **Accepted:** 27 June 2024

Abstract We introduce CURDIS, a template for algorithms to discretize arcs of regular curves by incrementally producing a list of support pixels covering the arc. In this template, algorithms proceed by finding the tangent quadrant at each point of the arc and determining which side the curve exits the pixel according to a tailored criterion. These two elements can be adapted for any type of curve, leading to algorithms dedicated to the shape of specific curves. While the calculation of the tangent quadrant for various curves, such as lines, conics, or cubics, is simple, it is more complex to analyze how pixels are traversed by the curve. In the case of conic arcs, we found a criterion for determining the pixel exit side. This leads us to present a new algorithm, called CURDIS-C, specific to the discretization of conics, for which we provide all the details. Surprisingly, the criterion for conics requires between one and three sign tests and four additions per pixel, making the algorithm efficient for resource-constrained systems and feasible for fixed-point or integer arithmetic implementations. Our algorithm also perfectly handles the pathological cases where the conic intersects a pixel twice or changes quadrants multiple times within this pixel, achieving this generality at the cost of potentially computing up to two square roots per arc. We illustrate the use of CURDIS for the discretization of different curves, such as ellipses, hyperbolas, and parabolas, even when they degenerate into lines or corners.

Key words Computer graphics, Curve discretization, Conics, Rasterization, Ellipse drawing, Conic spline, Conic pencil.

1 Introduction

A common problem in computer graphics and computer vision is *drawing* portions of *geometric curves* (line segments, ellipse arcs, Bézier or spline curves, for instance), which are continuous by nature, on a discrete image. In practice, images are represented by a matrix of pixel values and drawing a curve boils down to assigning the values of the pixels through which the curve in question passes. This assignment requires (1) to discretize the curve; that is, to find the *discrete curve support or domain*, which is a subset of pixels close to the curve in a certain way (to be defined), and (2) to choose a *curve function* that assigns a value to the corresponding support pixels.

In the literature, the words discretization, rasterization, and drawing are sometimes used interchangeably. In the following, we consider that drawing a curve means discretizing it and then choosing a curve function which assigns a value to the support pixels.

When we would like to draw a curve for visualization purposes, that is, to create an *image* of a curve, it often happens that the choice of the curve function is implicit. Indeed, we simply compute the curve support (we discretize the curve) and set the color of the obtained support pixels as requested by the user. This is obviously equivalent to a constant curve function. However, it is well known that constant curve functions often produce pixelated, and therefore unnatural, images. A possible solution to this problem is to use an anti-aliased drawing in which the curve function assigns a value to support pixels that depends on the distance between the curve and the center of that pixel.

From another side, in computer vision, curve drawing may be involved in some typical applications such as detecting features or analyzing a discrete image along a path that is a geometric curve, or determining how a given geometric curve fits the contours of an object. In this case, one way to proceed is to start by detecting *edge elements*, also called edgels, which are represented by their location (pixel) in the image and by the local image orientation (image gradient) at these pixels. Subsequently, the image processing algorithm discretizes the curve of interest and compares its orientation at the support pixels with the orientation of the closest image edgel. In this application, to assist the image processing operation, we chose a curve function that assigns the orientation of the curve normal to the support pixels.

The choice of the curve function is thus highly application dependent and, in this work, we will only consider the problem of curve discretization, which is the first step of a curve drawing process.

Outline. The purpose of this paper is twofold. First, we present CURDIS, an algorithmic template for the discretization of curves that is independent of the type of curve being discretized. Second, we describe how to instantiate this template for conics by providing implementation details specific to this type of curve. In Section 2, we review previous works on curve discretization. In particular, we discuss the various ways to define a curve support and select one of them. This section also introduces our conventions for mathematical expressions, image coordinate systems, and pixel grid representations. Section 3 details the principles of our algorithmic template and provides all necessary details for implementing the algorithms. In Section 4 and Appendix A, we describe the mathematical representation of conics and conic arcs that we use. In Section 5, we present the algorithm named CURDIS-C for discretizing conics (ellipse, circle, parabola, hyperbola, quadratic Bézier, or conic splines) by incrementally covering them with pixels. Finally, Section 6 discusses the strengths and limitations of CURDIS, and concludes the paper.

2 Problem description and related work

In an image, the *curve support* is a subset of pixels that are close to the curve according to a specified criterion. Conceptually, discretizing a curve involves two main steps: (1) sampling the curve, and (2) quantizing the sampled points. The sampling step reduces the curve to a finite list of points on the curve, and quantization replaces each of these sampled point with the center of the nearest grid pixel.

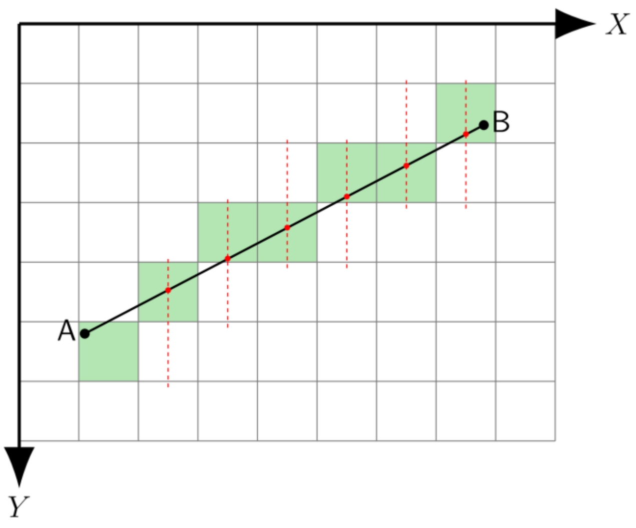
There are different methods for sampling a curve. Sampling points can be distributed, uniformly or non-uniformly, along the curve, similar to one-dimensional signal sampling, and thus independently of the pixel grid. In this case, as described in [6], the curve is replaced by a polygonal approximation, that can be obtained either by sampling the parameter of the curve (when represented with parametric equations) or by iterative subdivision. Efficient methods for drawing ellipse arcs using this approach are described in [5], [18]. The challenge lies in choosing the appropriate sampling step: it should be small enough to avoid gaps in the curve but large enough to minimize duplicate pixels.

Alternatively, sampling points can be chosen as intersections of the curve either with grid lines passing through pixel centers or with pixel boundary lines. Algorithms implementing this approach do not necessarily specify explicitly sampling points; instead, they directly provide a list of support pixels (quantized points). These algorithms incrementally construct the list by adding pixels one by one as the point moves along the curve, with the corresponding pixel shifting on the grid toward one of its neighbors. This type of approach is often referred to as *incremental* and is well-suited for traversing a curve in a discrete image.

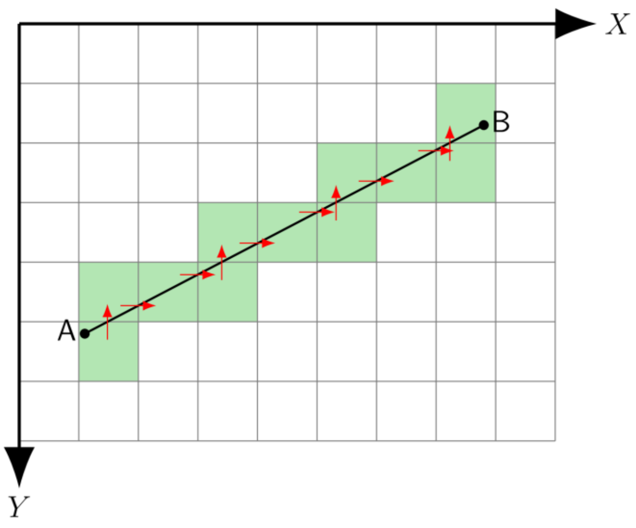
2.1 Curve support

An incremental method begins by defining which pixels should belong to the discrete support of the curve and then implements an algorithm to achieve this. Defining the curve support is not as simple as it seems. In [7], Freeman proposes two techniques to decide which neighboring pixel to include in the curve support.

Grid-intersect quantization. The first technique is called *grid-intersect* quantization. According to this technique, we look for the intersection points of the curve with grid lines passing through the pixel centers. When the tangent to the curve arc is more horizontal than vertical, we calculate the intersections of the arc with the vertical grid lines, and vice versa. Figure 1a illustrates this technique for discretizing a line segment. The segment is more horizontal than vertical, and we consider its intersections with vertical grid lines displayed as dotted red lines. The intersection points (the red point) define the pixels (colored green) used to quantize the curve. Bresenham's algorithms and their variants [2], [3], [14] are examples of this approach. As shown in Figure 1a, it is possible to obtain a series of pixels that may not completely cover the curve.



(a) Grid-intersect.



(b) Square-box.

Figure 1 Two approaches to quantize a line segment.

Square-box quantization. The second technique is *square-box* quantization, illustrated in Figure 1b for a line segment. The square-box discretization technique selects all the pixels crossed by the curve, in contrast to the grid-intersect approach. Conceptually, we start from the pixel containing the starting point A; this pixel initiates the list of support pixels. Then, we travel the curve, and whenever we cross a horizontal or vertical pixel boundary line, we enter a new pixel, which is then added to the list of support pixels. This process continues until endpoint B is reached. Note that square-box discretization can be considered a variant of grid-intersect discretization, in which only one pixel index changes at a time.

Comparison of quantization techniques. The curve support provided by grid-intersect quantization contains fewer pixels than that obtained by square-box quantization. This can be either an advantage or a disadvantage, depending on the application. Moreover, the square-box technique updates pixels through horizontal or vertical movement, which may be easier (or not) to implement than the diagonal and side movements of the grid-intersect technique. Furthermore, the square-box technique involves determining the quadrant of the tangent only, whereas the grid-intersect technique likely requires distinguishing between tangent octants.

In practice, the choice between these two methods is determined by the underlying application. When drawing curves for visualization purposes, it may be unnecessary to have all the pixels covering the curve, making grid-intersect quantization a suitable choice. However, anti-aliased drawing and certain computer vision applications may benefit from a curve support that fully covers the curve in the image.

Our work will focus on incremental curve discretization algorithms where the curve support fully covers the curve and is obtained using the square-box quantization technique.

2.2 Related work

As explained, while traveling the curve, incremental implementations have to decide which pixels to add to the curve support. In theory, this decision is based on curve intersections with lines, depending on the chosen quantization technique. However, in practice, authors deduce information about the position of these intersection points from the value of an error function, also known as residue, at pixel grid midpoints, pixels centers or pixel corners. To the best of our knowledge, this error function is always a type of algebraic distance based on the implicit algebraic equation satisfied by the Cartesian coordinates of the curve points.

Bresenham [3] was the first to propose discretizing lines with a linear error function representing the equation of the line. In a subsequent paper [2], he used the quadratic function defining circles for discretizing them. He established that this approach was equivalent to the midpoint method introduced in [8], and that both provided the best fit. Later, Van Aken [17] compared Bresenham's extended algorithm for axis-aligned ellipses (referred to as the two-point method) with a midpoint method for ellipses, which Kappel [9] further enhanced. In all these methods, the error function is based on the equation of the ellipse. DaSilva [4] extended previous approaches to ellipses in general position and managed pathological cases of very thin ellipses.

Furthermore, Pitteway [14] used the same principle for conics; the function defining the equation of the conic served as the error function. However, he later showed in [15] that his algorithm did not work in certain cases, such as when the conic rotates rapidly and its tangent orientation changes multiple times across the same pixel. Pratt [16] improved Pitteway's method and Banissi [1] extended the algorithm to manage general conics, even thin hyperbolas. Zingl [19] proposed to extend this approach to curves of higher degree and described implementations for curves up to degree 3. However, his algorithm (like Pitteway's algorithm [14]) is only applicable to curves that do not fold or rotate rapidly on themselves.

All the algorithms discussed implement incremental grid-intersect quantization techniques. They may be modified to obtain square-box discretization algorithms for lines, circles, or axis-aligned ellipses. However, to the best of our knowledge, CURDIS is the first general algorithmic template for incremental square-box discretization of conics or general curves.

2.3 Algorithmic principles of discretization solutions

In the literature, solutions to curve discretization problems are based on two types of conceptual elements. The first group of elements includes all algorithmic aspects of the solution that are reasonably independent of the type of curve to be discretized. The second group encompasses all the equations specific to the curve that are necessary for the concrete implementation of the conceptual algorithm described by the elements of the first group. Most of the time, authors do not separate these aspects and present solutions where algorithmic and mathematical elements are intermixed.

One of our objectives is to clearly separate the algorithm, independent of the curve, from the mathematical part, specific to the curve, in order to present an algorithmic template useful for the largest possible class of curves. This template, called CURDIS, will use various existing algorithmic principles encountered in applications of incremental curve discretization.

Like many authors, we assume that a pixel on the curve is known and define a criterion to decide the next pixel to add to the curve support. The idea is to construct the curve support incrementally, starting from the beginning of the arc and then moving from one pixel to the next until the end of the arc.

The criterion for deciding the next pixel to add to the support is systematically based on an error function that represents a sort of distance to the curve. This illustrates the mix between algorithmic and mathematical aspects. Conversely, the CURDIS template recognizes that this criterion is specific to the curve and does not impose using the error function as the sole method for deciding the next pixel to include. While the error function can certainly be part of the criterion, it is probably not the only one (except for straight lines).

Moreover, another widely used concept in literature is to consider the direction of the tangent to the curve, processing only the neighboring pixels located in this direction. Specifically, knowing the quadrant or octant of this tangent restricts the tests to just two or three neighbors. CURDIS, which implements a square-box quantization technique, only needs to compute the tangent quadrant. In contrast, grid-intersect quantization techniques, common in most existing algorithms, often require octant information.

In addition, in the literature, the separation of the problem based on the tangent's orientation is sometimes presented as an implementation trick to accelerate calculations. We believe that this separation is more conceptual and necessary for the generality of the CURDIS strategy.

Except in the case of straight lines, the quadrant or octant of the tangent generally varies along the curve arc. There are two known strategies to address this variation: (1) Frequently, authors verify that the tangent quadrant/octant remains unchanged in the newly added pixel and, if a change is detected, they adapt their algorithm accordingly (see [1], [4]). (2) Some authors detect, in advance, the points where the tangent to the curve changes quadrant or octant and divide the curve into segments with constant orientation (see [12], [19]).

Conceptually, these two strategies are not very different. Both assume that the curve can be divided into sub-arcs with constant quadrants or octants. In Section 3, we present our algorithm that follows the second strategy.

We note that dividing the curve into sub-arcs with a constant tangent quadrant, and thus knowing the last pixel of each sub-arc, allows us to simplify the criterion for deciding the next pixel to add to the support when nearing the end of the sub-arc.

The CURDIS template builds on many existing ideas prevalent in the curve discretization domain. Its originality lies somewhat in the arrangement of ideas, in the emphasis on the generality of the solution, and especially in the use of the last row and column information of the sub-arc to simplify the criterion.

2.4 Notations

Mathematical notations. In the following, we will use:

$\mathsf{sans\ serif}$	font for geometric entities (the point P, the line d, the conic f, etc.),
$italic$	font for projective or homogeneous quantities (the projective coordinate X, the line coefficient u, etc.),
Roman	font for Cartesian or non-homogeneous quantities (the Cartesian coordinate X, the ellipse major semi-axes a, etc.),
$\mathbf{bold\ roman}$	fonts for vectors of point Cartesian coordinates (such as $\mathbf{X}$),
$\boldsymbol{bold\ italic}$	fonts for vectors of point projective coordinates (such as $\boldsymbol{X}$) or homogeneous coefficients of lines (such as $\boldsymbol{u}$).

To save space, we will denote column vector as a list separated by semicolons. For instance, the vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ will also be noted $(a; b; c)$.

Concerning the row vectors, it is often necessary to add extra space between the components to clearly separate them in more complex expressions. When no confusion is possible, we will represent them as comma-separated lists. For instance, the row vector $(a \quad b \quad c)$ will be noted (a, b, c).

Continuous domain. We restrict our attention to a finite (rectangular) domain of the plane of width W and height H. We assume that the arc of curve is completely included in this domain. We choose the top-left corner of the image domain as the origin of the reference frame (not the center of the top-left pixel), with the X axis directed from left to right, and the Y axis directed from top to bottom (see Figures 1a and 1b). We consider that the rectangular domain is open to the right and the bottom. It means, that we always have $0 \leq \mathrm{X} < \mathrm{W}$ and $0 \leq \mathrm{Y} < \mathrm{H}$.

Because the Y axis is directed downward, we work in a reversed axes system, and the positive (trigonometric) rotation direction is clockwise (not counterclockwise). The quadrants are numbered accordingly, as depicted in Figure 2a. The quadrants are semi-open intervals; they contain their minimum value, but not their maximum value (the first quadrant is defined by the interval $\left[0, \frac{\pi}{2}\right[$, for instance).

Grid of pixels. We cover this domain with a grid or matrix of pixels, with n rows and m columns. Moreover, we consider square pixels with a unit width and height, so that $\mathrm{H} = n$ and $\mathrm{W} = m$.

When a point S (represented by $\mathbf{S} = (\mathrm{X}; \mathrm{Y})$) belongs to the pixel p_{ij}, its integer indexes (i, j) are given by $j = \lfloor \mathrm{X} \rfloor$ and $i = \lfloor \mathrm{Y} \rfloor$, where $\lfloor x \rfloor$ is the *floor* function (the largest integer less than or equal to x).

The point P_{ij} located at the center of the pixel p_{ij} has Cartesian coordinates $\mathbf{P}_{ij} = (j + 0.5; i + 0.5)$.

3 Description of CURDIS: An algorithmic template for incremental curve discretization

As outlined earlier, there exist several ways to discretize a curve. In this paper, we consider only the square-box quantization technique. Therefore, we define the discrete support of a curve as the list of all the pixels crossed by the curve.

In the following, we present CURDIS, an algorithmic template for curve discretization according to the square-box technique to provide a list of connected pixels going from one end of the arc to the other and containing all the pixels crossed by the curve. In Section 5, we will apply it to define and implement an algorithm specific to the discretization of arcs of conics.

3.1 Curve description

We consider an arc of a curve s defined by its parametric equations in Cartesian coordinates as follows

$$\mathbf{S}(t) = \left(\mathrm{X}_S(t); \mathrm{Y}_S(t)\right) \quad \text{with } t \in [a, b], \tag{1}$$

and $\mathrm{X}_S(t), \mathrm{Y}_S(t) \in C_1([a,b])$. The first point of the arc is represented by $\mathbf{A} = \mathbf{S}(t = a)$ and the last point by $\mathbf{B} = \mathbf{S}(t = b)$. There is no cusp or gap in the curve. In addition, we also hypothesize that the curve has no multiple points, therefore $\mathbf{S}(t) = \mathbf{S}(t')$ only if $t = t'$ in the interval $[a, b]$. The tangent vector $\mathbf{T}(t) = d\mathbf{S}(t)/dt$ is thus well defined for all points and changes continuously from $\mathbf{A}$ to $\mathbf{B}$.

If we eliminate t from the the parametric equations ($\mathrm{X} = \mathrm{X}_S(t)$ and $\mathrm{Y} = \mathrm{Y}_S(t)$), we may theoretically obtain a relation (let say $\gamma(\mathbf{X}) = 0$) between the Cartesian coordinates of points belonging to the curve. A curve may also be defined implicitly by this relation. In this representation, the normal to the curve is given by $\nabla\gamma(\mathbf{X}) = (\partial\gamma/\partial\mathrm{X}; \partial\gamma/\partial\mathrm{Y})$. The tangent vector is obtained by rotating this normal by an angle of $\pm\frac{\pi}{2}$, depending on the normal direction and the direction of travel along the curve.

We define the pixel containing the curve point $\mathbf{S}(t)$ as $p(t) = p_{i(t)j(t)}$, where $i(t) = \lfloor \mathrm{Y}_S(t) \rfloor \in \{0, \cdots, n-1\}$ and $j(t) = \lfloor \mathrm{X}_S(t) \rfloor \in \{0, \cdots, m-1\}$ are respectively the row and column indexes in the pixel matrix.

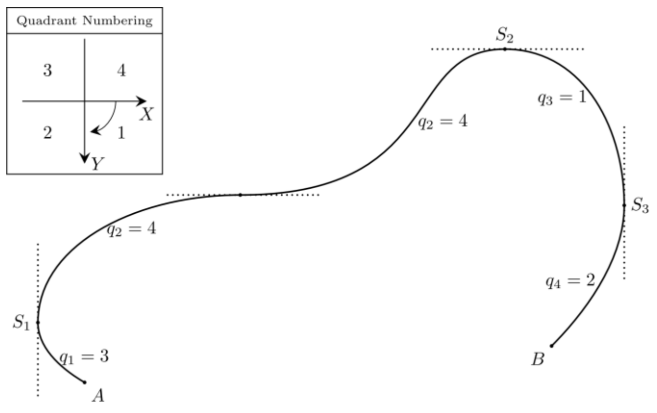
3.2 Curve partition into constant tangent quadrant sub-arc

We begin by analyzing the quadrant $q(t)$ of the orientation of the tangent vector to the curve. Formally, we define it as the quadrant of the angle between the tangent vector and the X-axis. This function $q(t)$ is piecewise constant, defined on $[a, b]$, and takes its values in the discrete set $\{1,2,3,4\}$.

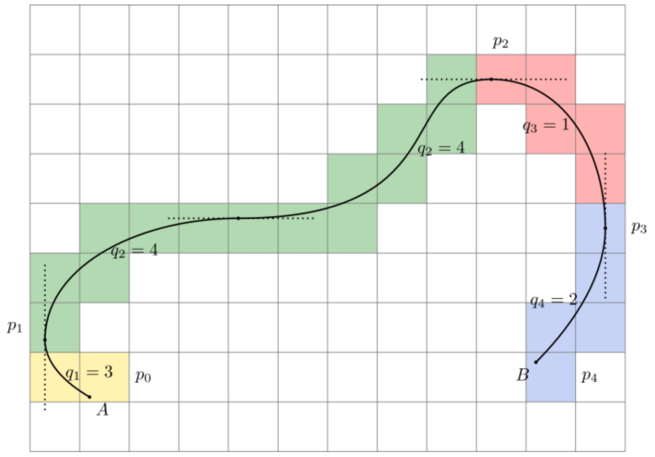
Next, we divide the interval $[a, b]$ into K sub-intervals for which the quadrant function $q(t)$ is constant. For this purpose, we define;

1. $\{t_0 = a, t_1, \cdots, t_K = b\}$ (with $a = t_0 < t_1 < \cdots < t_K = b$) as the increasing sequence of parameter values t where the quadrant changes (or where the curve arc begins or ends), as well as
2. $\{\mathbf{S}_0 = \mathbf{A}, \cdots, \mathbf{S}_k = \mathbf{S}(t_k) = (\mathrm{X}_k; \mathrm{Y}_k), \cdots, \mathbf{S}_K = \mathbf{B}\}$ as the corresponding sequence of quadrant change points (or where the curve arc begins or ends), and also
3. $\{q_0, q_1, \cdots, q_{K-1}\}$ (with $q_k \in \{1,2,3,4\}$ and $k \in \{0, \cdots, K-1\}$) as the sequence of values of the quadrant function $q(t)$ on the successive intervals $]t_k, t_{k+1}[$ of the parameter t.

An illustration of this partition process into sub-arcs is given in Figure 2a.



(a) Curve partition in constant tangent quadrant sub-arcs.



(b) Sub-arcs discretization.

Figure 2 Discretization of a curve split in segment with constant tangent quadrant. S_k are points where the tangent is either horizontal or vertical. q_k are the sub-arcs tangent quadrant.

Since the curve is C_1 (continuous, and its tangent vector is also continuous), the quadrant can only change when one of its components becomes zero; that is, when the tangent vector becomes horizontal or vertical. This condition is necessary but not sufficient, as at certain inflection points where the curvature of the arc changes sign, the tangent may have a zero component without a change in its quadrant. Therefore, we define the values t_k of the parameter as those where the tangent vector is horizontal or vertical, and which are not inflection points.

Algorithm 1 summarizes discussions of Subections 3.1 and 3.2.

Algorithm 1 CURDIS: Partition of curves into constant tangent quadrant sub-arcs.

Parametric representation	Implicit representation	
$\mathbf{S}(t)$ with $t \in [a,b]$, and $\mathbf{S}(a) = \mathbf{A}$, $\mathbf{S}(b) = \mathbf{B}$	$\gamma(\mathbf{X}) = 0$, and $\gamma(\mathbf{A}) = 0$, $\gamma(\mathbf{B}) = 0$	Curve arc from point A to B
$\mathbf{T}(t) = d\mathbf{S}(t)/dt$	$\mathbf{T} \perp \nabla\gamma(\mathbf{X})$	Tangent vector to the curve
$d\mathrm{X}_S(t)/dt = 0 \rightarrow$ Vertical tangent $d\mathrm{Y}_S(t)/dt = 0 \rightarrow$ Horizontal tangent	$\partial\gamma/\partial\mathrm{Y} = 0 \rightarrow$ Vertical tangent $\partial\gamma/\partial\mathrm{X} = 0 \rightarrow$ Horizontal tangent	One of the tangent components is zero and not an inflection point
$\{\mathbf{S}_0 = \mathbf{A}_r, \cdots, \mathbf{S}_k, \cdots, \mathbf{S}_K = \mathbf{B}\}$		List of tangent quadrant change points (or where the curve arc begins or ends).
$\{q_0, q_1, \cdots, q_{K-1}\}$ with $q_k \in \{1,2,3,4\}$ and $k \in \{0, \cdots, K-1\}$		List of values taken by the tangent quadrant between these change points.

3.3 Quadrant change point discretization

The tangent quadrant change pixels are defined as those containing $\mathbf{S}_k$ points (where the quadrant changes or the curve arc starts or ends). They are denoted by $p_k = (i_k, j_k) = pixel(\mathbf{S}_k) = pixel(\mathrm{X}_k, \mathrm{Y}_k)$.

Usually, we calculate the row and column indexes of this pixel in the grid respectively with the formulas $i_k = \lfloor \mathrm{Y}_k \rfloor$ and $j_k = \lfloor \mathrm{X}_k \rfloor$. However, this can pose a problem.

Indeed, when $q_{k-1} = 1$ and $q_k = 2$ or vice versa, the orientation of the tangent vector at $t = t_k$ is vertical, and around point $\mathbf{S}_k$, the curve approaches the vertical line $\mathrm{X} = \mathrm{X}_k$ from the left, touches it (tangentially at point $\mathbf{S}_k$) without crossing it, and then moves away while remaining on the left side of this line. If X_k is exactly an integer (meaning the line $\mathrm{X} = \mathrm{X}_k$ is a vertical boundary of pixels in the grid), then the curve touches a pixel boundary without entering it. In this case, if we discretize point $\mathbf{S}_k$ by taking $j_k = \lfloor \mathrm{X}_k \rfloor$, we obtain a pixel p_k that the curve does not traverse; it simply touches it. This situation can be problematic later on, and in this case, we choose to discretize the column with the formula $j_k = \lfloor \mathrm{X}_k \rfloor - 1$.

Similarly, when $q_{k-1} = 2$ and $q_k = 3$ or vice versa, the orientation of the tangent vector at $t = t_k$ is horizontal, and around point $\mathbf{S}_k$, the curve approaches the horizontal line $\mathrm{Y} = \mathrm{Y}_k$ from above, touches it (tangentially at point $\mathbf{S}_k$) without crossing it, and then moves away while remaining above this line. In this case, when Y_k is exactly an integer, we choose to discretize the row of pixel p_k with the formula $i_k = \lfloor \mathrm{Y}_k \rfloor - 1$ instead of $i_k = \lfloor \mathrm{Y}_k \rfloor$.

Algorithm 2 presents the *pixel* function that takes one quadrant change point as argument and returns the corresponding discretized pixel.

Algorithm 2 CURDIS: Discretization of quadrant change points.

for $k \in \{0, \cdots, K\}$: $p_k \leftarrow pixel(\mathbf{S}_k)$	Discretization of tangent quadrant change points, using the *pixel* function defined below.
function $pixel(\mathbf{S}_k) \to p_k$:	
$i_k \leftarrow \lfloor \mathrm{Y}_k \rfloor$, $j_k \leftarrow \lfloor \mathrm{X}_k \rfloor$	Initialize the pixel of the tangent quadrant change points with the floor function.
if $q_{k-1}, q_k \in \{1,2\}$, $\mathrm{X}_k \in \mathbb{Z}$, and $\mathrm{X}_S(t) \le \mathrm{X}_k$ around $\mathbf{S}_k$: $j_k \leftarrow j_k - 1$	If the curve touches a pixel from the left without entering it, adapt the column index
if $q_{k-1}, q_k \in \{2,3\}$, $\mathrm{Y}_k \in \mathbb{Z}$, and $\mathrm{Y}_S(t) \le \mathrm{Y}_k$ around $\mathbf{S}_k$: $i_k \leftarrow i_k - 1$	If the curve touches a pixel from the top without entering it, adapt the row index
return $p_k(i_k, j_k)$	Return the computed pixel

3.4 Sub-arc support

It is entirely possible for two or more consecutive pixels in the list $\{p_k\}$ to be identical. This occurs when the curve changes quadrants multiple times within the same pixel. In this scenario, for all indices k such that $p_k = p_{k+1}$, we can choose to remove the redundant element $k+1$ from the lists $\{t_k\}$, $\{q_k\}$, $\{\mathbf{S}_k\}$, $\{\mathbf{P}_k\}$ and $\{p_k\}$. If the lists completely collapse, resulting in $K = 0$ and only one pixel remains, then the arc is entirely covered by this pixel and the problem is solved. Otherwise, we have $K \ge 1$, and the conditions $p_k \ne p_{k+1}$ and $q_k \ne q_{k+1}$ hold for the remaining pixel and quadrant lists.

However, this duplicate removal operation may also be performed later. In such cases, it is the responsibility of the sub-arc discretization process to account for the fact that two consecutive quadrant change points may fall within the same pixel. Subsection 3.5 describes a sub-arc discretization process that follows this approach, thus not requiring that quadrant change pixels be different.

Each sub-arc $\mathrm{s_k}$ (defined by $\mathbf{S}(t)$ with $t \in [t_k, t_{k+1}]$ in parametric representation) has a starting point $\mathbf{S}_k$, and an ending point $\mathbf{S}_{k+1}$. The result of the discretization of this sub-arc is a list of pixels starting at p_k and ending just before p_{k+1}. In fact, we decide to include the pixel p_{k+1} in the curve support of $\mathrm{s_{k+1}}$ instead of $\mathrm{s_k}$. An illustration is provided in Figure 2b. There is an exception to this rule for the last pixel p_K, which is added to the curve support of the last sub-arc $\mathrm{s_{K-1}}$.

If we have previously removed the redundant quadrant change pixels or when $p_k \neq p_{k+1}$, the sub-arc support may not be empty (it contains at least the starting pixel p_k of the sub-arc). Otherwise, when $p_k = p_{k+1}$, the sub-arc support is empty, and we begin processing the next sub-arc.

If the sub-arc support is not empty, all its pixels are crossed by the curve and, except possibly for the first pixel p_k, the quadrant of the curve's tangent vector is constant across these pixels; it is equal to q_k.

3.5 Sub-arc discretization

We consider that we have computed the quadrant change points and divided the arc into sub-arcs. We are now discretizing the sub-arc $\mathrm{s_k}$, assuming $q_k = 4$. The process for other quadrants is similar.

Initially, we know the starting point $\mathbf{S}_k$ of this sub-arc and the corresponding pixel $p_k = pixel(\mathbf{S}_k)$. We also know its final point $\mathbf{S}_{k+1}$ and the corresponding pixel $p_{k+1} = pixel(\mathbf{S}_{k+1})$, which is not part of the sub-arc support.

We initialize a working pixel p with the first sub-arc pixel p_k and build the curve support step by step. At each step, we first check that the working pixel p is not the last pixel p_{k+1}. If it is not, we add it to the curve support and start looking for the next pixel to add.

With our assumptions, we know that the tangent is oriented to the top and to the right (since the tangent is in the fourth quadrant). Therefore, the curve entered the pixel from the bottom or left side and must exit the pixel from its top or right side.

If the working pixel has reached the last row (the uppermost in our case) of the sub-arc, then the next pixel to add cannot be the top neighbor; it must be the right neighbor. Similarly, if the working pixel has reached the last column (the rightmost in our case) of the sub-arc, then the next pixel to add cannot be the right neighbor; it must be the top neighbor.

When the working pixel has reached neither the last row nor the last column, we need another method, referred to as an *exit criterion*, to determine the pixel exit side. This exit criterion, tailored to the specific curve to discretize, may not be defined within the generic CURDIS template, and our method can only be used if we manage to design and implement it efficiently. Based on this *exit criterion*, we determine which of the neighboring pixels is the next working pixel.

When the working pixel p reaches the first pixel p_{k+1} of the next sub-arc (or p_K, the endpoint **B** pixel), the discretization of sub-arc $\mathrm{s_k}$ is complete, and we proceed to discretize the next sub-arc, if any.

In Algorithm 3, the step displayed greenish, specific to CURDIS, will prove to be important in instantiation of CURDIS for conic arc discretization, as explained in Section 5.

Algorithm 3 CURDIS: Algorithmic template for discretizing constant tangent quadrant sub-arcs of curves.

$p \leftarrow p_k \Leftrightarrow i \leftarrow i_k,\ j \leftarrow j_k,$ $Support_k \leftarrow [\ ]$	We initialize the current working pixel p with the starting pixel of the sub-arc and the curve support of this sub-arc with an empty list.
while $i > i_{k+1}$ or $j < j_{k+1}$:	Whenever the starting pixel of the next sub-arc has not been reached:
Add p to $Support_k$	We add the current working pixel to the curve support.
if $i = i_{k+1}$: $j \leftarrow j + 1$	If the working pixel row is equal to the last sub-arc pixel row, then the right neighbor is the only possible one.
else if $j = j_{k+1}$: $i \leftarrow i - 1$	If the working pixel column is equal to the last sub-arc pixel column, then the top neighbor is the only possible one.
else: $(i, j) \leftarrow ExitCriterion$	Otherwise, a curve-specific *exit criterion* determines the side effectively crossed by the curve and which neighboring pixel to choose.

3.6 How to instantiate the CURDIS template?

In conclusion, CURDIS is based on two functions that have to be calculated, in order to implement the discretization of a given curve:

1. A function to *compute the quadrant of the tangent vector* at each point of the curve and identify the points where this quadrant changes. Most of the time, these points are located where one of the tangent components is zero (and not an inflection point), making the tangent either horizontal or vertical.
2. An *exit criterion* to compute through which side the arc of the curve exits a pixel (knowing the tangent quadrant of the curve on this pixel).

In the following, we will present an instantiation of CURDIS for arc of conics.

3.7 Computation or detection of quadrant change points

As explained in Subsection 2.3, the quadrant change points $\mathbf{S}_k = \mathbf{S}(t_k)$ may be either (1) computed in advance, before starting to follow the curve and to add pixels in the curve support, or (2) detected on the fly, during the pixel marching process.

The choice between these two strategies is algorithmic. Indeed, conceptually, in both cases, we consider that the curve is such that it is possible to divide it into subarcs with constant quadrant.

Algorithm 3 presents a version of CURDIS where we compute in advance the quadrant change points $\mathbf{S}_k$. However, we may notice that it is not necessary to have their exact location, it is enough to know that they exist and are located in some pixels $p_k = p(t_k)$. Furthermore, the only thing we really need for this algorithm to work, is a way to ensure that the row and column of new support pixels have not reached yet the last row and column of the current sub-arc.

Indeed, whenever we are able to define a *row criterion* (or *column criterion*) to determine if we have reached or not the last row (or column) of the sub-arc, the previous algorithm can be adapted to detect the quadrant change points (and divide the arc) on the fly, instead of computing them in advance. We will not investigate this possibility further in this paper, as our discretization algorithm for conic arcs computes the quadrant change points in advance.

4 Representation of conics and arcs of conic

There exist various ways to define conics or arcs of conic and most applications have a natural format for describing and manipulating them. These may be geometric parameters (such as center, orientation, and semi-axes) or an algebraic representation of the curve (Cartesian or parametric equations). When drawing or discretization algorithms have to interface with other processing algorithms, it may be necessary to convert the data and switch from one representation of the curve arc to another. This conversion step is not free in terms of computing time, and, ideally, it should be considered when comparing the complexity and calculation times of several algorithms.

This discussion is beyond the scope of this paper, and we choose to represent conics by their implicit algebraic equation, as described in Subsection 4.1, and arcs of conics by conic splines, as detailed in Subsection 4.6 and Appendix A. This mathematical representation will enable us to compute their tangent vector as well as the points where this tangent changes quadrant.

Moreover, the exit criterion required for CURDIS-C instantiation of Algorithm 3 will be defined in Section 5 using the conic equation expressed in a Cartesian coordinate system translated to one of the corners of the working pixel (see Subsections 4.2 and 4.3). Additionally, in this algorithm, each time we move from one working pixel to the next, we also update the conic equation (see Subsection 4.4).

4.1 Conic equation in arbitrary coordinate system

A conic f will thus be defined by the set of the points P whose Cartesian coordinates $\mathbf{X} = (\mathrm{X};\mathrm{Y})$ verify a second degree equation (see [10]) whose the most general form is given by

$$f(\mathbf{X}) = a\mathrm{X}^2 + 2b''\mathrm{XY} + a'\mathrm{Y}^2 + 2b'\mathrm{X} + 2b\mathrm{Y} + a'' = 0\,. \tag{2}$$

In projective coordinates, this equation may be expressed in matrix form $f(\boldsymbol{X}) = \boldsymbol{X}^T\mathbf{f}\boldsymbol{X} = 0$, where $\boldsymbol{X} = (X;Y;Z)$ is a vector of projective point coordinates and the matrix of the conic homogeneous coefficients is defined by

$$\mathbf{f} = \begin{pmatrix} a & b'' & b' \\ b'' & a' & b \\ b' & b & a'' \end{pmatrix}. \tag{3}$$

We notice that a conic is defined by 6 homogeneous algebraic coefficients or by 5 independent parameters.

4.2 Conic equation in translated coordinate system

In Section 5, we will define mathematically the exit criterion for the discretization algorithm of arc of conics. This criterion is mainly based on the analysis of the intersection of the conic with some horizontal and vertical lines crossing at some point $\mathbf{Z} = (\mathrm{X}_Z;\mathrm{Y}_Z)$.

In order to simplify the analysis of these intersection points, we translate the origin of the axis system to point $\mathbf{Z}$ (and we possibly reverse the orientation of the new axes). The coordinate transform formulas are

$$\begin{cases} \mathrm{X} = \mathrm{X}_Z + s_x x \\ \mathrm{Y} = \mathrm{Y}_Z + s_y y\,, \end{cases} \tag{4}$$

where x and y are the coordinates of a point relatively to the new reference frame and $s_x, s_y = \pm 1$ are parameters allowing axis orientation reversal, if needed.

After this change of variables, the algebraic equation of the conic (2) becomes

$$F_Z(x,y) = F_Z + D_Z x + E_Z y + Ax^2 + Bxy + Cy^2 = 0\,, \tag{5}$$

where

$$\begin{aligned}&A = a\,,\ \ C = a' \quad \text{and} \quad B = 2s_x s_y b''\,,\\ &D_Z = 2s_x(a\mathrm{X}_Z + b''\mathrm{Y}_Z + b') \quad \text{and} \quad E_Z = 2s_y(b''\mathrm{X}_Z + a'\mathrm{Y}_Z + b)\,,\\ &F_Z = a\mathrm{X}_Z^2 + 2b''\mathrm{X}_Z\mathrm{Y}_Z + a'\mathrm{Y}_Z^2 + 2b'\mathrm{X}_Z + 2b\mathrm{Y}_Z + a''\,.\end{aligned} \tag{6}$$

We notice that A, B and C are independent of the point $\mathbf{Z}$, D_Z and E_Z are linear functions of $\mathbf{Z}$ and F_Z is equal to the value $f(\mathbf{Z})$ of the conic function in the original coordinate system (Equation (2)). As usual, $F_Z = f(\mathbf{Z}) = F_Z(0{,}0)$ is considered as a signed algebraic distance between the point $\mathbf{Z}$ and the conic. In the following, when no ambiguity is possible, we remove the subscript Z and use the notations D, E, and F.

4.3 Conic equation in pixel corner coordinate system

More specifically, we are interested in analyzing the intersection of the conic with the horizontal and vertical pixel boundary lines that intersect at a given pixel corner. The corner and the boundary lines of interest depend on the quadrant of the tangent to the curve in the current pixel.

We now consider the specific case of an arc of conic whose tangent is in quadrant 4. The other cases can be analyzed similarly. Since the tangent is in the fourth quadrant, the curve entered into the pixel by the bottom or the left side and must exit the pixel by its top or right side.

We denote the row and column index of the current pixel as i and j respectively. Therefore, the tangent to the curve is in the direction of the top right corner of the current pixel, whose Cartesian coordinates are $\mathbf{Z}_{ij}^{(q=4)} = (j+1; i)$. The pixel boundary lines of interest for quadrant 4 are the horizontal line $\mathrm{Y} = i$ and the vertical line $\mathrm{X} = j+1$. We thus translate the origin of the axis system to the corner $\mathbf{Z}_{ij}^{(4)}$ and we orient the new axes in such a way that the coordinates of points inside the (i,j) pixel are positive. Therefore, in the case of quadrant 4, we must reverse the x axis direction and we have $s_x = -1$ and $s_y = +1$ in Equation (4). The coordinate transform formulas are $\mathrm{X} = j + 1 - x$ and $\mathrm{Y} = i + y$.

4.4 Updating of conic coefficients

In the course of Algorithm 3, described in Subsection 3.5, we move from the current pixel to one of its neighbors. When we arrive at a new pixel, we have to consider a new top right corner point and we update the translated equation (5) accordingly.

But, the first step is to compute the conic equation in the translated frame of a corner of point A. If the tangent is oriented in quadrant 4, this is the top-right corner $\mathbf{Z}_A^{(4)}$ and, using Equation (4), we obtain the initial values of D, E and F. We recall that A, B, and C are constant (excepted for the sign of B, that may vary when quadrant changes).

In the subsequent steps, we move either horizontally or vertically by one pixel only. If we move horizontally to the right, point ordinates do not change but new abscissas are incremented. The transform formulas (4) are $x = -1 + x'$ and $y = 0 + y'$ and Equation (5) becomes

$$F_{Z'}(x', y') = F_Z - D_Z + A + (D_Z - 2A)x' + (E_Z - B)y' + Ax'^2 + Bx'y' + Cy'^2 = 0\,. \tag{7}$$

Therefore, we update the variables with $D_{Z'} = D_Z - 2A$, $E_{Z'} = E_Z - B$ and $F_{Z'} = F_Z - D_Z + A$. Likewise, if we move vertically upwards, the transform formulas (4) are $x = x'$ and $y = -1 + y'$ and we update the variables with $D_{Z'} = D_Z - B$, $E_{Z'} = E_Z - 2C$ and $F_{Z'} = F_Z - E_Z + C$.

These updating equations may be expressed simply as described in Table 1, where D'' and E'' are intermediate variables.

Horizontal	Vertical
$D'' = D - A$	$E'' = E - C$
$F' = F - D''$	$F' = F - E''$
$D' = D'' - A$	$E' = E'' - C$
$E' = E - B$	$D' = D - B$

Table 1 Update formulas for error function F, and coefficients D and E, when moving from one pixel to the next. D'' and E'' are intermediate variables. For diagonal moves, both formulas (horizontal and vertical) need to be applied.

4.5 Tangent to the conic

To apply our framework described in Section 3 to the arc of conics, we need to compute the quadrant of their tangent. This is simple because there are no inflection points, and the curvature has the same direction at all points (the arc rotates in the same direction everywhere). Therefore, the quadrant may only change at points where the tangent is vertical or horizontal, which is easy to compute. The only drawback of these formulas is requiring the calculation of up to two square roots per arc of conics (because a conic is quadratic in X and Y).

So, given the quadrant of the tangents at point A and B and the direction of travel of the arc, we know which quadrant changing points should be computed and in which order we will meet them. For instance, if the tangent at A is in the fourth quadrant and the tangent at B is in the second quadrant, then we know that the tangent of a point traveling the arc in the positive direction of rotation will be in quadrant 4 at the beginning, then it will first change to quadrant 1 and then to quadrant 2. In this case we have to compute the uppermost point of the arc, with a horizontal tangent (between quadrant 4 and 1) and the rightmost point of the arc, with a vertical tangent (between quadrant 1 and 2).

4.6 Arc of conics

An arc of a conic is somewhat different from a whole conic. In addition to the conic itself, the two endpoints of the arc must also be specified. Theoretically, a point on a conic can be specified by a single parameter, such as a curvilinear abscissa, and an arc of a conic is thus fully described by 7 independent parameters: 5 for the conic and 2 for the arc endpoints. Such a representation is rarely used in practice.

In Appendix A, we detail our mathematical representation of arcs of conics, based on conic splines and pencils of conics. Here, we summarize the aspects of this representation that will be used in the next section to describe the exit criterion for the discretization of arcs of conics.

We consider that arcs of conics are described by their two endpoints, A and B, the tangent lines at these endpoints, and a third point on the arc between the two endpoints (see Appendix A).

Ellipses and parabolas divide the plane into two domains: one convex (*inside* the conic) and the other concave (*outside* the conic). Hyperbolas divide the plane into three domains: two convex (considered the *interior* of the conic) and one concave (*outside* the conic). Because the endpoints A and B must define an arc, they are on the same branch of the conic, and their midpoint, M, is always *inside* the conic (in a convex domain).

We also assume that we always traverse the arc of conics in the positive direction of rotation. If this is not the case, we simply swap the endpoints of the arc. Therefore, the curvature of the conic (its *interior*) is directed to the right when traveling along the curve from A to B.

Additionally, with the assumptions described in Appendix A, the error function $F(x,y)$ in Equation (5) is positive inside the conic, negative outside, and zero on the conic.

5 CURDIS-C: An algorithm for conic discretization

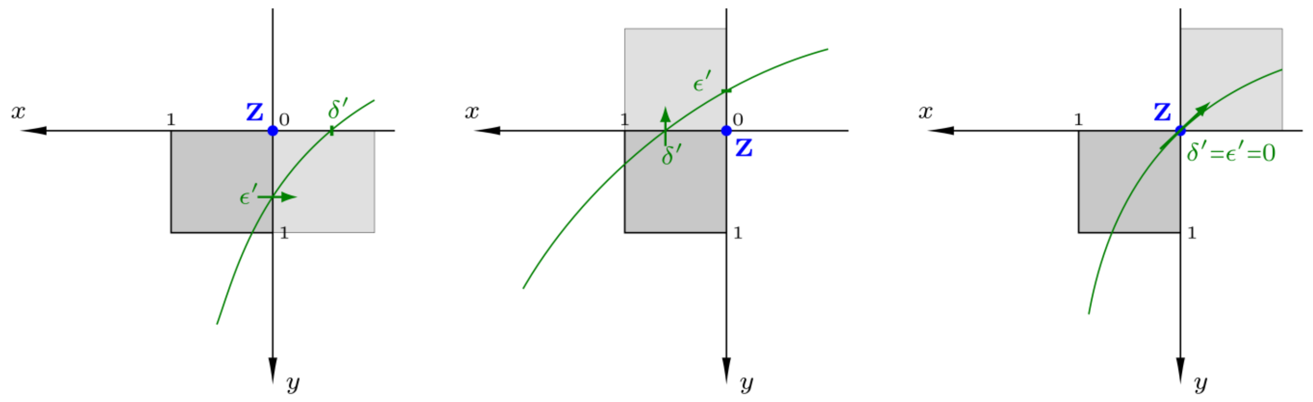
In this section, we present CURDIS-C, a particular implementation of CURDIS applied to conic arcs.

According to our framework, CURDIS-C requires (1) a function computing the quadrant of the tangent vector (and the points where this quadrant changes), and (2) an exit criterion determining through which side the arc of the conic exits a pixel.

We now discuss the details for calculating the exit criterion. For that, we consider the specific case of an arc of conic whose tangent is in quadrant 4. The other cases can be analyzed similarly.

We assume that we have reached neither the last row nor the last column of the arc. This means that the traversed conic arc will necessarily cross the vertical line containing the right side of the pixel as well as the horizontal line containing the top side of the pixel. The question is, obviously, in what order!

Therefore, we should study 3 cases: (1) the curve crosses the vertical line first, making the next pixel the right neighbor (see Figure 3a), (2) the curve crosses the horizontal line first, making the next pixel the top neighbor (see Figure 3b), (3) the curve passes through the top-right corner, crossing both lines simultaneously, making the next pixel the top-right diagonal neighbor (see Figure 3c).

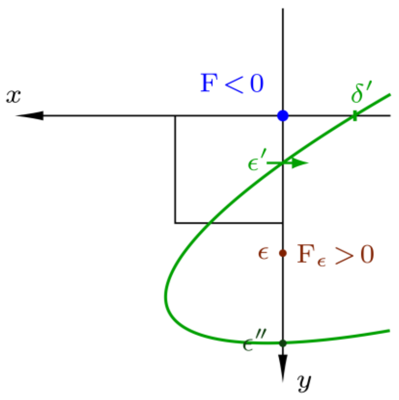


(a) The arc first crosses the vertical axis and enters the right neighbor. **(b) The arc first crosses the horizontal axis and enters the top neighbor.** **(c) The arc passes through the top right corner and enters the top right neighbor.**

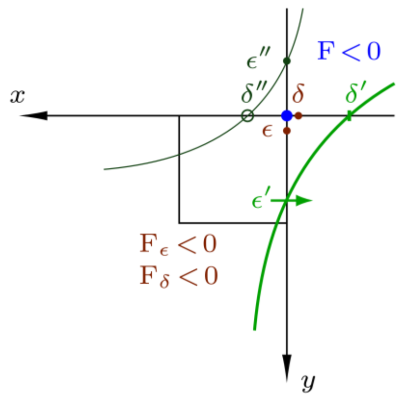
Figure 3 The three possibilities for conic arcs, with tangent in quadrant 4, to exit a pixel.

In Figure 3, we assume that the coordinate system has been translated to the top-right corner of the working pixel, as described in Section 4. The conic arc we are discretizing intersects the x axis at $(\delta', 0)$ and y axis at $(0, \epsilon')$. Both points belong to the same branch of the conic. As the tangent is in quadrant 4, the curvature of this branch is directed downward and to the right, since we choose to traverse the arc in the positive direction of rotation (see Subsection 4.6).

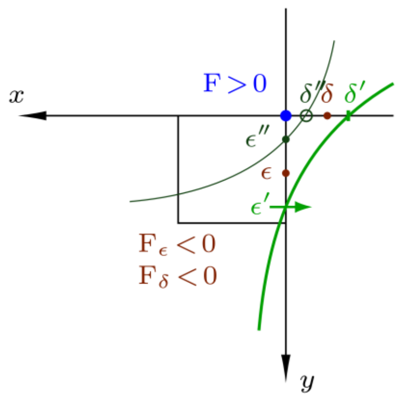
As explained in Section 2, when addressing similar problems, authors compare the sign of the algebraic distance function F computed at appropriate points near the curve (typically the corner $\mathbf{Z}$). We recall that the error function F is positive inside the conic, negative outside, and zero on the conic (see Subsection 4.6 and Appendix A). However, this error function does not represent the algebraic distance to the arc of interest but rather to the conic as a whole. Consequently, if the conic is (1) a highly flattened ellipse or parabola, (2) a very sharp branch of a hyperbola, or (3) a hyperbola that is very flat and close to its asymptotes, it is possible to encounter pixels that are very close to two distinct arcs of the curve or even pixels that are crossed twice by the conic. We illustrate some of these scenarios in Figure 4.



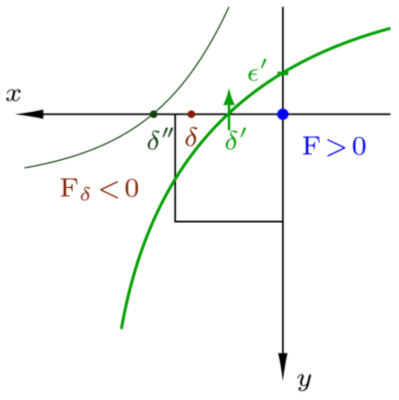
(a) Scenario $H34$, $H44$, or $H54$.



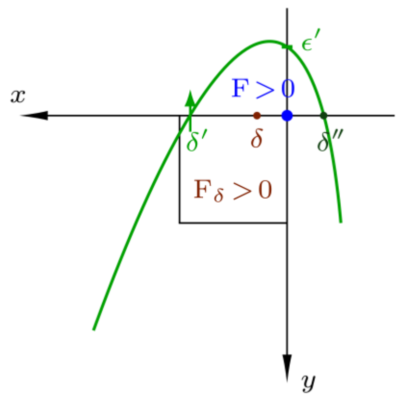
(b) Scenario $H33$.



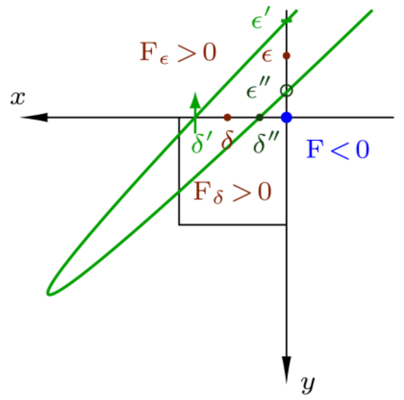
(c) Scenario $H11$.



(d) Scenario $V33$, $V34$ or $V35$.



(e) Scenario $V43$, $V44$, or $V45$.



(f) Scenario $V11$.

Figure 4 Six possible scenarios where a pixel is crossed by a conic with a tangent in quadrant 4. The discretized conic arc, passing through intersection points δ' and ϵ', is in bright green. The other branch of the conic is in dark green. δ'' and ϵ'' are the second intersection points with the axes (see Subsection 5.2). δ and ϵ are midpoints of $\delta'-\delta''$, and $\epsilon'-\epsilon''$ respectively (see Subsection 5.3). Sub-figures are described by scenario codes explained in Subsection 5.1.

In such cases, the value of the error function F is insufficient for a comprehensive analysis of the arc's behavior. In fact, the value and sign of F depend on the relative position of $\mathbf{Z}$, the top-right corner of the pixel, with respect to the intersection points of the conic with the x and y axes. Our exit criterion for discretizing conic arcs is based on a detailed analysis of these intersection points.

We recall that, if a line intersects a conic at a real point, it will also intersect this conic at a second real point (possibly at infinity). The case where this second point coincides with the first is not relevant; indeed, if the horizontal or vertical axes were tangent to the conic, we would have reached the last row or the last column of the arc.

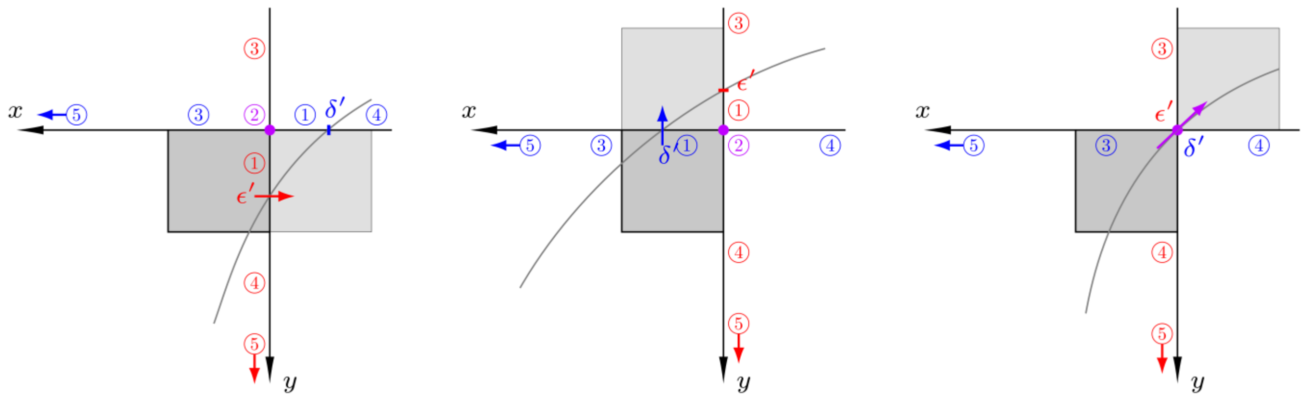
In Figure 4, we also display second intersection points of the conic with the x and y axes. The conic, containing the arc we are discretizing, intersects a second time x axis at $(\delta'', 0)$ and y axis at $(0, \epsilon'')$. We will discuss in Subsection 5.2 the information that can be extracted from these second intersection points.

Furthermore, in Subsection 5.3, we introduce $(\delta, 0)$ the midpoint of $(\delta', 0)$ and $(\delta'', 0)$, the conic intersection points with the x axis. Similarly, we introduce $(0, \epsilon)$ the midpoint of $(0, \epsilon')$ and $(0, \epsilon'')$, the conic intersection points with the y axis. These midpoints are also displayed in Figure 4. Then, we analyze midpoint positions with respect to the conic thank to the sign of the algebraic distances $F_\delta = F(\delta, 0)$ and $F_\epsilon = F(0, \epsilon)$, which are also illustrated in Figure 4.

This analysis of conic intersections with both axes enable us to define our exit criterion in Subsection 5.5.

5.1 Pixel traversed by conic arcs: Possible scenarios

We analyze the possible positions of the second intersection points δ'' and ϵ'' of the conic with the x and y axes.



(a) The conic exits through the right side. **(b) The conic exits through the top side.** **(c) The conic exits through the corner.**

Figure 5 Five possible positions for the second intersection points of the conic with both axes. Sub-figures display these positions for the three possible exit sides (with tangent in quadrant 4). The positions are described by blue numbers for δ'' and by red numbers for ϵ''. Number 2, at the corner, is colored magenta because it represents the position 2 for both δ'' and ϵ''

As illustrated in Figure 5, there are 5 possible positions for second intersection points δ'' and ϵ'':

Position 1 lies between the corner **Z** and the first intersection point. In this case, we have $0 < |\epsilon''| < |\epsilon'|$ and $0 < |\delta''| < |\delta'|$. Moreover, the second intersection points δ'' and ϵ'' have the same sign as their corresponding first intersection point δ' and ϵ'. This position is not possible when $\delta' = \epsilon' = 0$ (the conic exits through the pixel top right corner). Therefore, there is no position 1 in Figure 5c. When the arc exits the pixel through the right side (Figure 5a), this corresponds to $\delta' < \delta'' < 0$ and $0 < \epsilon'' < \epsilon'$. When the arc exits the pixel through the top side (Figure 5b), this corresponds to $0 < \delta'' < \delta'$ and $\epsilon' < \epsilon'' < 0$.

Position 2 is exactly at the corner, and we have $\epsilon'' = 0$ and $\delta'' = 0$. This position is not possible when $\delta' = \epsilon' = 0$ (the conic exits through the pixel top right corner). Therefore, there is no position 2 in Figure 5c.

Position 3 is exterior to the conic arc, and we have $\epsilon'' < \min(0, \epsilon')$ and $\delta'' > \max(0, \delta')$. When the arc exits the pixel through the right side (Figure 5a) or the top right corner (Figure 5c), this corresponds to $\delta'' > 0$ and $\epsilon'' < 0$. When the arc exits the pixel through the top side (Figure 5b), this corresponds to $\delta'' > \delta'$ and $\epsilon'' < \epsilon'$.

Position 4 is interior to the conic arc, and we have $\epsilon'' > \max(0, \epsilon')$ and $\delta'' < \min(0, \delta')$. When the arc exits the pixel through the right side (Figure 5a), this corresponds to $\delta'' < \delta'$ and $\epsilon'' > \epsilon'$. When the arc exits the pixel through the top side (Figure 5b) or the top right corner (Figure 5c), this corresponds to $\delta'' < 0$ and $\epsilon'' > 0$.

Position 5 is at infinity, and we have $\epsilon'' \to \infty$ and $\delta'' \to \infty$.

Therefore, when a conic exits through the right side of a pixel, there are 25 theoretical combined scenarios (5 possibilities for both δ'' and ϵ''). Fortunately, not all combinations are possible, reducing the number to 11. Similarly, when the arc exits through the top side, there are also 25 theoretical combinations, which again reduce to 11 possible scenarios. Additionally, when the arc exits the pixel through the top-right corner, there are 9 theoretical scenarios, all of which are possible. In total, there are 31 possible scenarios for a conic to cross a pixel. We will detail these scenarios in Subsection 5.5.

In the following discussion, we will refer to scenarios using a three character code. The first character in the code indicates the exit side (H for horizontal, V for vertical, and D for diagonal). The second and third characters are numbers between 1 and 5, indicating the position of the second intersection point respectively with the x and y axis.

5.2 Intersection points of the conic with translated axes

We recall that we express the conic equation in a frame translated to the top right corner of the current pixel (see Subsection 4.3). The equation of the conic is thus $F + Dx + Ey + Ax^2 + Bxy + Cy^2 = 0$ (see Equation (5)). In this translated frame, the conic intersects the horizontal top boundary line of the pixel (the x axis of equation $y = 0$) at points with abscissas δ' and δ'', which are solutions of

$$F + Dx + Ax^2 = 0\,. \tag{8}$$

Similarly, the conic intersects the vertical right boundary line of the pixel (the y axis of equation $x = 0$) at points with ordinates ϵ' and ϵ'', which are solutions of

$$F + Ey + Cy^2 = 0\,. \tag{9}$$

As previously explained, the conic arc we are discretizing intersects the x axis at $(\delta', 0)$ and y axis at $(0, \epsilon')$. Therefore, these points belong to the same branch of the conic. However, we have no information about the branch to which the second intersection points $(\delta'', 0)$ and $(0, \epsilon'')$ belong. With our assumptions (curvature directed to the right and downward), $(\delta'', 0)$ belongs to the same branch as $(\delta', 0)$ if and only if δ'' is to the right of δ' ($\delta'' < \delta'$). Similarly, $(0, \epsilon'')$ belongs to the same branch as $(0, \epsilon')$ if and only if ϵ'' is below ϵ' ($\epsilon'' > \epsilon'$).

Note that computing the roots of these equations is not always sufficient to solve our problem. When these roots are close to each other, we may not be able to determine which one belongs to the arc of the curve we are discretizing and which one belongs to another part of the conic.

Nevertheless, these equations provide important information. Indeed, Equations (8) and (9) have two real and distinct solutions because the conic always crosses the lines $y = 0$ and $x = 0$ twice. The discriminants of these equations must be strictly positive, meaning that

$$D^2 - 4AF > 0 \quad \text{and} \quad E^2 - 4CF > 0\,. \tag{10}$$

Additionally, the sign of the trinomials $F + Dx + Ax^2$ and $F + Ey + Cy^2$ is always the sign of A and C (the coefficients of the second-degree terms) excepted when the variable (x or y) lies between the two roots.

The top-right corner of the pixel (**Z**) lies on the vertical line (with abscissa $x = 0$). Therefore, if $\operatorname{sign}(F) = \operatorname{sign}(A)$, the two intersection points of the conic with the horizontal line are on the same side as the corner. Conversely, if $\operatorname{sign}(F) \neq$

$\text{sign}(A)$, the corner lies between the two intersection points.

Similarly, the top-right corner of the pixel ($\mathbf{Z}$) also lies on the horizontal line (with ordinate $y = 0$). Therefore, if $\text{sign}(F) = \text{sign}(C)$, the two intersection points of the conic with the vertical line are on the same side as the corner. Conversely, if $\text{sign}(F) \neq \text{sign}(C)$, the corner lies between the two intersection points.

5.3 Midpoint of the intersection points

Obviously, it is not recommended to explicitly compute the solutions of Equations (8) and (9), as this would require calculating two square roots. However, it is easy to obtain the midpoints of the intersection points, which will be used in our exit criterion.

If $A \neq 0$, the midpoint of the two intersection points of the conic with the horizontal line $y = 0$ has an abscissa

$$\delta = \tfrac{1}{2}(\delta' + \delta'') = -\tfrac{D}{2A} \quad \text{and} \quad \text{sign}(\delta) = -\text{sign}(D)\text{sign}(A)\,. \tag{11}$$

The value of the trinomial at the midpoint $(\delta; 0)$ is

$$F_\delta = F + D\delta + A\delta^2 = -\tfrac{D^2-4AF}{4A} \quad \text{and} \quad \text{sign}(F_\delta) = -\text{sign}(A)\,. \tag{12}$$

Similarly, if $C \neq 0$, the midpoint of the two intersection points of the conic with the vertical line $x = 0$ has an ordinate

$$\epsilon = \tfrac{1}{2}(\epsilon' + \epsilon'') = -\tfrac{E}{2C} \quad \text{and} \quad \text{sign}(\epsilon) = -\text{sign}(E)\text{sign}(C)\,. \tag{13}$$

The value of the trinomial at the midpoint $(0; \epsilon)$ is

$$F_\epsilon = F + E\epsilon + C\epsilon^2 = -\tfrac{E^2-4CF}{4C} \quad \text{and} \quad \text{sign}(F_\epsilon) = -\text{sign}(C)\,. \tag{14}$$

In addition, we recall that the error functions (F_δ and F_ϵ) are positive inside the conic, negative outside, and zero on the conic (see Subsection 4.6).

5.4 Horizontal and vertical asymptotes ($A = 0$ and $C = 0$)

When $A = 0$, the point at infinity of the x axis (with projective coordinate $(1; 0; 0)$) belongs to the conic described by Equation (5). We know that $D \neq 0$ because $D^2 = D^2 - 4AF > 0$ and the finite intersection point is $\delta' = -\frac{F}{D}$. The conic may be a hyperbola with a horizontal asymptote or a parabola with a horizontal axis.

Likewise, when $C = 0$, the point at infinity of the y axis (with projective coordinate $(0; 1; 0)$) belongs to the conic described by Equation (5). We know that $E \neq 0$ because $E^2 = E^2 - 4CF > 0$ and the finite intersection point is $\epsilon' = -\frac{F}{E}$. The conic may be a parabola with a vertical axis or a hyperbola with a vertical asymptote.

5.5 Exit criterion

In Subsection 5.1, we showed that there are 31 possible scenarios for a conic to cross a pixel. In this section, we will first analyze one of these scenarios, in order to illustrate how results from Subsections 5.2, 5.3, and 5.4 can be used to determine the sign of coefficients A, C, D, E, and F for this scenario. Then we will present tables showing the signs of A, C, D, E, and F for all possible scenarios. Finally, we will deduce our exit criterion from these tables.

Let us consider Figure 4c, where the conic arc is in quadrant 4 and exits the pixel through the right side ($\epsilon' > 0$ and $\delta' < 0$). In this scenario, we assume that the second intersection points ϵ'' and δ'' both lie between the corner and the first intersection point (position 1, see Subsection 5.1). Therefore, we have $0 < \epsilon'' < \epsilon'$ and $\delta' < \delta'' < 0$. The character code of this scenario is $H11$.

Because the second intersection points are not at infinity, we know that $A\ ,C \neq 0$. Moreover, the corner is interior to the conic and does not lie in the intervals $]\delta', \delta''[$ and $]\epsilon'', \epsilon'[$. From the discussion in Subsection 5.2, we deduce that $F > 0$, $\text{sign}(A) = \text{sign}(F) = +1$, and $\text{sign}(C) = \text{sign}(F) = +1$.

Furthermore, the midpoints ϵ and δ have the same sign as the corresponding intersection points; $\epsilon > 0$ and $\delta < 0$. From the discussion in Subsection 5.3 and since $A, C > 0$, we deduce that $\text{sign}(D) = -\text{sign}(A)\text{sign}(\delta) = +1$ and $\text{sign}(E) = -\text{sign}(C)\text{sign}(\epsilon) = -1$.

Finally, in this scenario, we have $A > 0$, $C > 0$, $D > 0$, $E < 0$, and $F > 0$.

We conduct a similar analysis for all 31 scenarios and present the results in Table 2 and Table 3.

Table 2 contains all situations where either A or C, or both are zero. As previously explained, this occurs when the conic is either (1) a parabola with a horizontal or vertical axis, or (2) a hyperbola with a horizontal or vertical asymptote. Remarkably, we notice that, despite the diversity of situations, the error function F is sufficient to determine the exit side.

Code	**Exit**	$\boldsymbol{\delta''}$	$\boldsymbol{\epsilon''}$	$\boldsymbol{F}$	$\boldsymbol{A}$	$\boldsymbol{C}$	$\boldsymbol{D}$	$\boldsymbol{E}$	**Figure**
$H53$	Right	$\to \infty$	< 0	$-$	0	$+$	?	?	
$H54$	Right	$\to \infty$	$> \epsilon'$	$-$	0	$-$	?	$+$	4a
$H35$	Right	> 0	$\to \infty$	$-$	$+$	0	?	?	
$H45$	Right	$< \delta'$	$\to \infty$	$-$	$+$	0	$+$	?	
$H55$	Right	$\to \infty$	$\to \infty$	$-$	0	0	?	?	
$V53$	Top	$\to \infty$	$< \epsilon'$	$+$	0	$+$	?	?	
$V54$	Top	$\to \infty$	> 0	$+$	0	$-$	?	?	
$V35$	Top	$> \delta'$	$\to \infty$	$+$	$+$	0	$-$	?	4d
$V45$	Top	< 0	$\to \infty$	$+$	$-$	0	?	?	4e
$V55$	Top	$\to \infty$	$\to \infty$	$+$	0	0	?	?	
$D53$	Corner	$\to \infty$	< 0	0	0	$+$	?	$+$	
$D54$	Corner	$\to \infty$	> 0	0	0	$-$	?	$+$	
$D35$	Corner	> 0	$\to \infty$	0	$+$	0	$-$	?	
$D45$	Corner	< 0	$\to \infty$	0	$-$	0	$-$	?	
$D55$	Corner	$\to \infty$	$\to \infty$	0	0	0	?	?	

Table 2 Possible scenarios for a conic arc (with tangent in quadrant 4 and $A = 0$ or $C = 0$) to intersect a pixel. Question marks indicate elements whose values are unknown and not used by the algorithm.

Table 3 contains the other situations where neither A nor C is zero. In this table, we separate the cases depending on the sign of A or C, which are known in advance and do not change over the course of the algorithm. Therefore, in a practical situation, we first test these signs and only consider the 3 or 5 possible cases. When A and C have not the same sign, that is ($A > 0$ and $C < 0$) or ($A < 0$ and $C > 0$), the sign of the error function F is sufficient to determine the exit side. Otherwise, we need to consider the sign of F and D (or E) to determine the exit side.

Code	Exit	δ''	ϵ''	F	A	C	D	E	Figure
$H11$	Right	$\in]\delta', 0[$	$\in]0, \epsilon'[$	$+$	$+$	$+$	$+$	$-$	4c
$H33$	Right	> 0	< 0	$-$	$+$	$+$	?	?	4b
$V33$	Top	$> \delta'$	$< \epsilon'$	$+$	$+$	$+$	$-$	$+$	4d
$H22$	Right	$= 0$	$= 0$	0	$+$	$+$	$+$	$-$	
$D33$	Corner	> 0	< 0	0	$+$	$+$	$-$	$+$	
$H34$	Right	> 0	$> \epsilon'$	$-$	$+$	$-$	?	$+$	4a
$V34$	Top	$> \delta'$	> 0	$+$	$+$	$-$	$-$	?	4d
$D34$	Corner	> 0	> 0	0	$+$	$-$	$-$	$+$	
$H43$	Right	$< \delta'$	< 0	$-$	$-$	$+$	$-$	?	
$V43$	Top	< 0	$< \epsilon'$	$+$	$-$	$+$	?	$+$	4e
$D43$	Corner	< 0	< 0	0	$-$	$+$	$-$	$+$	
$H44$	Right	$< \delta'$	$> \epsilon'$	$-$	$-$	$-$	$-$	$+$	4a
$V11$	Top	$\in]0, \delta'[$	$\in]\epsilon', 0[$	$-$	$-$	$-$	$+$	$-$	4f
$V44$	Top	< 0	> 0	$+$	$-$	$-$	?	?	4e
$V22$	Top	$= 0$	$= 0$	0	$-$	$-$	$+$	$-$	
$D44$	Corner	< 0	> 0	0	$-$	$-$	$-$	$+$	

Table 3 Possible scenarios for a conic arc (with tangent in quadrant 4 and $A \neq 0$ and $C \neq 0$) to intersect a pixel. Question marks indicate elements whose values are unknown and not used by the algorithm.

Finally, Algorithm 4 describes the exit criterion for determining through which side the conic arc exits a pixel. It requires at most 3 sign tests and 2 comparison tests to verify that we have not reached the last row or column of the current sub-arc.

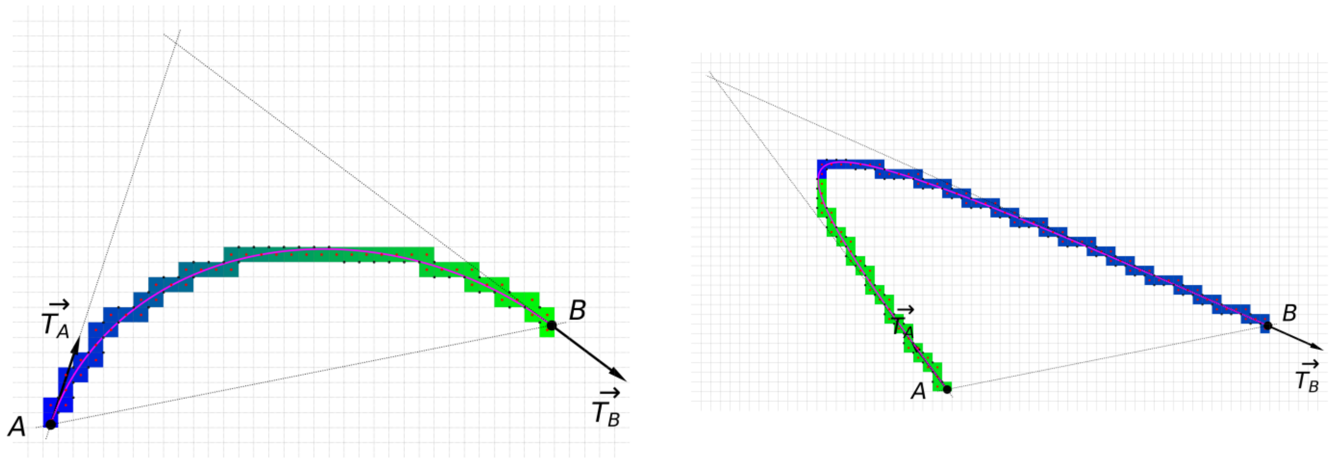
Algorithm 4 Computation of the pixel exit side of an arc of conic whose tangent is in the fourth quadrant.

```
If ( $A = 0$ or $C = 0$) or ( $A > 0$ and $C < 0$) or ( $A < 0$ and $C > 0$)
    if $F < 0$:          Exit → Right
    else if $F > 0$:     Exit → Top
    else:                Exit → Corner

If ( $A > 0$ and $C > 0$)
    if $F < 0$:          Exit → Right
    else if $F > 0$:
        if $D > 0$:      Exit → Right
        else:            Exit → Top
    else:
        if $D > 0$:      Exit → Right
        else:            Exit → Corner

If ( $A < 0$ and $C < 0$)
    if $F > 0$:          Exit → Top
    else if $F < 0$:
        if $D < 0$:      Exit → Right
        else:            Exit → Top
    else:
        if $D > 0$:      Exit → Top
        else:            Exit → Corner
```

5.6 Results

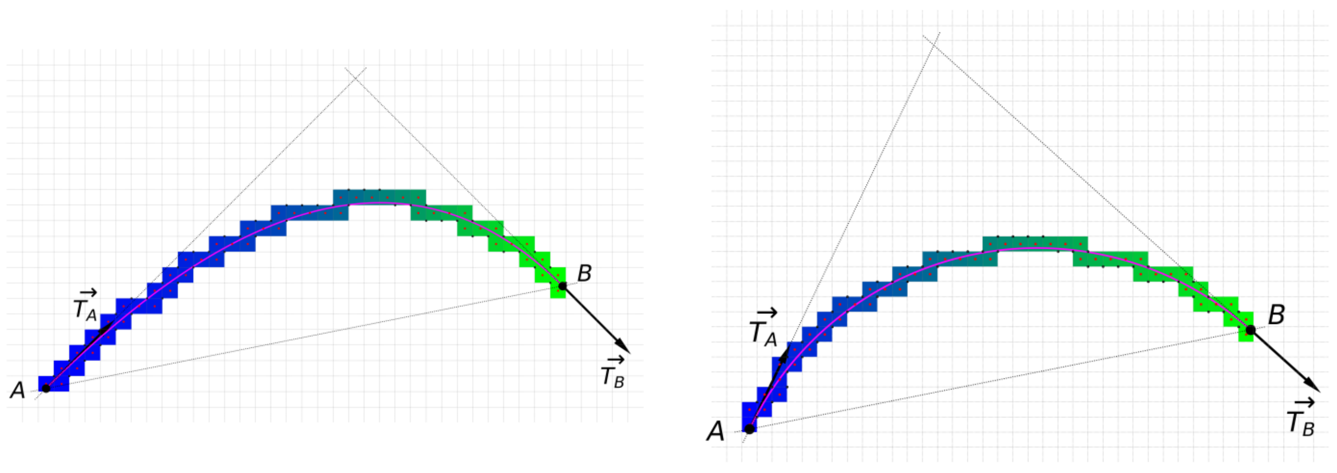
CURDIS-C can discretize all kinds of conic arcs, from ellipses (Figure 6a) to hyperbolas (Figure 6b), or parabolas, which are the quadratic Bézier curves (Figure 7a), and obviously circles (Figure 7b). The algorithm perfectly handles extreme cases such as ellipses that are very flat, or even reduced to a segment (Figure 8a), or hyperbolas that are very sharp, or even reduced to a corner (Figure 8b).

In Figures 6, 7, and 8, the color of the support pixels indicates the direction of the tangent to the conic. This information is naturally provided by the algorithm using the translated conic coefficients D and E. Additionally, all support pixels display a red point at their center and a black point at the corner in the direction of the tangent quadrant.



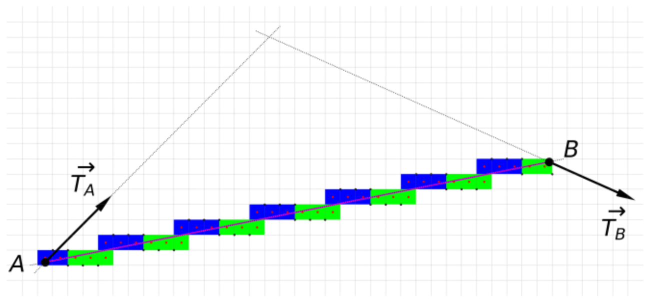
(a) **(b)**

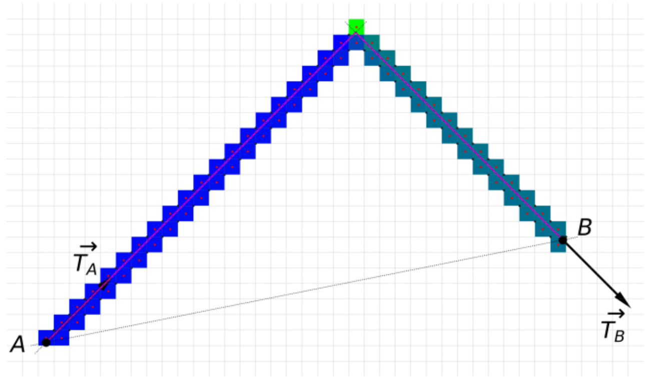
Figure 6 (a) Ellipse arc and (b) Hyperbola arc discretization.



(a) **(b)**

Figure 7 (a) Quadratic Bézier curve (Parabola) and (b) Circle arc discretization.





(a) **(b)**

Figure 8 (a) *Segment* (extremely flat ellipse) and (b) *Corner* (extremely sharp hyperbola) discretization.

6 Discussion and conclusions

We propose CURDIS a framework for discretizing arcs of regular curves (at least C_1) that incrementally produces a list of pixels that completely cover the arc of the curve (according to our definition of discretization). Implementing CURDIS requires two elements: (1) a way to calculate the quadrant of the direction of the tangent to the curve at any point on the arc, and (2) a criterion to decide through which side the curve exits the crossed pixels. Calculating the quadrant of the tangent is straightforward for a whole range of interesting curves in practice, such as lines, conics, or cubics. However, analyzing how the curve traverses the pixels (and thus how it enters and exits them) is a more complex problem. This complexity increases as the class of curve arcs considered becomes broader. In this case, there is a greater chance of encountering a wide variety of possible curve behaviors.

In this paper, we provide a criterion for determining the exit side of a pixel for conics, which constitute the general class of algebraic curves of degree 2. This criterion requires, for each pixel, testing the sign of a maximum of 3 quantities (the minimum being 1). In addition, the general algorithm requires comparing 2 quantities (the pixel row and column indices) to their limit values for each pixel. Furthermore, the variables needed to calculate these quantities to be tested are easily obtained incrementally based on their values at the previous pixel. Updating these variables requires only 4 additions, which allows the algorithm to be fast and ubiquitous. It may even be possible to consider an implementation in fixed-point or integer arithmetic if necessary.

A difficulty of the algorithm lies in the need to know the points of the conic arc where the tangent is perfectly horizontal or vertical. Thus, there are between zero and four points to obtain. This is not very complicated, but it still requires calculating, at most, two square roots. If one wants to avoid this calculation, it is necessary to add one or more tests at each pixel to verify that the tangent has not changed quadrant. Therefore, there is an application tradeoff between the difficulty (and time) of calculating the two square roots and the additional cost of the quadrant change test.

Complete pixel coverage is the definition we have chosen for discretization (the *square-box* approach). However, this is not necessarily the most used, at least for curve drawing. One may wonder about extending our methodology to the *grid-intersect* discretization case, and at least two approaches can be considered.

For example, instead of testing how the curve exits the pixel, one has to test how it crosses the vertical grid lines if the tangent is predominantly horizontal, and vice versa. The grid lines are those that pass through the centers of the pixels. In this solution, it is likely that one will also have to detect the points of the curve where the orientation of the tangent changes octant. The arc will then need to be subdivided based on changes in octants rather than quadrants.

Another way to proceed without calculating octant changes, is to apply our algorithm as is but *skipping* some pixels, for example if the absolute value of the error function F is small (less than a threshold to be defined). In this case, one can consider that the curve passes through the corner of the pixel and one then chooses the diagonal pixel as the next pixel. The same kind of approach can be applied by testing if the exit point of the curve is close to the corner of the pixel, in which case we move to the diagonal pixel.

It is interesting to note that our method provides (at the cost of 4 additional additions per pixel) the error function and the tangent vector at the center of each pixel. This information is very useful, respectively, for anti-aliased drawing and image processing along a conic arc.

Finally, in theory, we can obviously consider extending our algorithm in n dimensions. In this case, it is a matter of finding the list of voxels (in n dimensions) that completely cover a curve. The calculation of quadrant change points must be replaced by the calculation of points where one of the n components of the tangent vector passes through 0. We will also need to find a criterion to decide through which of the n faces (or hyper-faces) the curve exits the voxel!

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used ChatGPT, an AI language model developed by OpenAI, for assistance in translating and proofreading the English text of this article. After using this service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declaration of competing interest

We declare that we have no conflict of interest.

References

[1] Banissi, E. and Golipour, M.K. A new general incremental algorithm for conic section. In *International Conference on Computer Graphics, Imaging and Visualization*, pages 46-51, Singapore, August 2014. Institute of Electrical and Electronics Engineers (IEEE).

[2] Bresenham, Jack E. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100-106, February 1977.

[3] Bresenham, Jack E. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25-30, 1965.

[4] Da Silva, Dilip. Raster algorithms for 2D primitives. Master's thesis, Brown University, Providence, Rhode Island, USA, May 1989.

[5] Fellner, Dieter W. and Helmberg, Christoph. Robust rendering of general ellipses and elliptical arcs. *ACM Transactions on Graphics*, 12(3):251-276, July 1993.

[6] Foley, James and Van Dam, Andries and Feiner, Steven and Hughes, John. *Computer graphics: principles and practice*. Addison-Wesley, second edition, 1990.

[7] Freeman, Herbert. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57-97, March 1974.

[8] Horn, Berthold K.P. Circle generators for display devices. *Computer Graphics and Image Processing*, 5(2):280-288,

June 1976.

[9] Kappel, Michael R. An ellipse-drawing algorithm for raster displays. *Fundamental Algorithms for Computer Graphics*, pages 257-280, 1985.

[10] Ladegaillerie, Yves. Géométrie Affine, Projective, Euclidienne et Anallagmatique. Ellipses, Paris, September 2003.

[11] Latour, Philippe and Van Droogenbroeck, Marc. Dual approaches for elliptic hough transform. In *Discrete Geometry for Computer Imagery (DGCI)*, volume 11414 of *Lecture Notes in Computer Science*, pages 367-379. Springer, 2019.

[12] Laurent, Pierre-Jean and Le Méhauté, Alain and Schumaker, Larry L. A new curve tracing algorithm and some applications. In *Curves and Surfaces*, pages 267-270. Academic Press, January 1991.

[13] Pavlidis, Theodosios. Curve fitting with conic splines. *ACM Transactions on Graphics*, 2(1):1-31, January 1983.

[14] Pitteway, Mike L.V. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *The Computer Journal*, 10(3):282-289, March 1967.

[15] Pitteway, Mike L.V. Algorithms of conic generation. In *Fundamental Algorithms for Computer Graphics*, volume 17 of NATO ASI Series, pages 219-237. Springer Berlin Heidelberg, 1985.

[16] Pratt, Vaughan. Techniques for conic splines. In *ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 151-159, San Francisco, California, USA, July 1985. ACM Press.

[17] Van Aken, Jerry R. An efficient ellipse-drawing algorithm. *IEEE Computer Graphics and Applications*, 4(9):24-35, 1984.

[18] Van Aken, Jerry R. A fast parametric ellipse algorithm. *arXiv*, abs/2009.03434, 2020.

[19] Zingl, Alois. A rasterizing algorithm for drawing curves. Technical note (https://zingl.github.io/Bresenham.pdf), 2016.

Appendix A Conic splines and pencil of conics

In many applications, arcs of conics are naturally described by their two endpoints A and B, the tangent lines a and b at these endpoints, and a third point on the arc between the two endpoints (see Figure 9). We can define the tangent lines at the endpoints by their direction vectors $\vec{\mathrm{T}}_\mathrm{A}$ and $\vec{\mathrm{T}}_\mathrm{B}$, or alternatively by their direction coefficients $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$, which are respectively orthogonal to $\vec{\mathrm{T}}_\mathrm{A}$ and $\vec{\mathrm{T}}_\mathrm{B}$. If we denote P as the intersection point of tangents a and b, we see that the points A, P, and B are the 3 support points of a conic spline or a quadratic Bézier curve (arc of a parabola).

Therefore, we choose to represent arcs of conics as conic splines, as described in [11], [13], [16]. In this framework, we construct a conic by ensuring that (1) it passes through two known points A and B (which also define the arc endpoints), and (2) the tangent to the conic at these points is orthogonal to two known vectors $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$. This representation is especially useful when we need to follow an arc of a conic in an image and compare the direction of the conic's normal with the orientation (image gradients) of nearby image edgels.

However, these 4 pieces of information (A, B, $\vec{\mathrm{G}}$, and $\vec{\mathrm{H}}$) are not sufficient (5 parameters are necessary to define a conic), and there exists an infinity of conics satisfying these four conditions. This set or family of conics depends on 1 parameter; it is the *pencil* of bitangent conics passing through A and B orthogonally to $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$. In Figure 9, the bluish curves are examples of hyperbolas in the pencil, the greenish curves are examples of ellipses in the pencil, and the red curve is the (unique) parabola of the pencil.

We describe different ways to specify a particular conic in the pencil in the next sections.

A.1 Scalar and cross products

We use the following algebraic notations. The scalar product of two vectors $\mathbf{P}$ and $\mathbf{Q}$ (representing the 2D Cartesian coordinates of some points P and Q), is denoted by

$$\lambda_{\mathrm{P}}^{\mathrm{Q}} = \mathrm{X}_P \mathrm{X}_Q + \mathrm{Y}_P \mathrm{Y}_Q \,, \tag{15}$$

and a *kind of* cross product reduced to one dimension by

$$\mu_{\mathrm{P}}^{\mathrm{Q}} = \mathrm{X}_P \mathrm{Y}_Q - \mathrm{X}_Q \mathrm{Y}_P \,. \tag{16}$$

Likewise, if $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S}$ are the Cartesian coordinates of four points, then we note $\mathbf{PQ} = \mathbf{Q} - \mathbf{P}$ and $\mathbf{RS} = \mathbf{S} - \mathbf{R}$ the free vectors joining respectively $\mathbf{P}$ to $\mathbf{Q}$, and $\mathbf{R}$ to $\mathbf{S}$. We then define

$$\begin{aligned} \lambda_{\mathrm{PQ}}^{\mathrm{RS}} &= \mathrm{X}_{PQ} \mathrm{X}_{RS} + \mathrm{Y}_{PQ} \mathrm{Y}_{RS} \,, \\ \mu_{\mathrm{PQ}}^{\mathrm{RS}} &= \mathrm{X}_{PQ} \mathrm{Y}_{RS} - \mathrm{X}_{RS} \mathrm{Y}_{PQ} \,. \end{aligned} \tag{17}$$

A.2 The pencil equation and the $\triangle$ PAB triangle

In this section, and the following, we extensively use the scalar and cross product notations (respectively lambda λ and mu μ) introduced in Section A.1.

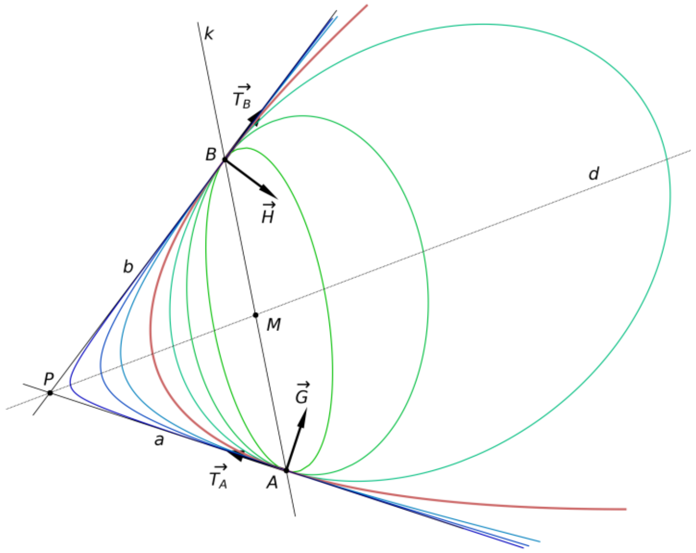


Figure 9 The image points A, B, and the associated direction coefficient vectors of the tangents $\vec{\mathrm{G}}$, $\vec{\mathrm{H}}$, may represents contour elements (edgels) in the image. Compatible conics belong to a pencil of bitangent conics. The conics are tangent to the lines a and b. The line d passing through the midpoint M (of A, B) and the intersection point P (of the tangents a, b) is a diameter of all the conics in the pencil.

We first describe the elements of Figure 9:

- M is the midpoint of A and B, and $\vec{\mathrm{K}}$ is defined as the vector $\frac{1}{2}\overrightarrow{\mathrm{AB}}$.
- $\mu_{\mathrm{K}}^{\mathrm{MX}} = \mathrm{X}_K(\mathrm{Y} - \mathrm{Y}_M) - \mathrm{Y}_K(\mathrm{X} - \mathrm{X}_M) = 0$ is the equation of line k, joining the points A and B.
- $\lambda_{\mathrm{G}}^{\mathrm{AX}} = \vec{\mathrm{G}} \cdot \overrightarrow{\mathrm{AX}} = \mathrm{X}_G(\mathrm{X} - \mathrm{X}_A) + \mathrm{Y}_G(\mathrm{Y} - \mathrm{Y}_A) = 0$ is the equation of the tangent a passing through the point A and

orthogonal to the direction $\vec{\mathrm{G}}$.

- $\lambda_{\mathrm{H}}^{\mathrm{BX}} = \vec{\mathrm{H}} \cdot \overrightarrow{\mathrm{BX}} = \mathrm{X}_H(\mathrm{X} - \mathrm{X}_B) + \mathrm{Y}_H(\mathrm{Y} - \mathrm{Y}_B) = 0$ is the equation of the tangent b passing through the point B and orthogonal to the direction $\vec{\mathrm{H}}$.
- P is the intersection point of the two tangents a and b. It is also the pole of the line k for all conics in the pencil. When $\mu_{\mathrm{G}}^{\mathrm{H}} = 0$, the tangents are parallel, and the pole P is a point at infinity.
- The vector $\vec{\mathrm{D}}$ is parallel to the line MP. When the tangents are not parallel ($\mu_{\mathrm{G}}^{\mathrm{H}} \neq 0$), we have $\mathbf{D} = \mu_{\mathrm{G}}^{\mathrm{H}} \mathbf{MP}$. When $\mu_{\mathrm{G}}^{\mathrm{H}} > 0$, this vector $\vec{\mathrm{D}}$ is oriented in the same direction than the vector $\overrightarrow{\mathrm{MP}}$ and oriented in the inverse direction otherwise.

The general form of the implicit algebraic equation of a conic is given by Equation (2). It is possible to obtain its coefficients in terms of the endpoint coordinates A and B and the tangent direction coefficients $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$ by identifying it with the general equation of a conic in the pencil (see [10] for instance). Therefore, we have

$$f(\mathbf{X}) = 2\lambda_{\mathrm{G}}^{\mathrm{AX}} \lambda_{\mathrm{H}}^{\mathrm{BX}} + \frac{\kappa}{\mu_{\mathrm{K}}^{\mathrm{D}}} {\mu_{\mathrm{K}}^{\mathrm{MX}}}^2 = 0\,, \tag{18}$$

where κ is a parameter that defines which conic of the pencil is effectively represented by the equation. It is one way to provide the missing piece of information that allows us to define completely one and only one specific conic of the pencil when A, B, $\vec{\mathrm{G}}$, and $\vec{\mathrm{H}}$ are given. It is the parameter of this family of conics. We call it the *index* of the conic in the pencil.

In the pencil, the conic is an ellipse when $\kappa > {\mu_{\mathrm{G}}^{\mathrm{H}}}^2$, which tends to the double line k when $\kappa \to +\infty$. It is a parabola when $\kappa = {\mu_{\mathrm{G}}^{\mathrm{H}}}^2$, and a hyperbola when $\kappa < {\mu_{\mathrm{G}}^{\mathrm{H}}}^2$. The conic is the pair of tangents ab when $\kappa = 0$.

A.3 Conditions on endpoints, tangents and traversing direction

We first note that ellipses and parabolas divide the plane into two domains: one convex (*inside* the conic) and the other concave (*outside* the conic). Hyperbolas divide the plane into three domains: two convex (considered the *interior* of the conic) and one concave (*outside* the conic). Because the points A and B must define an arc, they are on the same branch of the conic, and the point M is always *inside* the conic (in a convex domain).

In the following, we will use the abbreviation $\kappa' = \kappa/\mu_{\mathrm{K}}^{\mathrm{D}}$, and the pencil equation becomes $f(\mathbf{X}) = 2\lambda_{\mathrm{G}}^{\mathrm{AX}} \lambda_{\mathrm{H}}^{\mathrm{BX}} + \kappa' {\mu_{\mathrm{K}}^{\mathrm{MX}}}^2 = 0$. For this equation to effectively represent a conic, certain hypotheses on the data A, B, $\vec{\mathrm{G}}$, $\vec{\mathrm{H}}$, and κ are necessary:

1. We always assume that the endpoints A and B are distinct, and the normal direction $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$ are not null; that is
$$\mathrm{A} \neq \mathrm{B}\ ,\ \ \vec{\mathrm{G}} \neq \vec{0}\,, \quad \text{and} \quad \vec{\mathrm{H}} \neq \vec{0}\ . \tag{19}$$
2. In addition, it is not possible to build a pencil of conics from a pair of points when the point P is on the line k (the $\triangle$ PAB triangle is flat). Therefore, we assume that the lines a and b are different. This implies that the point B (resp. A) do not belong to a (resp. b), and that the direction coefficients $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$ are not orthogonal to k. In mathematical terms,
$$\lambda_{\mathrm{G}}^{\mathrm{K}} \neq 0 \quad \text{and} \quad \lambda_{\mathrm{H}}^{\mathrm{K}} \neq 0\,. \tag{20}$$
3. Strictly negative values of κ correspond to hyperbolas whose branches are not on the same side of the tangents as the midpoint M. Therefore, there is no arc joining A to B on these conics, and, in the following, we always consider
$$\kappa \geq 0\,. \tag{21}$$

4. We also consider that $\vec{\mathrm{G}}$ and $\vec{\mathrm{H}}$ are both directed either toward the interior or toward the exterior of the conic. If this is not the case, we simply change the sign of their components. For instance, in Figure 9, they are both oriented toward the interior of the conic. Mathematically, this is equivalent to imposing that $\lambda_{\mathrm{G}}^{\mathrm{K}}$ and $\lambda_{\mathrm{H}}^{\mathrm{K}}$ do not have the same sign, and we should have
$$\mu_{\mathrm{K}}^{\mathrm{D}} = 2\lambda_{\mathrm{G}}^{\mathrm{K}}\lambda_{\mathrm{H}}^{\mathrm{K}} < 0\,. \tag{22}$$
5. Finally, we assume that we always traverse the arc of conics in the positive direction of rotation. If this is not the case, we simply swap the endpoints of the arc. Therefore, the curvature of the conic (its interior) is directed to the right when traveling along the curve from A to B. This direction of rotation is determined by $\operatorname{sign}(\mu_{\mathrm{G}}^{\mathrm{H}})$. When Condition (22) is satisfied, the arc is oriented in the positive direction when
$$\mu_{\mathrm{G}}^{\mathrm{H}} > 0\,. \tag{23}$$

In addition, with our assumptions, the function $f(\mathbf{X})$ in Equation ((18)) is positive inside the conic. Indeed, the midpoint $\mathbf{M}$ is always inside the conic and we can verify that
$$f(\mathbf{M}) = 2\lambda_{\mathrm{G}}^{\mathrm{AM}}\lambda_{\mathrm{H}}^{\mathrm{BM}} + \kappa'{\mu_{\mathrm{K}}^{\mathrm{MM}}}^2 = 2\lambda_{\mathrm{G}}^{\mathrm{K}}\left(-\lambda_{\mathrm{H}}^{\mathrm{K}}\right) = -\mu_{\mathrm{K}}^{\mathrm{D}} > 0\,. \tag{24}$$

A.4 Pixel corner equation

We define an arc of conic with A, B, $\vec{\mathrm{G}}$, $\vec{\mathrm{H}}$, and κ, and we assume that Conditions (19), (20), (21), (22), and (23) are satisfied. The pencil equation (18) is expressed in an arbitrary frame and corresponds to Equation (2). We translate the frame to point Z (and possibly reverse axes) with formulas $\mathrm{X} = \mathrm{X}_Z + s_x x$ and $\mathrm{Y} = \mathrm{Y}_Z + s_y y$ (see Equation (4)). By identification with Equation (5) ($F + Dx + Ey + Ax^2 + Bxy + Cy^2 = 0$), we obtain the coefficients
$$\begin{aligned}
A &= 2\mathrm{X}_G\mathrm{X}_H + \kappa'\mathrm{Y}_K^2\ , \\
B &= 2s_x s_y(\mathrm{X}_H\mathrm{Y}_G + \mathrm{X}_G\mathrm{Y}_H - \kappa'\mathrm{X}_K\mathrm{Y}_K), \\
C &= 2\mathrm{Y}_G\mathrm{Y}_H + \kappa'\mathrm{X}_K^2, \\
D(\mathbf{Z}) &= 2s_x\left(\lambda_{\mathrm{H}}^{\mathrm{BZ}}\mathrm{X}_G + \lambda_{\mathrm{G}}^{\mathrm{AZ}}\mathrm{X}_H - \kappa'\mu_{\mathrm{K}}^{\mathrm{MZ}}\mathrm{Y}_K\right)\ , \\
E(\mathbf{Z}) &= 2s_y\left(\lambda_{\mathrm{H}}^{\mathrm{BZ}}\mathrm{Y}_G + \lambda_{\mathrm{G}}^{\mathrm{AZ}}\mathrm{Y}_H + \kappa'\mu_{\mathrm{K}}^{\mathrm{MZ}}\mathrm{X}_K\right)\ , \\
F(\mathbf{Z}) &= 2\lambda_{\mathrm{G}}^{\mathrm{AZ}}\lambda_{\mathrm{H}}^{\mathrm{BZ}} + \kappa'{\mu_{\mathrm{K}}^{\mathrm{MZ}}}^2\ .
\end{aligned} \tag{25}$$

A.5 Third point of an arc of conic in the pencil

However, choosing a value for the index κ is not always straightforward in practice. Therefore, we instead use an equivalent parameter: the abscissa ξ_E of the intersection point of the conic with the line MP. This line is always the diameter of the conic, whose direction is conjugate to that of the secant k (line AB). E is the intersection point of this diameter with the traveled arc of conic AEB. The tangent at this point E is parallel to the secant k, and it is also the point on the arc farthest from the line k.

We express the abscissa ξ_E by taking M as the origin and $\overrightarrow{\mathrm{MP}}$ as the positive direction, if P is not at infinity (otherwise, the direction of the abscissas is arbitrarily chosen). In practice, this abscissa can be expressed in pixel units, or as a pixel distance from the secant, or by taking the length of segment $\overline{\mathrm{MP}}$ as the unit.

First, let us consider that E is located inside the $\triangle$ PAB triangle. When E is at the midpoint between M and P, the conic is a parabola. When it is closer to M, the conic is an ellipse, and when it is closer to P, the conic is a hyperbola. When E is exactly at M, the conic is the double line k, and when it is at P, the conic is the pair of lines ab. If the point E is outside the segment $\overline{\mathrm{MP}}$ on the M side, the conic arc is necessarily an arc of ellipse, and it is also the larger of the two arcs connecting A and B on this ellipse. The point E may not lie outside the segment $\overline{\mathrm{MP}}$ on the P side because in that case, we would not be able to construct an arc AEB.

Thus, the choice of abscissa ξ_E instead of index κ to specify the conic in the pencil also allows determining, in the case of ellipses, which of the two possible arcs is of interest. Indeed, index κ is not sufficient for this purpose. It would then be necessary to add another piece of information or to impose traveling along the arc necessarily in the positive direction and ask the user to provide points A and B in the appropriate order to process the chosen arc.