Convex Hull of Grid Points below a Line or a Convex Curve

H. Balza-Gomez, J-M. Moreau, D. Michelucci
École Nationale Supérieure des Mines de Saint-Étienne,
158 cours Fauriel, F 42023 Saint-Étienne cedex 2

Abstract

Consider a finite non-vertical, and non-degenerate straight-line segment $s = [s_0, s_1]$ in the Euclidian plane $\mathbb{E}^2$. We give a method for constructing the boundary of the upper convex hull of all the points with integral coordinates below (or on) s, with abscissa in $[x(s_0), x(s_1)]$. The algorithm takes $O(\log n)$ time, if n is the length of the segment. We next show how to perform a similar construction in the case where s is a finite, non-degenerate, convex arc on a quadric curve. The associated method runs in $O(k \log n)$, where n is the arc's length and k the number of vertices on the boundary of the resulting hull. This method may also be used for a line segment; in this case, $k = O(\log n)$, and the second method takes $O(k^2)$ time, compared with $O(k)$ for the first.

1 Upper Hull of Grid Points below a Line Segment

This paper will consider integral hulls, i.e. convex hulls of set of points with integral coordinates in the Euclidean plane (grid points). A paper on $3D$ lattice convex hulls is Reveillès and Yaacoub's [RY95].

This section describes a method for computing the boundary of the upper convex hull of all the grid points located below a given non-degenerate, non-vertical, line segment s (including those possibly *on* it). Obviously, only the upper hull boundary requires some computation. Hence, we shall use the terms "hull" and "upper hull" indifferently in the sequel, unless where specified otherwise.

The supporting line of the segment has equation $y = ax + b$, and is unambiguously described by a and b, since s is supposed not to be vertical; typically, these coefficients are rational numbers, although the method also applies to any computable field, i.e. a field in which the usual exact arithmetic operations (sum, difference, product, division, sign, floor) are available.

Convention: We assume w.l.o.g. that the slope of s is non-negative ($a \geq 0$): this allows a shorter presentation, and the other situation ($a < 0$) is deduced by symmetry.

Since we want the convex hull of the integral points below s, we may restrict the problem to a segment the endpoints of which have integral *abscissæ*, say $x_0 = \lceil x(s_0) \rceil \in \mathbb{Z}$ and $x_1 = \lfloor x(s_1) \rfloor \in \mathbb{Z}$. Such a segment is the hypothenuse of the rectangle triangle with vertices (x_0, ax_0+b), (x_1, ax_0+b), and (x_1, ax_1+b). Let $n = \min(x_1 - x_0, \lfloor a(x_1 - x0) \rfloor)$ be the length (in number of unit intervals) of the shortest edge in this triangle; then the brute-force method to compute the upper hull boundary takes $O(n)$ time. We summarize this method in sub-section 1.1.2, where it is used as a subroutine of our own algorithm, which we now detail.

1.1 Constructing the Upper Hull Boundary

The method we propose runs in $O(\log n)$ time. It proceeds in two steps:

1. The first step consists of computing the boundary of a *prehull*, which yields a superset – actually a superlist – of the final upper hull boundary. It is a superlist because it generally contains:

- points aligned with extremal vertices of the final hull, but that are not part of it[1],
- but also some points that were previously found to lie on the convex hull boundary, but do not any more, due to concatenation (see below).

2. The second step uses the brute-force method to scan the prehull boundary, and remove from it those points that are not part of the final list, in time proportional to the size of the boundary of the prehull (and, incidentally, that of the final upper hull boundary), i.e. $O(\log n)$.

1.1.1 The First Step.

Some notations.

We define $A = (x_0, ax_0 + b), B = (x_1 > x_0, ax_1 + b)$. AB is the segment. Set $y_0 = \lfloor ax_0 + b \rfloor$, and let $C = (x_0, y_0)$ be the grid point just below A (or equal to A when A is a grid point). We also need $D = (x_1, y_0)$.

$ABCD$ is called the "reference trapezoid" of segment AB. The trapezoid is said to be *vertical* when the angle between CD and AB is superior or equal to 45 degrees ($a \geq 1$) (see Fig. 2) and *horizontal* otherwise (see Fig. 3 and 4).

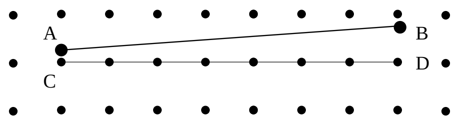

Figure 1: When $\lfloor ax_0 + b \rfloor = \lfloor ax_1 + b \rfloor$, the upper hull boundary is obviously the segment from vertex $(x_0, \lfloor ax_0 + b \rfloor)$ to vertex $(x_1, \lfloor ax_1 + b \rfloor)$.

If the segment is almost horizontal (see Fig. 1), i.e. if $\lfloor ax_0 + b \rfloor = \lfloor ax_1 + b \rfloor$, then the upper hull (and prehull) boundary is segment CD.

Otherwise, we distinguish two cases, depending on whether the reference trapezoid is vertical or horizontal, making use of the following lemma, that we give without proof:

Let A be a closed set in $\mathbb{E}^2$, and U an unimodular transformation of $\mathbb{E}^2$. Let $H()$ denote the convex hull of a subset in the plane. Then

$$U^{-1}(H(U(A))) \equiv H(A) \tag{1}$$

We recall that the restriction to $\mathbb{Z}^2$ of an unimodular transformation of $\mathbb{E}^2$ is an affine bijection of $\mathbb{Z}^2$; as a consequence, the associated matrix has integer entries and determinant ± 1. Geometrically, such transformations preserve areas, grid points, alignments, and convexity. Lemma 1 will be applied to reference trapezoids.

Constructing the convex hull of A may be done by applying the reciprocal (U^{-1}) to the convex hull of the image by U of A. The principle of the algorithm will be to reduce the original problem to one with smaller size, by means of a well-chosen unimodular transformation, constructing the upper hull boundary of the image, and then deducing the original upper convex hull by inverse image. Let us now see how we can change a vertical trapezoid into an horizontal one, and *vice versa*.

Vertical trapezoid.

Since we have assumed, w.l.o.g., that $a \geq 0$, we necessarily have $a \geq 1$ (see Fig. 2). We reduce this case to the next ($0 \leq a < 1$) by means of a unimodular transformation.

[1]If $H_1, H_2, \ldots H_n$ is a *maximal* chain of aligned points of the boundary of a convex hull, only H_1 and H_n are *extremal* vertices: a vertex on the boundary of the convex hull is either an extremity, or it is common to two adjacent segments with different slopes. Thus, for instance, the hull of the segment between point $O = (0,0)$ and $M = (1000,1000)$ only has two extremal vertices (O and M), among all the 1001 integral points that the full segment comprises.

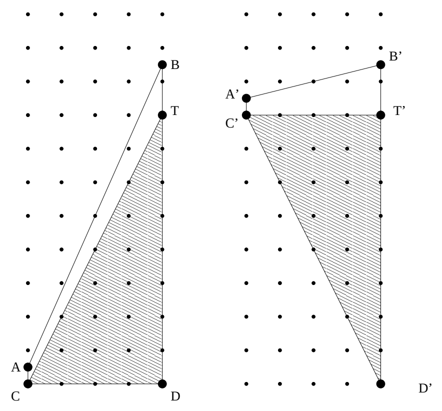

Figure 2: A vertical reference trapezoid.

Let $T = (x_T = x_1, y_T = y_0 + (x_1 - x_0)\lfloor a \rfloor)$, i.e. T is the grid point below B such that CT has slope $\sigma = \lfloor a \rfloor$.

Consider that C is the origin of our coordinate system, and consider the following transformation $U : (x, y) \rightsquigarrow (x', y')$, defined by

$$\begin{cases} x' &= x \\ y' &= y - \sigma x \end{cases} \tag{2}$$

Transformation U is obviously unimodular, leaves C invariant, and maps T to $T' \equiv D$ (see Fig. 2), and more generally vertically shifts vertical lines proportionally to their distance to the vertical line passing through C, so as to make line CT horizontal.

Since $A'B'$ has slope $(a - \lfloor a \rfloor)$ less than 1, its reference trapezoid $A'B'C'T'$ is horizontal, has smaller size than $ABCD$ (the original trapezoid for segment AB), and an upper hull boundary with similar characteristics (number of vertices) to the ones of $ABCD$, due to Lemma 1.

Hence, we only need to construct the upper prehull boundary for segment $A'B'$, and then deduce that for segment AB by applying U^{-1}. This takes us to the second case.

Horizontal trapezoid.

Here, we necessarily have $a < 1$ (see Figure 3 and 4). The initial trapezoid $ABCD$ contains a smaller vertical one, $A_2B_2C_2D_2$, with inverted axis[2], which we obtain as follows:

- B_2 is the intersection point between AB and the horizontal line with equation $y = y_C + 1$;
- A_2 is the intersection point between AB and the horizontal line with equation $y = \lfloor y_B \rfloor$.
- C_2 is the grid point just to the right of A_2 (or A_2 if A_2 is a grid point).
- D_2 follows in a straightforward fashion.

Note that the left convex prehull boundary, h_2, of the grid points located to the right of segment A_2B_2 may be computed, after some rotation and symmetry, with our method, by a recursive call. Now, the prehull boundary of AB is the concatenation of: grid point C, h_2, and grid point $K = (x_1, \lfloor ax_1 + b \rfloor)$, unless $C_2 = K$. This should be obvious from Fig. 3 and 4. As mentioned earlier, note that some points originally on h_2 may become non-extremal after concatenation with C, as in Fig. 4. This justifies the existence of a second step, to remove possibly invalid vertices.

In some cases (see Fig. 5), we can reduce an horizontal trapezoid to a vertical one not completely contained in the horizontal one. The condition is that the triangle IAC does not contain grid points other than C, where I is the intersection point between line AB and line CD. In all such cases, $B_2 \equiv I$.

[2]By this, we mean that C_2D_2 is vertical, while CD was horizontal.

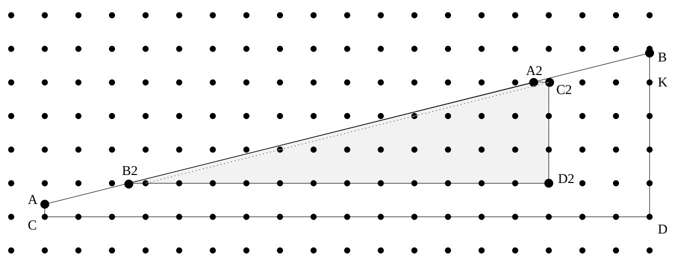

Figure 3: An horizontal reference trapezoid.

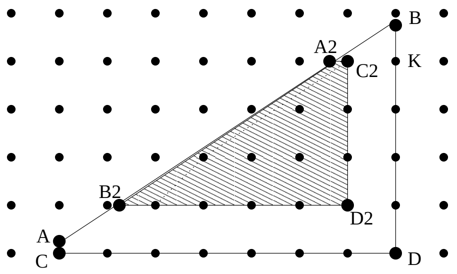

Figure 4: An horizontal reference trapezoid. Note that some vertices on the upper hull boundary of $A_2B_2C_2D_2$ do not belong to that of $ABCD$.

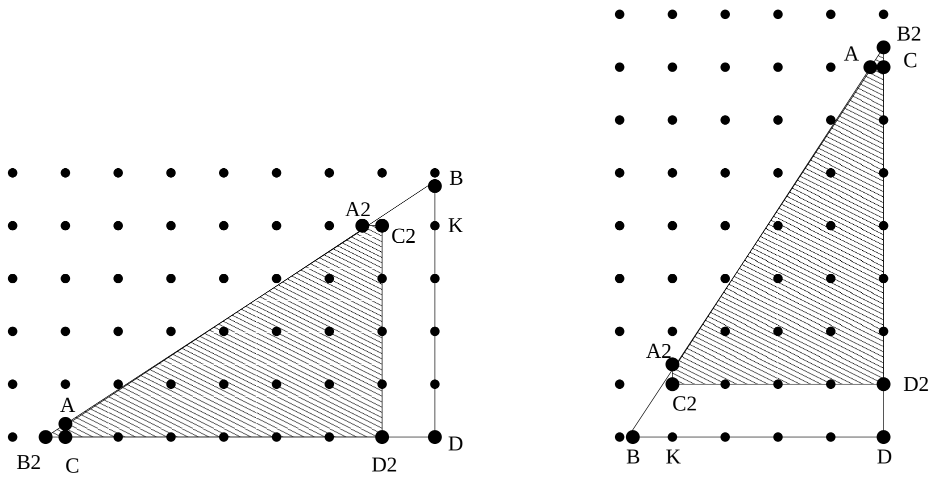

Figure 5: Refining the method.

Running time of the first step.

A vertical trapezoid with (integral) basis w and height h is reduced to an horizontal one with basis w and height $h' = h \bmod w$. An horizontal trapezoid with basis w and height h is reduced to a vertical trapezoid with basis $w' < h$ and height $h' < w$. Thus, there are at most $O(\log(\max(w,h)))$ such reductions.

Similarly, the upper prehull boundary of a vertical trapezoid has as many points as that of the horizontal trapezoid it is reduced to. The upper prehull boundary of an horizontal trapezoid has at most 2 (thus $O(1)$) vertices more than that of the vertical trapezoid it is reduced to.

In the basic case which stops recursion, the upper prehull boundary has 2 vertices. Thus the upper prehull boundary of a trapezoid with size $n = \max(w,h)$ has $O(\log n)$ vertices. Since the prehull is a superset of the hull, the latter also has $O(\log n)$ vertices.

The hull may have $\Theta(\log n)$ vertices in the worst case: as is well known, for a segment from $(0,0)$ excluded to $(q,p) \in \mathbb{Z}^2$, the vertices (x,y) of the upper hull of integral points $(0 < x \leq q, y \leq px/q)$ below the segment, and the lower hull of integral points $(0 < x \leq q, y \geq px/q)$ above the segment, are given by the successive convergents $\frac{y}{x}$ of the continued fraction expansion (CFE) of the rational slope p/q (Klein's theorem). The smallest inputs (q,p) for which there are k convergents (thus k vertices in both hulls) are the two successive Fibonacci numbers $q = F_k$ and $p = F_{k-1}$. We remind the reader that Fibonacci numbers are defined by : $F_0 = 0$, $F_1 = 1$, $F_k = F_{k-1} + F_{k-2}$, that $F_k = (\Phi^k - \Phi'^k)/\sqrt{5} = \lfloor \Phi^k/\sqrt{5} \rceil$ where $\Phi = (1+\sqrt{5})/2 \approx 1.618\ldots$ is the golden number, and $\Phi' = (1-\sqrt{5})/2 \approx -0.618\ldots$ is its conjugate, and that $k = \lfloor \log_\Phi F_k\sqrt{5} \rceil$. The notation $\lfloor n \rceil$ stands for n rounded to the nearest integer. See [GKP89] for a more detailed account.

Of course, we want the running time of the first step to be $O(\log n)$, which takes a little effort: we can reach this complexity $O(\log n)$ if the function `prehull()`, after each of its $O(\log n)$ recursive calls, only performs $O(1)$ operations. Thus the function `prehull()` is typically allowed to append or prepend two new vertices to the list resulting from the current recursive call (i.e. the upper prehull boundary h'), but not to apply any kind of linear transformation to the $O(\log n)$ vertices of h'.

A solution is to let function `prehull()` receive h' in the correct coordinate system, as a result of its recursive call. Thus the coordinate system should be passed to it as an argument (*via* some transformation matrix, say). Function `prehull()` then uses this matrix to transform the $O(1)$ vertices it adds to the current upper prehull boundary, thus only performing $O(1)$ operations in addition to its recursive call.

This method has some similarity with Euclid's algorithm ([GKP89]), and may be interestingly compared to the algorithm proposed by N. Kanamaru, T. Nishizeki and T. Asano ([KNA94]), for counting the number (or giving the list) of all the grid points inside a given triangle.

1.1.2 The Second Step.

Here, the upper prehull boundary is a list of grid points, ordered by increasing abscissæ. This step removes those grid points it contains that are not vertices of the upper hull, using a technique akin to the well-known *Graham scan* ([PS85]) for constructing the convex hull of a set of points in the plane, arranged in some compatible order:

```
Scan the list:
  Let $q$ be the current point, $p$ its predecessor, and $r$ its successor in the list.
  If $pqr$ forms a strictly convex angle:
    Step forward:
      Set $q \leftarrow r$ (and update $p$ and $r$).
  Otherwise:
    Remove $q$ from the list, and
    Step back:
      Set $q \leftarrow p$ (and update $p$ and $r$).
```

This simplified presentation ignores special cases occurring at the head or at the tail of the list, but it is enough to assume pqr to be declared convex when either p or r does not exist.

The method runs in time linear in the size of the list, k: there are at most $O(k)$ back steps, since each one of them removes an item from the original list; and the number of forward steps cannot exceed k plus the number of back steps.

2 Upper Convex Hull of Grid Points below a Convex Curve

2.1 The Problem

This section describes a method for computing the upper convex hull boundary of all the grid points below a given finite, convex and bi-monotonic arc of a quadric (parabola, hyperbola, ellipse), including those possibly *on* it. Bi-monotonic arcs have at most one intersection point with any line parallel to either of the two prescribed perpendicular directions (say Ox and Oy). From now on, we shall only consider arcs which are monotonic with respect to Ox and Oy: all quadrics may be partitionned in such arcs in constant time.

Arcs are defined by the integral-coefficient equation of the quadric, $F(x, y) = ax^2 + by^2 + cxy + dx + ey + f = 0$, and by the linear inequalities which their points must satisfy. For instance, the maximal arc of quadric where points have normal oriented towards the north-east quadrant carries inequalities: $F'_x \leq 0$, and $F'_y \geq 0$ where $F'_x(x, y) = 2ax + cy + d$ and $F'_y(x, y) = 2by + cx + e$ are the derivatives of F in x and y, respectively.

Note that, in this formulation, the (possibly non rational) coordinates of arcs endpoints need neither be explicitly (i.e. with numbers), nor exactly represented.

2.2 Constructing the Upper Hull Boundary

We suppose that a method to compute intersection points between the arc at hand and any ray is available. A *ray* is a half straight line, defined by an origin (x_0, y_0), and a direction vector (u, v); in our setting, x_0, y_0, u and v are integers (incidentally, the last two are supposed to be coprimes). Points (x, y) on the ray can be parameterized by t: $x = x_0 + ut$, $y = y_0 + vt$. Substituting x and y in the quadric's equation $F(x, y) = 0$, we get a quadratic equation $E(t) = 0$ with integral coefficients, which is (almost always, see Section 2.4) trivially solved. Only intersection points which satisfy the arc's inequalities are kept.

To compute the upper hull, we walk along the arc, starting from a known point S of the hull, say its leftmost point. The principle is to maintain a basis (I, J) of two vectors (see Fig. 6) such that:

- $I = (I_x, I_y)$ and $J = (J_x, J_y)$ have integral coordinates,
- their determinant is equal to $+1$:

  $$\begin{vmatrix} I_x & I_y \\ J_x & J_y \end{vmatrix} = +1$$

  This *unimodularity* condition guarantees that all grid points have integer coordinates relatively to the coordinates system (S, I, J), and conversely, all points with integral coordinates relatively to (S, I, J) are grid points. Moreover, the segment $[S, S + J[$ is cut by the curve, and will always be as we follow the curve.
- The last constraint on (S, I, J) is that the next vertex (T, to be computed) after S on the convex hull boundary has non-negative integral coordinates relatively to (S, I, J).

For instance, if the arc to be followed has normal oriented towards North-East, then the initial values for I and J are $I_0 = (1, 0)$ and $J_0 = (0, 1)$; other cases are found by symmetry.

The method computes the next point T after S on the convex hull; then it jumps to T, i.e. T becomes the new origin S, and the basis (I, J) is reset to (I_0, J_0) – a constant for a given arc.

It remains to explain how the basic step, `NextPoint`(arc, S, I, J) computes T. We shall use the following notations: if P is a point and V a vector, $P + V$ is the point Q such that $\overrightarrow{PQ} = V$; `Ray`(P, U) is the ray with origin P and direction U.

There are several possible configurations for the arc S, I, and J, as discussed below. First note that, for the sake of clarity, figures use an orthonormal basis (I, J), though actually only the initial basis (I_0, J_0) is orthonormal, (with the exception of Fig. 6 where we have represented the process "as is", for a comparison). This convention is justified by the fact that the transformation applied to (I,J) to get "orthonormal figures" is unimodular, i.e. it preserves area, alignments and grid points.

2.2.1 Case 1.

In case 1 (see Fig. 7, 8, and 9), the arc does not cross ray $\mathtt{Ray}(S + J,\ I)$, or crosses it twice (possibly tangentially) inside the interval $]S + J, S + J + I[$.

Then, if the arc crosses $\mathtt{Ray}(S + I,\ I)$ at point P, we are in case 1.1 (Fig. 7); T is known and it is the grid point just before P, i.e. if P has coordinates $(P_x, P_y = 0)$ in coordinate system (S, I, J), then T has coordinates $(T_x = \lfloor P_x \rfloor, T_y = 0)$; this should be obvious on Fig. 7.

Otherwise, the arc does not cross $\mathtt{Ray}(S + I,\ I)$. We are either in case 1.2 or 1.3.

We are in case 1.2 (see Fig. 8) if we have reached the arc's end: we clip Ray$(S,\ I)$ by all linear inequalities which are part of the arc's definition. Let P be the right endpoint of the clipped ray. Then T has coordinates $(T_x = \lfloor P_x \rfloor, T_y = 0)$; this should be obvious on Fig. 8.

We are in case 1.3 (see Fig. 9) if the arc is tangent to an asymptote parallel to I, and if we accept as input arcs with infinite length; here T is the point at infinity $(+\infty, 0)$. This kind of case occurs for instance with the hyperbola: $0 < x,\ 0 < y,\ xy - 1 \geq 0$. Of course, there is the risk that the upper convex hull boundary of infinite arcs has an infinite number of vertices, and in such a case, the method never stops! Thus, we mention case 1.3 only for completeness: it is *a priori* impossible.

Thus, in case 1, the method terminates and returns T.

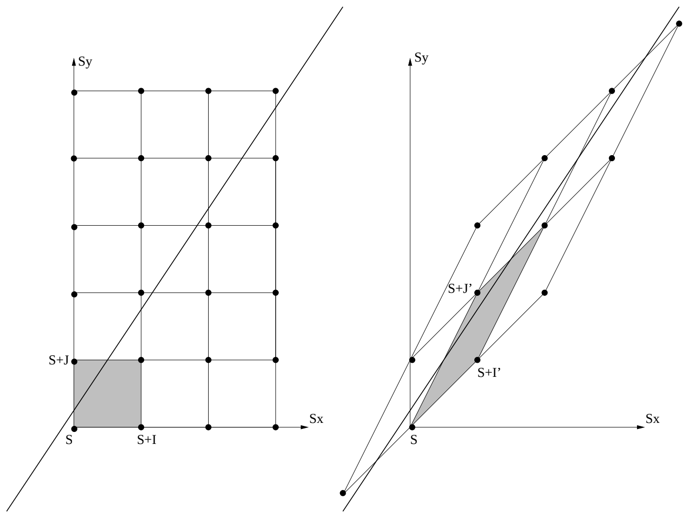

Figure 6: On the left handside, basis (I, J) is the initial one (I_0, J_0). We are in the second case (as defined below), and the new basis (right handside diagram) is $I' = I + J, J' = I + 2J$. In both diagrams, we have drawn the lattice generated by translating the elementary unit "cell" (shadowed). The set of all vertices in each lattice coincides with $\mathbb{Z}^2$, thanks to unimodularity.

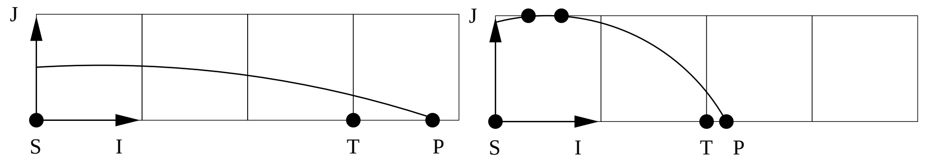

Figure 7: Case 1.1.

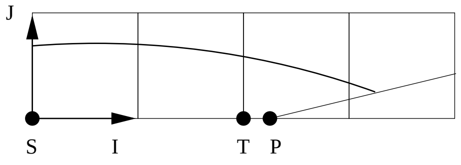

Figure 8: Case 1.2.

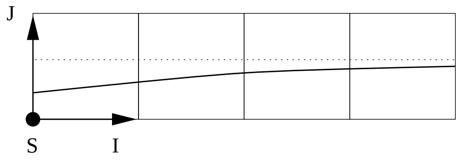

Figure 9: Case 1.3.

2.2.2 Case 2.

In case 2 (see Fig. 10), the arc crosses $\texttt{Ray}(S+J,I)$ once, and the intersection point belongs to $[S+J, S+J+I[$. Due to monotonicity, either the arc cuts $\texttt{Ray}(S+I,\ J)$ in one point P and we are in case 2.1, or it terminates before and we are in case 2.2. A case similar to case 1.3, with an asymptote parallel to J, is impossible.

In case 2.1, P exists and has coordinates $(P_x = 1, P_y)$ in (S, I, J). Let $b = \lfloor P_y \rfloor$, $I' = I + bJ$, $J' = I' + J$. Note that (S, I', J') fulfills the prescribed requirements (unimodularity, etc). Then we recursively compute $T = \texttt{NextPoint}(\text{arc},\ S,\ I',\ J')$.

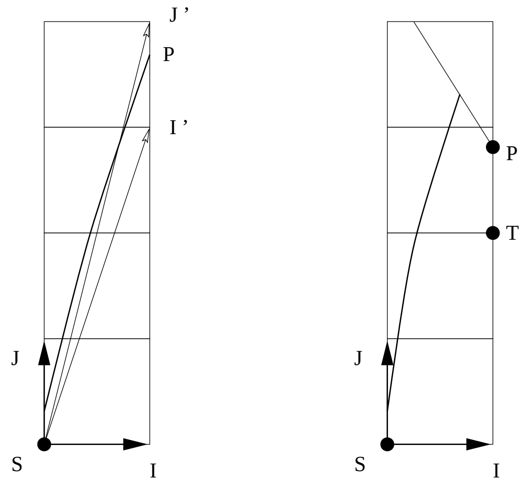

Figure 10: Case 2. Left: case 2.1. Right: case 2.2.

In case 2.2, $\texttt{Ray}(S+I,J)$ is clipped by all linear inequalities of the arc's definition. Let P be the highest endpoint of the clipped ray. Then T has coordinates $(T_x = 1, T_y = \lfloor P_y \rfloor)$.

2.2.3 Case 3.

In case 3 (see Fig. 11), the arc crosses $\texttt{Ray}(S+J,I)$ at least once, but the intersection point closest to $S+J$ (call it $P = (P_x, P_y = 1)$) does not belong to $[S+J, S+J+I[$. Then let $a = \lceil P_x \rceil$, $I' = aI + J$, $J' = I' - I$; again (S, I', J') fulfills the prescribed requirements (unimodularity, etc), and we compute recursively $T = \texttt{NextPoint}(\text{arc}, S,\ I',\ J')$.

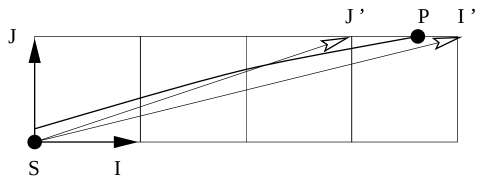

Figure 11: Case 3.

2.3 Justification and Running Time

That we obtain the upper convex hull in the fashion described above is inherently justified by the fact that the curve arc is convex and we always select the *next* local integral maximum under the arc.

Indeed, method `NextPoint`(arc, S, I, J) basically mimics the *Continued Fraction Expansion* of the rational slope $\frac{p}{q}$ of segment ST, and hence selects the best possible approximations in increasing order. More precisely, method `NextPoint` starts with basis I_0, J_0, and, in the worst case, either transforms I_n, J_n into

$$\left\{ \begin{array}{lcl} I_{n+1} & = & I_n + J_n \\ J_{n+1} & = & I_n + 2J_n \end{array} \right. \tag{3}$$

in case 2, or into

$$\left\{ \begin{array}{lcl} I_{n+1} & = & 2I_n + J_n \\ J_{n+1} & = & I_n + J_n \end{array} \right. \tag{4}$$

in case 3 (in all other cases, it terminates). At step k, $I_k = (x_k \; y_k)$ and $J_k = (x_k \; y_k)$ have coordinates x_k, y_k, x'_k, y'_k superior or equal to 2^{k-1} (proof by recurrence). After a logarithmic number of steps, I_k length becomes greater than the arc length, and method `NextPoint` terminates.

This implies that method `NextPoint` performs $O(\log(\max(p,q)))$ operations; here the most expensive operation is the resolution of a quadratic equation with integral coefficients. It is usually assumed (although debatable) that such an operation takes constant time. p and q are bounded by the length (say n) of the minimal rectangle enclosing the arc. Thus `NextPoint` runs in $O(\log n)$, and is called $O(k)$ times if there are k vertices on the hull boundary.

Hence, the method runs in $O(k \log n)$ time. It may be used on a straight line segment: in this case, $k = O(\log n)$, and it runs in $O(k^2)$ time, compared to $O(k)$ for the method of the previous section, designed for straight lines only.

The running time $O(k \log n)$ may perhaps be improved:

- First, the previous running time analysis was a bit pessimistic to remain simple. More optimistically, if $(\frac{b_i}{a_i})$ is the sequence of slopes along the hull, then the number of operations is
$$\sum_{i=1}^{k-1} \log(\max(a_i, b_i)).$$
- Then the method itself may perhaps be improved: at each vertex, it starts with basis I, J equal to I_0, J_0, the same initial basis for all arcs; we do not use the fact that the slope of $S_k S_{k+1}$ is likely to be close to that of $S_{k-1} S_k$, i.e. the first reduced fractions of these two slopes are equal. For the time being, we have not investigated such tracks.

2.4 Accuracy Problems

A straightforward implementation using floating-point arithmetic faces difficulties due to inaccuracy. Typically, the program will loop at some hull vertices: solving a second degree equation is not so trivial!...

Basically, we need to solve quadratic equations $E(t) = 0$, with integral coefficients, in such a way that if a root is an integer, we must compute it exactly, but otherwise, it is enough to know that it belongs to some interval $]z, z+1[$ with $z \in \mathbb{Z}$. This is easily done.

Integers on 32 bits may be not sufficient: a *BigInteger* library should be used to prevent overflows (provoked, among other things, by the repeated application of unimodular transformations). The bad news is that basic operations no longer take constant time. The good news is that an integer arithmetic is sufficient: no (very time consuming) quadratic arithmetic (providing the exact square root together with the usual exact operations $+$, $-$, $\times$, $/$, and comparisons) is required.

2.5 Number of Vertices

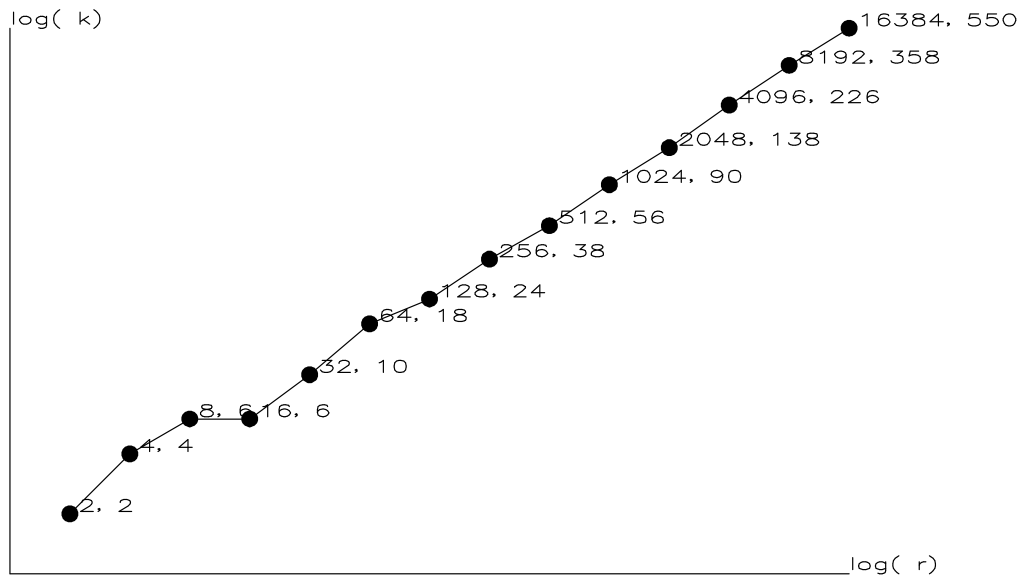

Figure 12: This diagram shows the number k of vertices in upper hulls of grid points below quarter circles with various radii $r = 2, 4, 8, 16 \ldots 16384$. Scales are logarithmic.

Our method is $O(k \log n)$, the brute force one $O(n)$: clearly our method is of interest only if $k \log n < n \Leftrightarrow k < \frac{n}{\log n}$. Thus the question arises to bound k in function of n.

We are still working on this subject. This section only gives measures of k for quarter circles with variable radius r. Figure 12 shows a log-log diagram with k in ordinate and r in abscissa. Circles have equation: $x^2 + y^2 \leq r^2$. Tested radii are: $r = 2, 4, 8 \ldots 2^{14}$. The arc is limited by: $x \geq 0, y \geq 0$. For r large enough, say $r \geq 128$, points $(r, k(r))$ are roughly aligned, so that $k \approx 0.86 r^{0.669}$. Of course, it is an empirical statement, not a theorem.

Asymptotically, $O(n^{0.669})$ is smaller than $O(\frac{n}{\log n})$, so for large enough radius values, our walking method will perform faster than the brute force one.

2.6 Extension to other Curves

As a matter of fact, our solution applies to any convex curve arc, provided a method to compute the intersection points between the arc and any ray $(x(t), y(t))$ is available. In the case of an algebraic curve of degree d with integral coefficients, computing such an intersection reduces to finding the roots of a polynomial in t with degree d. As is already the case for quadrics, it suffices to know when roots are integral, and otherwise to detect an isolating interval not containing integers.

The difficulty is rather in characterizing arcs. A possibility is to partition the curve into arcs with Collins's cylindric algebraic decomposition algorithm. However, the preprocessing step is no longer constant, as it depends on the algebraic degree of the curve.

We have not investigated quadrics with algebraic but not integral coefficients; however, such quadrics are subsets of algebraic curves with integral coefficients and higher degrees, that elimination methods (e.g.resultants or Gröbner bases) enable to compute; for instance, the line with equation $x - \sqrt{2}y = 0$ is a subset of $x^2 - 2y^2 = 0$, a quadric with only integral coefficients; thus its upper convex hull boundaries may be computed with our method for quadrics, using only integer arithmetic; they can also be computed with the method for line segments, but some arithmetic in the quadratic field $\mathbb{Q}[\sqrt{2}]$ is required. We won't go into such subtleties, due to lack of space.

3 Conclusion

This paper has presented methods for computing upper convex hulls of grid points below a segment or an arc. We are currently investigating the possibility of improving these methods, by exploiting geometric properties of such hulls, like the presence of periodic patterns (i.e. automorphisms). This question is related to Number Theory, especially to quadratic number fields, their units and the Pell-Fermat equation.

Aknowledgements

The authors wish to thank Jean-Michel Muller and Vincent Lefèvre, from LIP, ENS Lyon, for having mentioned such problems to them. They also wish to thank an anonymous referee for suggestions that helped improve the original version.

References

[GKP89] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: a Fondation for Computer Science*. Addison-Wesley Publishing Company, 1989.

[KNA94] N. Kanamaru, T. Nishizeki, and T. Asano. Efficient Enumeration of Grid Points in a Convex Polygon and its Application to Integer Programming. *International Journal of Computational Geometry and Applications*, 4(1):69–85, 1994.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry – An Introduction*. Springer-Verlag, New York, N.Y., 1985.

[RY95] J-P Reveillès and G. Yaacoub. A Sublinear 3d Convex Hull Algorithm for Lattices. In *Actes du 5 ième colloque DGCI, Clermont-Ferrand, France*, pages 219–230, 1995.