Efficient computation of the outer hull of a discrete path*

S. Brlek[1], H. Tremblay[1], J. Tremblay[1], and R. Weber[1]

Laboratoire de Combinatoire et d'Informatique Mathématique,
Université du Québec à Montréal,
CP 8888 Succ. Centre-ville, Montréal (QC) Canada H3C 3P8
brlek.srecko@uqam.ca, hugo.tremblay@lacim.ca,
jerome.tremblay@uqam.ca

Abstract. We present here a linear time and space algorithm for computing the outer hull of any discrete path encoded by its Freeman chain code. The basic data structure uses an enriched version of a the data structure introduced by Brlek, Koskas and Provençal. Since path intersection is achievable in linear time and space thanks to the use of quaternary trees for representing points in the discrete plane $\mathbb{Z} \times \mathbb{Z}$ on which neighborhood links are added. Then, by combining the well-known wall follower algorithm we obtain the desired result with two passes resulting in a global linear time and space algorithm. As a byproduct, the convex hull is obtained as well.

Keywords: Freeman code, lattice paths, radix tree, discrete figures, outer hull, convex hull.

1 Introduction

The ever-growing use of digital screens in industrial, military and civil applications gave rise to a new branch of study of discrete objects: digital geometry, where objects are sets of pixels. In particular, their various geometric properties play an essential role, for allowing the design of efficient algorithms for recognizing patterns and extracting features: these are mandatory steps for an accurate interpretation of acquired images.

Convex objects play a prominent role in several branches of mathematics, namely functional analysis, optimization, probability and mathematical physics (see [1] for a detailed account of convex geometry and applications). In Euclidean geometry, given a finite set of points, the problem of finding the smallest convex figure containing all of them gave rise to the geometric notion of *convex hull*. On the practical side, the computation of the convex hull proved to be one of the most fundamental algorithm as it has many applications ranging from operational research [2] to design automation [3]. It is also widely used in computer graphics, and particularly in image processing [4]. For example, the Delauney

* with the support of NSERC (Canada)

triangulation of a d-dimensional set of points in Euclidean space is equivalent to finding the convex hull of a set of $d+1$-dimensional points [5].

It is well known that for the Euclidean case, algorithms for computing the convex hull of a set $S \subset \mathbb{R}^2$ run in $\mathcal{O}(n \log n)$ time where $n = |S|$ (see [6, 7]). One can also show that such algorithms are optimal up to a linear constant (see [8–10] for the general case).

By restraining the problem to computing the convex hull of simple polygons, linear asymptotic bounds are achieved (see [11, 12]). The digital version of this problem is a little more involved. First instance, one can compute the convex hull of a set of pixels S by first computing the Euclidean convex hull of S and then digitalizing the result [13]. This automatically yields $\mathcal{O}(n \log n)$ asymptotical bounds in the worst case. However, linear asymptotic bounds are obtained when considering discrete paths encoded by elementary steps. Indeed, Brlek et al. provided a linear time algorithm for computing the discrete convex hull of non self-intersecting closed paths in the square grid [14]. It is based on a very efficient factorization of the path in Lyndon words.

The situation is more complicated for intersecting paths. In this paper, we describe a linear algorithm for computing the outer hull of any discrete path. This goal is achieved by using the data structure described in [15] where the authors designed a linear time and space algorithm for detecting path intersection. It rests on the encoding of points in the discrete plane $\mathbb{Z} \times \mathbb{Z}$ by quaternary trees deduced from the radix order representation of binary coordinate points. Then, each path is dynamically encoded by adding a pointer for each step of the discrete path encoded on the four letter alphabet {0,1,2,3}. Starting from that, the wall follower algorithm used for maze solutions allows to take at each intersection the rightmost available step. The resulting two-passes algorithm is linear in space and time. As a byproduct, the convex hull of any discrete path is computed in linear time.

2 Preliminaries

Given a finite alphabet Σ, a *word* w is a function $w : [1, 2, \ldots, n] \longrightarrow \Sigma$ denoted by its sequence of letters $w = w_1 w_2 \cdots w_n$, and $|w| = n$ is its *length*. For $a \in \Sigma$, $|w|_a$ is the number of letters a in w. The set of all words of length k is denoted by Σ^k. Consequently, $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ is the set of all finite words on Σ where $\Sigma^0 = \{\varepsilon\}$, the set consisting of the empty word. Σ^* together with the operation of concatenation form a monoid called *the free monoid on* Σ. For any word $w \in \Sigma^*$, the k^{th} power of w is defined recursively by $w^k = w^{k-1} \cdot w$ with $w^0 = \varepsilon$.

Combinatorics on words imposed itself throughout the years as a very efficient tool to study digital geometry, that is the branch of geometry dealing with discrete sets of pixels (see [16, 17]). There is a bijection between the set of pixels and $\mathbb{Z}^2$ obtained by mapping $(a, b) \in \mathbb{Z}^2$ to the unitary square whose bottom left vertex coordinate is (a, b). Therefore, we may consider pixels as elements of $\mathbb{Z}^2$. By definition, a *discrete figure* S is a set of pixels, i.e. $S \subset \mathbb{Z}^2$. Also, S is called 4-*connected* if each pair of pixels share a common edge and 8-*connected*

if each pair of pixels share a common edge or vertex. Since any discrete figure is a disjoint collection of 8-connected figures, we will consider from now on that discrete figures are 4 or 8-connected.

A convenient way of representing discrete figures without hole is to use a word describing its contour. In 1961, Herbert Freeman proposed an encoding of discrete objects by specifying their contour using the four elementary steps $(\rightarrow, \uparrow, \leftarrow, \downarrow) \simeq (0, 1, 2, 3)$ [18]. This encoding provides a convenient representation of discrete paths in $\mathbb{Z}^2$. By definition, a *discrete path* P is a sequence of points $P = \{p_1, p_2, \ldots, p_n\}$ where p_i and p_{i+1} are neighbors for $1 \leq i < n$. Intuitively, two points u and v are neighbors if and only if $u = v \pm e$ where $e \in \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$. For convenience, we include the unit line segment between any two consecutive points (i.e. *linel*) of the discrete path P as part of P. This allows us to define a topology on any discrete path by replacing linels with rectangles of very small width (see Section 4).

It is clear from these definitions that any discrete path P is represented by a word $w \in \mathcal{F}^*$ where $\mathcal{F} = \{0, 1, 2, 3\}$ is the Freeman alphabet. For example, the word $w = 001100322223$ represents the discrete path shown in Figure 1(a). One says that a word $w \in \mathcal{F}^*$ is *closed* if and only if $|w|_0 = |w|_2$ and $|w|_1 = |w|_3$. Further, w is called *simple* if it codes a non self-intersecting discrete path. For instance, $w = 001100322223$ is non-simple and closed.

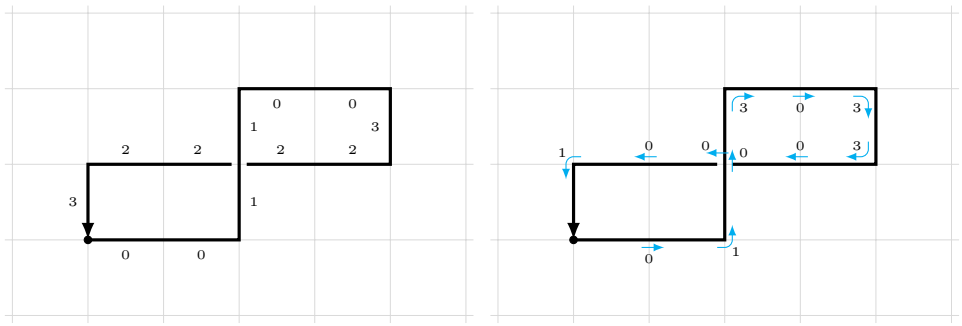


Fig. 1: (a) A discrete path coded by the word $w = 001100322223$; (b) and its first difference word $\Delta(w) = 01030330003$

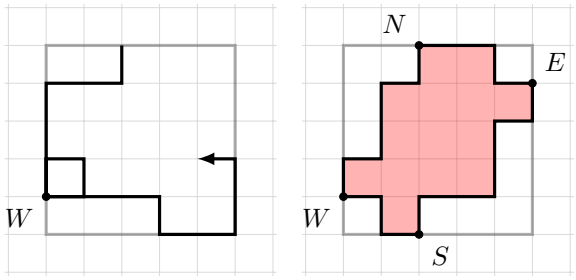
It is sometimes useful to consider encoding of paths with turns instead of elementary steps. Such encoding is obtained from the contour word $w = w_1 \cdots w_n$ by setting

$$\Delta(w) = (w_2 - w_1)(w_3 - w_2) \cdots (w_n - w_{n-1})$$

where subtraction is computed modulo 4. $\Delta(w)$ is called the *first differences word of* w. Letters of $\Delta(w) \in \mathcal{F}^*$ are interpreted via the bijection $(0, 1, 2, 3) \simeq$

(foward, left turn, u-turn, right turn). For example, one can verify in Figure 1(b) that $\Delta(w) = 01301101301$ and that it codes the turns of w.

Now, every path w is contained in a smallest rectangle, or bounding box such that we can define the point W as in Figure 2(a). W is easily obtained in linear time by keeping track of the extremum coordinates while reading the word. It is worth mentioning that in the case of a closed simple path u, this coordinate corresponds to the point W of the standard decomposition of u obtained by considering the following four extremal points of the bounding box: W (lowest on the left side), N (leftmost on the top side), E (highest on the right side) and S (rightmost on the bottom side) (see Figure 2(b)).



(a) (b)

Fig. 2: (a) Smallest rectangle containing a discrete path and the point W; (b) Standard decomposition of a self-avoiding closed path

3 Outer and convex hull

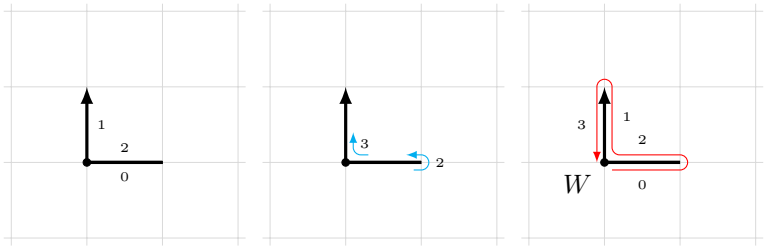
Now, let S be a 8-connected discrete figure. The *outer hull of* S, denoted $\mathrm{Hull}(S)$ is the boundary of the discrete figure without hole and of minimal area containing S, i.e. the non self-intersecting path following the exterior contour of S. Definition 1 extends this notion of the outer hull to any discrete path.

Definition 1. *Let P be any discrete path. Then,* $\mathrm{Euclidean}(P) \subset \mathbb{R}^2$ *is the simply connected subset of* $\mathbb{R}^2$ *of minimal area containing P and* $\mathrm{Hull}(P)$ *is the unique boundary of* $\mathrm{Euclidean}(P)$*, i.e.* $\mathrm{Hull}(P) = \partial(\mathrm{Euclidean}(P))$.

The difference between Definition 1 and the preceding one lies in the use of an Euclidean figure instead of a discrete figure to describe the outer hull. This choice is not arbitrary as it allows the treatment of discrete line segments by embedding them in an Euclidean figure of area 0 (namely the line segment itself, see Figure 3 for an example using the path coded by $w = 021$).

By definition, the boundary of a line segment in Euclidean space $\mathbb{R}^2$ is the segment itself. We will express such boundary in $\mathbb{Z}^2$ by a closed and simple

word, e.g. the horizontal line segment coded by 0 will be expressed by 02 (see Figure 3). The following Proposition 2 ensures that Definition 1 is a convenient generalization of the outer hull to discrete paths.



(a) (b) (c)

Fig. 3: (a) The path $w = 021$, (b) its first diference word $\Delta(w) = 23$ and (c) its outer hull $\text{Hull}(w) = 0213$

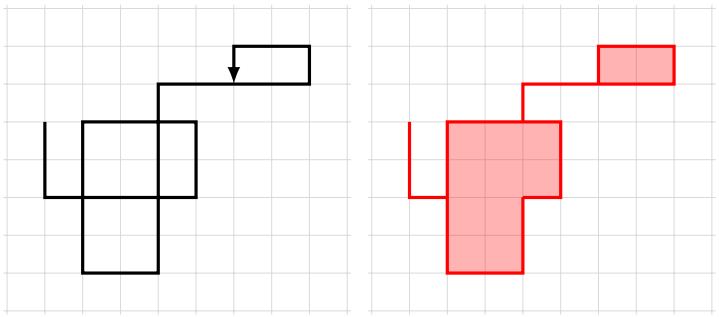
Proposition 2 *Let P be any discrete path. Then,* $\text{Hull}(P)$ *is a simple and closed discrete path. Furthermore, if P codes the boundary of a discrete figure S then* $\text{Hull}(P) = \text{Hull}(S)$ *(i.e. the two definitions coincide).*

Proof. By definition, $\partial(\text{Euclidean}(P))$ is a subgraph of P, that is any vertex and edge of $\partial(\text{Euclidean}(P))$ is in P (see Figure 4 for an example). Consequently, it is a finite union of connected line segments of $\mathbb{R}^2$. Since any finite union of closed sets is closed, $\partial(\text{Euclidean}(P))$ is closed (see [19]). Now, we show that it is also simple. Let $\overline{AB}$ and $\overline{CD}$ be two segments of $\partial(\text{Euclidean}(P))$ intersecting each other at vertex x. Consider the right angles $(\overline{Ax}, \overline{xD})$ and $(\overline{Cx}, \overline{xB})$ sharing only the common vertex x. Since the line segments associated to those angles form two non-intersecting 8-connected paths, $\partial(\text{Euclidean}(P))$ is simple. Finally, if P codes the boundary of a discrete figure S, then P is simple and closed by definition. This gives $P = \text{Hull}(P)$ and since $\text{Hull}(S)$ is the boundary $\partial(S)$ of S by definition, we have

$$\text{Hull}(S) = \partial(S) = P = \text{Hull}(P).$$

Since there is a bijection between discrete paths in $\mathbb{Z}^2$ and words on $\mathcal{F}$, we identify the path P with its coding word w and we write $\text{Hull}(w)$ instead of $\text{Hull}(P)$.

Finally, we recall some basic notions concerning digital convexity, for which a detailed exposure appears in [14, 17]. Let S be an 8-connected discrete figure. S is *digitally convex* if it is the Gauss digitalization of a convex subset R of $\mathbb{R}^2$, i.e. $S = \text{Conv}(R) \cap \mathbb{Z}^2$. The *convex hull of* S, denoted $\text{Conv}(S)$ is the convex Euclidean form of minimal area containing S. In the case of a closed simple



(a) (b)

Fig. 4: (a) A self-intersecting path P; (b) Euclidean(P) and its boundary

path w, Conv(w) is given by the Spitzer factorization of w (see [20, 14]). Given $w = w_1 w_2 \cdots w_n \in \{0,1\}^*$, one can compute the NW part of this factorization as follows: Start with the list $(b_1, b_2, \ldots, b_n) = (w_1, w_2, \ldots, w_n)$. If the slope $\rho(b_i) = |b_i|_1/|b_i|_0$ of b_i is strictly smaller than that of b_{i+1} for some i, then

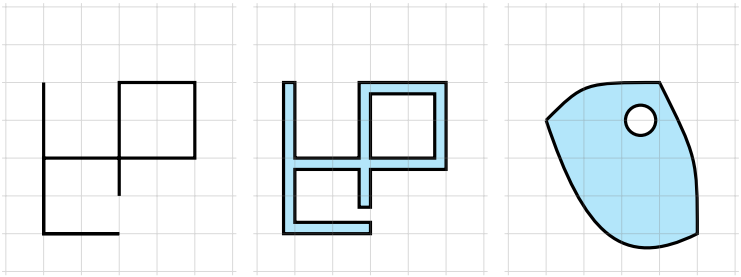
$$(b_1, b_2, \ldots, b_k) = (b_1, \ldots, b_{i-1}, b_i b_{i+1}, b_{i+2}, \ldots, b_k).$$

By repeating this process until it is no longer possible to concatenate any words, one obtains the Spitzer factorization of w. The NE, SE and SW parts of the factorization are obtained by rotations.

4 Algorithm

Let $w \in \mathcal{F}^*$ be a discrete path and $G_w = (V, E)$ its graph representation where V and E are respectively the vertices and edges sets of w. Remark that the application $g : w \mapsto G_w$ is not bijective since it is not injective (see Figure 5 for instance). Now, G_w admits the following topology $\mathcal{T}(G_w)$: Identify any edge to the rectangle $[0,1] \times [0,\delta]$ where $\delta \in \mathbb{R}$ is small and gluing them at coincident vertices, considering that the intersection of such rectangles is in $\mathcal{T}(G_w)$. Intuitively, this topology is obtained by inflating the edges of G_w as in Figure 5. One can easily show that $\mathcal{T}(G_w)$ is closed under arbitrary unions and finite intersections thus making it a topology. Moreover, the connectedness of G_w implies that of $\mathcal{T}(G_w)$. Further informations on topological aspects of graph theory can be found in [21]. With this topology, G_w can be seen as a connected maze. Now, $\mathcal{T}(G_w)$ is homeomorphic to a connected region R as in Figure 5. The problem of determining the outer hull of G_w is thus equivalent to describing the boundary $\partial(R)$ of R. Now, provided one starts on $\partial(R)$, the solution can be achieved by using the so-called "right-hand rule" or "wall follower algorithm" for traversing mazes. Applying this rule, the idea of our algorithm is to walk along the path, starting at an origin point on the outer hull and turning

systematically right at each intersection and returning to the origin point. The preceding discussion guarantees that the resulting walk is then precisely the outer hull of w.



(a) (b) (c)

Fig. 5: (a) The paths $w_1 = 333300221100001122333$ and $w_2 = 220022111133000011223333$ have the same graph representation G_w; (b) The maze associated to G_w by the topology $\mathcal{T}(G_w)$ and (c) a homeomorphic connected region R

To efficiently implement this procedure, several problems must be addressed. First, as stated before, the walk needs to start on a coordinate of the outer hull, otherwise the resulting path may not describe the correct object. This can be solved by choosing the point W associated with the contour word w as the starting point. Secondly, in the case of a path that returns to W before continuing on (the simplest of which is the path coded by $w = 021$, see Figure 3), one must make sure that the algorithm does not stop until every such sub-paths have been explored. An easy solution to this is to keep a list of all neighbors of W. One can easily show that this list has at most two elements. Finally, one needs to recognize intersections and decide of the rightmost turn. We solve this problem by using a quaternary tree structure keeping information on neighborhood relations. This so-called radix quadtree structure was first introduced by Brlek et al. in [15] for detecting path intersections. Given a discrete path w starting at $(x, y) \in \mathbb{N}^2$ and staying in the first quadrant, the *quaternary tree structure associated to* w (see [17] and [15] for the generalization to all four quadrants) is described as follows: $G = (N, R, T)$ is a quaternary tree where N is the set of vertices associated to points in the plane and R and T are sets of directed edges representing respectively paternity and neighborhood relations. More precisely, there is a vertex $r \in R$ from (x, y) to (x', y') if and only if (x', y') is a child of (x, y), that is if $(x', y') = (2x + \alpha, 2y + \beta)$ where $(\alpha, \beta) \in \{0, 1\}^2$. In the same manner, there is a vertex $t \in T$ from (x, y) to (x', y') if and only if (x', y') is a neighbor of (x, y), that is if $(x', y') = (x, y) + \boldsymbol{e}$ where $\boldsymbol{e} \in \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ (see Example 3). One should note that the quadtree structure is described in [15] with

unidirectional edges while in this paper all edges are considered bidirectional. Further, by following the procedure described in [15] to build the quadtree, one adds neighborhood links between non-visited nodes during the recursion process. This is easily fixed by adding a boolean label to each neighborhood edge indicating if that specific edge is part of the discrete path or if it has been added by a recursive call. This ensures that the points u and v are neighbors if and only if there is a neighborhood edge between these two vertices in the quadtree structure.

Example 3 *Let $w = 001100322223$ be the word coding the discrete path in Figure 1 translated to the origin. The quaternary tree structure associated to w is represented in Figure 6. Parenthood and neighborhood relations are respectively represented by black and red edges. Visited nodes are marked by red squares. Note that the actual tree being infinite, several unvisited nodes are not drawn for the sake of readability.*

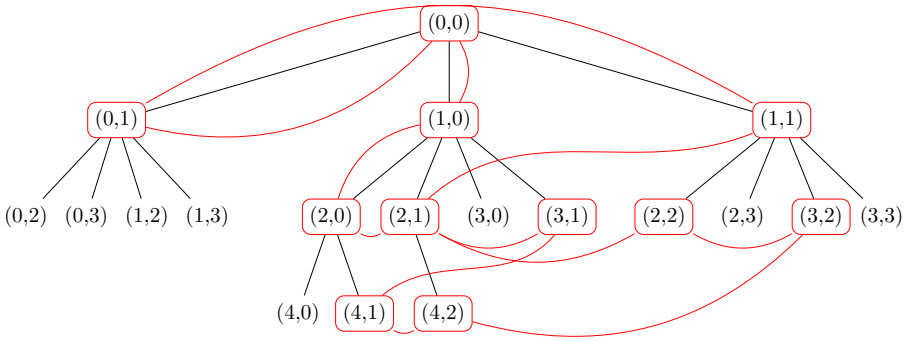


Fig. 6: Quaternary tree corresponding to the word $w = 001100322223$. Neighborhood edges added by recursive calls are omitted.

It is worth mentioning that this structure is computed in linear time and space. Moreover, it can be generalized to any discrete path as opposed to paths staying in the first quadrant.

This gives rise to the following Algorithm 1 to compute the outer hull of a discrete path w: Begin by constructing the quaternary tree G associated to w. Then, starting at the point W of the bounding box, travel along the path w. At intersections, for each neighbor v of the current coordinate c, compute $\mathrm{Step}(v - c)$, the letter associated to the vector $\boldsymbol{v} - \boldsymbol{c}$ (e.g. $\mathrm{Step}((1,0)) = 0$, $\mathrm{Step}((0,1)) = 1$, $\mathrm{Step}((-1,0)) = 2$ and $\mathrm{Step}((0,-1)) = 3$). Then, compute the first difference $\Delta(w_c \cdot \mathrm{Step}(v-c))$ of the word formed by the current letter w_c of w and $\mathrm{Step}(v - c)$. This gives the turn associated to v. Finally, choose the turn

closest to 3 (i.e. the rightmost turn). This procedure ends when returning to the point W.

Algorithm 1 Outer hull

Require: A word $w \in \mathcal{F}^*$ coding a discrete path
Ensure: A simple word $w' \in \mathcal{F}^*$ describing Hull(w)
1: Construct the quaternary tree G associated to w rooted in W
2: Let `W` be the leftmost lowest coordinate on the bounding box of w
3: Let `N` be the set of all visited neighbors of W
4: `c` $\leftarrow$ `W` $+ (1, 0)$ if it is in `N` or `W` $+ (0, 1)$ otherwise
5: $w' = \text{Step}(\texttt{c} - \texttt{W})$
6: **while** `c` $\neq$ `W` and `N` $\neq \varnothing$ **do**
7: `turn` $= 2 \mod 4$
8: **for** each neighbor `v` of `c` **do**
9: **if** $[\text{Step}(\texttt{v} - \texttt{c}) - \text{Lst}(w')] + 1 \mod 4 \leq [\texttt{turn}] + 1 \mod 4$ **then**
10: `turn` $\leftarrow \text{Step}(\texttt{v} - \texttt{c}) - \text{Lst}(w')$
11: `next` $\leftarrow$ `v`
12: **end if**
13: **end for**
14: $w' = w' \cdot \text{Step}(\texttt{next} - \texttt{c})$
15: remove `c` from `N`
16: `c` $\leftarrow$ `next`
17: **end while**
18: **return** w'

Theorem 4 (Correctness of Algorithm 1) *For any word $w \in \mathcal{F}^*$, Algorithm 1 returns* Hull(w).

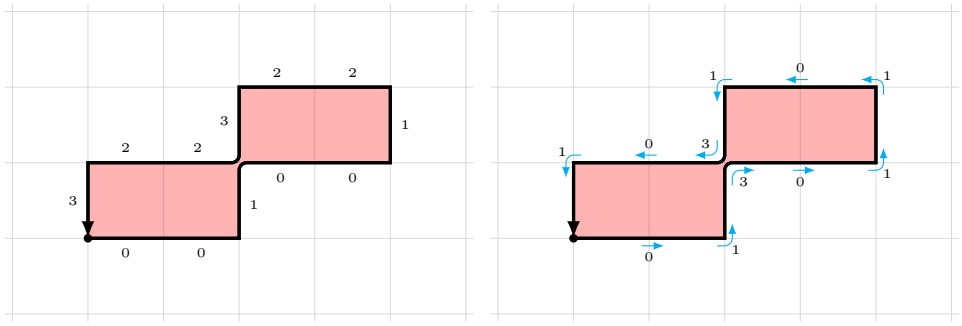
Proof. Let Hull(w) be of length $k \in \mathbb{N}^+$. We use the following loop invariant:

> At the start of the i^{th} iteration of the while loop in Line 6, w' is a prefix of length i of the contour word associated to Hull(w).

The invariant holds the first time Line 6 is executed, since at that time, w' is the first step of the outer hull of w computed at Line 5. Now, assume the invariant holds before the i^{th} iteration of the loop. Then, Lines 8 to 13 find the rightmost turn at the current coordinate `c`. Then in Line 14, w' is concatenated with the step of this turn. By the right-hand rule for solving simply connected maze, considering rightmost turns yields coordinates on the outer hull of w. Consequently, at the end of the iteration, w' is a prefix of the contour word associated to Hull(w) of length $i+1$. Finally, at the end of the loop, w' is a prefix of the contour word associated to Hull(w) of length k, that is $w' = \text{Hull}(w)$. Note that since any neighbor of W is on Hull(w), Line 15 will effectively remove every element from `N` yielding, at termination, an empty set. □

We end this section by showing that Algorithm 1 is linear in time and space. First, the quaternary tree structure is constructed in linear time (see [15]). Also, as stated before, the point W is easily computed in linear time. Consequently, computations in Line 1 are done in linear time. Next, Line 2, 4 and 5 each take constant time. Moreover, the set `N` is constructed in linear time by accessing neighborhood informations of the root in the quaternary tree structure, so Line 3 takes linear time. Now, since any coordinate has at most four neighbors, the **for** loop in Line 8 is executed at most four time per iteration of the **while** loop. Line 15 takes constant time. This is due to the fact that `N` contains at most two elements. Since instructions in Line 7, 9, 10, 11, 14 and 16 are all computed in constant time, at most $k(4c_1 + c_2)$ computations occur during the execution of the **while** loop where $k \in \mathbb{N}^+$ is the length of $\mathrm{Hull}(w)$ and $c_1, c_2 \in \mathbb{R}$ some constants, thus making Algorithm 1 linear in time. Finally, it follows from the fact that the quaternary tree structure occupies linear space that our algorithm is also linear in space.

Example 5 *Consider the word* $w = 001100322223$ *of Example 3. Then, Algorithm 1 yields* $w' = 001001223223$ *(see Figure 7). One can easily verify that* w' *is a simple path describing the outer hull of* w*, so* $\mathrm{Hull}(w) = w'$.



(a) $w' = 001001223223$ (b) $\Delta(w') = 01301101301$

Fig. 7: Outer hull of $w = 001100322223$

5 Convex hull of discrete paths

We now show how Algorithm 1 can be used to compute in linear time and space the convex hull of any discrete path. It relies on the following obvious result:

Proposition 6 *Let* $w \in \mathcal{F}^*$ *be a boundary word coding a discrete path. Then,*

$$\mathrm{Conv}(w) = \mathrm{Conv}(\mathrm{Hull}(w)).$$

Proof. If w is simple, then $\text{Hull}(w) = w$ so the claim holds. Now, suppose w is non-simple. Then by definition, $\text{Hull}(w)$ is the boundary of w. Since, $\text{Conv}(w)$ is the convex figure of minimal area containing w, it must also contain $\text{Hull}(w)$ and thus $\text{Conv}(w) = \text{Conv}(\text{Hull}(w))$.

Recall that $\text{Hull}(w)$ is simple (i.e. non self-intersecting) for any path w. Proposition 6 then yields a very simple procedure for computing the convex hull of a discrete path using Brlek et al. simple path convex hull algorithm (see [14]):

1. Start by computing $\text{Hull}(w) = w'$;
2. Compute $\text{Conv}(w')$.

It is clear that the preceding procedure computes the convex hull of a discrete path in linear time and space. Indeed, we showed in Section 4 that the first step is computed in linear time and space. Furthermore, it is shown in [14] that the second step is computed in a similar fashion.

6 Concluding remarks

We presented an algorithm for computing the outer hull of a discrete path. This led to a procedure for computing the convex hull of any discrete figure. Our algorithm is a significant improvement over the convex hull algorithm presented in [14] in the sense that computations can be made on any discrete path as opposed to non self-intersecting ones. Moreover, we proved that such computations can be made in linear time and space.

Instead of computing the outer hull of a discrete path P as described in this paper, one could want to compute the largest simply connected isothetic polygon such that all integers points on its boundary are visited by P. Although some modifications to our algorithm are necessary in order to perform such computations, the time complexity would not change.

In addition, this research begs to be generalized to three dimensional discrete spaces, that is geometry in Euclidean space $\mathbb{R}^3$ studying sets of unit cubes. Also, applications of our algorithm is not limited to convex hull problems. We plan on using it to study various path intersections problems such as primality, union, intersection and difference of discrete figures.

References

1. Gruber, P.M.: Convex and discrete geometry. Springer-Verlag (2007)
2. Sherali, H., Adams, W.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal on Discrete Mathematics **3**(3) (1990) 411–430
3. Kim, Y.S.: Recognition of form features using convex decomposition. Computer-Aided Design **24**(9) (1992) 461–476

4. Kim, M.A., Lee, E.J., Cho, H.G., Park, K.J.: A visualization technique for DNA walk plot using k-convex hull. In: Proceedings of the fifth international conference in Central Europe in computer graphics and visualization, Plzeň , Czech Republic, Západočeská univerzita (1997) 212–221
5. Okabe, A., Boots, B., Sugihara, K.: Spacial tesselations: Concepts and applications of Voronoi diagrams. Wiley (1992)
6. Graham, R.A.: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters **1**(4) (1972) 132–133
7. Chan, T.M.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete & Computational Geometry **16** (1996) 361–368
8. Yao, A.C.C.: A lower bound to finding the convex hulls. PhD thesis, Stanford University (April 1979)
9. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discrete & Computational Geometry **10** (1993) 377–409
10. Goodman, J.E., O'Rourke, J.: Handbook of discrete and computational geometry. second edn. CRC Press (2004)
11. McCallum, D., Avis, D.: A linear algorithm for finding the convex hull of a simple polygon. Information Processing Letters **9**(5) (1979) 201–206
12. Melkman, A.: On-line construction of the convex hull of a simple polyline. Information Processing Letters **25** (1987) 11–12
13. Chaudhuri, B.B., Rosenfeld, A.: On the computation of the digital convex hull and circular hull of a digital region. Pattern Recognition **31**(12) (1998) 2007–2016
14. Brlek, S., Lachaud, J.O., Provençal, X., Reutenauer, C.: Lyndon + Christoffel = digitally convex. Pattern Recognition **42** (2009) 2239–2246
15. Brlek, S., Koskas, M., Provençal, X.: A linear time and space algorithm for detecting path intersection. Theoretical Computer Science **412** (2011) 4841–4850
16. Blondin Massé, A.: À l'intersection de la combinatoire des mots et de la géométrie discrète: Palindromes, symétries et pavages. PhD thesis, Université du Québec à Montréal (February 2012)
17. Provençal, X.: Combinatoire des mots, géométrie discrète et pavages. PhD thesis, Université du Québec à Montréal (September 2008)
18. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers **EC-10**(2) (1961) 260–268
19. Kelley, J.L.: General topology. second edn. Springer-Verlag (1955)
20. Spitzer, F.: A combinatorial lemma and its application to probability theory. Transactions of the American Mathematical Society **82** (1956) 323–339
21. Vella, A.: A fundamentally topological perspective on graph theory. PhD thesis, University of Waterloo (2005)