

# Fast Exact Convex and Concave Curvature in Digital Topography

J. Askeland, W. Du

San Jose State University, One Washington Square,  
San Jose, California USA, 95192  
jake.askeland@gmail.com winncy.du@sjsu.edu

**Abstract.** Characterizing digital arcs as concave, convex, or straight with exact results has received little recent attention. We propose a simple linear algorithm which is both efficient and highlights the underlying geometric structure of digital curves. Empirical results suggest a runtime improvement over previous algorithms.

## 1 Introduction

A common problem when processing pixel-based digital images is recognition and characterization of arcs and lines as concave, convex, or straight. This is especially important when one is approximating Euclidean geometry for tasks such as curve-fitting, vectorization, shape recognition, or path-finding.

A linear time algorithm to determine end-points of digital curve segments was published in 1995 by Debled-Renneson and Reveilles [5] which represents the problem in a moving frame.

One recent algorithm by Coeurjolly, et al., for digital circle detection can be modified to segment digital arcs into convex and concave parts, however its run-time is  $\mathcal{O}(n^{4/3} \cdot \log(n))$  [4]. An earlier linear time algorithm which also handles this problem, by Roussillon, et al. [16] uses a partial convex hull algorithm and Pick's formula. Their algorithm as stated is not intended to give an exact measure, though it is sufficient to do so.

Using recent methodologies and definitions by Brlek, et al. [2], Dorksen-Reiter and Debled-Renneson [10], and Eckhardt [6], the algorithm proposed in this paper is an efficient and intuitive linear approach to the problem of exact concave or convex arc characterization.

We begin with a preliminary discussion of digital curvature, convexity, polyominoes, and combinatorics on words. We then use these concepts in conjunction with an intuitive definition of digital curvature to formulate a methodology and to derive a curvature encoding scheme. Next we provide a linear time algorithm for computing this encoding. After our algorithm is discussed, we provide an application in path-finding and make a run-time comparison to Roussillon's algorithm [16].

## 2 Preliminaries

A *curve* in this paper refers to a non-intersecting, closed curve which bounds an object or obstacle. An *arc* is a segment of a curve which can be defined as convex, concave, or straight. The interior of curves and arcs define objects or obstacles which are typically considered foreground and the space between these objects is the background.

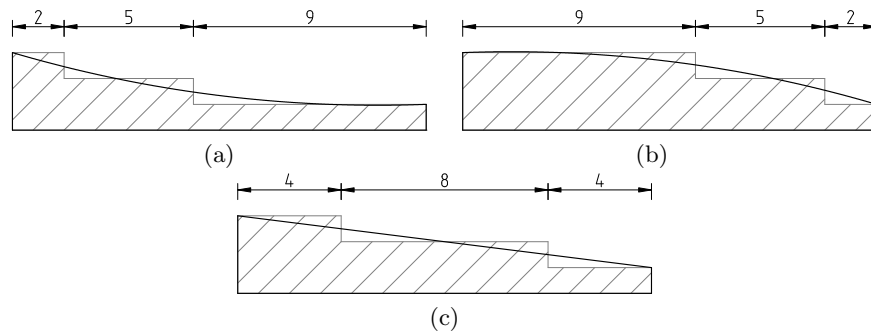


Fig. 1: Digitization of a (a) concave, (b) convex and (c) linear arc. These display how digital curvature is an approximation of their continuous counterparts and give insight into our methodology.

In Euclidean 2-space –  $\mathbb{E}^2$  – an arc along a curve is convex iff it is a closed, connected portion of a *convex set*<sup>1</sup>  $S$  contained within the interior of a curve. We may also say a straight line between any two points on that arc is contained inside the arc because it is contained in the convex set to which the arc belongs. We will distinguish, however, between a straight arc and a convex arc by saying a straight arc has points satisfying the chord property [15]. On a concave arc there exists at least two points such that a line between them has points which do not belong to the interior of the curve or to the curve itself.

In a grid space –  $\mathbb{Z}^2$  – such as that of a digital image, characterizing arcs as concave, convex, or straight is less straightforward. Our definitions from  $\mathbb{E}^2$  still apply but with exceptions. A mapping from  $\mathbb{E}^2$  to  $\mathbb{Z}^2$  causes diagonal lines to become steps. See figure 1 for examples.

The inflection of the continuous arcs in figures 1a and 1b determine if their digital representations have a longer or shorter step size. Digital cells are darkened if their area is occupied by an average pixel value below 50% of maximum. 50% of the area of a cell must fall on the interior of a curve in  $\mathbb{E}^2$  to be considered a member of the resultant digital object. Step size is then a direct translation of the inflection of a continuous arc.

<sup>1</sup> A “set such that a [line segment] connecting any two points of the set is completely contained within the set” [3,9]

Perhaps the most important factor involved in the process of digital curvature analysis is the use of a good definition of digital convex and concave curvature. In 2004 Dorksens-Reiter and Debled-Rennesson [10] developed such a definition using arithmetic geometry.

**Definition 1 (Digital Convex and Concave Curves[10]).** *A curve “is called convex (concave) whenever the sequence of slopes of fundamental segments is increasing (decreasing)...”*

Recent developments in combinatorics on words as a geometric computation tool have made conceptualization of the methodology used here more intuitive and accessible. Words have been used to describe digital arcs with a variety of alphabets to solve many problems efficiently, such as polyomino area and center of mass, digital straightness, and digital hull convexity [1,2,8].

## 2.1 Polyominoes

Throughout this paper we describe digital contours which enclose some area made up of unit squares. Because of the successes of recent research using words to encode the contours of polyominoes, we use the definitions in the following subsections as a basis for our final methodology.

A walk along the lines of a square lattice is a *path*. A path is called a self-avoiding walk (or SAW) iff the walk does not intersect itself [13]; this occurs when walking the perimeter of a *simply connected polyomino* (figure 2.a). A closed path is a *contour* [2,13]. If two squares on a lattice, inside a contour, share a common edge then they are said to be *edge-connected* [13]. See figure 3a for an example of a walk like the kind in this paper.

A *polyomino* is defined as a set of edge-connected unit squares on a lattice, inside a contour [2,13]. Some special types are polyominoes with holes or multiple points and polyominoes which are corner-connected. Figure 2 depicts these types.

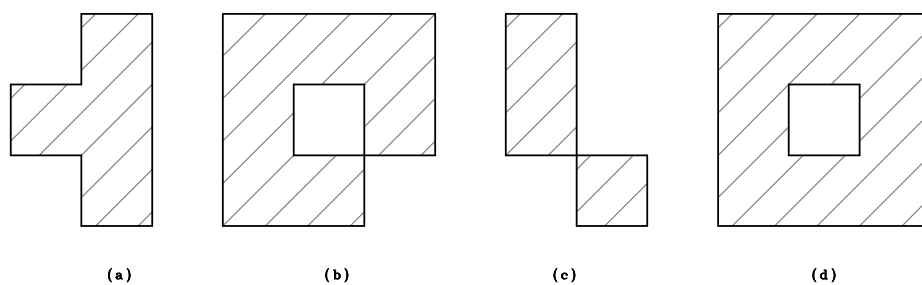


Fig. 2: Examples of (a) simply connected, (b) multiple point, (c) corner connected polyominoes and (d) a polyomino with a hole. Polyominoes are a useful conceptualization when working with digital image masks.

## 2.2 Combinatorics on Words

The rules governing the combinatorics of words are defined in M. Lothaire’s books [12,13] as follows. An alphabet  $\Sigma$  is a set of letters, which, for the purposes of this paper is a finite set. A finite sequence of concatenated letters in  $\Sigma$  may be defined as a word  $w$ . This word would then contain the letters  $w_1w_2w_3\dots w_n$ ,  $w_i \in \Sigma$ . The length  $k$  prefix of  $w$  is  $w[0\dots k - 1]$ . The length of  $w$  is denoted  $|w|$ . The count of a specific letter  $l$  in  $w$  is denoted  $|w|_l$ .

Again, using Lothaire’s notation [12,13], the set of all words will be noted as  $\Sigma^*$ . The *empty word* will be  $\varepsilon$ , making the set of all non-empty words  $\Sigma^+ = \Sigma^* - \varepsilon$ .

Patterns such as  $A\dots A$  stand for a set of consecutive letters  $A$  of arbitrary count 0 to  $\infty$ .

## 2.3 Methodology

The technique presented in this paper seeks out digital steps by pattern matching within each obstacle’s enclosing curve. This can be done by first walking around an obstacle and encoding its curve in a relative-direction word, with an alphabet such as  $\{L, R, F\}$ , meaning “Compared to the immediately previous edge, we are traversing this edge in a  $\{Left, Right, Forward\}$  orientation.” This encoding is rotationally invariant, allowing for pattern matching without alphabet rotations such as those used in [2,5]. The  $\{L, R, F\}$  alphabet can also be obtained by simple iterative mappings from 4- and 8-directional Freeman chain codes or from the alphabet described in [2].

Now that we have rotational invariance, we can reclassify  $L$  and  $R$  as *fundamentally -concave* or *-convex* (see definition 2, below) and use a new alphabet  $\{C, X, F\}$ .

**Definition 2 (Fundamental Curvature).** *An intersection on a grid which is a member of a polyomino’s contour is a fundamentally convex vertex if it is touching exactly one cell belonging to a polyomino. It is a fundamentally concave vertex if either it is touching exactly three polyomino cells or if it is touching exactly two polyomino cells who are opposite one another. It is neither in all other cases.*

Words of the  $\{C, X, F\}$  alphabet, called *intermediate-curvature words*, have the desirable feature that steps along the boundary of a polyomino are encoded in only two patterns:  $F\dots FCX$  or  $XCF\dots F$ .

Mapping from  $\{L, R, F\}$  to  $\{C, X, F\}$  alphabets is fast because algorithms for obstacle tracing typically work by applying a  $90^\circ$  or  $270^\circ$  rotation when they encounter a wall. Because the trace maintains its desired rotation as either clockwise or counter-clockwise we can say that if any  $90^\circ$  rotation corresponds to a convex turn, all  $90^\circ$  rotations do. The same is true of  $270^\circ$  rotations. The two are mutually exclusive, so when it is known which is which, we can apply the appropriate mappings  $L \Rightarrow X : C$  and  $R \Rightarrow C : X$ .

Within intermediate curvature-words, steps are found which take the form  $F\dots FCX$  or  $XCX\dots F$ , graphically corresponding to what we will now call *up-steps* and *down-steps*, respectively. These steps then collect as *staircases* – sets of adjacent steps of the same form – such as  $FFFFFFCXFFFCX$  and  $XCXCFFF$  (both appear in figure 3c). Algorithms using these words will need to collect staircases, into a data structure containing both the index of each  $X$  and the preceding (in the case of an up-step) or the succeeding (for a down-step) number of  $F$ s in the pattern (later referred to as an  $F$ -count).

An  $F$ -count is the unit of measure we link to definition 1. As  $F$ -counts increase in an up-step or decrease in a down-step, we are on a convex curve. As they decrease on up-steps or increase on down-steps, we are on a concave curve.

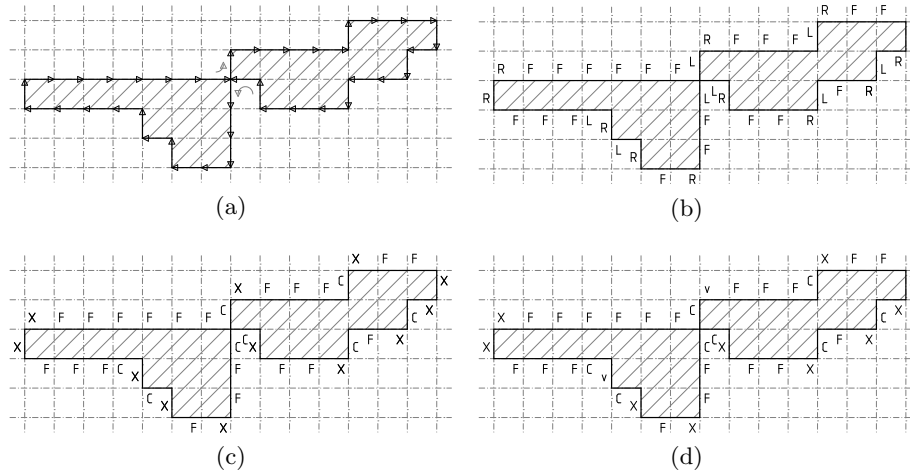


Fig. 3: (a) A closed path SAW around an obstacle. Arrows point in the direction of traversal and encoding. (b) relative-direction word, (c) intermediate-curvature word and (d) final-curvature word encoding of an obstacle.

In the case of  $FFFFFFCXFFFCX$ , figure 3c shows two up-steps of decreasing length, thus the first  $X$  is a fundamentally convex vertex along a concave segment. Likewise,  $XCXCFFF$  is a pair of down-steps of increasing length and the second  $X$  is a member of a concave segment. We will again re-classify these words under a new alphabet  $\{C, X, F, v\}$ , where  $v$  now stands for a fundamentally convex vertex along a concave segment:  $FFFFFFCvFFFCX$ ,  $XCvCFFF$ . Figure 3d displays these changes with  $v$ 's. Words of this alphabet are called final-curvature words and those substrings containing  $v$ ,  $CF\dots FC$ , or  $FCF$  are concave, and the rest are convex.

### 3 Encoding

Using the concepts in the preceding sections and our definitions of convexity, concavity, and straightness in a grid-space geometry, we derive an algorithm for classifying digital arc vertices as convex or concave and we suggest steps to take in order to also detect and encode digital straightness.

Algorithms (1) and (2) detect and store indices and lengths of up-steps and down-steps in contiguous sets, ordered by index. An index refers to the location of the X in a step pattern.

Algorithm (3) then determines when two contiguous steps make a concave or a convex arc and overwrites the appropriate index in the curvature-word. We are left with a final-curvature word of the alphabet  $\{X, C, F, v\}$  (see figure 3d).

GETCURVATUREWORD converts a relative-direction word into a curvature-word by locating a vertex, testing its fundamental curvature, and re-classifying all vertices accordingly (see figures 3b and 3c). The input argument “Point  $p$ ” refers to a grid-intersecting vertex along the curve of a polyomino.

---

#### Algorithm 1: FINDCXCURVATURE( $C$ )

---

**Input:** Intermediate curvature-word  $C$  from alphabet  $\{C, X, F\}$   
 Let  $CXindex$  be an integer array of arbitrary size;  
 Let  $CXlength$  be an integer array of arbitrary size;  
 Let  $fCount := 0$ ;  
**for**  $i := 1$  **to**  $|C| - 1$  **do**  
   **if**  $SUBSTRING(C_i \text{ to } C_{i+1}) \neq CX$  **then**  
     **if**  $R_i = F$  **then**  $fCount := fCount + 1$ ;  
     **else**  $fCount := 0$ ;  
      $i := i + 1$ ;  
   **end**  
   **else**  
      $i := i + 2$ ;  
     Append  $i - 1$  to  $CXindex$ ;  
     Append  $fCount$  to  $CXlength$ ;  
      $fCount := 0$ ;  
   **end**  
**end**  
**return**  $\{CXindex, CXlength\}$ ;

---

### 4 Path-finding In Bitmaps

Since 1989 [7] robots have been able to map their surroundings into pixel-based formats. Recent game AI have topographic information but are typically held to

---

**Algorithm 2:** FINDXCCURVATURE( $C$ )
 

---

**Input:** Curvature-word  $C$  from alphabet  $\{C, X, F\}$   
 Let  $\text{XCindex}$  be an integer array of arbitrary size;  
 Let  $\text{XClength}$  be an integer array of arbitrary size;  
 Let  $\text{fCount} := 0$ ;  
**for**  $i := 1$  **to**  $|C| - 1$  **do**  
   **if**  $\text{SUBSTRING}(C_i \text{ to } C_{i+1}) \neq \text{XC}$  **then**  $i := i + 1$ ;  
   **else**  
     Let  $j := i$ ;  
      $i := i + 2$ ;  
     Let  $k := i$ ;  
     **while**  $C_i = F$  **and**  $i < |C|$  **do**  
        $i := i + 1$ ;  
     **end**  
     Append  $j$  to  $\text{XCindex}$ ;  
     Append  $i - k$  to  $\text{XClength}$ ;  
   **end**  
**end**  
**return**  $\{\text{XCindex}, \text{XClength}\}$ ;  


---

---

**Algorithm 3:** SETCONCAVECURVATURE( $\mathcal{M}, R, P$ )
 

---

**Input:** Relative-word  $R$  from alphabet  $\{L, R, F\}$ , Point array  $P \subset \mathcal{M}$   
   of Points along some curve, Traversability Map  $\mathcal{M} \subset \mathbb{Z}^2$   
 Let  $C := \text{GETCURVATUREWORD}(\mathcal{M}, R, P)$ ;  
 Let  $\{\text{CXindex}, \text{CXlength}\} := \text{FINDCXCURVATURE}(C)$ ;  
 Let  $\{\text{XCindex}, \text{XClength}\} := \text{FINDXCCURVATURE}(C)$ ;  
**for**  $i := 2$  **to**  $|\text{CXindex}|$  **do**  
   Let  $\text{isAdjacent} := \text{CXindex}_i - \text{CXindex}_{i-1} = \text{CXlength}_i + 2$ ;  
   **if**  $\text{isAdjacent}$  **and**  $\text{CXlength}_{i-1} \geq \text{CXlength}_i$  **then**  
      $C_{\text{CXindex}_{i-1}} := v$ ;  
      $i := i + 1$ ;  
   **end**  
**end**  
**for**  $i := 2$  **to**  $|\text{XCindex}|$  **do**  
   Let  $\text{isAdjacent} := \text{XCindex}_i - \text{XCindex}_{i-1} = \text{XClength}_{i-1} + 2$ ;  
   **if**  $\text{isAdjacent}$  **and**  $\text{XClength}_{i-1} \leq \text{XClength}_i$  **then**  
      $C_{\text{XCindex}_i} := v$ ;  
      $i := i + 1$ ;  
   **end**  
**end**  


---

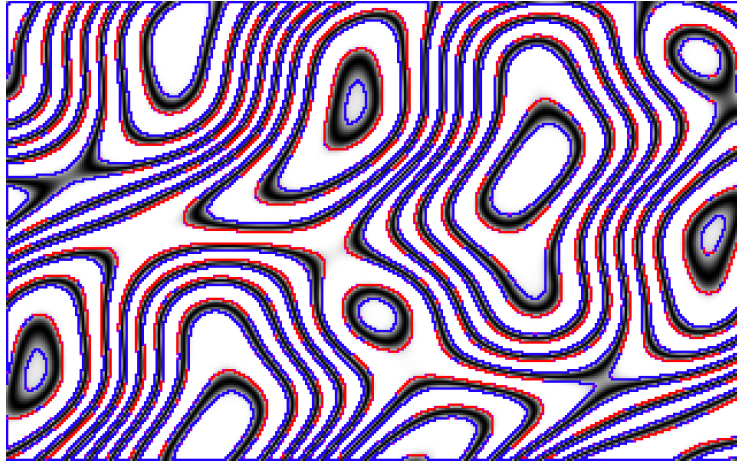


Fig. 4: Curvature about a ripple. Convex segments in red, concave segments in blue, and convex points in purple.

maps which have been created and path-mapped before being released to players. By thinning the vertices used in path-finding, we reduce the time required to acceptable end-user limits in both cases. Finding these vertices quickly, then, is our goal.

To do so we can model our map as polygons smattered with polygonal obstacles, and take advantage of the concept of Euclidean measure, which states that in a Euclidean plane, shortest paths are “non-self-intersecting polygonal path[s] with corners at obstacle vertices” [9].

Since only convex vertices are required for shortest paths in Euclidean 2-space, we can use our algorithm to find convex end-points, then construct shortest path trees. Each convex vertex in the map needs only appear once in such trees so storage of a tree can be handled in an array. Thus for all-pairs shortest paths, we have a *shortest path matrix*, whose cells contain the information “to get from vertex  $i$  to vertex  $j$ , which vertex does one travel to next?”

Other notable research in path finding and path planning includes [11,14,17].

Figure 5 shows a result of the technique detailed in this paper as used on images from [19]. The map on the left shows a section of a building, plotted autonomously by robots. Our algorithm was used to determine the curvature of the impassible regions of the map under the consideration the robots would have a footprint of approximately 25 square pixels. The map on the right depicts vertices at either end-point of each convex segment of the map’s contour. These make up the minimum set of vertices required for path-finding via a path matrix.

In grid-space, the smallest unit is the unit cell and so Euclidean distance is defined within the tolerance of  $\pm 1/2$  a unit square. An algebraic proof shows that if integer path weights are scaled by  $2l - 1$  where  $l$  is at least the maximum number of vertices visited along a shortest path (or simply  $2n - 1$  as it is not

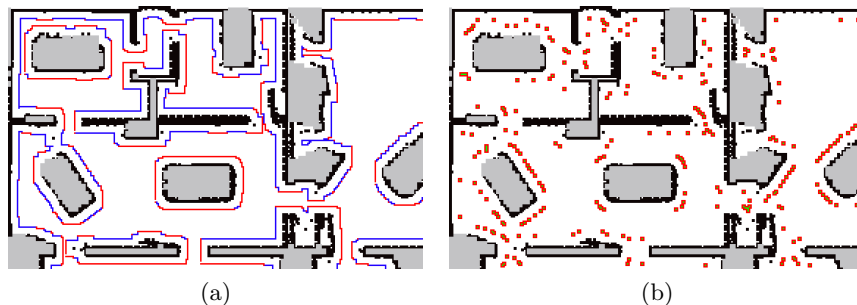


Fig. 5: A portion of a robot-generated map from after curvature processing. (a) shows curvature (red for convex, blue for concave), (b) shows points useful in path-finding. If curvature was not considered before determining the points necessary for path-finding, one could expect about twice the points and thus about four times the run-time of an  $\mathcal{O}(n^2 + nm)$  all pairs path-finding algorithm such as an all-pairs implementation using Thorup’s [18].

generally efficient to accurately find  $l$  before finding all-pairs shortest paths), we ensure the decision between true Euclidean shortest paths and the output of any pathfinding algorithm is not affected by rounding errors. The maximum value of an integer and the size of the intended images should be considered before using this approach.

Then, for graphs with integer weights, such as ours, Thorup’s algorithm [18] (linear in time for single-source shortest paths) yields an all-pairs shortest paths algorithm in  $\mathcal{O}(n^2 + nm)$  time. With this and the algorithm described here, an all-pairs shortest paths matrix can be calculated in the most efficient manner currently available.

## 5 Results

An implementation of our algorithm was compared with Roussillon, et al.’s algorithm at  $\alpha = 0.0^2$  (provided by Tristan Roussillon). In order to make a direct comparison, our implementation did not incorporate straight-line detection. The results (figure 6) suggest a 10.2 times average improvement. Measurements were taken at the point of execution at which the desired inputs for each algorithm were available: a relative direction word, list of points, and image for ours and a list of points generated from an 8-connected Freeman chain code for Roussillon’s.

Roussillon’s method of calculating a partial convex hull was designed to allow smoothing and this method is more costly than our method of combinatorics on words, which accounts for the difference in run-time.

<sup>2</sup> In Roussillon, et al. [16],  $\alpha$  is a relaxation threshold for curvature considered to be concave or convex. When  $\alpha = 0.0$ , all curvature is either concave or convex and no smoothing occurs, which is equivalent to our algorithm when no straight-line detection is incorporated.

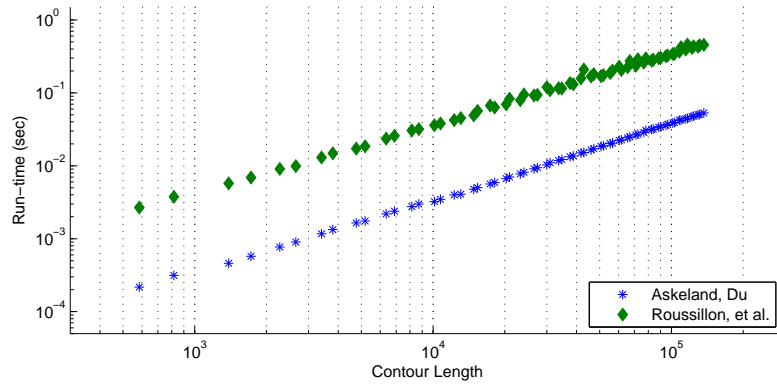


Fig. 6: Run-time comparison of Askeland, Du and of Roussillon, et al. Roussillon’s partial convex hull method allows smoothing and is more costly than our method.

## 6 Conclusion

The algorithm supplied generates a word-encoding of exact convex and concave curvature in grid-space thus determining this curvature faster than previously known algorithms. Further research can be done to fully incorporate digitally straight segments and to precisely apply a smoothing coefficient, which would make this methodology a member of a body of work toward an emerging digital calculus.

## 7 Acknowledgments

Kate Isaacs and Alex Tsui for their helpful discussions, and Tristan Roussillon for adapting and supplying an implementation of his algorithm for comparison with ours.

## References

1. S. Brlek, G. Labelle, and A. Lacasse. The discrete green theorem and some applications in discrete geometry. *Theoretical Computer Science*, 346(2):200–225, 2005.
2. S. Brlek, X. Proven, and J.-O. Lachaud. Combinatorial view of digital convexity. In *Discrete Geometry for Computer Imagery*, volume 4992/2008 of *Lecture Notes in Computer Science*, pages 57–68. Springer Berlin / Heidelberg, 2008.
3. B. B. Chaudhuri and Azriel Rosenfeld. On the computation of the digital convex hull and circular hull of a digital region. *Pattern Recognition*, 31(12):2007–2016, 1998.

4. D. Coeurjolly, Y. Gérard, J.-P. Reveillés, and L. Tougne. An elementary algorithm for digital arc segmentation. *Discrete Applied Mathematics*, 139(1-3):31–50, 2004.
5. Isabelle Debled-Rennesson and Jean-Pierre Reveillés. A linear algorithm for segmentation of digital curves. *IJPRAI*, 9(4):635–662, 1995.
6. Ulrich Eckhardt. Digital lines and digital convexity. pages 209–228, 2001.
7. A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.
8. Panama Geer, Harry W. McLaughlin, and Keith Unsworth. Cellular lines: An introduction. In Michel Morvan and Éric Rémila, editors, *Discrete Models for Complex Systems, DMCS'03*, volume AB of *DMTCS Proceedings*, pages 167–178. Discrete Mathematics and Theoretical Computer Science, 2003.
9. Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of discrete and computational geometry*. CRC Press, Inc., Boca Raton, FL, USA, 1997.
10. Hélène Reiter Dorksén and Isabelle Debled Rennesson. Convex and Concave Parts of Digital Curves. In none, editor, *Geometric Properties from Incomplete Data*, page 15 p. Kluwer, 2004. Contribution à un ouvrage.
11. D. Kuan, J. Zamiska, and R. Brooks. Natural decomposition of free space for path planning. In *1985 IEEE Int. Conf. Robotics and Automation*, volume 2, pages 168–173, Coleman Avenue, Santa Clara, CA, 3 1985. FMC Corporation.
12. M. Lothaire. *Combinatorics on Words*. Cambridge University Press, New York, NY, USA, 1997.
13. M. Lothaire, editor. *Applied Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, New York, NY, USA, 2005.
14. Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *ACM Journal on Experimental Algorithmics*, 11:2.8, 2006.
15. A. Rosenfeld. Digital straight line segments. *IEEE Trans. Comput.*, 23(12):1264–1269, 1974.
16. Tristan Roussillon, Isabelle Sivignon, and Laure Tougne. Robust Decomposition of a Digital Curve into Convex and Concave Parts. In *International Conference on Pattern Recognition (ICPR 2008)*, December 2008.
17. Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3310–3317, 1994.
18. Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 5 1999.
19. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.