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

Mouhammad Said, Jacques-Olivier Lachaud

{Mouhammad.Said,Jacques-Olivier.Lachaud}@univ-savoie.fr, University of Savoie, France

Abstract

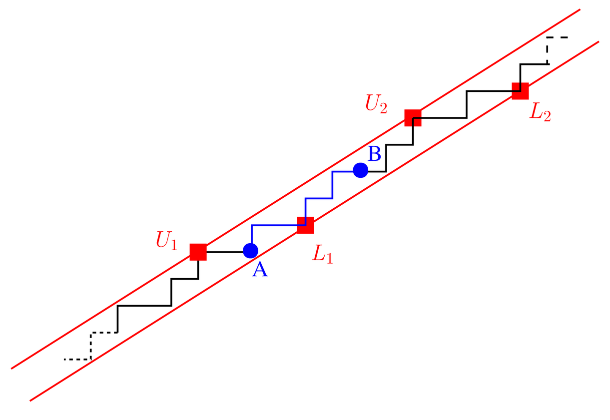
We address the problem of computing the exact characteristics of any subsegment of a digital straight line with known characteristics. We present a new algorithm that solves this problem, whose correctness is proved. Its principle is to climb the Stern-Brocot tree of fraction in a bottom-up way. Its 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.

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

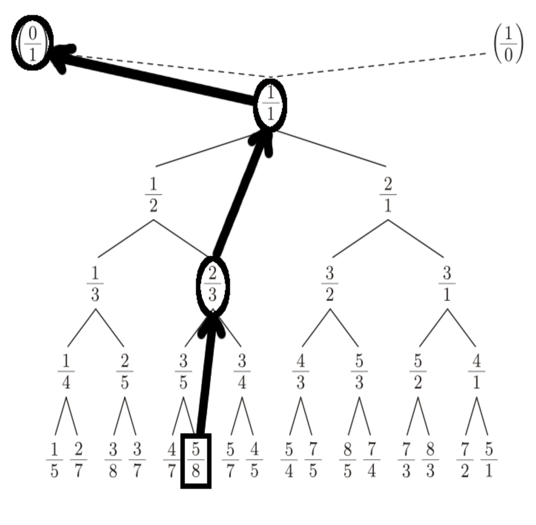
1 Introduction

Objective:

- Present a fast algorithm which computes the exact (minimal) characteristics of a DSS that is a subset of a known DSL.
- Determine these characteristics by moving in a bottom-up way along the Stern-Brocot Tree [1].
- Prove the correctness of this algorithm.
- Compare this algorithm with the SmartDSS algorithm published in DGCI 2009 [3] and the classical DSS recognition algorithm [2].
- Apply this result to compute the multiresolution of a digital object, since analytic formulas give the multiresolution of DSL.



DSL D(5,8,0) and DSS AB(2,3,1)

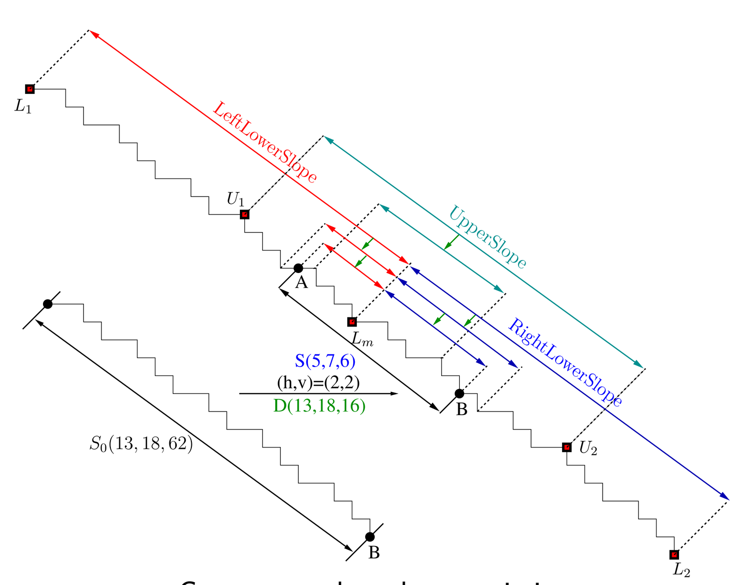


Stern-Brocot Tree

2 A coarsening algorithm for computing the characteristics of a subsegment included in a known DSL

Overview of the algorithm.

```
Function ReversedSmartDSS ( In D : DSL(α, β), In L1, L2 :
Points of Z², In A, B : Points of Z² ) : DSS (a, b, µ);
Var Lp1, Lp2 : Point of Z²;
Var dL : integer /* The horizontal distance between L1 and L2 */;
Var S : slope;
begin
    if (Ax == Bx) then return (1, 0, Ax);
    if (Ay == By) then return (0, 1, Ay);
    dL ← L2x − L1x;
    if (dL ≥ 3β) or (dL == 2β and (A == L1 or B == L2)) or
    (A == L1
      and B == L2) then return (α, β, αL1x + βL1y) ;
    /* S is included in two patterns of D */;
    if (dL == 2β) then return DSSWithinTwoPatterns
    (D, L1, L2, A, B);
    /* S is included in one pattern of D */;
    (D(α′, β′), Lp1, Lp2) ← NewLowerBound(D, L1, L2, A, B);
    if (Lp1 == L1) and (Lp2 == L2) then
        return FinalSlope(D, L1, L2, Lp1, Lp2, A, B);
    return ReversedSmartDSS(DSL(α′, β′), Lp1, Lp2, A, B);
end
```



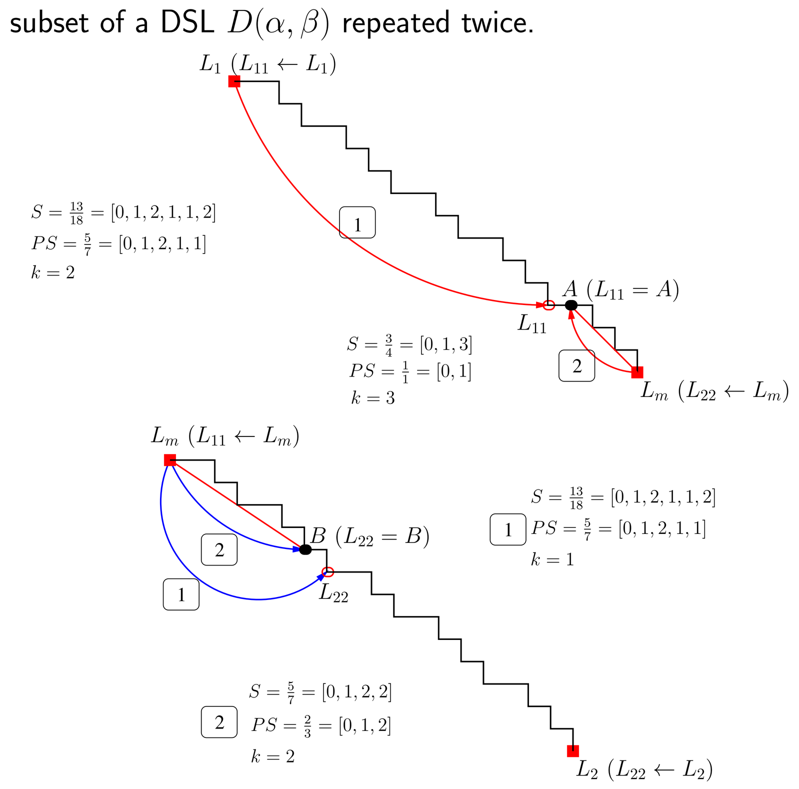
Computes the characteristics (a, b, μ) of a DSS S that is some subset of a DSL D of slope $\frac{\alpha}{\beta}$ and lower leaning points L_1 and L_2

S is included in two patterns of D

```
Function DSSWithinTwoPatterns( In D : DSL (α, β), In L1, L2 : Lower bounds of D, In A, B : Point of Z²) : DSS (a, b, µ);
begin
    L ← true, R ← true, U ← true, i ← 0, Lm ← MiddleLowerBound(L1, D);
    CF ← ContinuedFraction(D), U1 ← FirstUpperBound(D, L1, L2), U2 ← SecondUpperBound(U1, D);
    while i < |CF| do
        if (L and R and U) then
            S1 ← LowerSlope(DL, L1, Lm, L2, A, true) /* Left Lower Slope */;
            S2 ← LowerSlope(DR, L1, Lm, L2, B, false) /* Right Lower Slope */;
            S3 ← UpperSlope(DU, U1, U2, A, B) /* Upper Slope */;
            if (L1 >= A or L2 <= B or (U1 >= A and U2 <= B)) then
                DS ← DeepestSlope( S1 , S2 , S3 )/* DS The deepest slope */;
            if L1 >= A then L ← false; if L2 <= B then R ← false;
            if (U1 >= A and U2 <= B) then U ← false;
            if (L1 >= A and DS == S1) or (L2 <= B and DS == S2) or (U1 >= A and U2 <= B and DS == S3) then break;
        else if ((L and R) or (L and U) or (R and U)) then
            S1 ← (R and U) ? LowerSlope(DR, L1, Lm, L2, B, false) : LowerSlope(DL, L1, Lm, L2, A, true) ;
            S2 ← (L and R) ? LowerSlope(DR, L1, Lm, L2, B, false) : UpperSlope(DU, U1, U2, A, B);
            if (((L and (R or U)) and (L1 >= A)) or (((R and (L or U)) and (L2 <= B)) or (((U and (L or R)) and (U1 >= A
            and U2 <= B))) then DS ← DeepestSlope( S1 , S2 );
            if (((R and U) and ((L2 <= B and DS == S1) or (U1 >= A and U2 <= B and DS == S2)))
                or ((L and U) and ((L1 >= A and DS == S1) or (U1 >= A and U2 <= B and DS == S2)))
                or ((L and R) and ((L1 >= A and DS == S1) or (L2 <= B and DS == S2)))) then break;
            if (((L and U) or (L and R)) and (L1 >= A)) then L ← false;
            if (((R and U) or (L and R)) and (L2 <= B)) then R ← false;
            if (((R and U) or (L and U)) and (U1 >= A and U2 <= B)) then U ← false;
        else
            if L then DS ← LowerSlope(DL, L1, Lm, L2, A, true);
            else if R then DS ← LowerSlope(DR, L1, Lm, L2, B, false);
            else DS ← UpperSlope(DU, U1, U2, A, B);
            if ((L and L1 >= A) or (R and L2 <= B) or (U and U1 >= A and U2 <= B)) then break;
    a ← DSx, b ← DSy, µ ← aLmx + bLmy;
    return (a, b, µ);
end
```

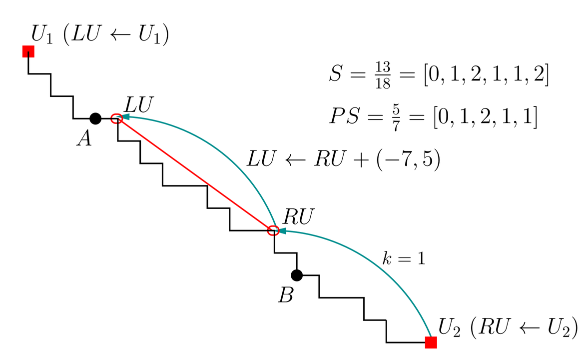
Computes the characteristics (a, b, μ) of a DSS S that is some subset of a DSL $D(\alpha, \beta)$ repeated twice.

```
Function LowerSlope( In D : DSL (α, β), InOut L1, Lm, L2 : Lower bounds
of D, In X(A or B) : Point of Z², In Left : Boolean): DSS (a, b);
begin
    (P, L11, L22) ← Left ? (L1, L1, Lm) : (L2, Lm, L2) ;
    parity ← Parity(D);
    PS ← PreviousSlope(D);
    if (parity is odd) then
        k ← NumberOfCoveringSubPatterns(L11, X, PS, true, false);
        P ← L11 − k(−PSy, PSx);
    else
        k ← NumberOfCoveringSubPatterns(L22, X, PS, true, false);
        P ← L22 − k(PSy, −PSx);
    if Left then L11 = P;
    else L22 = P;
    (a, b) ← (|Py − Lmy|, |Lmx − Px|);
    (L1, L2) ← Left ? (L11, L2) : (L1, L22);
    return (a, b);
end
```



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) $(A, B \in D)$.

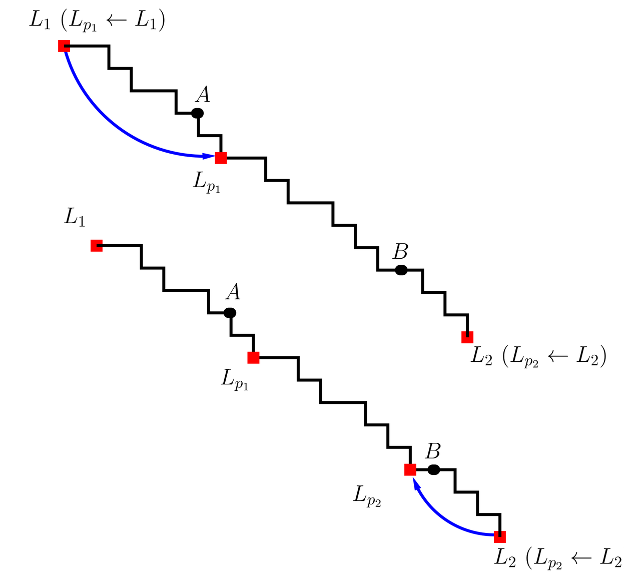
```
Function UpperSlope( In D : DSL (α, β), In U1, U2: Upper bound of D, In
A, B : Points of Z²) : DSS (a, b) ;
begin
    LU ← U1, RU ← U2;
    if (LU > A and RU < B) then
        return (α, β);
    parity ← Parity(D), PS ← PreviousSlope(D);
    if (parity is odd) then
        if (RU > B) then
            k ← NumberOfCoveringSubPatterns(RU, B, PS, true, false);
            RU ← RU − k(PSy, −PSx);
        if (LU < A) then
            LU ← RU − (PSy, −PSx);
    else
        if (LU < A) then
            k ← NumberOfCoveringSubPatterns(LU, A, PS, true, false);
            LU ← LU − k(−PSy, PSx);
        if (RU > B) then
            RU ← LU − (−PSy, PSx);
    (a, b) ← (LUy − RUy, RUx − LUx);
    return (a, b);
end
```



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 $(A, B \in D)$.

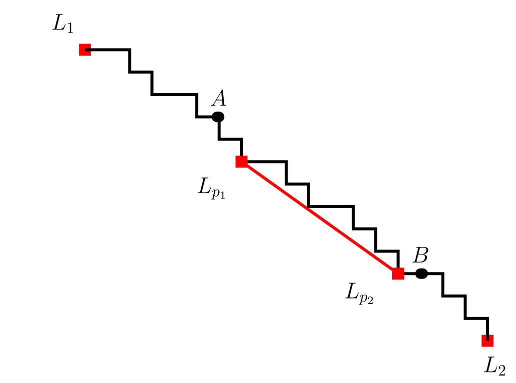
S is included in one pattern of D

```
Function NewLowerBounds( In D : DSL (α, β), In L1, L2, A, B: Points of Z²)
: (DSL, Point of Z², Point of Z²);
begin
    parity ← Parity(D), PS ← PreviousSlope(D);
    L ← parity ? L1 : L2, coveringA ← parity ? true : false;
    coveringB ← parity ? false : true;
    k1 ← NumberOfCoveringSubPatterns(L, A, PS, coveringA, true);
    k2 ← NumberOfCoveringSubPatterns(L, B, PS, coveringB, true);
    if (parity is odd) then
        Lp1 ← L1 − k1(−PSy, PSx), V2 ← L1 − k2(−PSy, PSx);
        Lp2 ← (V2 ≤ L2) ? V2 : L2;
    else
        Lp2 ← L2 − k2(PSy, −PSx), V1 ← L2 − k1(PSy, −PSx);
        Lp1 ← (V1 ≥ L1) ? V1 : L1;
    k ← ( parity is odd and Lp2! = L2) or ( parity is even and Lp1! = L1) ?
    (Lp2x − Lp1x)/PSy : 1;
    (α, β) ← ((Lp1y − Lp2y)/k, (Lp2x − Lp1x)/k);
    return (DSL(α, β), Lp1, Lp2);
end
```



Updates in $O(1)$ the slope of the DSL D according to the change of the two lower leaning points. .

```
Function FinalSlope( In D : DSL (α, β), In L1, L2, Lp1, Lp2, A, B: Points of
Z²) : DSS (a, b, µ);
begin
    PS ← PreviousSlope(D), PPS ← PreviousSlope(PS), parity ← Parity(D);
    if (Lp2 == B) then
        Lp1 ← Lp1 − ((parity) ? (−PSy, PSx) : (−PPSy, PPSx) );
        (a, b) ← (Lp1y − Lp2y, Lp2x − Lp1x), µ ← aL2x + bL2y;
    else if (Lp1 == A) then
        Lp2 ← Lp2 − ((parity) ? (PPSy, −PPSx) : (PSy, −PSx) );
        (a, b) ← (Lp1y − Lp2y, Lp2x − Lp1x), µ ← aL1x + bL1y;
    else
        (a, b) ← PS;
        µ ← a(parity ? L1x : L2x)+ b(parity ? L1y : L2y);
    return (a, b, µ);
end
```



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

3 Correctness and computational complexity

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

Proposition 2 Algorithm $ReversedSmartDSS$ 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 $\frac{a}{b} = [u_0, u_1, \cdots, u_{n'}]$.

Timing measures: Computation times of the (h, v)-covering of various digital shapes with our proposed approach.

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

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 Timing). Our algorithms are sensitive to the depth of the input DSL and output DSS, and are clearly sublinear.

References

[1] F. de Vieilleville and J.-O. Lachaud. Revisiting digital straight segment recognition. In A. Kuba, K. Palágyi, and L.G. Nyúl, editors, *Proc. Int. Conf. Discrete Geometry for Computer Imagery (DGCI'2006), Szeged, Hungary*, volume 4245 of *LNCS*, pages 355–366. Springer, October 2006.

[2] I. Debled-Rennesson and J.-P. Reveillès. A linear algorithm for segmentation of discrete curves. *International Journal of Pattern Recognition and Artificial Intelligence*, 9:635–662, 1995.

[3] M. Said, J.-O. Lachaud, and F. Feschet. Multiscale Discrete Geometry. In *Proc. International Conference on Discrete Geometry for Computer Imagery (DGCI2009)*, volume 5810 of *Lecture Notes in Computer Science*, pages 118–131, Montréal, Québec Canada, 2009. Springer.