Computing the Characteristics of a SubSegment of a Digital Straight Line in Logarithmic Time

Mouhammad Said[1,2] and Jacques-Olivier Lachaud[1]

[1] Laboratoire de Mathématiques, UMR 5127 CNRS, Université de Savoie, 73376 Le Bourget du Lac, France
{mouhammad.said,jacques-olivier.lachaud}@univ-savoie.fr
[2] LAIC, Univ. Clermont-Ferrand, IUT, Campus des Cézeaux, 63172 Aubière Cedex, France

Abstract. We address the problem of computing the exact characteristics of any subsegment of a digital straight line with known characteristics. We present a new algorithm that solves this problem, whose correctness is proved. Its principle is to climb the Stern-Brocot tree of fraction in a bottom-up way. Its worst-time complexity is proportionnal to the difference of depth of the slope of the input line and the slope of the output segment. It is thus logarithmic in the coefficients of the input slope. We have tested the effectiveness of this algorithm by computing a multiscale representation of a digital shape, based only on a digital straight segment decomposition of its boundary.

Keywords: standard lines, digital straight segment recognition, Stern-Brocot tree.

1 Introduction

Digital Straight Lines (DSL) and Digital Straight Segments (DSS) are useful to describe the geometry of a digital shape (coding, discrete geometric estimators, feature detection) and this explains why they have been so deeply studied (see the survey [9] or [8]). When a straight line is digitized on a grid, we obtain a sequence of grid points defining a digital straight line segment. Methods of recognizing digital straight segments are known since long. In one of the first methods, Freeman [7] suggested to analyze the regularity in the pattern of the directions in the chain code [6] of a digital curve. Anderson and Kim [1] have presented a deep analysis of the properties of DSS and suggested a different algorithm based on calculating the convex hull of the points of digital curves to be analyzed. In [10], Kovalevsky presented a *new classification of digital curves* into boundary curves and visual curves. Boundary curves and lines are a useful mean for fast drawing of regions defined by their boundaries. Modern DSS recognition algorithms achieves a computational complexity of $O(n)$, if n is the number of input points and with a computing model where arithmetic operations takes $O(1)$ time. Interestingly, this complexity is obtained by algorithms based on arithmetic properties [12,11], combinatorial properties [16], or dual-space construction [5]. These algorithms are optimal, when no further information is known. However, in some case, we may already know that a set of points is included in some DSL of known characteristics. This happens for instance when computing the multiresolution of a digital object, since analytic formulas give the multiresolution of DSL. We assume that the digital shape was previously polygonalized as a sequence of M DSS, for instance by a simple greedy DSS segmentation algorithm or with the Minimum Perimeter Polygon [13]. We then calculate the characteristics of each DSS when changing the resolution of the grid (see [14] for more details about the covering of a DSL). In this case, a faster computation of the DSS parameters is possible.

Many works deal with the relations between irreducible rational fractions and digital lines (see [4] for characterization with Farey series, and [18] for a link with decomposition into continuous fractions). In [2], Debled and Réveillès first introduced the link between the Stern-Brocot tree and the recognition of digital line. Recognizing a piece of digital line is like going down the Stern-Brocot tree up to the directional vector of the line. To sum up, the classical online DSS recognition algorithm **DR95** [2] (also reported in [8]) updates the DSS slope when adding a point that is just exterior to the current line (weak exterior points). In [17], De Vieilleville and Lachaud have revisited a classical arithmetically-based DSS recognition algorithm with new parameters related to a combinatorial representation of DSS. New analytic relations have been established and the relation with the Stern-Brocot tree has been made explicit.

The main contribution of this paper is to present a fast algorithm which computes the exact (minimal) characteristics of a DSS that is some subset of a DSL of known characteristics (see [15] for more details about minimal characteristics). More precisely, the input DSL, say D, is given as the continued fraction of its slope. The DSS is specified by the positions of its two endpoints A and B. Furthermore, the two lower leaning points of D surrounding A and B are given as input. This new algorithm, called `ReversedSmartDSS`[1], determines the characteristics of the DSS by moving in a bottom-up way along the Stern-Brocot tree, a famous tree structure that represents positive fractions. We prove the correctness of this algorithm in Proposition 1. We further show in Proposition 2 that its worst-case computational complexity is $\Theta(k-k')$, where $[u_0; u_1, \ldots, u_k]$ is the continued fraction of the slope of the input DSL and $[u_0; u_1, \ldots, u_{k'}]$ is the continued fraction of the slope of the output DSS. This result assumes a computing model where standard arithmetic operations are in $O(1)$, which is a reasonnable assumption when analyzing digital shapes.

This complexity is first to compare with the `SmartDSS` algorithm [14], whose worst-case computational complexity is $\Theta(\sum_{i=0}^{k'} u_i * \delta)$, where δ is the number of patterns of the output DSS. The average of this sum for all fractions $\frac{a}{b}$ with $a+b=n$ is experimentally lower than $\log^2 n$, and this sum is upper bounded by n and lower bounded by 1. This complexity is secondly to compare with the complexities of classical DSS recognition algorithms (e.g., DR95 [3], see

[1] This name is in opposition to the SmartDSS algorithm [14], because it moves along the Stern-Brocot in a reversed direction.

also [9]), whose complexities are at best $\Theta(n)$. The `ReversedSmartDSS` algorithm was implemented and tested and various experimental results show that this algorithm performs better than the `SmartDSS` algorithm and the classical DSS recognition algorithms (see Table 1).

Table 1. Computation times of the (h, v)-covering of various digital shapes with our proposed approach. The digital shapes are: a circle of radius 2000; a flower with 5 petals, mean radius 5000 and variability of radius 7000; a polygon with 8 sides and radius 2000. The symbol # stands for "number of".

Shape	Flower			Circle			Polygon		
# points	67494			16004			15356		
# segments	1991			574			44		
h, v	2	4	10	2	4	10	2	4	10
# points (h, v)	33744	16870	6750	8000	4000	1600	7676	3840	1532
Smart DSS									
# points tested	19352	11254	4367	5413	2977	1019	782	667	527
timings (ms)	3.1286	2.6446	2.2914	0.997	0.8902	0.7618	0.1258	0.1142	0.0946
Reversed Smart DSS									
timings (ms)	2.364	2.103	2.078	0.758	0.702	0.625	0.104	0.097	0.084

In Section 2 we describe this new algorithm. We show the correctness of this new algorithm in Section 3, as well as its computational complexity. The last section concludes and presents future works.

2 A Coarsening Algorithm for Computing the Characteristics of a Subsegment Included in a Known DSL

We recall first that a standard *digital straight line* D in the fourth quadrant is some subset of the digital plane $\{(x, y) \in \mathbb{Z}^2, \nu \leq \alpha x + \beta y < \nu + \alpha + \beta\}$, where $(\alpha, \beta, \nu) \in \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}$ form the *characteristics* of D. The fraction $\frac{\alpha}{\beta}$ is the *slope* of the DSL. Any 4-connected piece of a DSL is a *digital straight segment* (DSS). The *characteristics* of a DSS are the characteristics of the "simplest" DSL covering it (the one with the smallest possible α). Lastly, a *pattern* (resp. *reversed pattern*) is the Freeman chaincode joining two consecutive lower leaning points (resp. upper leaning points). Patterns are a combinatorial characterization of DSL.

In the next section, we describe our new algorithm for determining the characteristics of any subsegment S of a digital straight line D in the fourth quadrant, whose characteristics (α, β, ν) are known. We make use of the property that the slope of S is either $\frac{\alpha}{\beta}$ or any one of the ancestors of $\frac{\alpha}{\beta}$ in the Stern-Brocot tree ([15], see also Proposition 3 of [14]). The principle of our new algorithm is to follow a bottom-up way in the Stern-Brocot tree.

2.1 Overview of the Algorithm

Algorithm 1 is the general algorithm for computing the exact characteristics of S knowing the characteristics (α, β, μ) of its covering line D. This algorithm

```
Function ReversedSmartDSS ( In $D$ : $DSL(\alpha, \beta)$, In $L_1, L_2$ : Points of $\mathbb{Z}^2$, In $A, B$ : Points
of $\mathbb{Z}^2$ ) : DSS $(a, b, \mu)$;
Var $Lp_1, Lp_2$ : Point of $\mathbb{Z}^2$;
Var $dL$ : integer /* The horizontal distance between $L_1$ and $L_2$ */;
Var $S$ : slope;
begin
1       if $(A_x == B_x)$ then return $(1, 0, A_x)$;
2       if $(A_y == B_y)$ then return $(0, 1, A_y)$;
        $dL \leftarrow L_{2_x} - L_{1_x}$;
3       if $(dL \geq 3\beta)$ or $(dL == 2\beta$ and $(A == L_1$ or $B == L_2))$ or $(A == L_1$
            and $B == L_2)$ then return $(\alpha, \beta, \alpha L_{1_x} + \beta L_{1_y})$ ;
        /* $S$ is included in two patterns of $D$ */;
4       if $(dL == 2\beta)$ then return DSSWithinTwoPatterns $(D, L_1, L_2, A, B)$;
        /* $S$ is included in one pattern of $D$ */;
5       $(D(\alpha', \beta'), Lp_1, Lp_2) \leftarrow$ NewLowerBound$(D, L_1, L_2, A, B)$;
6       if $(Lp_1 == L_1)$ and $(Lp_2 == L_2)$ then
            return FinalSlope$(D, L_1, L_2, Lp_1, Lp_2, A, B)$;
7       return ReversedSmartDSS$(DSL(\alpha', \beta'), Lp_1, Lp_2, A, B)$;
end
```

Algorithm 1. Main algorithm. Computes the characteristics (a, b, μ) of a DSS S that is some subset of a DSL D of slope $\frac{\alpha}{\beta}$ and lower leaning points L_1 and L_2 (the ones surrounding the segment AB). The DSS is defined by a starting point A and an ending point B $(A, B \in D)$.

thus computes the simplest DSL covering S. The segment S is defined by its two endpoints A and B. Lastly, we give also as input the two lower leaning points of D which surround A and B. Note that these input data are all known if the DSL D was recognized by a classical recognition algorithm (e.g., DR95 [3]).

If A and B have the same abscissa (or same ordinate), then this algorithm stops and return $(1, 0, A_x)$ (or $(0, 1, A_y)$), which are the obvious results. Otherwise, this algorithm then checks the horizontal distance between L_1 and L_2 to measure the number of patterns covering the segment. Should this distance be: (1) three times greater than β, (2) equal to 2β, A and L_1 are superposed, or B and L_2 are superposed, or (3) A and L_1 are superposed, and B and L_2 are superposed, then the algorithm stops and returns (α, β) (line 3). Indeed, in these case, the DSS contains at least one pattern, so the characteristics follow.

Otherwise, if this horizontal distance is equal to 2β — S is included in two patterns of D — then the characteristics are computed by the function `DSSWithinTwoPatterns` (line 4), described by Algorithm 2.

2.2 S Is Included in Two Patterns of D

The function `DSSWithinTwoPatterns` (Algorithm 2) is really the core of this DSS recognition method. In its loop, three different patterns are tested progressively, so as to find the first that has exactly the sought slope. It is easy to see that, since the segment is included in two patterns, the middle lower leaning point L_m is the lowest point of the segment. Therefore, if the slope of the segment is defined by a pattern (i.e. defined by two lower leaning point), then one of the extremities of the pattern is this point L_m. We thus test in sequence the possible patterns to the left and to the right of L_m. However, the slope of a DSS may also be defined by a reversed pattern (i.e. defined by two upper leaning points). We thus test also in sequence the possible reversed patterns.

The progressive computation of these three sequences of patterns (left pattern, right pattern, reversed pattern) is done in a parallel manner. More precisely, they are run consecutively one step at each time:

- line 4 with a call to `LowerSlope`(…, true) computes the next left patterns and stops when the lower leaning point L_1 overtakes or reaches A,
- line 5 with a call to `LowerSlope`(…, false) computes the next right patterns and stops when the lower leaning point L_2 overtakes or reaches B,
- line 6 with a call to `UpperSlope` computes the next reversed pattern and stops when the upper point RU overtakes towards the left or reaches B and the upper point LU overtakes towards the right or reaches A.

The conditions at lines 3, 7 and 8 corresponds to three possible cases where respectively no, one, or two pattern computation(s) is/are stopped. At each iteration, the pattern, reversed or not, with deepest slope is the candidate solution. Various tests of the positions of the current leaning points with respect to the segment AB allow to determine if this candidate is valid. If it is true, the function exits and returns the characteristics of the elected pattern. If it is false, the function loops and computes the new three patterns. This function loops at most k times, where k is the depth of the input slope.

Algorithm 3 computes in $O(1)$ the characteristics of the lower bound in the left (or right according to the boolean variable $Left$) of L_m. In this algorithm, we fix L_m, move L_1 right towards A (or L_2 left towards B in the other case) and calculate the value of the new slope between the new L_1 and L_m (or L_m and the new L_2).

Algorithm 4 determines in $O(1)$ the positions of the new upper leaning points LU and RU, according to the parity of slope (line 1,3) (see [17] for more details about the parity of the slope). We have moved LU and RU either towards the left in the odd case or towards the right in the even case. But before moving, we must calculate the number of subpatterns, that is covering the point A in the even case (line 2) or B in the odd case (line 4).

Example. Let us look at a run of Algorithm 1 for the DSL D of slope $13/18 = [0, 1, 2, 1, 1, 2]$ (Fig. 1), for the subsegment AB. Here the segment S is included in two patterns 13/18. Since the condition on line 3 of Algorithm 1 is fulfilled, we call `DSSWithinTwoPatterns` (Algorithm 2) to compute the characteristics (a, b, μ) of S.

For the first step, left pattern has slope 3/4 (call `LowerSlope`, see Fig. 2,a), right pattern has slope 5/7 (call `LowerSlope`, see Fig. 2,c), and reversed pattern has slope 5/7 (call `UpperSlope`, see Fig. 2,b). As Algorithm 4 is stopped, we thus compute the deepest slope of 3/4, 5/7 and 5/7, which is 5/7. Since the deepest slope coincides with the slope returned by the stopped algorithm, Algorithm 2 stops and returns this slope (final result $(5, 7, 6)$).

Let us now give some explanations of (Fig. 2,a). In the first step, we fix L_{11} at L_1 and L_{22} at L_m, and we compute the previous slope $PS = 5/7$ of $S = 13/18$. Since the parity of S is odd, then we calculate the number k of subpatterns

```
Function DSSWithinTwoPatterns( In $D$ : DSL $(\alpha,\beta)$, In $L_1, L_2$ : Lower
bounds of $D$, In $A, B$ : Point of $\mathbb{Z}^2$) : DSS $(a,b,\mu)$;
Var $U_1, U_2, L_m$ : Point of $\mathbb{Z}^2$; L, R, U : boolean;
Var $S_1, S_2, S_3, DS$ : slope;
Var $D_L, D_R, D_U$ : DSL $(\alpha,\beta)$, CF : Continued Fraction of $D$;
begin
    $L$ ←true, $R$ ←true, $U$ ←true, i ←0 ;
1   $L_m$ ←MiddleLowerBound($L_1, D$), $CF$ ←ContinuedFraction($D$);
2   $U_1$ ←FirstUpperBound($D, L_1, L_2$), $U_2$ ←SecondUpperBound($U_1, D$);
    while $i < |CF|$ do
3       if ($L$ and $R$ and $U$) then
4           $S_1$ ←LowerSlope($D_L, L_1, L_m, L_2, A, true$) /* Left Lower Slope */;
5           $S_2$ ←LowerSlope($D_R, L_1, L_m, L_2, B, false$) /* Right Lower Slope */;
6           $S_3$ ←UpperSlope($D_U, U_1, U_2, A, B$) /* Upper Slope */;
            if ($L_1 >= A$ or $L_2 <= B$ or ($U_1 >= A$ and $U_2 <= B$)) then
                $DS$ ←DeepestSlope( $S_1$ , $S_2$ , $S_3$ )/* DS The deepest slope */;
            if $L_1 >= A$ then $L$ ←false;
            if $L_2 <= B$ then $R$ ←false;
            if ($U_1 >= A$ and $U_2 <= B$) then $U$ ←false;
            if ($L_1 >= A$ and $DS == S_1$) or ($L_2 <= B$ and $DS == S_2$) or
            ($U_1 >= A$ and $U_2 <= B$ and $DS == S_3$) then break;
7       else if (($L$ and $R$) or ($L$ and $U$) or ($R$ and $U$)) then
            $S_1$ ←($R$ and $U$) ? LowerSlope($D_R, L_1, L_m, L_2, B, false$)
                                : LowerSlope($D_L, L_1, L_m, L_2, A, true$) ;
            $S_2$ ←($L$ and $R$) ? LowerSlope($D_R, L_1, L_m, L_2, B, false$)
                                : UpperSlope($D_U, U_1, U_2, A, B$);
            if (((($L$ and $U$) or ($L$ and $R$)) and ($L_1 >= A$)) or ((($R$ and $U$) or
            ($L$ and $R$)) and ($L_2 <= B$)) or ((($R$ and $U$) or ($L$ and $U$)) and
            ($U_1 >= A$ and $U_2 <= B$))) then $DS$ ←DeepestSlope( $S_1$ , $S_2$ );
            if ((($R$ and $U$) and (($L_2 <= B$ and $DS == S_1$) or ($U_1 >= A$ and
            $U_2 <= B$ and $DS == S_2$))) or (($L$ and $U$) and (($L_1 >= A$ and
            $DS == S_1$) or ($U_1 >= A$ and $U_2 <= B$ and $DS == S_2$))) or (($L$
            and $R$) and (($L_1 >= A$ and $DS == S_1$) or ($L_2 <= B$ and
            $DS == S_2$)))) then break;
            if ((($L$ and $U$) or ($L$ and $R$)) and ($L_1 >= A$)) then $L$ ←false;
            if ((($R$ and $U$) or ($L$ and $R$)) and ($L_2 <= B$)) then $R$ ←false;
            if ((($R$ and $U$) or ($L$ and $U$)) and ($U_1 >= A$ and $U_2 <= B$)) then
                $U$ ←false;
8       else
            if $L$ then $DS$ ←LowerSlope($D_L, L_1, L_m, L_2, A, true$);
            else if $R$ then $DS$ ←LowerSlope($D_R, L_1, L_m, L_2, B, false$);
            else $DS$ ←UpperSlope($D_U, U_1, U_2, A, B$);
            if (($L$ and $L_1 >= A$) or ($R$ and $L_2 <= B$) or ($U$ and $U_1 >= A$ and
            $U_2 <= B$)) then break;
    $a$ ←$DS_x$, $b$ ←$DS_y$, $\mu$ ←$aL_{m_x} + bL_{m_y}$;
    return $(a,b,\mu)$;
end
```

Algorithm 2. Computes the characteristics (a,b,μ) of a DSS S that is some subset of a DSL $D(\alpha,\beta)$ repeated twice

```
Function LowerSlope( In $D$ : DSL $(\alpha, \beta)$, InOut $L_1, L_m, L_2$ : Lower bounds of $D$, In $X(A$ or $B)$ : Point of $\mathbb{Z}^2$, In Left : Boolean): DSS $(a, b)$;
Var $P, L_{11}, L_{22}$ : Points of $\mathbb{Z}^2$ /* $L_{11}$ and $L_{22}$ two lower leaning points */;
Var $PS$ : slope;
Var $k$ : integer;
Var $parity$ : boolean;
begin
    $(P, L_{11}, L_{22}) \leftarrow$Left ? $(L_1, L_1, L_m)$ : $(L_2, L_m, L_2)$ ;
    $parity \leftarrow$Parity($D$);
    $PS \leftarrow$PreviousSlope($D$);
    if ($parity$ is odd) then
        $k \leftarrow$NumberOfCoveringSubPatterns($L_{11}, X, PS, true, false$);
        $P \leftarrow L_{11} - k(-PS_y, PS_x)$;
    else
        $k \leftarrow$NumberOfCoveringSubPatterns($L_{22}, X, PS, true, false$);
        $P \leftarrow L_{22} - k(PS_y, -PS_x)$;
    if $Left$ then $L_{11} = P$;
    else $L_{22} = P$;
    $(a, b) \leftarrow (|P_y - L_{m_y}|, |L_{m_x} - P_x|)$;
    $(L_1, L_2) \leftarrow$Left ? $(L_{11}, L_2)$ : $(L_1, L_{22})$;
    return $(a, b)$;
end
```

Algorithm 3. Computes in $O(1)$ the Lower (Left or Right) characteristics (a, b) of a DSS that is some subset of a DSL D, given a starting point A and an ending point L_m (Left part) or given a starting point L_m and an ending point B (Right part) $(A, B \in D)$ (Left pattern is L_1L_m and Right pattern is L_mL_2)

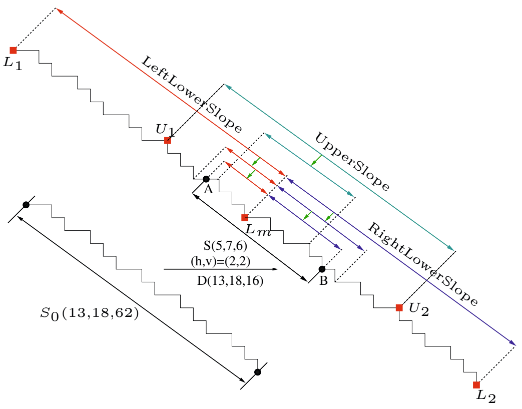

Fig. 1. A DSL $D(13, 18, 16)$ with two patterns between L_1 and L_2 and an odd depth slope. S (AB) is the covering of S_0 by the tiling $(h, v) = (2, 2)$, and also a subset of D (AB is included in two patterns of D). Lower and upper leaning points are drawn as red boxes. The (red, blue or cyan) arrows represent bottom-up move along the Stern-Brocot Tree.

5/7 that is covering A from L_{11} to the right, such that this covering reaches or overtakes A. It is impossible to take $k = 3$, because in this case this displacement from L_{11} overtakes L_{22}. So we take $k = 2$, and L_{11} is moved of two subpatterns 5/7 toward the right. We have now a new DSS $S(3, 4)$ between the new L_{11} and L_{22}. In the second step, we calculate the previous slope $PS = 1/1$ of $S = 3/4$. Since the parity of S is even, then we calculate the number k of subpatterns

```
Function UpperSlope( In $D$ : DSL $(\alpha, \beta)$, In $U_1, U_2$: Upper bound of $D$, In $A, B$ : Points of
$\mathbb{Z}^2$) : DSS $(a, b)$ ;
Var $LU, RU$ : Point of $\mathbb{Z}^2$ /* Left and Right upper leaning points */;
Var $S, PS$ : slope /* PS the previous slope of S */;
Var $k$ : integer /* Number of subpatterns covering A or B */;
Var $parity$ : boolean /* parity of slope continued fraction */;
begin
    $LU \leftarrow U_1$, $RU \leftarrow U_2$;
    if ($LU > A$ and $RU < B$) then
        return $(\alpha, \beta)$;
    $parity \leftarrow$Parity($D$), $PS \leftarrow$PreviousSlope($D$);
1   if ($parity$ is odd) then
        if ($RU > B$) then
2           $k \leftarrow$NumberOfCoveringSubPatterns($RU, B, PS, true, false$);
            $RU \leftarrow RU - k(PS_y, -PS_x)$;
        if ($LU < A$) then
            $LU \leftarrow RU - (PS_y, -PS_x)$;
3   else
        if ($LU < A$) then
4           $k \leftarrow$NumberOfCoveringSubPatterns($LU, A, PS, true, false$);
            $LU \leftarrow LU - k(-PS_y, PS_x)$;
        if ($RU > B$) then
            $RU \leftarrow LU - (-PS_y, PS_x)$;
    $(a, b) \leftarrow (LU_y - RU_y, RU_x - LU_x)$;
    return $(a, b)$;
end
```

Algorithm 4. Computes in $O(1)$ the Upper characteristics (a, b) of a DSS that is some subset of a DSL D (U_1 and U_2 are two upper leaning points of D), given a starting point A and an ending point B ($A, B \in D$)

1/1 that is covering A from L_{22} to the left, such that this covering reaches or overtakes A. It is impossible to take $k = 4$, because in this case this displacement from L_{22} overtakes L_{11}. So we take $k = 3$, and L_{11} is moved of three subpatterns 1/1 toward the left and we obtain a new DSS from the new L_{11} to L_{22}. As L_{11} reach A, this algorithms stops and returns the left slope 1/1.

2.3 S Is Included in One Pattern of D

In the remaining case, S is included in one pattern only. The function `NewLowerBounds` (Algorithm 5) attempts to move L_1 toward A with k_1 subpatterns and L_2 toward B with k_2 subpatterns, according to the parity of the depth of the development of the slope in continued fractions (it means that $Parity(D)$ is true if D has a slope with even depth, false otherwise) and to the previous convergent of the slope (line 5) (if the depth of the slope is a $k - th$ convergent, then the previous slope is a $(k-1) - th$ convergent). This displacement is constrained by the fact that the leaning points must still surround A and B (see Fig. 3,(a),(b)). In the case where L_1 and L_2 did not move, we can conclude on the characteristics with a call to `FinalSlope` (Algorithm 6). In this algorithm, we focus on the three cases. Firstly, if the second lower leaning point Lp_2 is equal to B and Lp_1 moves toward A with either one previous slope or previous previous slope, according to the parity of the depth of the slope. Secondly, if the first lower leaning point Lp_1 is equal to A and Lp_2 moves towards B. Finally, if the lower leaning points

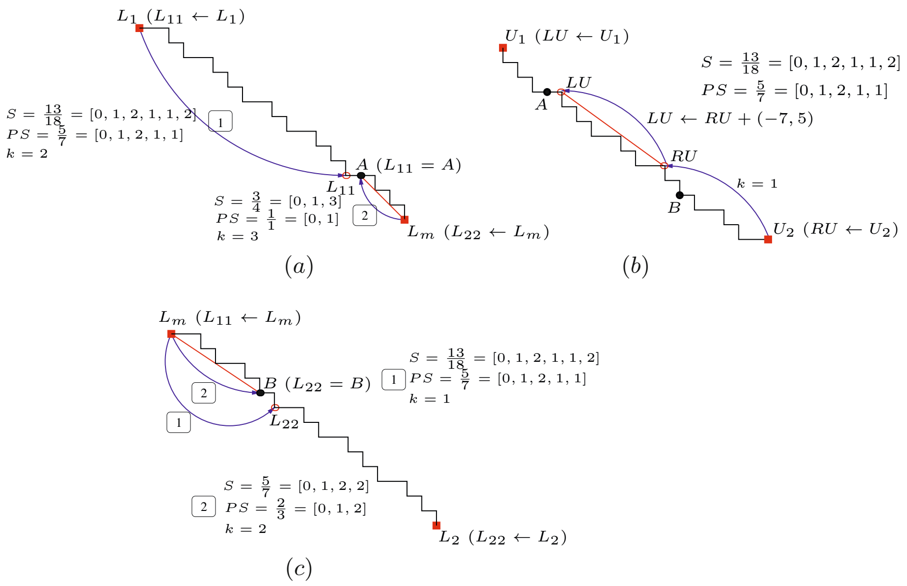
L_1 $(L_{11} \leftarrow L_1)$
$S = \frac{13}{18} = [0,1,2,1,1,2]$
$PS = \frac{5}{7} = [0,1,2,1,1]$
$k = 2$
1
A $(L_{11} = A)$
L_{11}
$S = \frac{3}{4} = [0,1,3]$
$PS = \frac{1}{1} = [0,1]$
$k = 3$
2
L_m $(L_{22} \leftarrow L_m)$
(a)
U_1 $(LU \leftarrow U_1)$
$S = \frac{13}{18} = [0,1,2,1,1,2]$
$PS = \frac{5}{7} = [0,1,2,1,1]$
LU
A
$LU \leftarrow RU + (-7,5)$
RU
$k = 1$
B
U_2 $(RU \leftarrow U_2)$
(b)
L_m $(L_{11} \leftarrow L_m)$
B $(L_{22} = B)$
$S = \frac{13}{18} = [0,1,2,1,1,2]$
$PS = \frac{5}{7} = [0,1,2,1,1]$
$k = 1$
L_{22}
$S = \frac{5}{7} = [0,1,2,2]$
$PS = \frac{2}{3} = [0,1,2]$
$k = 2$
L_2 $(L_{22} \leftarrow L_2)$
(c)

Fig. 2. The DSS U_1U_2 (resp. L_1L_m and L_mL_2) of characteristics $(13, 18, 16)$, which is a subset of L_1L_2 of Fig. 1. The blue arrows represent the move between the upper (lower) leaning points, and k is the number of subpatterns covering B (A and B).

```
Function NewLowerBounds( In $D$ : DSL $(\alpha,\beta)$, In $L_1, L_2, A, B$: Points of $\mathbb{Z}^2$) : (DSL, Point of $\mathbb{Z}^2$, Point of $\mathbb{Z}^2$);
Var $L, V_1, V_2$ : Point of $\mathbb{Z}^2$;
Var $k, k_1, k_2$ : integer;
Var $parity, covering_A, covering_B$ : boolean;
Var $PS$ : slope;
begin
    $parity \leftarrow$Parity$(D)$, $PS \leftarrow$PreviousSlope$(D)$;
    $L \leftarrow parity\ ?\ L_1 : L_2$, $covering_A \leftarrow parity\ ?\ true : false$;
    $covering_B \leftarrow parity\ ?\ false : true$;
    $k_1 \leftarrow$NumberOfCoveringSubPatterns$(L, A, PS, covering_A, true)$;
    $k_2 \leftarrow$NumberOfCoveringSubPatterns$(L, B, PS, covering_B, true)$;
    if ($parity$ is odd) then
        $Lp_1 \leftarrow L_1 - k_1(-PS_y, PS_x)$, $V_2 \leftarrow L_1 - k_2(-PS_y, PS_x)$;
        $Lp_2 \leftarrow (V_2 \leq L_2)\ ?\ V_2 : L_2$;
    else
        $Lp_2 \leftarrow L_2 - k_2(PS_y, -PS_x)$, $V_1 \leftarrow L_2 - k_1(PS_y, -PS_x)$;
        $Lp_1 \leftarrow (V_1 \geq L_1)\ ?\ V_1 : L_1$;
    $k \leftarrow$( parity is odd and $Lp_2! = L_2$) or ( parity is even and $Lp_1! = L_1$) ?
    $(Lp_{2_x} - Lp_{1_x})/PS_y$ : 1;
    $(\alpha,\beta) \leftarrow ((Lp_{1_y} - Lp_{2_y})/k, (Lp_{2_x} - Lp_{1_x})/k)$;
    return $(DSL(\alpha,\beta), Lp_1, Lp_2)$;
end
```

Algorithm 5. Updates in $O(1)$ the slope of the DSL $D(\alpha, \beta)$ according to the change of the two lower leaning points from (L_1, L_2) to (Lp_1, Lp_2)

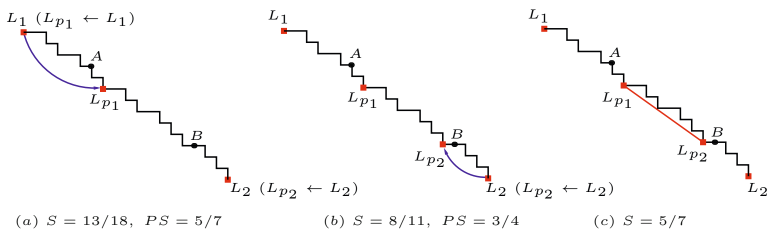
L_1 ($L_{p_1} \leftarrow L_1$)
A
L_{p_1}
B
L_2 ($L_{p_2} \leftarrow L_2$)
(a) $S = 13/18$, $PS = 5/7$
L_1
A
L_{p_1}
B
L_{p_2}
L_2 ($L_{p_2} \leftarrow L_2$)
(b) $S = 8/11$, $PS = 3/4$
L_1
A
L_{p_1}
B
L_{p_2}
L_2
(c) $S = 5/7$

Fig. 3. L_1L_2 is a DSS of characteristics $(13, 18, 16)$. The blue arrows represent the move of the lower leaning points. (a) $S = L_1L_2$, L_1 is moved of one subpatterns 5/7 towards the right. (b) $S = L_{p_1}L_2$, L_2 is moved of one subpatterns 3/4 towards the left. (c) Lp_1 and Lp_2 represent the lower leaning points of AB.

```
Function FinalSlope( In $D$ : DSL $(\alpha,\beta)$, In $L_1, L_2, Lp_1, Lp_2, A, B$: Points of $\mathbb{Z}^2$) : DSS
$(a, b, \mu)$;
Var $PS, PPS$ : slope;
Var $parity$ : boolean;
begin
    $PS \leftarrow$PreviousSlope($D$), $PPS \leftarrow$PreviousSlope($PS$), parity $\leftarrow$Parity($D$);
    if ($Lp_2 == B$) then
        $Lp_1 \leftarrow Lp_1 - ((parity)\ ?\ (-PS_y, PS_x) : (-PPS_y, PPS_x)\ )$;
        $(a, b) \leftarrow (Lp_{1_y} - Lp_{2_y}, Lp_{2_x} - Lp_{1_x})$, $\mu \leftarrow aL_{2_x} + bL_{2_y}$;
    else if ($Lp_1 == A$) then
        $Lp_2 \leftarrow Lp_2 - ((parity)\ ?\ (PPS_y, -PPS_x) : (PS_y, -PS_x)\ )$;
        $(a, b) \leftarrow (Lp_{1_y} - Lp_{2_y}, Lp_{2_x} - Lp_{1_x})$, $\mu \leftarrow aL_{1_x} + bL_{1_y}$;
    else
        $(a, b) \leftarrow PS$;
        $\mu \leftarrow a(parity\ ?\ L_{1_x} : L_{2_x}) + b(parity\ ?\ L_{1_y} : L_{2_y})$;
    return $(a, b, \mu)$;
end
```

Algorithm 6. Computes in $O(1)$ the characteristics (a, b, μ) of a DSS in the case where $L_1 == Lp_1$ and $L_2 == Lp_2$

did not move. We therefore conclude that AB has the slope of Lp_1Lp_2 (see Fig. 3,(c)). If at least one of the leaning points has moved, then we recursively call `ReversedSmartDSS` but with an input DSL with a slope depth at least one smaller than the slope depth of D.

3 Correctness and Computational Complexity

We sketch here the proof of the correctness of algorithm `ReversedSmartDSS` (Proposition 1). Proposition 2 establishes the time complexity of this algorithm, as a function of the depth of the slopes of the input DSL and output DSS.

Proposition 1. *For any DSL D such that $A, B \in D$, Algorithm 1 computes the characteristics of the segment $S = [AB]$ included in D.*

Proof. We prove by induction on n (the depth of the slope of D) that Algorithm 1 computes the characteristics of the segment S in the fourth quadrant. The initial steps $n = -1$ or $n = 0$ are obvious since D is just a vertical or horizontal segment and has slope 1/0 or 0/1. In this case, Algorithm 1 returns the correct characteristics $(1, 0, A_x)$ or $(0, 1, A_y)$. The induction hypothesis is that this algorithm has a correct output for all points A, $B \in D$ of slope $[u_0, u_1, \cdots, u_n]$. We shall prove that the output is also correct for any points A, B in some DSL D with a depth of its slope equal to $n + 1$.

If the horizontal distance between the lower leaning points L_1 and L_2 verifies one of the three conditions of line 3, then Algorithm 1 stops and returns the characteristics of the segment AB which are trivially in this case the characteristics of D itself (i.e. depth is $n + 1$).

Otherwise, if this distance is equal to 2β, then we call Algorithm 2. There, three algorithms are run in a parallel manner: (i) `LowerSlope`(..., true) at line 4 for the pattern to the left of L_m, (ii) `LowerSlope`(..., false) at line 5 for the pattern to the right of L_m, and (iii) `UpperSlope` at line 6 for the reversed pattern containing L_m. More precisely, they are run consecutively one iteration each time. This algorithm does not depend on the induction hypothesis. It just calculates the largest pattern contained in $[AB]$ either to the left or right of the lowest point L_m, and the largest reversed pattern containing L_m. Since patterns characterize a DSS slope, the deepest slope is exactly the slope of AB.

The last case occurs when S is included in only one pattern of D. We then look for a simpler DSL than D that contains AB by moving the lower leaning points toward AB (function `NewLowerBounds`). The simpler DSL is one of the left or right ancestor of the slope of D in the Stern-Brocot tree, according to its parity. If such a simpler DSS includes AB, then the lower leaning points are moved. Here, we recursively call `ReversedSmartDSS` but with this new input data, where the input DSL has a slope depth less or equal to n. The induction hypothesis applies in this case. If no simpler DSS contains AB, the function `FinalSlope` determines directly (without loop) the correct characteristics, with simple checks. We can therefore conclude in all cases. □

We assume that we have stored all the convergents of the slope of D before running Algorithm 1. We further assume a computing model where standard arithmetic operations takes $O(1)$. Note that the largest integer used in the presented algorithms is lower than $\alpha^2 + \beta^2$, if D has slope $\frac{\alpha}{\beta}$, for a frame centered on the DSS.

Proposition 2. *Algorithm 1 takes $O(n - n')$ time complexity, where n is the depth of the input DSL D with slope $\frac{\alpha}{\beta} = [u_0, u_1, \cdots, u_n]$ and n' is the depth of the output DSS S with slope $\frac{a}{b} = [u_0, u_1, \cdots, u_{n'}]$.*

Proof. Computation of Algorithm 1 on line 1, 2 and 3 is clearly $O(1)$. Otherwise, if S is included in two patterns of D (line 4), we then apply Algorithm 2 (detailed in the previous proposition). In the core of this algorithm, we call three algorithms where the time complexity of each algorithm is $O(1)$. Furthermore, we repeat either the same condition (line 3) until one of the three algorithms is stopped, line 7 if one is stopped, or line 8 if two are stopped. Since the stopping occurs when the output slope is reached, the number of steps of

Algorithm 2 is the difference between the depth slope of the input DSL and the one of the output DSS, that is $O(n - n')$.

Otherwise, it means that S is included in one pattern of D. In this case, we compute in $O(1)$ the positions of the new two lower leaning points L_1 and L_2 of D by calling `NewLowerBounds` (line 5). When the lower leaning points do not move, then the function `FinalSlope` determines the correct characteristics in $O(1)$. Otherwise, we recursively call `ReversedSmartDSS` but with an input DSL with a slope depth less or equal to $n - 1$. The preceding arguments recursively hold. With this observation, the worst-case computational complexity of Algorithm 1 is clearly $O(n - n')$. □

Lamé's theorem implies that Algorithm 1 takes at most $O(\log(\max(\alpha, \beta)))$ time.

Timing measures. Execution times were measured for some contours (Table 1). These times were obtained on a 2.10 GHz Intel Core 2 Duo. The listed numbers include the computation time for subsampling contours and the associated number of tested points in the `SmartDSS` algorithm [14] and the computation of the characteristics of each segment in the `ReversedSmartDSS` algorithm.

4 Conclusion

We have presented a novel fast DSS recognition algorithm with guaranteed logarithmic complexity, in the special case where a DSL container is known. The algorithm principle is to move in a bottom-up way along the Stern Brocot Tree, starting from the initial known DSL slope. Finally, we have used this algorithm to efficiently compute the exact multiscale covering of a digital contour (Table 1). Our algorithms are sensitive to the depth of the input DSL and output DSS, and are clearly sublinear.

References

1. Anderson, T.A., Kim, C.E.: Representation of digital line segments and their preimages. Computer Vision, Graphics, and Image Processing 30(3), 279–288 (1985)
2. Debled-Rennesson, I.: Etude et reconnaissance des droites et plans discrets. Ph.D. thesis, Université Louis Pasteur, Strasbourg (1995)
3. Debled-Rennesson, I., Reveillès, J.P.: A linear algorithm for segmentation of discrete curves. International Journal of Pattern Recognition and Artificial Intelligence 9, 635–662 (1995)
4. Dorst, L., Smeulders, A.W.M.: Discrete representation of straight lines. IEEE transactions Pattern Analysis Machine Intelligence 6, 450–463 (1984)
5. Dorst, L., Smeulders, A.W.M.: Discrete straight line segments: Parameters, primitives and properties. In: Vision Geometry, Series Contemporary Mathematics, pp. 45–62. American Mathematical Society, Providence (1991)
6. Freeman, H.: On the encoding of arbitrary geometric configurations. Transactions on Electronic Computer 10(2), 260–268 (1961)
7. Freeman, H.: Computer processing of line-drawing images. ACM Comput. Surv. 6(1), 57–97 (1974)

8. Klette, R., Rosenfeld, A.: Digital Geometry - Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Francisco (2004)
9. Klette, R., Rosenfeld, A.: Digital straightness–a review. Discrete Applied Mathematics 139(1-3), 197–230 (2004), `http://www.sciencedirect.com/science/article/B6TYW-49YD4PJ-4/2/8a755750eee9d6517adbff2f20ee7dc2`, the 2001 International Workshop on Combinatorial Image Analysis
10. Kovalevsky, V.: Applications of digital straight segments to economical image encoding. In: Ahronovitz, E. (ed.) DGCI 1997. LNCS, vol. 1347, pp. 51–62. Springer, Heidelberg (1997)
11. Kovalevsky, V.A.: New definition and fast recognition of digital straight segments and arcs. In: International Conference on Pattern Analysis and Machine Intelligence, pp. 31–34 (1990)
12. Kovalevsky, V.A., Fuchs, S.: Theoretical and experimental analysis of the accuracy of perimeter estimates. In: Forster, W., Ruwiedel, S. (eds.) Robust Computer Vision, pp. 218–242 (1992)
13. Montanari, U.: A note on minimal length polygonal approximation to a digitized contour. Communications of the ACM 13(1), 41–47 (1970)
14. Said, M., Lachaud, J.-O., Feschet, F.: Multiscale Discrete Geometry. In: Brlek, S., Reutenauer, C., Provençal, X. (eds.) DGCI 2009. LNCS, vol. 5810, pp. 118–131. Springer, Heidelberg (2009), `http://hal.archives-ouvertes.fr/hal-00413681/en/`
15. Sivignon, I., Dupont, F., Chassery, J.M.: Digital intersections: minimal carrier, connectivity, and periodicity properties. Graphical Models 66(4), 226–244 (2004)
16. Troesch, A.: Interprétation géométrique de l'algorithme d'euclide et reconnaissance de segments. Theor. Comput. Sci. 115(2), 291–319 (1993)
17. de Vieilleville, F., Lachaud, J.O.: Revisiting digital straight segment recognition. In: Kuba, A., Palágyi, K., Nyúl, L.G. (eds.) DGCI 2006. LNCS, vol. 4245, pp. 355–366. Springer, Heidelberg (2006), `http://www.lama.univ-savoie.fr/~lachaud/Publications/LACHAUD-JO/publications.html#deVieilleville06a`
18. Yaacoub, J.: Enveloppes convexes de réseaux et applications au traitement d'images. Ph.D. thesis, Université Louis Pasteur, Strasbourg (1997), `http://lsiit.u-strasbg.fr/Publications/1997/Yaa97`